Optimization Techniques for Land-use Planning

Optimization Project
N. Murphy

September 17, 2018

Contents

1 Problem Description

The aim of this project is to explore various optimization techniques in order to optimally allocate a set of 9 pre-specified land-use types to a 400ha region subject to mutiple objectives and constraints. The idea is to develop the brackish peat meadow which is criss-crossed by water and populated by vast amounts of vegetation and bird life as part of a nature and recreation development plan. As part of the development plan, three additive objectives were identified to transform the current region subject to constraints such as the total amount of land allocated to each of the land-use types and to certain areas of land in the region which cannot be converted from their current uses.

2 Formulation of the problem

In order to set up the problem, the 400ha region is divided into 1ha blocks of land each of which containing one single land-use type. This constitutes the first set of constraints to the optimisation problem. This allows us to illustrate the region as a 20x20 grid and visualise the allocated land-use types of each block in the grid

giving a convenient representation of the region and planned developments as given by the optimal solutions we obtain. Figure 1 below illustrates the 400ha region as a grid and the land-use types which currently exist in the region.

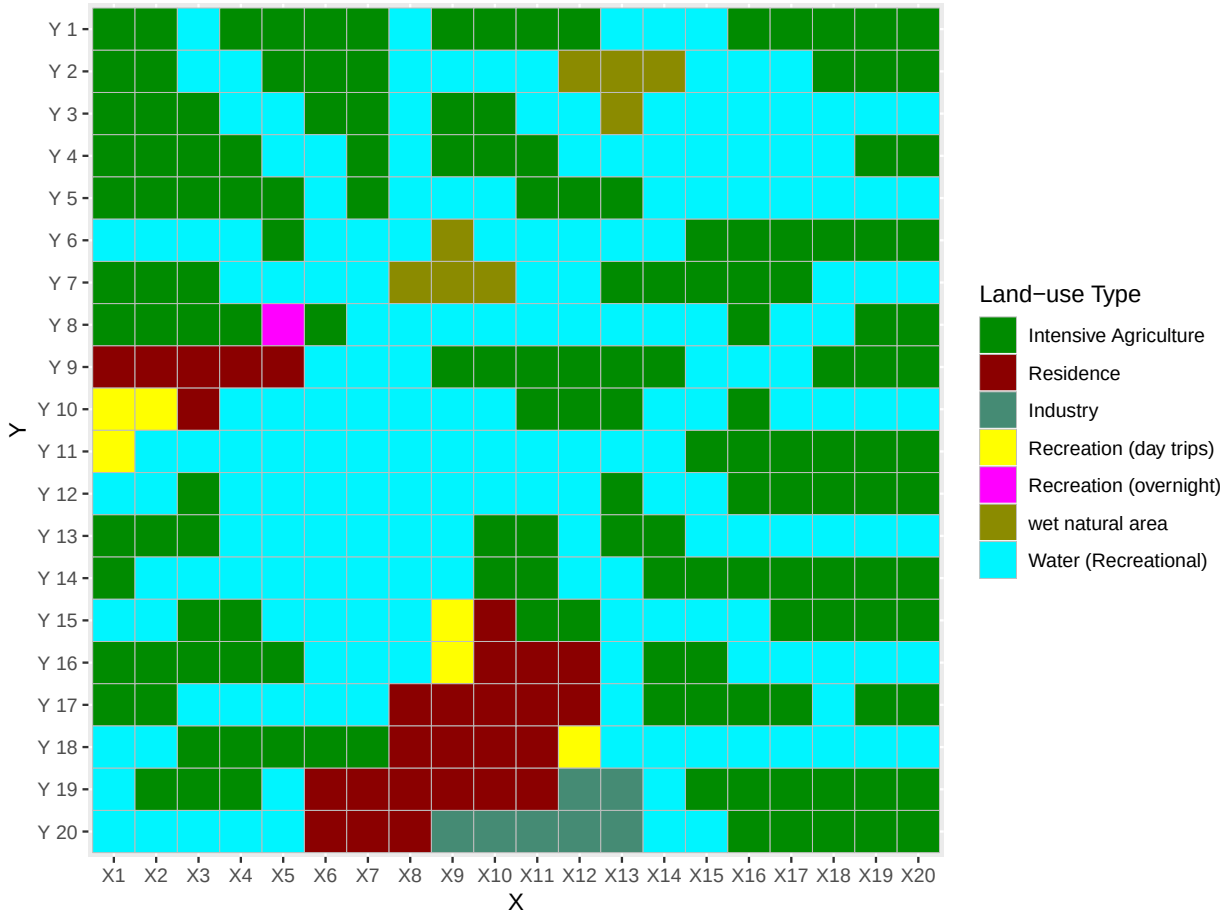


Figure 1: A simplified illustration of the 400ha region showing the land types that currently occupy each 1ha block of land.

As mentioned above, there are three objectives that were identidied as part of the development plan. They are as follows:

1) maximise the natural value of the region
2) maximise the recreatioinal value of the region
3) minimise the cost of changing land use

In order to structure the problem efficiently, we will specify a lanuse map as a binary vector of 3600 (400x9) elements. Here, each element will specify the row, the column and the land-use type allocated. For instance, the first cell of the grid (1,1) will be represented by the first 9 elements of the 3600 long vector. 8 of the 9 elements will be zero and one single element, say the k_{th} element, will hold a one which specifies that the first grid cell will be allocated landuse k. The next 9 elements in the vector will represent the second grid cell and so on. We will go from left to right in each row of the grid amd proceed row by row down the grid in this manner. This vector representation will make it a ot easier to construct the problem as an integer program. Hence, we will have 3600 binary decision variables in our LP.

The plot below illustrates the amount of value added for each grid cell to nature (a) and recreation (a), (b)

and (c). Table 1 then specifies how each of these value maps are incorporated into the objective functions for nature and recreation.

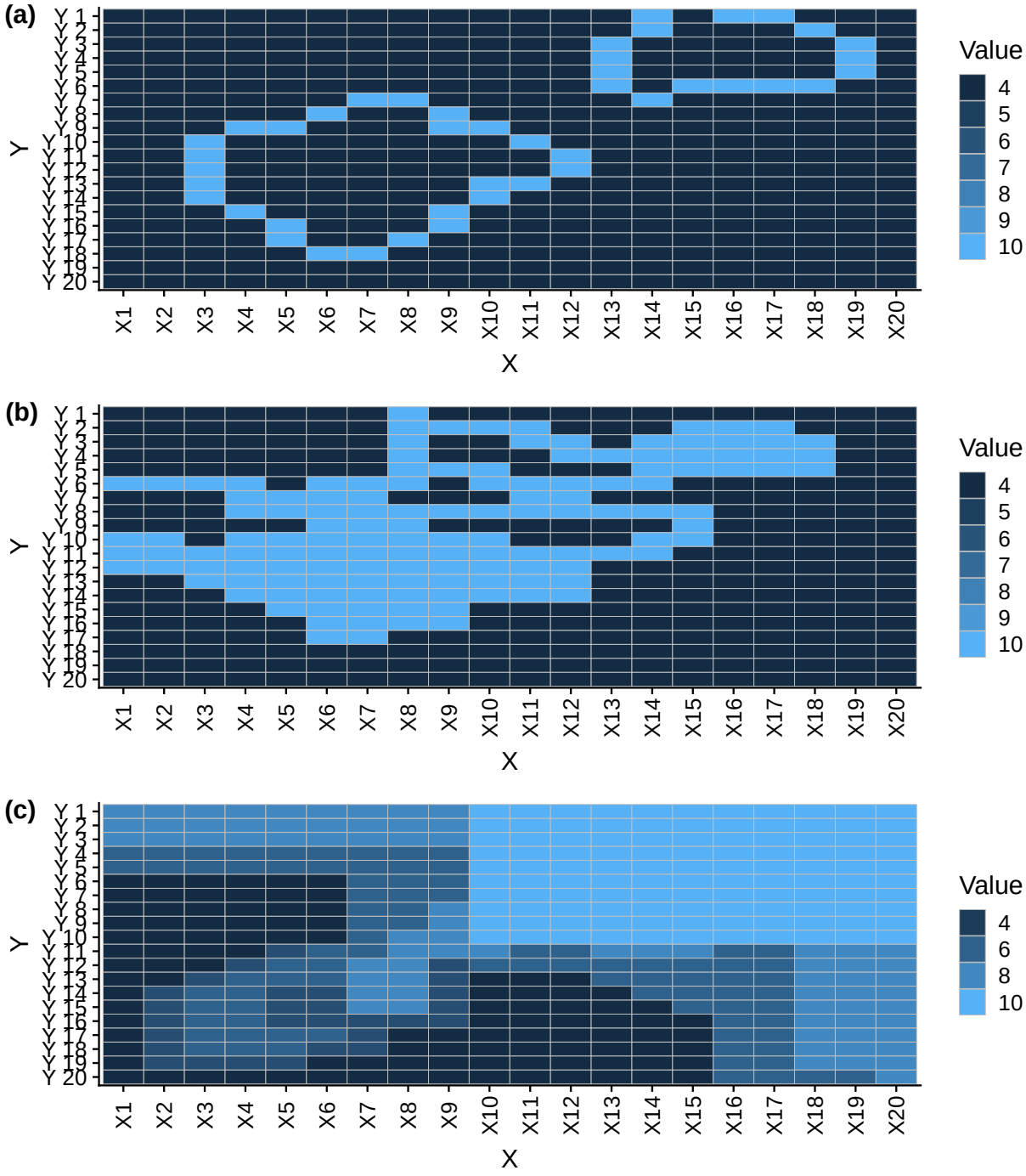


Figure 2: Value maps for nature (a) and recreation (a), (b) and (c).

Table 1: Values for nature and recreation per land-use type..

	Land-use type (k)	Nature value	Recreation value
1	1 intensive agriculture	4	6
2	2 extensive agriculture	map figure 3(a)	map figure 2(a)
3	3 residence	3	3
4	4 industry	1	1
5	5 recreation (day trips)	5	map figure 2(b)
6	6 recreation (overnight)	5	map figure 2(c)
7	7 wet natural area	map figure 2(a)	7
8	8 water (recreational use)	7	map figure 2(b)
9	9 water (limited access)	map figure 2(a)	1

2.1 Problem Constraints

All of the objectives satisfy the same set of constraints. The first set of constraints we will enforce is that each cell in the grid must be allocated only one land use. Thus, the sum of the 3600 long vector for a grid map should add up to 400 as one land type falls in each cell and the rest of the elements are zero. This is implemented by creating a constraint that ensures the sum of all decision variables is equalt to 400.

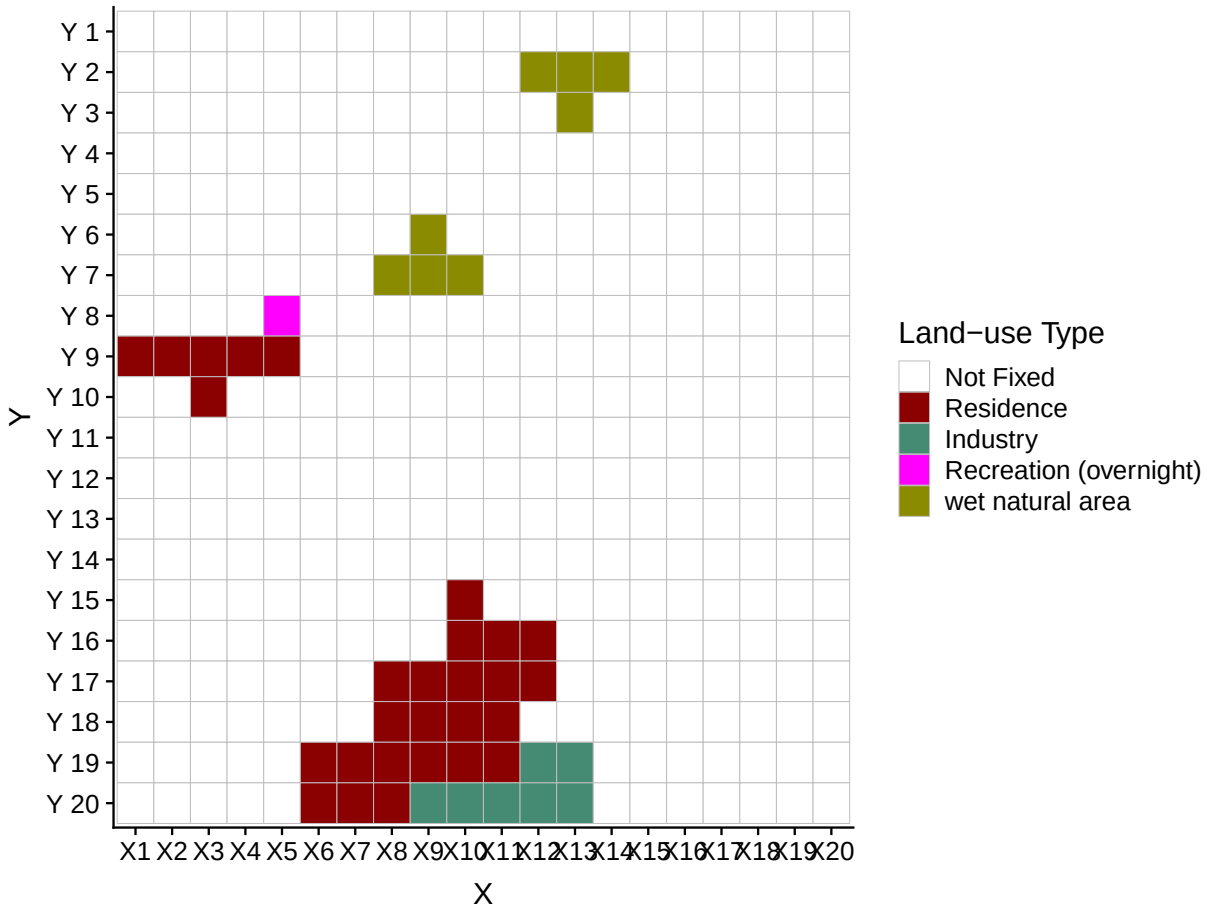
The next set of constraints pertains to the fact the only one landuse can be allocated to each cell. To implement this, we set up 400 constraints whereby each constraint will consisit of the sum of decisions variables that represent a given cell and those variables must sum up to one. For example, the sum of the first 9 decision variables representing the first cell which are the first 9 elements of the 3600 long vector must have a sum equal to one. The same goes for the next 9 elements after that and we continue on in this manner for all 400 nine element vectors of decision variables.

The next set of constraints pertain to the minimum and maximum total number of grid cells in the region that can be allocated to each land type k. This results in 18 constraints, a lower and upper bound for each land-use type. These are given in the following table:

Table 2: Minimum and maximum size per land-use type.

	Land-use type	Minimun	Maximum
1	Intensive Agriculture	80	150
2	Extensive Agriculture	20	65
3	Residence	20	45
4	Industry	5	15
5	Recreation (day trips)	0	70
6	Recreation (overnight)	0	35
7	wet natural area	0	30
8	Water (Recreational)	120	150
9	Water (limited access	0	60

The final set of constraints concern the fixed constraints as specified in the figure below. There are a total of 44 fixed contraints which fix a certain land-use type to cells in the grid as illustrated in the figure.



All in all there are a total of 463 constraints that need to be satisfied.

3 Linear Programming

In order to implement the LP we will use the *Rglpk* package in R. The *Rglpk_solve_LP* function takes on 6 input parameters. The first input is the objective function which we will specify as a 3600 element vector of coefficients pertaining to a given objective which we will discuss in more detail in the next 3 subsections. The seocond input is the constraint matrix containing the coefficients of the constraint matrix which we will input as a simple triplet matrix using the *simple_triplet_matrix* function from the slam package which is used due to the sparseness of the constraints. The rest of the inputs are the constraint inequalities as a vector, the right hand side of the constraints that each cosntraint must satisfy and whether it is a maximization or minimization problem specified as a boolean. We will now discuss the objective functions (coefficients) for each of the three objectives.

3.1 Objective 1: Natural Value

The natural value of the of the region is calcluated by first specifying the amount of natural value that each grid cell contributes to the overall natural value based on the land use type that is allocated to that cell. The natural value is the computed by summing up the total natural value in the 400 cell grid area given by the function

$$f_{nature}(\boldsymbol{\mu}) = \sum_{r=1}^{20}\sum_{c=1}^{20}\sum_{k=1}^{9} a_{rck}\mathscr{X}_{rck}$$

where the value assigned to each land-use type k in a given grid cell (r, c) is a_{rck}. $\mathscr{X}_{rck}$ will take on a value of 1 or 0 depending on whther land type k has been allocated to grid cell (r, c). a_{rck} is specified in table 2 where the value will either be constant over all grid cells or will be based on the value maps illustrated in figure 2 and thus the natural value will depend on the row (r) and column (c) index of the grid.

Implementing *Rglpk_solve_LP* on the natural value objective function results in an optimal total nature value given by 2773. The plot of the grid for the optimal natural value for the area is plotted below:

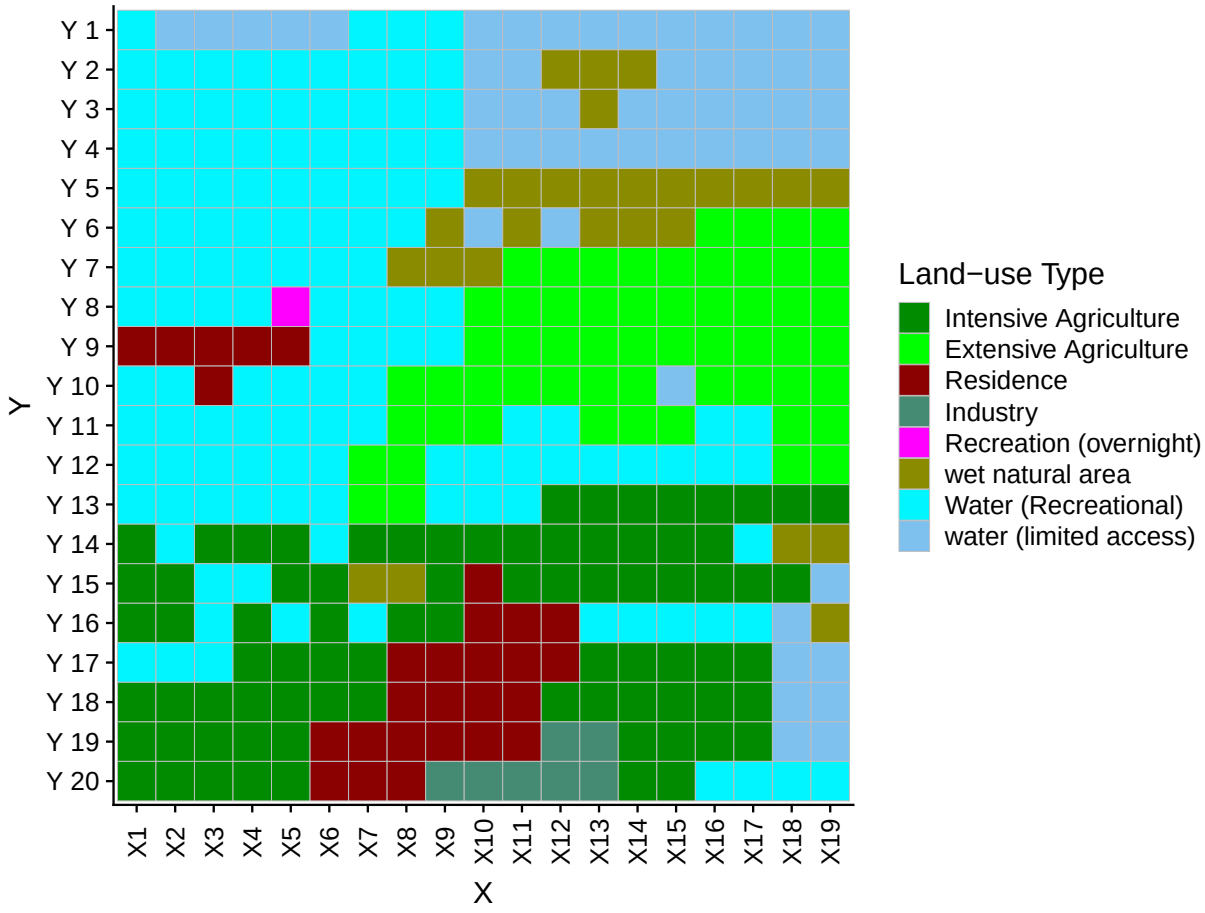


Figure 3: Optimal land-use allocation for maximizing the natural value of the region.

3.2 Objective 2: Recreational Value

The recreational value of the of the area is calcluated similarly to the natural value by first specifying the amount of recreational value that each grid cell contributes to the overall recreational value based on the land use type that is allocated to that cell. The recreational value is the computed by summing up the total recreational value in the 400 cell grid area given by the function

$$f_{recreational}(\boldsymbol{\mu}) = \sum_{r=1}^{20}\sum_{c=1}^{20}\sum_{k=1}^{9} a_{rck}\mathscr{X}_{rck}$$

where the value assigned to each land-use type k in a given grid cell (r, c) is a_{rck}. $\mathscr{X}_{rck}$ will take on a value of 1 or 0 depending on whther land type k has been allocated to grid cell (r, c). Again, as in the natural value case, a_{rck} is specified in table 1 where the value will either be constant over all grid cells or will be based on the value maps in figure 2 and thus the recreational value will depend on the row (r) and column (c) index of the grid.

Implementing *Rglpk_solve_LP* on the recreational value objective function results in an optimal (maximum) total recreational value given by 3151. The plot of the grid for the optimal recreational value for the area is plotted below:

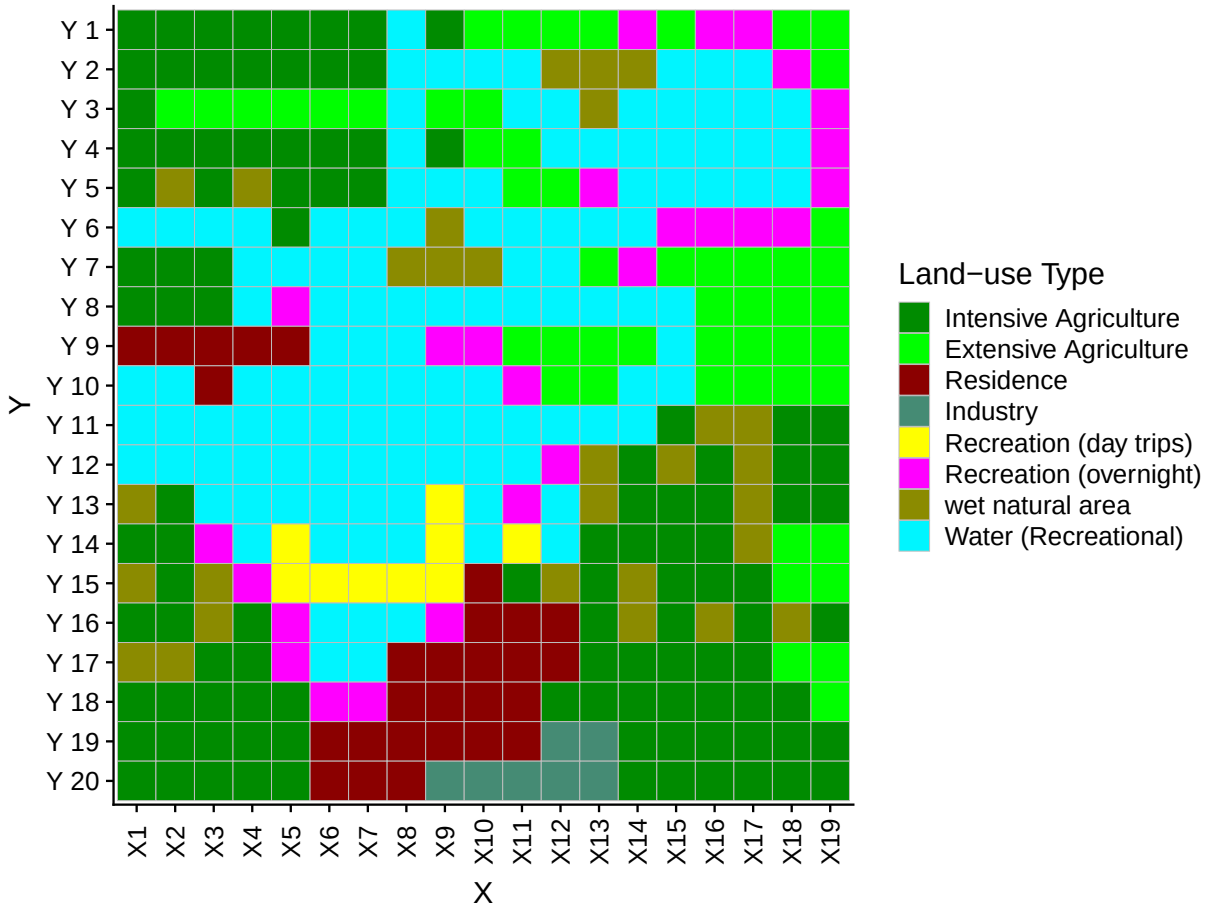


Figure 4: Optimal land-use allocation for maximizing the recreational value of the region.

3.3 Objective 3: Cost of Changing Land Uses

The third objective concerns the minimization of the cost of changing a land use from its current use, as illustrated in Figure 1, to a new land use type. How it works is that for each pair of land uses, l and k, there is a cost associated with changing from land use l to land use k which are illustrated in table 3 below. It must be noted that some changes are infeasible and hence we have set infeasible conversions to have a cost of

-1000000. We will implement the problem as a maximization problem and thus maximize revenue which is equivalent to minimizing costs.

Table 3: Values for nature and recreation per land-use type..

	Land-use type	1	2	3	4	5	6	7	8	
1	Intensive Agriculture	0	-75	150	150	-225	0	-150	-300	
2	Extensive Agriculture	75	0	150	150	-150	75	-75	-225	
3	Residence	-1000000	-1000000	0	-1000000	-10000	-10000	-1000000	-1000000	-1
4	Industry	-1000000	-1000000	-1000000	0	-10000	-10000	-1000000	-1000000	-1
5	Recreation (day trips)	150	75	3	300	0	150	0	-150	
6	Recreation (overnight)	0	-75	150	150	-150	0	-150	-300	
7	wet natural area	0	75	225	225	-75	150	0	-75	
8	Water (Recreational)	100	100	-1000000	-1000000	-1000000	-1000000	0	0	
9	water (limited access)	100	100	-1000000	-1000000	-1000000	-1000000	0	0	

Hence, for a given grid cell (r, c), the cost of changing a land use from land use l to land use k is represented by a_{rck}. All conversion costs for given land-use types are then added up to arrive at a total cost in a similar fashion to the previous objectives. Positive values will be associated with revenues and negative values as costs. The objective function is then specified as follows

$$f_{cost}(\boldsymbol{\mu}) = \sum_{r=1}^{20} \sum_{c=1}^{20} \sum_{k=1}^{9} a_{rck} \mathscr{X}_{rck}$$

Implementing *Rglpk_solve_LP* on the cost objective function results in an optimal (maximum) total revenue (minimum cost) value given by 11950. The plot of the grid for the optimal revenue value for the area is illustrated below:

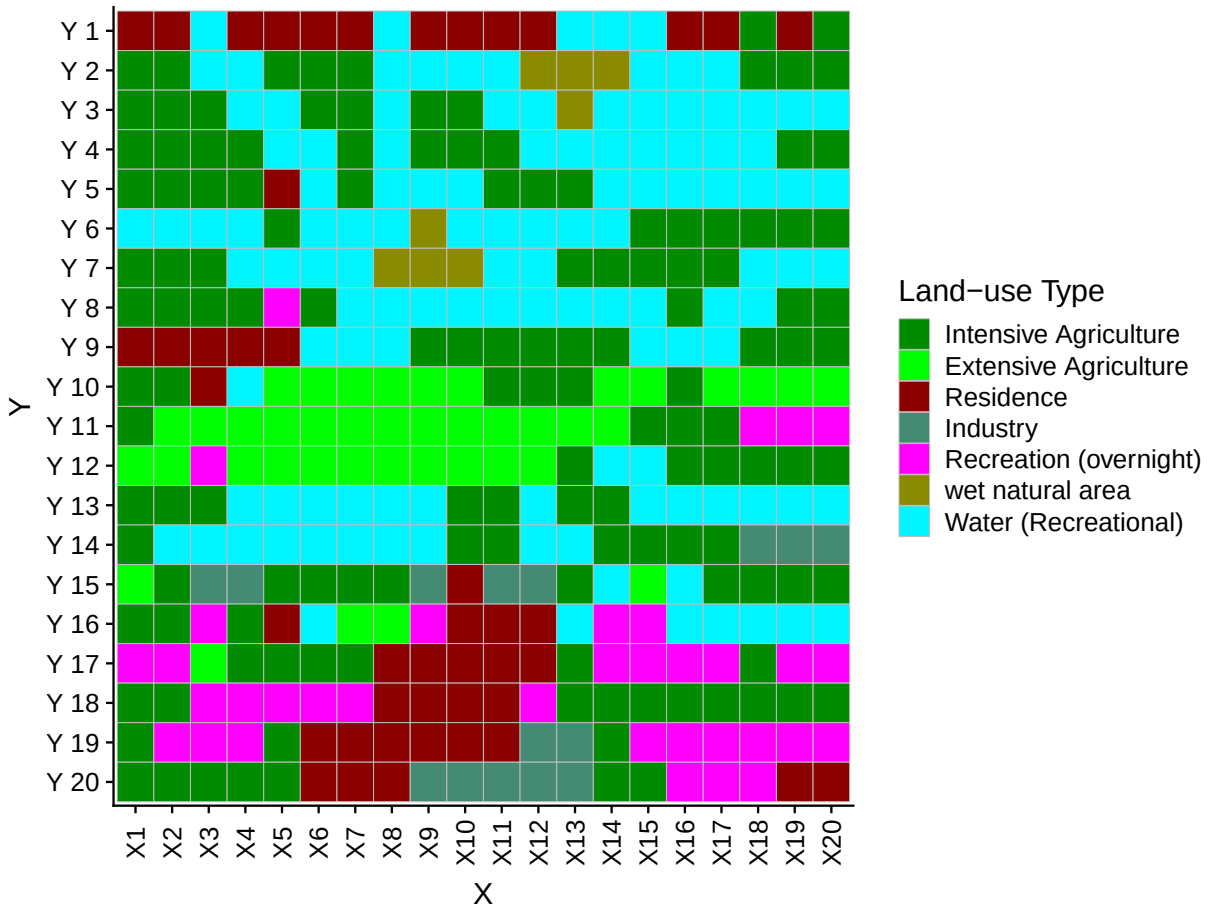


Figure 5: Optimal land-use allocation for minimizing the cost of changing land-use types from their current type.

4 Genetic Algorithms

4.1 Single Objective Optimization

In this section we repeat the same process as we did in section 2 but instead of using Linear Programming to solve the optimization problems, we make use of a genetic algorithm (GA) and compare the corresponding results to that of LP. GA's are population-based optimization methods whereby in every iteration of the algorithm, we deal with a set (popolation) of solutions. The basic idea of a GA is to move a population of solutions towards 'good' regions of the search space. Usually a good region will imply that the fitness of an individual(s) is high there i.e. they obtain high values of the objective function.

4.1.1 Design of GA

The form of the genetic algorithm which we implemented starts with a set of 10000 randomly selected landuse maps which will be represented as a binary vector of 3600 elements. At each generation, randomly paired parent solutions are randomly selected, and "crossed-over"" to generate a pair of "child" solutions for each pair of parents. The best half of the total parents and children solutions were retained to form the next

parent population. The iterarative process continues for a selected number of generations or until convergence in the objective function value.

4.1.1.1 Initial Solution Generation

To generate a population of slutions we first allocate the fixed land-use types to each 3600 long individual solution. For each solution, we then sample a value for each land use type between the minimum and maximum size of the land-use types as specified in table 1.

4.1.1.2 Evaluation

Evaluation is the process through which we compute the total value of the objective function (fitness) for each parent solution at each generation of the algorithm. The constraints are also checked for each candidate solution and their fitness is penalized if any constraints are violated.

4.1.1.3 Selection

The selection process involves selecting a sample of parent solutions from the set of solutions produced at each iteration. Samples are taken based on a set of probabilites attached to each parent solution. The parent with the largest fitness is allocated a relative probabilty of 1 and the one with the worst fitness, a probability of 0.5. The rest of the parent solutions are given probabilities which are found by linearly interpolated between 1 and 0.5. During each selection round, 400 parents are sampled using such probabilities.

4.1.1.4 Reproduction

Reproduction or "cross-over" is the process of randomly pairing up parents that result from the selection process and swapping pairs of land use types between the solutions. We begin by computing all combinations of pairs of land use types except pairs in which the land use types are identical ($9^2 - 9 = 72$). For a given combination of land uses l and k, we locate the grid cells in each pair of parents where cells of parent 1 have a l allocated and cells of parent 2 have a k allocated in the same grid positions. Half of the number of such cells are allocated land use k in parent 1 and l in parent 2. This process is repeated for all pairs of parents and all combinations at each iteration. The best 400 parents and children are taken to the next iteration.

4.1.1.5 Mutation

During the mutation process, we define a mutation probability (rate) and sample a set of uniform random numbers between 0 and 1 thats the same size as our population. If the random number is less than the mutation probability, we mutate a chromosone. We then swap 2 random land uses in each chromosone that is to be muatated.

4.1.2 Objective 1

The maximum nature value attained by the GA was 2668 compared to 2773 in the LP. Below is the plot of the land use map resultings from the GA and for comparison, we have included the map from the LP implementation below it.

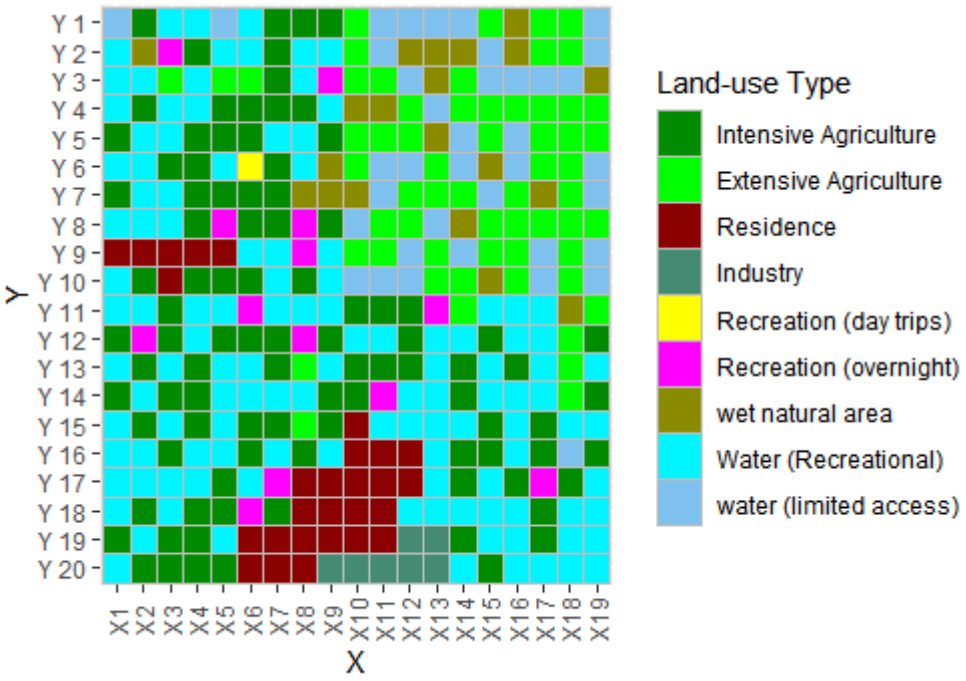


Figure 6: Optimal land-use allocation for maximizing natural value using the GA.

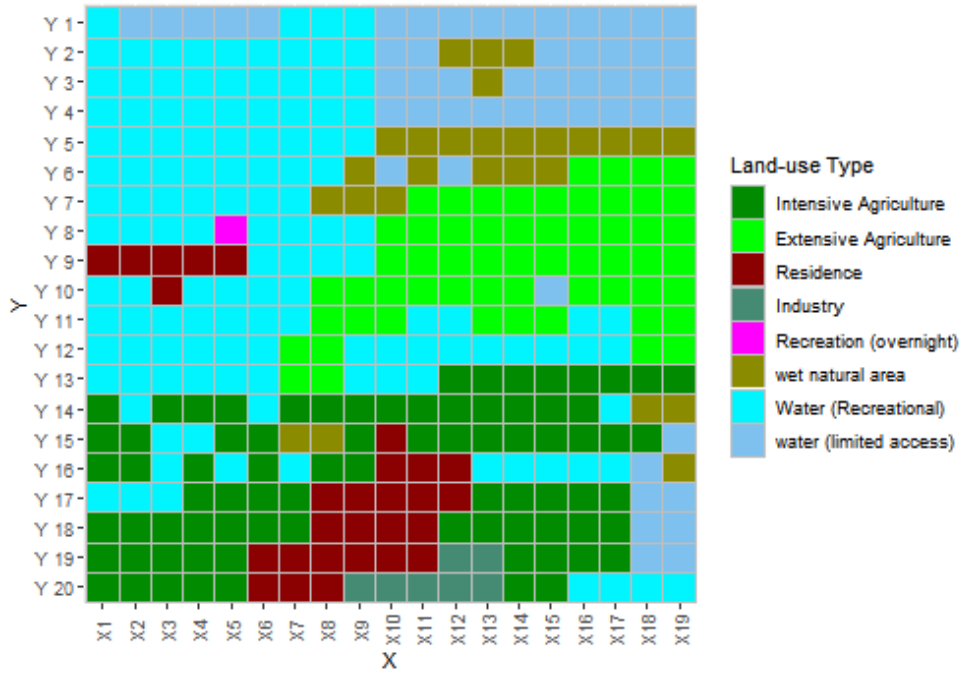


Figure 7: Optimal land-use allocation for maximizing natural value using the LP.

There doesn't seem to be an obvious similarity between the two solutions although there is some overlap in the top right corner with limited access water and extensive agriculture. The GA seems to allocate a few too many recreational land-use types.

The plot below illutrates the mean fitness of the population over generations. The dotted lines represent the fitness of the fittest individual (have to look closely to see the dotted line).

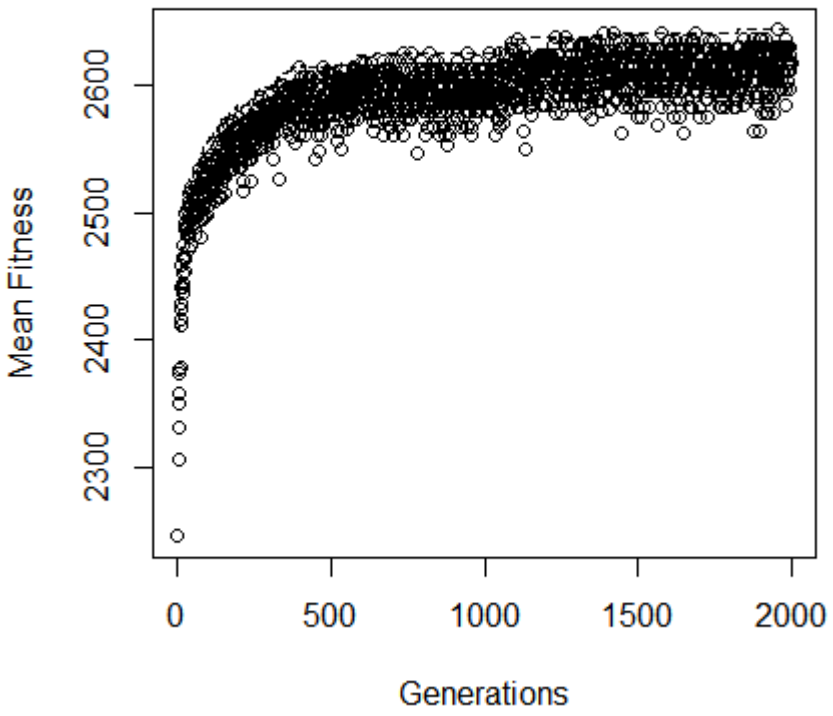


Figure 8: Mean fitness of population over generations when optimizing natrual value.

4.1.3 Objective 2

The maximum recreational value attained by the GA was 3056 compared to 3151 in the LP. Below is the plot of the land use map resultings from the GA and for comparison, we have included the map from the LP implementation below it.

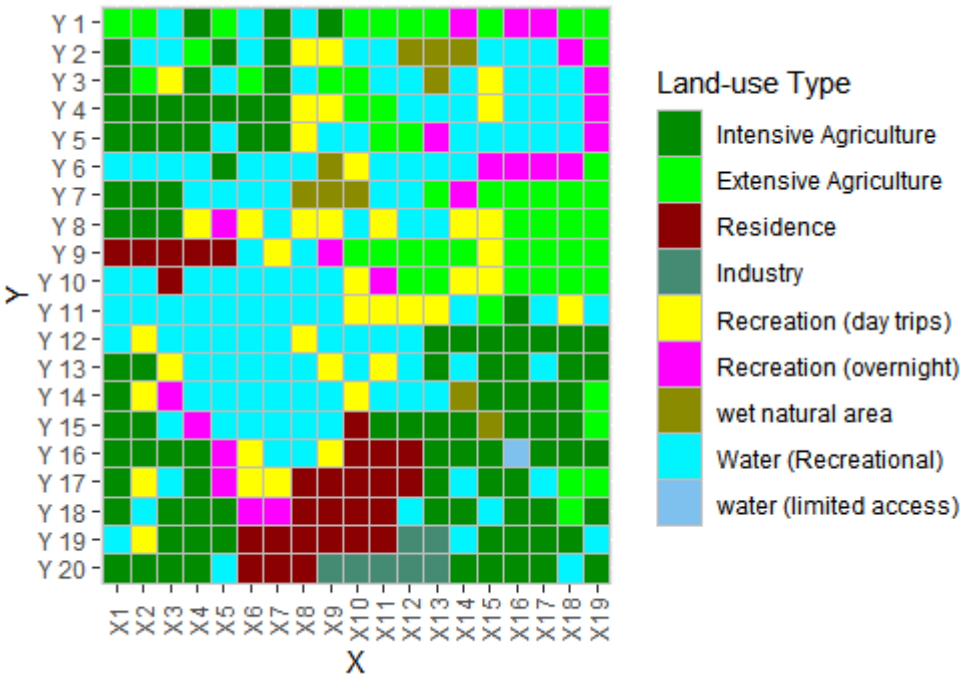


Figure 9: Optimal land-use allocation for maximizing recreational value using the GA.

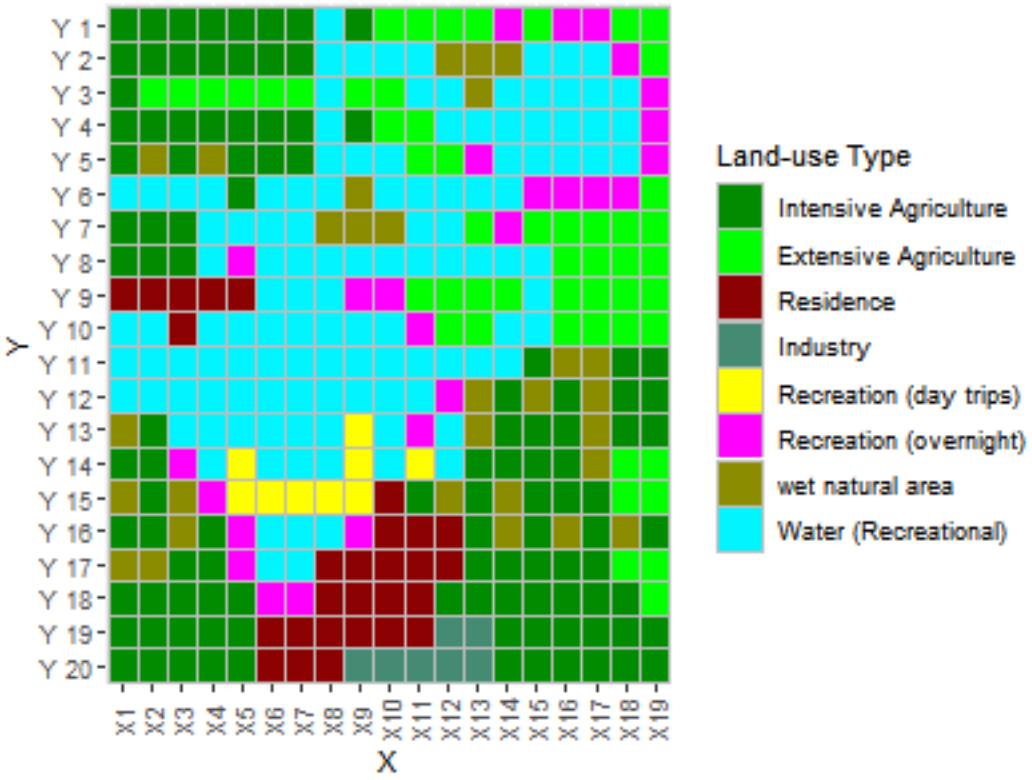


Figure 10: Optimal land-use allocation for maximizing recreational value using the LP.

The GA map for recreational map is probably the mst convinicing of the 3 objectives. It does quite well in allocating recreational land-use types in the correct general areas.

The plot below illutrates the mean fitness of the population over generations. The dotted line represents the fitness of the fittest individual in the population (best solution found by the GA).

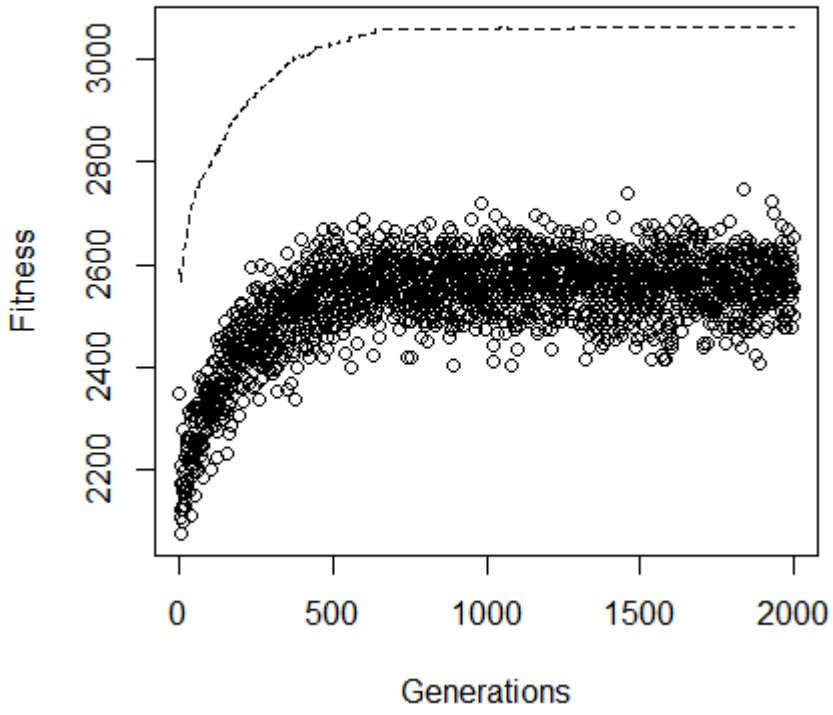


Figure 11: Mean fitness of population over generations when optimizing recreational value.

4.1.4 Objective 3

The minimum cost attained by the GA was 7325 compared to 11950 in the LP. Below is the plot of the land use map resultings from the GA and for comparison, we have included the map from the LP implementation below it.

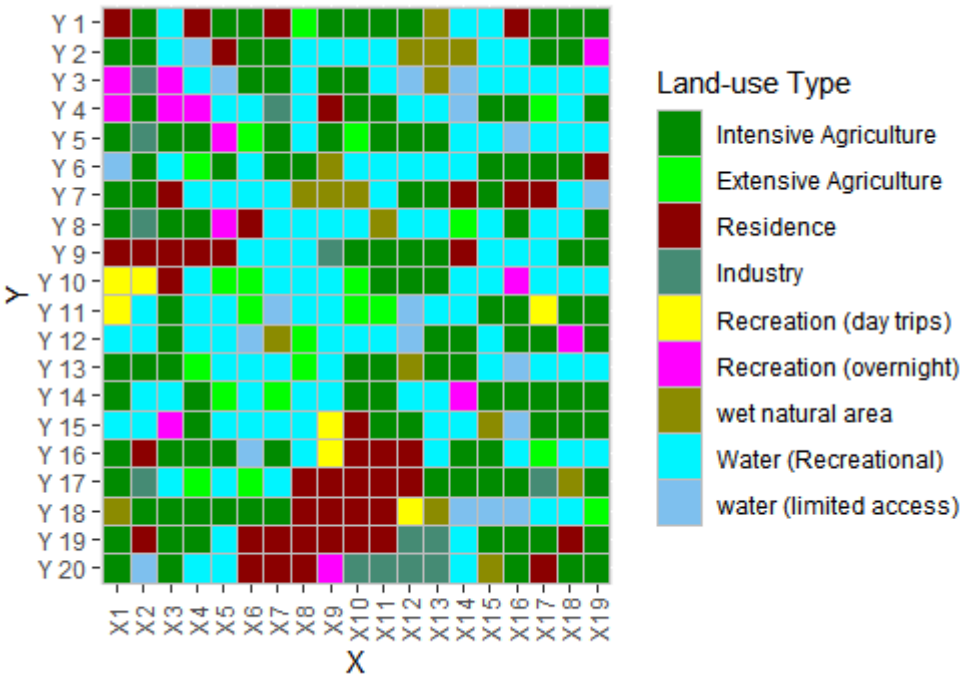


Figure 12: Optimal land-use allocation for minimizing the cost of changing land-use types from their current type using the GA.

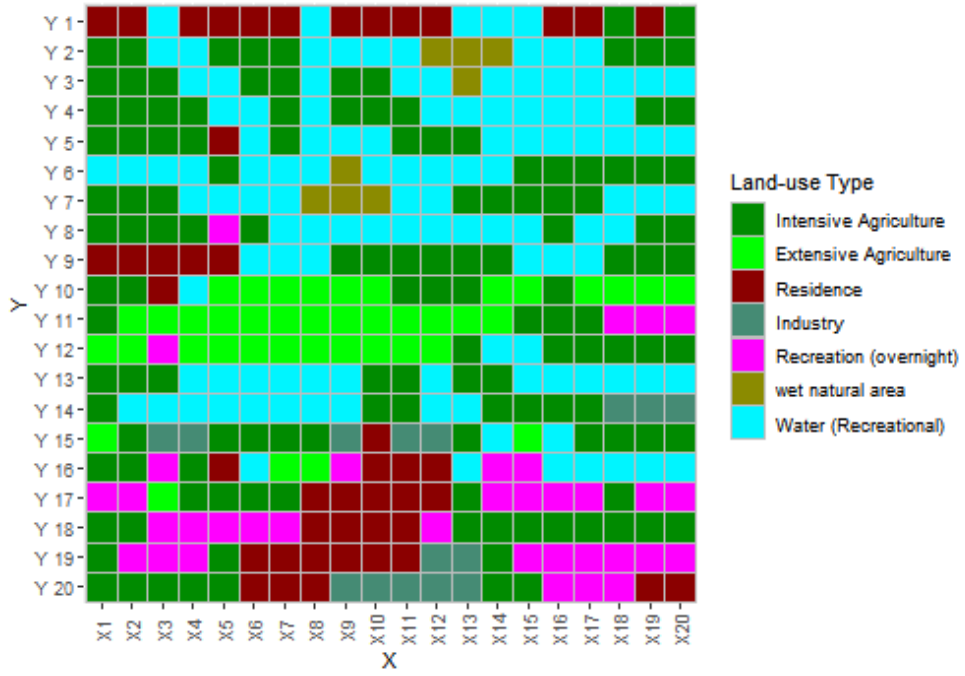


Figure 13: Optimal land-use allocation for minimizing the cost of changing land-use types from their current type using the LP.

There doesnt seem to be obvious similarities between the maps for cost minimization.

The plot below illutrates the mean fitness of the population over generations. The dotted lines represent the fitness of the fittest indivdual.

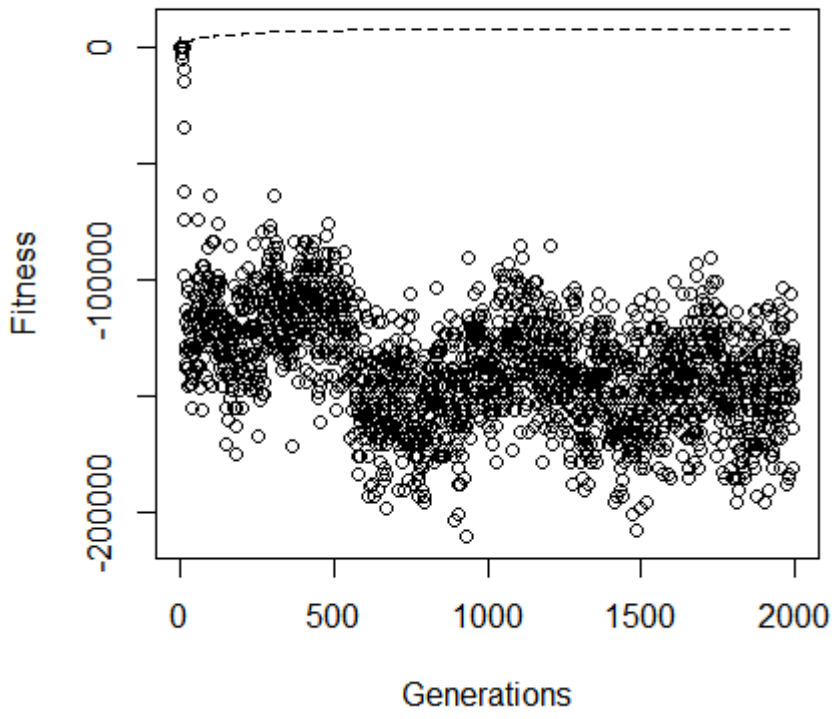


Figure 14: Mean fitness of population over generations when minimizing the cost of changing land-use types from their current type.

4.1.5 Comments on Implementation

After many trials of trying to prevent the GA from converging prematurely, I had to give up on numerous attempts to get it running properly due to time constraints. The values aren't all that far off the optimal values returned by the LP but the GA couldnt seem to get them exact.

5 Multiobjective Optimization

In most real-world optimization problems, decision makers are faced with maximizizing (minimizing) multiple objectives at once as in the case of the land use allocation problem at hand. Rather than just optimizaing each objective on its own as we have done above, we need a method of optimizng all 3 objectives at the same time. Obviously, in majority of cases, we will have to give up on performance on some objectives to perform better in another objective and so we need to find solutions that give a good balance between all the objectives.

5.1 Goal Programming

In light of the above paragraph, to find a solution with a good balance between objectives, we implement an Archimedean goal program. The idea of goal programming is to obtain a desireable level of performance, g_k from each objective $k = 1, \ldots, 3$. We then define the magnitude of the deviation of the objective from the goal value as δ_k. Assuming each objective is defined as z_k, we thus want $z_k + \delta_k \geq g_k$. The intuition is that we want to achieve an objective value as good as our goal and we will thus want to minimize the deviation from this goal δ_k. If goal achieved then $z_k \geq g_k$ and $\delta k = 0$, if not then $z_k \leq g_k$ and $\delta k > 0$ measures the underachievement. A constraint $z_k \geq g_k$ will be added to the LP for each objective k and our objective will be to minimize $f(\delta 1, \delta 2, \delta 3)$.

5.1.1 Archimedean Goal Programming

The goal prgramming approach we take is the Archimedean Goal Program. The approach is tominimize the following objective

$$\text{Minimize} \sum_{k=1}^{3} w_k \delta_k$$

subject to:

$$z_k + \delta_k \geq g_k \quad \text{for } k = 1, \ldots, n \tag{1}$$

$$\sum_{j=1}^{P} a_{ij} \leq b_i \quad \text{for } i = 1, \ldots, m \tag{2}$$

$$x_j \geq 0 \quad \text{for } j = 1, \ldots, P \tag{3}$$

The objective function will be a vector of length 3603, the first 3600 representing the decision variables for the land use grid which are all set to zero and the last 3 elements will represent the weights (w_k) attached to each deviation variable (δ_k).

All the constraint mentioned in the Constraints section (2.1) remain identical for this problem and are represented by the 2nd line in the above set of constraint for the goal program. We will need to add on a row of zeros to the constraint coefficient matrix for each objective (3 extra columns of zeros). The only additional constraints that we add are those in equation (1) of the constraints above. The 3 extra constraints will each be a vector of length 3603 where the first 3600 coefficients are the cofficients of each objective and element 3601, 3602 and 3603 in each of the 3 contraints for objective 1, 2 and 3 respectively will be a have a one (representing δk) with each of the other 2 elements set to zero.

To specify the goals for each objective, we first compute a payoff table. A payoff table is constructed by optimizing objective functions one-at-a-time and computing the values for the other objectives which are not being optimized in each case. The table is illustrated below. Rows represent the objective being optimized and column the value of each objective for a given optimization case. It is important to note that the infeasible solutions are set to a value of zero in this table (only one solution was infeasible wasthe cost chaning objective when optimizing the recreational objective).

Table 4: Payoff table for the 3 objectives. Rows represent the objective being optimized and column the value of each objective for a given optimization case.

	Obj 1	Obj 2	Obj 3
Obj 1	2773	2335	-21190
Obj 2	2367	3151	0
Obj 3	2040	2403	11950

We then compute the ranges of each of the objective which is done by taking the minimum and maximum in each column and taking that difference. We specify priority levels for each objective: a number between 0 (low importance) and 1 (high importance) specifying the importance of an objective. The goal is then taken to be the minimum value plus the product of the priority level and the objective range. We choose a priority level of 0.5 for all 3 objectives and get goals: 2406.5, 2743 and -4620 for objective 1, 2 and 3 respectively. Choosing a priority level of 0.5 for all 3 objectives gives us equal weights for the objective function: $w_1 = w_2 = w_3 = 0.3333$.

Again, we implement *Rglpk* to solve the LP and the optimal value returned is 0 which implies our goal objectives were met since the weighted deviations all sum up to zero. Below we plot the land use map for the solution to the multiobjective LP problem:

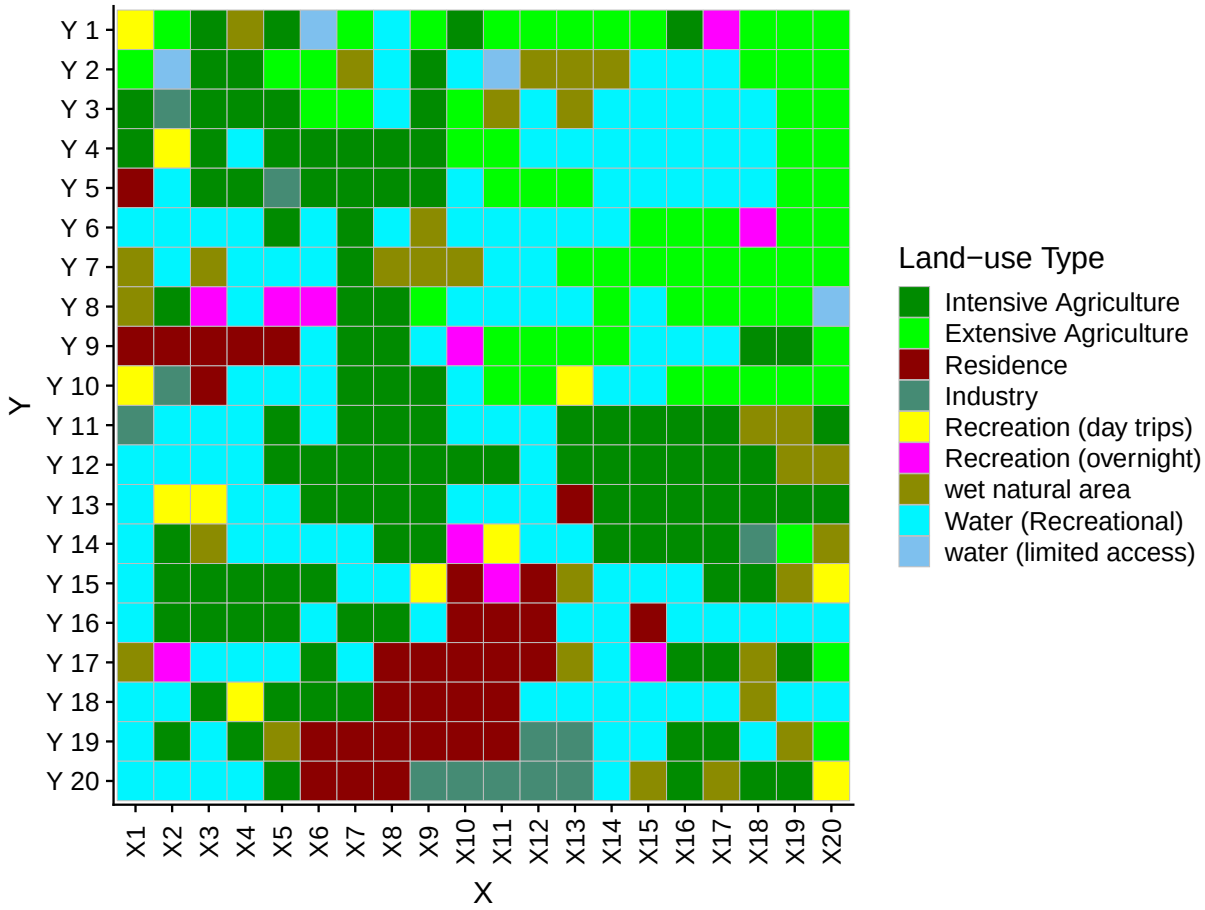


Figure 15: Optimal land-use allocation for the multiobjective land use problem.

6 References

[1] Stewart, T. J., Janssen, R., & van Herwijnen, M. (2004). A genetic algorithm approach to multiobjective land use planning. Computers & Operations Research, 31(14), 2293-2313.

[2] Janssen, R., van Herwijnen, M.,Stewart, T. J. & Aerts, J. (2008). Multiobjective decision support for land-use planning. Environment and Planning B: Planning and Design, 35, 740-756.

7 Appendix

Before running any of the below scripts, open the .Rproj file in the Land-use-planning folder. Apart from the `GA function and Implementation` script, the rest of the scripts can be run by just sourcing the R script. `GA function and Implementation` has instructions in the subsection below.

7.1 Natural Value

```
## Optimisation Project
#
# Reuired:
# 1. Rdata file containing land uses for figure
#
# Problem Specification:
#
# Author : N Murphy
#
## Version Control
# ~/R/Opt/Project/Land-use-planning

## 1. Clear environemnt and remove all plots
rm(list=ls()) # clear environment
#dev.off()

## 2. load required libraries
library(Rglpk)

## 3. Load previously prepared data
load("data/spatialDSS_inputs_fig5.RData")
load("data/landtypes.Rdata")
load("data/spatialDSS_inputs.RData")
Fig3a <- fig_3a
Fig3b <- fig_3b
Fig3c <- fig_3c

################################################################################
## 4. Data Processing

n <- 3600
nrows <- 20
ncols <- 20
nooflandtypes <- 9
```

```
## coefficients of the objective function
objcoeffs <- rbind(rep(4,400),as.vector(t(Fig3a)),rep(3,400),rep(1,400),rep(5,400),rep(5,400),as.vector

Objective <- numeric()
for (j in 1:400) {
  for (i in 1:9){
    Objective <- c(Objective,objcoeffs[i,j])
  }
}
Objective_nature <- as.matrix(Objective)
objcoeffs_nat <- objcoeffs

## coefficients of the constraint matrix
#predifine the constraints that make up the 3600x3600 matrix of constraints
int_agriconstmin_ind <- numeric()
int_agriconstmax_ind <- numeric()
ext_agriconstmin <- numeric()
ext_agriconstmax <- numeric()
residencemin <- numeric()
residencemax <- numeric()
industrymin <- numeric()
industrymax <- numeric()
recreationdaymin <- numeric()
recreationdaymax <- numeric()
recreationnightmin <- numeric()
recreationnightmax <- numeric()
wetareamin <- numeric()
wetareamax <- numeric()
waterrecreationalmin <- numeric()
waterrecreationalmax <- numeric()
waterlimitedmin <- numeric()
waterlimitedmax <- numeric()

# populate the column indices for each constraint
for (i in seq(1, n,9)) {
  int_agriconstmin_ind <- cbind(int_agriconstmin_ind,i)
  int_agriconstmax_ind <- cbind(int_agriconstmax_ind,i)
  ext_agriconstmin <- cbind(ext_agriconstmin,i+1)
  ext_agriconstmax <- cbind(ext_agriconstmax,i+1)
  residencemin <- cbind(residencemin,i+2)
  residencemax <- cbind(residencemax,i+2)
  industrymin <- cbind(industrymin,i+3)
  industrymax <- cbind(industrymax,i+3)
  recreationdaymin <- cbind(recreationdaymin,i+4)
  recreationdaymax <- cbind(recreationdaymax,i+4)
  recreationnightmin <- cbind(recreationnightmin,i+5)
  recreationnightmax <- cbind(recreationnightmax,i+5)
  wetareamin <- cbind(wetareamin,i+6)
  wetareamax <- cbind(wetareamax,i+6)
  waterrecreationalmin <- cbind(waterrecreationalmin,i+7)
  waterrecreationalmax <- cbind(waterrecreationalmax,i+7)
  waterlimitedmin <- cbind(waterlimitedmax,i+8)
  waterlimitedmax <- cbind(waterlimitedmax,i+8)
```

```
}

# land (grid) constraint column ind
landconstraint <- as.numeric(c(1:n))

#create a vector of the column indices of all constraints
colinds <- cbind(int_agriconstmin_ind,int_agriconstmax_ind,ext_agriconstmin,
                 ext_agriconstmax,residencemin,residencemax,
                 industrymin,industrymax,recreationdaymin,recreationdaymax,recreationnightmin,
                 recreationnightmax,wetareamin,wetareamax,waterrecreationalmin,
                 waterrecreationalmax, waterlimitedmin,waterlimitedmax,t(landconstraint))

# Compute the row indices of activity constraints
rowinds <- numeric()
for (i in 1:(18)){
  rowinds <- c(rowinds,rep(i,400))
}
#add land constraints (400 blocks maximum)
rowinds <- c(rowinds,rep(19,n))

## add figure 5 constraints
#column indices
Fig5indices <- which(Fig5!=0,arr.ind = T)
Fig5indices <- Fig5indices[order(Fig5indices[,1]), ]
Fig5values <- as.vector(Fig5[Fig5indices])
fixedconstraint_colinds <- numeric()

for (i in 1:length(Fig5values)){
  # the following formula gives the vector index of the grid position (index)
  fixedconstraint_colinds <- c(fixedconstraint_colinds,(Fig5indices[i,1]-1)*20*9 + Fig5indices[i,2]*9 -
}
fixedconstraint_colinds <- as.matrix(fixedconstraint_colinds)

#populate fixed constraints into vector indices of all other constraints
colinds <- c(colinds,fixedconstraint_colinds)
colinds <- as.matrix(colinds)

#row indices
for (i in 20:(19+length(Fig5values))){
  rowinds <- c(rowinds,i)
}
rowinds <- as.matrix(rowinds)

## add constraints for maximum of one activity per grid block
#blockconstraint <- numeric()
for (i in 64:(63+nrows*ncols)){
  rowinds <- c(rowinds,rep(i,9))
}
colinds <- c(colinds,1:n)

## create the spasre matrix of constraint coeffs
sparse_mat_nature <- simple_triplet_matrix(i = rowinds, j = colinds, v = c(rep(1,length(colinds))))
realmat <- as.matrix(sparse_mat_nature) #check that sparse matrix makes sense
```

```
#compute RHS of constraints
b_nature <- c(80,150,20,65,20,45,5,15,0,70,0,35,0,30,120,150,0,60,400,rep(1,44),rep(1,400))

# signs of the constraints
constraint_ineq_nature <- c(">=", "<=",">=", "<=",">=", "<=",">=", "<=", ">=","<=",">=", "<=",">=","<="

#variable types
var_types <- rep("B",n)

## Solve the LP problem
LPsolve <- Rglpk_solve_LP(obj = Objective_nature, mat = sparse_mat_nature, dir = constraint_ineq_nature

## plot the optimal solution
LPsolveSolution <- as.matrix(LPsolve$solution)
#LPsolveSolution
sum(LPsolveSolution)

## display solution as grid
#first populate solution into grid form
LPsolveSolutiongrid <- matrix(LPsolveSolution,nrow = 400,ncol = 9,byrow=TRUE)

Solutiongridvalues <- matrix(NA,nrows*ncols,1)
for(i in 1:(nrows*ncols)){
  Solutiongridvalues[i,1] <-  which(LPsolveSolutiongrid[i,]!=0,arr.ind = T)
}
Solutiongrid <- matrix(Solutiongridvalues,nrow = 20,ncol = 20,byrow=TRUE)

#define colours for ggplot
col = c("green4","green","red4","aquamarine4","yellow","magenta","yellow4","turquoise1","skyblue2")

#get unique values of solution in case htat some landtypes are missing
colourind <- sort(unique(c(Solutiongrid)))

library(ggplot2);
library(reshape2)
Soldf <- data.frame(indv=factor(paste("Y",1:20),
                                levels = rev(paste("Y",1:20))), as.matrix(Solutiongrid))
Soldf <- Soldf[,1:20]

Soldf <- melt(Soldf, id.var = 'indv')

nature_opt <- LPsolve$optimum
nature_sol <- LPsolve$solution

ggplot(Soldf, aes(variable, indv,fill=as.factor(value))) + theme(axis.text.x=element_text(angle=90,vjus
ggsave(filename="LP_nat_map.png",width = 500, height = 350, plot=last_plot(),limitsize = F,dpi = 300,de
```

7.2 Recreational Value

```
## Optimisation Project
#
```

```
# Reuired:
# 1. Rdata file containing land use figure
#
# Problem Specification:
#
# Author : N Murphy
#
## Version Control
# ~/R/Opt/Project/Land-use-planning

## 1. Clear environemnt and remove all plots
#rm(list=ls()) # clear environment
#dev.off()

## 2. load required libraries
library(Rglpk)

## 3. Load previously prepared data
load("data/spatialDSS_inputs_fig3a.RData")
Fig3a <- nature_values
load("data/spatialDSS_inputs_fig3b.RData")
load("data/spatialDSS_inputs_fig3c.RData")
load("data/spatialDSS_inputs_fig5.RData")


################################################################################
## 4. Data Processing
n <- 3600
nrows <- 20
ncols <- 20
nooflandtypes <- 9


## coefficients of the objective function
objcoeffs <- rbind(rep(6,400),as.vector(t(Fig3a)),rep(3,400),rep(1,400),as.vector(t(Fig3b)),as.vector(t

Objective <- numeric()
for (j in 1:400) {
  for (i in 1:9){
    Objective <- c(Objective,objcoeffs[i,j])
  }
}
Objective_rec <- as.matrix(Objective)
objcoeffs_rec <- objcoeffs

## coefficients of the constraint matrix
#predifine the constraints that make up the 3600x3600 matrix of constraints
int_agriconstmin_ind <- numeric()
int_agriconstmax_ind <- numeric()
ext_agriconstmin <- numeric()
ext_agriconstmax <- numeric()
residencemin <- numeric()
residencemax <- numeric()
```

```
industrymin <- numeric()
industrymax <- numeric()
recreationdaymin <- numeric()
recreationdaymax <- numeric()
recreationnightmin <- numeric()
recreationnightmax <- numeric()
wetareamin <- numeric()
wetareamax <- numeric()
waterrecreationalmin <- numeric()
waterrecreationalmax <- numeric()
waterlimitedmin <- numeric()
waterlimitedmax <- numeric()

# populate the column indices for each constraint
for (i in seq(1, nrows*ncols*nooflandtypes,9)) {
  int_agriconstmin_ind <- cbind(int_agriconstmin_ind,i)
  int_agriconstmax_ind <- cbind(int_agriconstmax_ind,i)
  ext_agriconstmin <- cbind(ext_agriconstmin,i+1)
  ext_agriconstmax <- cbind(ext_agriconstmax,i+1)
  residencemin <- cbind(residencemin,i+2)
  residencemax <- cbind(residencemax,i+2)
  industrymin <- cbind(industrymin,i+3)
  industrymax <- cbind(industrymax,i+3)
  recreationdaymin <- cbind(recreationdaymin,i+4)
  recreationdaymax <- cbind(recreationdaymax,i+4)
  recreationnightmin <- cbind(recreationnightmin,i+5)
  recreationnightmax <- cbind(recreationnightmax,i+5)
  wetareamin <- cbind(wetareamin,i+6)
  wetareamax <- cbind(wetareamax,i+6)
  waterrecreationalmin <- cbind(waterrecreationalmin,i+7)
  waterrecreationalmax <- cbind(waterrecreationalmax,i+7)
  waterlimitedmin <- cbind(waterlimitedmax,i+8)
  waterlimitedmax <- cbind(waterlimitedmax,i+8)
}

# land (grid) constraint column ind
landconstraint <- as.numeric(c(1:n))

#create a vector of the column indices of all constraints
colinds <- cbind(int_agriconstmin_ind,int_agriconstmax_ind,ext_agriconstmin,
                 ext_agriconstmax,residencemin,residencemax,
                 industrymin,industrymax,recreationdaymin,recreationdaymax,recreationnightmin,
                 recreationnightmax,wetareamin,wetareamax,waterrecreationalmin,
                 waterrecreationalmax, waterlimitedmin,waterlimitedmax,t(landconstraint))

# Compute the row indices of activity constraints
rowinds <- numeric()
for (i in 1:(18)){
  rowinds <- c(rowinds,rep(i,400))
}
#add land constraints (400 blocks maximum)
rowinds <- c(rowinds,rep(19,n))
```

```
## add figure 5 constraints
#column indices
Fig5indices <- which(Fig5!=0,arr.ind = T)
Fig5indices <- Fig5indices[order(Fig5indices[,1]), ]
Fig5values <- as.vector(Fig5[Fig5indices])
fixedconstraint_colinds <- numeric()

for (i in 1:length(Fig5values)){
  # the following formula gives the vector index of the grid position (index)
  fixedconstraint_colinds <- c(fixedconstraint_colinds,(Fig5indices[i,1]-1)*20*9 +  Fig5indices[i,2]*9 -
}
fixedconstraint_colinds <- as.matrix(fixedconstraint_colinds)

#populate fixed constraints into vector indices of all other constraints
colinds <- c(colinds,fixedconstraint_colinds)
colinds <- as.matrix(colinds)

#row indices
for (i in 20:(19+length(Fig5values))){
  rowinds <- c(rowinds,i)
}
rowinds <- as.matrix(rowinds)

## add constraints for maximum of one activity per grid block
#blockconstraint <- numeric()
for (i in 64:(63+nrows*ncols)){
  rowinds <- c(rowinds,rep(i,9))
}
colinds <- c(colinds,1:n)

## create the spasre matrix of constraint coeffs
sparse_mat_rec <- simple_triplet_matrix(i = rowinds, j = colinds, v = c(rep(1,length(colinds))))

#compute RHS of constraints
b_rec <- c(80,150,20,65,20,45,5,15,0,70,0,35,0,30,120,150,0,60,400,rep(1,44),rep(1,400))

# signs of the constraints
constaint_ineq_rec <- c(">=", "<=",">=", "<=",">=", "<=",">=", "<=", ">=","<=",">=", "<=",">=","<=",">=

#variable types
var_types <- rep("B",n)

## Solve the LP problem
LPsolve <- Rglpk_solve_LP(obj = Objective_rec, mat = sparse_mat_rec, dir = constaint_ineq_rec, rhs = b_

## plot the optimal solution
LPsolveSolution <- as.matrix(LPsolve$solution)
#LPsolveSolution
#sum(LPsolveSolution)

## display solution as grid
#first populate solution into grid form
LPsolveSolutiongrid <- matrix(LPsolveSolution,nrow = 400,ncol = 9,byrow=TRUE)
```

```
Solutiongridvalues <- matrix(NA,nrows*ncols,1)
for(i in 1:(nrows*ncols)){
  Solutiongridvalues[i,1] <-  which(LPsolveSolutiongrid[i,]!=0,arr.ind = T)
}
Solutiongrid <- matrix(Solutiongridvalues,nrow = 20,ncol = 20,byrow=TRUE)

#get unique values of solution in case htat some landtypes are missing
colourind <- sort(unique(c(Solutiongrid)))

col = c("green4","green","red4","aquamarine4","yellow","magenta","yellow4","turquoise1","skyblue2")


library(ggplot2);
library(reshape2)
Soldf <- data.frame(indv=factor(paste("Y",1:20),
                                levels = rev(paste("Y",1:20))), as.matrix(Solutiongrid))
Soldf <- Soldf[,1:20]

Soldf <- melt(Soldf, id.var = 'indv')

rec_opt <- LPsolve$optimum
rec_sol <- LPsolve$solution


load("data/landtypes.Rdata")

p <- ggplot(Soldf, aes(variable, indv,fill=as.factor(value))) +theme(axis.text.x=element_text(angle=90,
p + guides(fill=guide_legend(title="Land-use Type")) + labs(x = "X",y = "Y")
ggsave(filename="LP_rec_map.png",width = 400, height = 300, plot=last_plot(),limitsize = F,dpi = 300,de
```

7.3 Cost of Changing

```
## Optimisation Project
#
# Reuired:
# 1. Rdata file containing land uses for figure ... (a)
#
# Problem Specification:
#
# Author : N Murphy
#
## Version Control
# ~/R/Opt/Project/Land-use-planning

## 1. Clear environemnt and remove all plots
#rm(list=ls()) # clear environment
#dev.off()

## 2. load required libraries
library(Rglpk)
```

```
## 3. Load previously prepared data
cost_matrix <- read.table("data/cost_matrix.txt",sep="",header = FALSE)
# load("data/spatialDSS_inputs_fig3a.RData")
# Fig3a <- nature_values
# load("data/spatialDSS_inputs_fig3b.RData")
# load("data/spatialDSS_inputs_fig3c.RData")
load("data/spatialDSS_inputs_fig2c.RData")
load("data/spatialDSS_inputs_fig5.RData")
load("data/landtypes.Rdata")


################################################################################
## 4. Data Processing
n <- 3600
nrows <- 20
ncols <- 20
nooflandtypes <- 9


## coefficients of the objective function
Objective <- numeric()
for (i in 1:nrows){
  for (j in 1:ncols){
    for (k in 1:nooflandtypes){
      if(Fig2c[i,j]==k){
        Objective <- c(Objective,as.matrix(cost_matrix[k,]))
      }
    }
  }
}
Objective_cost <- as.matrix(Objective)

## coefficients of the constraint matrix
#predifine the constraints that make up the 3600x3600 matrix of constraints
int_agriconstmin_ind <- numeric()
int_agriconstmax_ind <- numeric()
ext_agriconstmin <- numeric()
ext_agriconstmax <- numeric()
residencemin <- numeric()
residencemax <- numeric()
industrymin <- numeric()
industrymax <- numeric()
recreationdaymin <- numeric()
recreationdaymax <- numeric()
recreationnightmin <- numeric()
recreationnightmax <- numeric()
wetareamin <- numeric()
wetareamax <- numeric()
waterrecreationalmin <- numeric()
waterrecreationalmax <- numeric()
waterlimitedmin <- numeric()
waterlimitedmax <- numeric()
```

```
# populate the column indices for each constraint
for (i in seq(1, nrows*ncols*nooflandtypes,9)) {
  int_agriconstmin_ind <- cbind(int_agriconstmin_ind,i)
  int_agriconstmax_ind <- cbind(int_agriconstmax_ind,i)
  ext_agriconstmin <- cbind(ext_agriconstmin,i+1)
  ext_agriconstmax <- cbind(ext_agriconstmax,i+1)
  residencemin <- cbind(residencemin,i+2)
  residencemax <- cbind(residencemax,i+2)
  industrymin <- cbind(industrymin,i+3)
  industrymax <- cbind(industrymax,i+3)
  recreationdaymin <- cbind(recreationdaymin,i+4)
  recreationdaymax <- cbind(recreationdaymax,i+4)
  recreationnightmin <- cbind(recreationnightmin,i+5)
  recreationnightmax <- cbind(recreationnightmax,i+5)
  wetareamin <- cbind(wetareamin,i+6)
  wetareamax <- cbind(wetareamax,i+6)
  waterrecreationalmin <- cbind(waterrecreationalmin,i+7)
  waterrecreationalmax <- cbind(waterrecreationalmax,i+7)
  waterlimitedmin <- cbind(waterlimitedmax,i+8)
  waterlimitedmax <- cbind(waterlimitedmax,i+8)
}

# land (grid) constraint column ind
landconstraint <- as.numeric(c(1:n))

#create a vector of the column indices of all constraints
colinds <- cbind(int_agriconstmin_ind,int_agriconstmax_ind,ext_agriconstmin,
                 ext_agriconstmax,residencemin,residencemax,
                 industrymin,industrymax,recreationdaymin,recreationdaymax,recreationnightmin,
                 recreationnightmax,wetareamin,wetareamax,waterrecreationalmin,
                 waterrecreationalmax, waterlimitedmin,waterlimitedmax,t(landconstraint))


# Compute the row indices of activity constraints
rowinds <- numeric()
for (i in 1:(18)){
  rowinds <- c(rowinds,rep(i,400))
}
#add land constraints (400 blocks maximum)
rowinds <- c(rowinds,rep(19,n))

## add figure 5 constraints
#column indices
Fig5indices <- which(Fig5!=0,arr.ind = T)
Fig5indices <- Fig5indices[order(Fig5indices[,1]), ]
Fig5values <- as.vector(Fig5[Fig5indices])
fixedconstraint_colinds <- numeric()

for (i in 1:length(Fig5values)){
  # the following formula gives the vector index of the grid position (index)
  fixedconstraint_colinds <- c(fixedconstraint_colinds,(Fig5indices[i,1]-1)*20*9 +  Fig5indices[i,2]*9 -
}
fixedconstraint_colinds <- as.matrix(fixedconstraint_colinds)
```

```
#populate fixed constraints into vector indices of all other constraints
colinds <- c(colinds,fixedconstraint_colinds)
colinds <- as.matrix(colinds)

#row indices
for (i in 20:(19+length(Fig5values))){
  rowinds <- c(rowinds,i)
}
rowinds <- as.matrix(rowinds)

## add constraints for maximum of one activity per grid block
#blockconstraint <- numeric()
for (i in 64:(63+nrows*ncols)){
  rowinds <- c(rowinds,rep(i,9))
}
colinds <- c(colinds,1:n)

## add constraints for land types that are infeasible to convert
for (i in 1:nrows){
  for (j in 1:ncols){
    for (k in 1:nooflandtypes){
      if(Fig2c[i,j]=="NA"){
        colinds <- c(colinds,(i-1)*20*9 +  (j-1)*9 + k)
      }
    }
  }
}

## create the sparse matrix of constraint coeffs
sparse_mat_cost <- simple_triplet_matrix(i = rowinds, j = colinds, v = c(rep(1,length(colinds))))
#realmat <- as.matrix(sparse_mat) #check that sparse matrix makes sense

#compute RHS of constraints
b_cost <- c(80,150,20,65,20,45,5,15,0,70,0,35,0,30,120,150,0,60,400,rep(1,44),rep(1,400))

# signs of the constraints
constaint_ineq_cost <- c(">=", "<=",">=", "<=",">=", "<=",">=", "<=", ">=","<=",">=", "<=",">=","<=",">=

#variable types
var_types <- rep("B",n)

## Solve the LP problem
LPsolve <- Rglpk_solve_LP(obj = Objective_cost, mat = sparse_mat_cost, dir = constaint_ineq_cost, rhs =
costs_opt <- LPsolve$optimum
options(scipen = 999)
cost_sol <- LPsolve$solution

## plot the optimal solution
LPsolveSolution <- as.matrix(LPsolve$solution)
#LPsolveSolution
sum(LPsolveSolution)

## display solution as grid
```

```
#first populate solution into grid form
LPsolveSolutiongrid <- matrix(LPsolveSolution,nrow = 400,ncol = 9,byrow=TRUE)

Solutiongridvalues <- matrix(NA,nrows*ncols,1)
for(i in 1:(nrows*ncols)){
  Solutiongridvalues[i,1] <-  which(LPsolveSolutiongrid[i,]!=0,arr.ind = T)
}

Solutiongrid <- matrix(Solutiongridvalues,nrow = 20,ncol = 20,byrow=TRUE)

library(ggplot2)
library(reshape2)
library(gplots)
Soldf <- data.frame(indv=factor(paste("Y",1:20),levels = rev(paste("Y",1:20))), as.matrix(Solutiongrid))
Soldf <- melt(Soldf, id.var = 'indv')

#get unique values of solution in case htat some landtypes are missing
colourind <- sort(unique(c(Solutiongrid)))

col = c("green4","green","red4","aquamarine4","yellow","magenta","yellow4","turquoise1","skyblue2")
col <- col[colourind]
## EOF


p <- ggplot(Soldf, aes(variable, indv,fill=as.factor(value)))+theme(axis.text.x=element_text(angle=90,v
p + guides(fill=guide_legend(title="Land-use Type")) + labs(x = "X",y = "Y")

ggsave(filename="LP_cost_map.png",width = 500, height = 350, plot=last_plot(),limitsize = F,dpi = 300,d
```

7.4 GA function and Implementation

To implement this script, uncomment the objective function which you want to maximize/minimize and source the script. For objectives 1 and 2, line 170 of the script must be set to 1000 and for objective 3 to -100000.

```
##Script to implement GA for each objective

# Reuired:
# 1. Rdata file containing land uses for figure ... (a)
#
# Problem Specification:
#
# Author : N Murphy
#
## Version Control
# ~/R/Opt/Project/Land-use-planning

## 1. Clear environemnt and remove all plots
#dev.off()

## 2. load required libraries
library(Rglpk)
```

```r
## 3. Load previously prepared data
load("data/spatialDSS_inputs_fig3a.RData")
Fig3a <- nature_values
load("data/spatialDSS_inputs_fig3b.RData")
load("data/spatialDSS_inputs_fig3c.RData")
load("data/spatialDSS_inputs_fig5.RData")
load("data/landtypes.Rdata")


ga <- function(sparse_mat, b, constraint_ineq, obj_wts, pop_size, num_gens){
  ########################################################
  ### Code for land-use planning project
  ########################################################
  load("data/spatialDSS_inputs_fig5.RData")
  Fig5 <- as.matrix(Fig5)
  Fig5indicesfixed <- which(Fig5!=0,arr.ind = T)
  Fig5indicesother <- which(Fig5==0,arr.ind = T)
  Fig5indicesfixed <- Fig5indicesfixed[order(Fig5indicesfixed[,1]), ]
  Fig5indicesother <- Fig5indicesother[order(Fig5indicesother[,1]), ]

  Fig5valuesfixed <- as.vector(Fig5[Fig5indicesfixed])
  #Fig5valuesother <- as.vector(Fig5[Fig5indicesother])


  # Generate initial population of solutions (given number of items and size of popn)
  initial_pop = function(varsize,popsize){

    rand_vect <- function(N, M, sd = 1) {
      vec <- rnorm(N, M/N, sd)
      if (abs(sum(vec)) < 0.01) vec <- vec + 1
      vec <- round(vec / sum(vec) * M)
      deviation <- M - sum(vec)
      for (. in seq_len(abs(deviation))) {
        vec[i] <- vec[i <- sample(N, 1)] + sign(deviation)
      }
      while (any(vec < 0)) {
        negs <- vec < 0
        pos  <- vec > 0
        vec[negs][i] <- vec[negs][i <- sample(sum(negs), 1)] + 1
        vec[pos][i]  <- vec[pos ][i <- sample(sum(pos ), 1)] - 1
      }
      vec
    }

    fixedconstraint_colinds <- numeric()

    for (i in 1:length(Fig5valuesfixed)){
      # the following formula gives the vector index of the grid position (index)
      fixedconstraint_colinds <- c(fixedconstraint_colinds,(Fig5indicesfixed[i,1]-1)*20*9 +  Fig5indices
    }

    #create objective matrix
    Objective <- obj_wts
```

```
  #standardize each row of the matrix
  Objective <- matrix(Objective,400,9,byrow = T)
  #compute selection valie for each cell
  sigma <- t(apply(Objective,1,function(x) { (x-min(x)+0.01)/(max(x)-min(x))}))
  Objective <- numeric()

  pop <- matrix(0,popsize,varsize)
  pop[,fixedconstraint_colinds] <- 1

  for (k in 1:popsize){
    #first generate a set of possible land use types for the non-fixed cells within the min-max range
    numvaluesforeachtype <- numeric()
    maxconstr <- numeric()
    minconstr <- numeric()
    for (i in seq(1,18,2)){
      numvaluesforeachtype <- c(numvaluesforeachtype,sample(b[i]:(b[i+1]-1),1)) #sample random number
      maxconstr <- c(maxconstr,b[i+1])
      minconstr <- c(minconstr,b[i])
    }
    sizediff <- 356 - sum(numvaluesforeachtype)

    if (sizediff > 0){
      #add a set of possible values to add to vector numvaluesforeachtype to make sum up to 356
      numvaluesforeachtype <- numvaluesforeachtype + rand_vect(9,sizediff)
    }else if (sizediff < 0){
      #add a set of possible values to add to vector numvaluesforeachtype to make sum up to 356
      numvaluesforeachtype <- numvaluesforeachtype - rand_vect(9,-sizediff)

      if(any(numvaluesforeachtype<0)){ #convert ay negative values to zero
        for (i in which(numvaluesforeachtype<0)){
          sizeofnegative <- abs(numvaluesforeachtype[i])
          numvaluesforeachtype[i] <- 0
          numvaluesforeachtype[1] <- numvaluesforeachtype[1] - sizeofnegative
        }
      }
    }


    #start allocating values to each individual
    #generate matrix for fixed constraints
    fixedconstr <- rep(0,3600)
    fixedconstr[fixedconstraint_colinds] <- 1
    fixedconstr <- matrix(fixedconstr,400,9,byrow = T)
    no_landuses_fixed <- apply(fixedconstr,2,sum)
    numvaluesforeachtype <- numvaluesforeachtype + no_landuses_fixed

    if (sum(numvaluesforeachtype)==400){
      pre_pop <- fixedconstr

      #loop from 1 to 400 allocating land uses based on probabilities
      numvaluesforeachtype <- numvaluesforeachtype - no_landuses_fixed

      cells <- which(apply(pre_pop,1,function(x){sum(x)==0})>0) #cells to still be filled
```

```
      number_types_allocated <- rep(0,9)
      for (j in 1:356) {
        cell_allocate <- sample(cells,1)
        cells <- cells[-c(which(cells==cell_allocate))]

        #compute the sacling factor for each land use to encourage achievment of
        constraintfactor <- (numvaluesforeachtype - number_types_allocated)/numvaluesforeachtype
        constraintfactor[is.nan(constraintfactor)] <- 0
        constraintfactor[is.na(constraintfactor)] <- 0

        value_to_allo <- sample(1:9,1,prob = sigma[cell_allocate,]*constraintfactor)
        pre_pop[cell_allocate,value_to_allo] <- 1 #allocate land use to correct column and row

        #print(value_to_allo)
        number_types_allocated[value_to_allo] <- number_types_allocated[value_to_allo]+1

      }
      if(any(apply(pre_pop,1,sum)==2)){
        toomany <- apply(pre_pop,1,sum)==2
        toolittle <- apply(pre_pop,1,sum)==0
        toomanyvec <- which(pre_pop[toomany,]>0)
        pre_pop[toomany,toomanyvec[2]] <- 0
        pre_pop[toolittle,toomanyvec[1]] <- 1
      }

      pop[k,] <- as.vector(t(pre_pop))
    } else {
      #generate the possible values for each of the empty cells to be allocated and then randomly scr
      Fig5valuesother <- numeric()
      Fig5valuesother <- c(rep(1:9,numvaluesforeachtype))
      Fig5valuesother <- sample(Fig5valuesother,length(Fig5valuesother),replace = FALSE)

      othercells_colinds <- numeric()
      for (i in 1:dim(Fig5indicesother)[1]){
        othercells_colinds <- c(othercells_colinds,(Fig5indicesother[i,1]-1)*20*9 +  Fig5indicesother
      }
      pop[k,fixedconstraint_colinds] <- 1
      pop[k,othercells_colinds] <- 1

    }


  }

  fitness_init <- evaluate_pop(pop,obj_wts,sparse_mat,b,constraint_ineq,fixedconstraint_colinds)[[1]]
  pop <- pop[-which(fitness_init < 1000),]  #(-100000)

  return(list(this_pop=pop,fixedconstraint_colinds=fixedconstraint_colinds))
}


# Evaluate the fitness value of each member of the population
# Eval = 0 if no. of land types allocated constraint exceeded
```

```
evaluate_pop = function(pop,obj_wts,sparse_mat,b,constraint_ineq,fixedconstraint_colinds){
  popsize = length(pop[,1])
  varsize = length(pop[1,])
  evals = rep(0,popsize)
  numconstraints <- dim(sparse_mat)[1]
  numconstrsatisfied <- matrix(0,popsize,1)
  min_max_constr <- matrix(0,2,9)
  min_max_constr[1,] <- b[seq(1,18,2)]
  min_max_constr[2,] <- b[seq(2,18,2)]

  total_obj <- apply(pop,1,function(x){sum(x*obj_wts)})

  for(i in 1:popsize){
    #total_constraint <- apply(sparse_mat,1,function(x){sum(pop[i,]*x)})

    #convert from vector to 400 by 9 matrix
    pop_temp <- matrix(pop[i,],nrow = 400,ncol = 9,byrow=TRUE)
    num_vals_per_type <- apply(pop_temp,2,sum)

    #check fixed constraints
    if (sum(pop[i,fixedconstraint_colinds])!=44){
      evals[i] <- 0
      #evals[i] <- total_obj[i] - 200*abs(sum(pop[i,fixedconstraint_colinds])!=44)
    } else if(sum(num_vals_per_type < min_max_constr[1,] |  num_vals_per_type > min_max_constr[2,])>0
      evals[i] <- 0
      #evals[i] <- total_obj[i] - 200*abs(sum(num_vals_per_type < min_max_constr[1,] |  num_vals_per_
    } else if (sum(pop[i,])!=400){
      evals[i] <- 0
      #evals[i] <- total_obj[i] - 200*abs(sum(pop[i,])-400)
    } else{
      evals[i] <- total_obj[i]
    }

  }
  return(list(evals=evals,fixedconstraint_colinds=fixedconstraint_colinds))
}

# Function for rank based selection: create
rank_based_selection = function(pop,fitness){
  popsize = length(pop[,1])
  varsize = length(pop[1,])
  #compute rank-based probabilities of surviving
  probs <- matrix(0,dim(pop))

  #get ranks of the populations fitness
  sorted_ranked_index <- order(unlist(fitness),decreasing = T, runif(length(unlist(fitness))))
  #organise population in descending order from most fit to least fit
  sorted_pop <- pop[sorted_ranked_index,]

  #specify size of poulation to be carried through iterations
  parent_size <- 400

  #linearly interpolation between 1 and xi=0.3
```

```
  probs <- approx(c(1,parent_size), y = c(1,0.5), method="linear", n=dim(sorted_pop)[1])$y

  #sample new population with above probabilities
  next_parents <- sorted_pop[sample(1:dim(sorted_pop)[1],size=parent_size,replace=TRUE,prob = probs),]

  return(next_parents)
}

# Function for crossover step (reproduction)
reproduce = function(parents,obj_wts,sparse_mat,b,constraint_ineq,fixedconstraint_colinds){
  #children <- matrix(NA,dim(parents)[1],dim(parents)[2])
  popsize <- dim(parents)[1]
  varsize <- dim(parents)[2]
  n_rows <- 20
  n_cols <- 20

  #randomly generate indices of mates - mates will be successive pairs of rows
  mateind <- sample(1:popsize,popsize,replace = FALSE)

  # reprodution
  children <- matrix(0,dim(parents)[1],dim(parents)[2])
  for (i in seq(1,popsize,2)){
    mates <- parents[mateind[i:(i+1)],]

    #convert parents into 400x9 grids to find possible swaps
    mate1 <- matrix(mates[1,],nrow = 400,ncol = 9,byrow=TRUE)
    mate2 <- matrix(mates[2,],nrow = 400,ncol = 9,byrow=TRUE)

    mate1values <- rep(0,n_rows*n_cols)
    mate2values <- rep(0,n_rows*n_cols)

    #get the vlaues in each of the cells fot both parents
    mate1values <-  unlist(apply(mate1,1,function(x){as.numeric(which(x!=0))}))
    mate2values <-  unlist(apply(mate2,1,function(x){as.numeric(which(x!=0))}))

    land_type_combs <- combn(c(1:9),2)
    #land_type_combs <- land_type_combs[,sample(1:36,10)]

    for (k in 1:dim(land_type_combs)[1]){
      randswapvals <- land_type_combs[,k]
      mate2valsforindsinmate <- which((mate1values==randswapvals[1])*(mate2values==randswapvals[2])>0)
      mate1valsforindsinmate <- which((mate1values==randswapvals[2])*(mate2values==randswapvals[1])>0)

      #calculate hwo many of first land type in first parent and second landtype in second parent div
      numberlandtypes_for_combo_to_swap1 <- length(mate2valsforindsinmate)/2
      numberlandtypes_for_combo_to_swap2 <- length(mate1valsforindsinmate)/2

      #sample half of indices to be swapped
      swapinds1 <- sample(mate2valsforindsinmate,numberlandtypes_for_combo_to_swap1,replace = F)
      swapinds2 <- sample(mate1valsforindsinmate,numberlandtypes_for_combo_to_swap2,replace = F)

      #swap for parent 1
      mate1[swapinds1,] <- 0
```

```
      mate1[swapinds1,randswapvals[2]] <- 1
      mate1[swapinds2,] <- 0
      mate1[swapinds2,randswapvals[1]] <- 1

      #swap for parent 2
      mate2[swapinds1,] <- 0
      mate2[swapinds1,randswapvals[1]] <- 1
      mate2[swapinds2,] <- 0
      mate2[swapinds2,randswapvals[2]] <- 1

    }

    #convert children back to vector form
    children[i,] <- as.vector(t(mate1))
    children[i+1,] <- as.vector(t(mate2))

  }

  #childrenfit <- evaluate_pop(children,obj_wts,sparse_mat,b,constraint_ineq,fixedconstraint_colinds)

  ## combine parents and children and return the top 'parent size' genes from the overall population
  totalpop <- rbind(parents,children)
  sorted_ranked_index <- order(evaluate_pop(totalpop,obj_wts,sparse_mat,b,constraint_ineq,fixedconstra

  bestpop <- totalpop[-sorted_ranked_index[-c(1:popsize)],]

  #return(totalpop)
  return(bestpop)
}

# Function for mutation step
mutations = function(pop,mutation_rate = 0.5){
  popsize <- dim(pop)[1]
  varsize <- dim(pop)[2]
  # MUTATE Generate the new population by mutation

  #which genes to mutate
  genesToMutate <- which(runif(popsize, min = 0, max = 1) < mutation_rate)

  #we then need to mutate 2 points in the gene as changing a single 1 to 0 (vice versa) will lead to
  # to do this, sample a row and cloumn index for each mutation and find another row and column in th
  pointsingenetomutaterows <- sample(1:20,length(genesToMutate),replace = T)
  pointsingenetomutatecols <- sample(1:20,length(genesToMutate),replace = T)

  #check if any mutations occur
  if (length(genesToMutate)!=0){
    for(i in 1:length(genesToMutate)){
      #convert parents into 20x20 grids of values 1 to 9 to find possible swaps
      genemutate <- matrix(pop[genesToMutate[i],],nrow = 400,ncol = 9,byrow=TRUE)
      if(sum(apply(genemutate,1,sum))==400){


      genemutatevalues <- rep(0,20*20)
```

```
        for(w in 1:(20*20)){
          genemutatevalues[w] <-  as.numeric(which(genemutate[w,]!=0))
        }
        #20x20 grids of values for land-use types
        genemutategrid <- matrix(genemutatevalues,nrow = 20,ncol = 20,byrow=TRUE)

        #locate what value is in the row and column of the current mutation
        mutvalue <- genemutategrid[pointsingenetomutaterows[i],pointsingenetomutaterows[i]]
        #change the value to anything but its current value
        valuenew <- sample((1:9)[-mutvalue],1)
        genemutategrid[pointsingenetomutaterows[i],pointsingenetomutaterows[i]] <- valuenew

        #find another index in the grid that has the current land-use as this new land-use and set it t
        genemutategrid[sample(which(genemutategrid==valuenew),1)] <- mutvalue

        #convert grids back into vectors of 3600 but first set the current gene thats being mutated to
        pop[genesToMutate[i],] <- 0
        for (m in 1:20){
          for (n in 1:20){
            pop[genesToMutate[i],(m-1)*20*9 + (n-1)*9 + genemutategrid[m,n]] <- 1
          }
        }
        }
      }
    }
    return(pop)
  }

  ## putting it all together
  varsize <- length(obj_wts)
  this_pop <- initial_pop(varsize,pop_size)
  fixedconstraint_colinds <- this_pop$fixedconstraint_colinds
  this_pop <- this_pop$this_pop

  max_evals = c()
  mean_evals = c()
  nconstraintssat = c()

  for(gen in 1:num_gens){
    evals = evaluate_pop(this_pop,obj_wts,sparse_mat,b,constraint_ineq,fixedconstraint_colinds)
    next_parents = rank_based_selection(pop=this_pop,fitness=evals$evals)
    next_children = reproduce(parents=next_parents,obj_wts,sparse_mat,b,constraint_ineq,fixedconstraint
    next_children = rank_based_selection(pop=next_children,fitness=evaluate_pop(next_children,obj_wts,s
    this_pop = mutations(pop=next_children,mutation_rate=0.5)
    #this_pop = next_children
    max_evals = c(max_evals,max(evals$evals))
    mean_evals = c(mean_evals,mean(evals$evals))
    #nconstraintssat <- c(nconstraintssat,max(evals[[2]]))
  }


  return(list(evals=evals$evals,pop=this_pop,maxeval=max_evals,meaneval=mean_evals,ngenerations = num_g
}
```

```
## EOF - GA

###############
## Constraints

#initialize
n <- 3600
nrows <- 20
ncols <- 20
nooflandtypes <- 9
## coefficients of the constraint matrix
#predifine the constraints that make up the 3600x3600 matrix of constraints
int_agriconstmin_ind <- numeric()
int_agriconstmax_ind <- numeric()
ext_agriconstmin <- numeric()
ext_agriconstmax <- numeric()
residencemin <- numeric()
residencemax <- numeric()
industrymin <- numeric()
industrymax <- numeric()
recreationdaymin <- numeric()
recreationdaymax <- numeric()
recreationnightmin <- numeric()
recreationnightmax <- numeric()
wetareamin <- numeric()
wetareamax <- numeric()
waterrecreationalmin <- numeric()
waterrecreationalmax <- numeric()
waterlimitedmin <- numeric()
waterlimitedmax <- numeric()

# populate the column indices for each constraint
for (i in seq(1, nrows*ncols*nooflandtypes,9)) {
  int_agriconstmin_ind <- cbind(int_agriconstmin_ind,i)
  int_agriconstmax_ind <- cbind(int_agriconstmax_ind,i)
  ext_agriconstmin <- cbind(ext_agriconstmin,i+1)
  ext_agriconstmax <- cbind(ext_agriconstmax,i+1)
  residencemin <- cbind(residencemin,i+2)
  residencemax <- cbind(residencemax,i+2)
  industrymin <- cbind(industrymin,i+3)
  industrymax <- cbind(industrymax,i+3)
  recreationdaymin <- cbind(recreationdaymin,i+4)
  recreationdaymax <- cbind(recreationdaymax,i+4)
  recreationnightmin <- cbind(recreationnightmin,i+5)
  recreationnightmax <- cbind(recreationnightmax,i+5)
  wetareamin <- cbind(wetareamin,i+6)
  wetareamax <- cbind(wetareamax,i+6)
  waterrecreationalmin <- cbind(waterrecreationalmin,i+7)
  waterrecreationalmax <- cbind(waterrecreationalmax,i+7)
  waterlimitedmin <- cbind(waterlimitedmax,i+8)
  waterlimitedmax <- cbind(waterlimitedmax,i+8)
}
```

```
# land (grid) constraint column ind
landconstraint <- as.numeric(c(1:n))

#create a vector of the column indices of all constraints
colinds <- cbind(int_agriconstmin_ind,int_agriconstmax_ind,ext_agriconstmin,
                 ext_agriconstmax,residencemin,residencemax,
                 industrymin,industrymax,recreationdaymin,recreationdaymax,recreationnightmin,
                 recreationnightmax,wetareamin,wetareamax,waterrecreationalmin,
                 waterrecreationalmax, waterlimitedmin,waterlimitedmax,t(landconstraint))

# Compute the row indices of activity constraints
rowinds <- numeric()
for (i in 1:(18)){
  rowinds <- c(rowinds,rep(i,400))
}
#add land constraints (400 blocks maximum)
rowinds <- c(rowinds,rep(19,n))

## add figure 5 constraints
#column indices
Fig5indices <- which(Fig5!=0,arr.ind = T)
Fig5indices <- Fig5indices[order(Fig5indices[,1]), ]
Fig5values <- as.vector(Fig5[Fig5indices])
fixedconstraint_colinds <- numeric()

for (i in 1:length(Fig5values)){
  # the following formula gives the vector index of the grid position (index)
  fixedconstraint_colinds <- c(fixedconstraint_colinds,(Fig5indices[i,1]-1)*20*9 +  Fig5indices[i,2]*9 -
}
fixedconstraint_colinds <- as.matrix(fixedconstraint_colinds)

#populate fixed constraints into vector indices of all other constraints
colinds <- c(colinds,fixedconstraint_colinds)
colinds <- as.matrix(colinds)

#row indices
for (i in 20:(19+length(Fig5values))){
  rowinds <- c(rowinds,i)
}
rowinds <- as.matrix(rowinds)

## add constraints for maximum of one activity per grid block
#blockconstraint <- numeric()
for (i in 64:(63+nrows*ncols)){
  rowinds <- c(rowinds,rep(i,9))
}
colinds <- c(colinds,1:n)

## create the spasre matrix of constraint coeffs
sparse_mat <- simple_triplet_matrix(i = rowinds, j = colinds, v = c(rep(1,length(colinds))))
#realmat <- as.matrix(sparse_mat_rec) #check that sparse matrix makes sense

#compute RHS of constraints
```

```
b <- c(80,150,20,65,20,45,5,15,0,70,0,35,0,30,120,150,0,60,400,rep(1,44),rep(1,400))

# signs of the constraints
constaint_ineq <- c(">=", "<=",">=", "<=",">=", "<=",">=", "<=", ">=","<=",">=", "<=",">=","<=",">=", "

############
## Natural

## coefficients of the objective function
objcoeffs <- rbind(rep(4,400),as.vector(t(Fig3a)),rep(3,400),rep(1,400),rep(5,400),rep(5,400),as.vector

Objective <- numeric()
for (j in 1:400) {
  for (i in 1:9){
    Objective <- c(Objective,objcoeffs[i,j])
  }
}

###############
## Recreational

## coefficients of the objective function
# objcoeffs <- rbind(rep(6,400),as.vector(t(Fig3a)),rep(3,400),rep(1,400),as.vector(t(Fig3b)),as.vector
#
# Objective <- numeric()
# for (j in 1:400) {
#   for (i in 1:9){
#     Objective <- c(Objective,objcoeffs[i,j])
#   }
# }
# Objective <- as.matrix(Objective)


###############
## Costs

## coefficients of the objective function
# Objective <- numeric()
# for (i in 1:nrows){
#   for (j in 1:ncols){
#     for (k in 1:nooflandtypes){
#       if(Fig2c[i,j]==k){
#         Objective <- c(Objective,as.matrix(cost_matrix[k,]))
#       }
#     }
#   }
# }
# Objective <- as.matrix(Objective)

# -+-+-+-+-+-+-+-+-+-+
## Genetic Algorithm |
```

```
num_gens <- 2000
initial_popsize <- 10000
ga_ans <- ga(sparse_mat, b, constraint_ineq, Objective, initial_popsize, num_gens)
mean_evals <- ga_ans$meaneval
max_evals <- ga_ans$maxeval

#plot mean fitness values over generations
plot(1:num_gens,mean_evals,xlab="Generations",ylab="Fitness",
     ylim=c(min(c(max_evals,mean_evals)),max(c(max_evals,mean_evals))))
lines(1:num_gens,max_evals,lty=2)


## find which individual gives max fitness and plot
max_indiv_fit <- which.max(ga_ans$evals)

LPsolveSolutiongrid <- matrix(ga_ans$pop[max_indiv_fit,],nrow = 400,ncol = 9,byrow=TRUE)

Solutiongridvalues <- matrix(NA,nrows*ncols,1)
for(i in 1:(nrows*ncols)){
  Solutiongridvalues[i,1] <-  which(LPsolveSolutiongrid[i,]!=0,arr.ind = T)
}
Solutiongrid <- matrix(Solutiongridvalues,nrow = 20,ncol = 20,byrow=TRUE)

Soldf <- data.frame(indv=factor(paste("Y",1:20),
                                levels = rev(paste("Y",1:20))), as.matrix(Solutiongrid))
Soldf <- Soldf[,1:20]

Soldf <- melt(Soldf, id.var = 'indv')
#get unique values of solution in case that some landtypes are missing
colourind <- sort(unique(c(Solutiongrid)))
col = c("green4","green","red4","aquamarine4","yellow","magenta","yellow4","turquoise1","skyblue2")

## plot grid
p <- ggplot(Soldf, aes(variable, indv,fill=as.factor(value))) + theme(axis.text.x=element_text(angle=90
p + guides(fill=guide_legend(title="Land-use Type")) + labs(x = "X",y = "Y")
```

7.5 Multiobjective Linear Program

```
## Optimisation Project
#
# Reuired:
# 1. Rdata file containing land uses for figure ... (a)
#
# Problem Specification:
#
# Author : N Murphy
#
## Version Control
# ~/R/Opt/Project/Land-use-planning

## 1. Clear environemnt and remove all plots
#rm(list=ls()) # clear environment
```

```
#dev.off()

## 2. load required libraries
library(Rglpk)

## 3. Load previously prepared data
load("data/spatialDSS_inputs_fig3a.RData")
Fig3a <- nature_values
load("data/spatialDSS_inputs_fig3b.RData")
load("data/spatialDSS_inputs_fig3c.RData")
load("data/spatialDSS_inputs_fig5.RData")
load("data/spatialDSS_inputs_fig2c.RData")
cost_matrix <- read.table("data/cost_matrix.txt",sep="",header = FALSE)

## coefficients of the objective function
objcoeffs <- rbind(rep(4,400),as.vector(t(Fig3a)),rep(3,400),rep(1,400),rep(5,400),rep(5,400),as.vector

Objective <- numeric()
for (j in 1:400) {
  for (i in 1:9){
    Objective <- c(Objective,objcoeffs[i,j])
  }
}
Objective_nature <- as.matrix(Objective)
objcoeffs_nat <- objcoeffs

## coefficients of the constraint matrix
#predifine the constraints that make up the 3600x3600 matrix of constraints
int_agriconstmin_ind <- numeric()
int_agriconstmax_ind <- numeric()
ext_agriconstmin <- numeric()
ext_agriconstmax <- numeric()
residencemin <- numeric()
residencemax <- numeric()
industrymin <- numeric()
industrymax <- numeric()
recreationdaymin <- numeric()
recreationdaymax <- numeric()
recreationnightmin <- numeric()
recreationnightmax <- numeric()
wetareamin <- numeric()
wetareamax <- numeric()
waterrecreationalmin <- numeric()
waterrecreationalmax <- numeric()
waterlimitedmin <- numeric()
waterlimitedmax <- numeric()

# populate the column indices for each constraint
for (i in seq(1, n,9)) {
  int_agriconstmin_ind <- cbind(int_agriconstmin_ind,i)
  int_agriconstmax_ind <- cbind(int_agriconstmax_ind,i)
  ext_agriconstmin <- cbind(ext_agriconstmin,i+1)
  ext_agriconstmax <- cbind(ext_agriconstmax,i+1)
```

```
  residencemin <- cbind(residencemin,i+2)
  residencemax <- cbind(residencemax,i+2)
  industrymin <- cbind(industrymin,i+3)
  industrymax <- cbind(industrymax,i+3)
  recreationdaymin <- cbind(recreationdaymin,i+4)
  recreationdaymax <- cbind(recreationdaymax,i+4)
  recreationnightmin <- cbind(recreationnightmin,i+5)
  recreationnightmax <- cbind(recreationnightmax,i+5)
  wetareamin <- cbind(wetareamin,i+6)
  wetareamax <- cbind(wetareamax,i+6)
  waterrecreationalmin <- cbind(waterrecreationalmin,i+7)
  waterrecreationalmax <- cbind(waterrecreationalmax,i+7)
  waterlimitedmin <- cbind(waterlimitedmax,i+8)
  waterlimitedmax <- cbind(waterlimitedmax,i+8)
}

# land (grid) constraint column ind
landconstraint <- as.numeric(c(1:n))

#create a vector of the column indices of all constraints
colinds <- cbind(int_agriconstmin_ind,int_agriconstmax_ind,ext_agriconstmin,
                 ext_agriconstmax,residencemin,residencemax,
                 industrymin,industrymax,recreationdaymin,recreationdaymax,recreationnightmin,
                 recreationnightmax,wetareamin,wetareamax,waterrecreationalmin,
                 waterrecreationalmax, waterlimitedmin,waterlimitedmax,t(landconstraint))

# Compute the row indices of activity constraints
rowinds <- numeric()
for (i in 1:(18)){
  rowinds <- c(rowinds,rep(i,400))
}
#add land constraints (400 blocks maximum)
rowinds <- c(rowinds,rep(19,n))

## add figure 5 constraints
#column indices
Fig5indices <- which(Fig5!=0,arr.ind = T)
Fig5indices <- Fig5indices[order(Fig5indices[,1]), ]
Fig5values <- as.vector(Fig5[Fig5indices])
fixedconstraint_colinds <- numeric()

for (i in 1:length(Fig5values)){
  # the following formula gives the vector index of the grid position (index)
  fixedconstraint_colinds <- c(fixedconstraint_colinds,(Fig5indices[i,1]-1)*20*9 + Fig5indices[i,2]*9 -
}
fixedconstraint_colinds <- as.matrix(fixedconstraint_colinds)

#populate fixed constraints into vector indices of all other constraints
colinds <- c(colinds,fixedconstraint_colinds)
colinds <- as.matrix(colinds)

#row indices
for (i in 20:(19+length(Fig5values))){
```

```
  rowinds <- c(rowinds,i)
}
rowinds <- as.matrix(rowinds)

## add constraints for maximum of one activity per grid block
#blockconstraint <- numeric()
for (i in 64:(63+nrows*ncols)){
  rowinds <- c(rowinds,rep(i,9))
}
colinds <- c(colinds,1:n)

## create the spasre matrix of constraint coeffs
sparse_mat_nature <- simple_triplet_matrix(i = rowinds, j = colinds, v = c(rep(1,length(colinds))))
realmat <- as.matrix(sparse_mat_nature) #check that sparse matrix makes sense

#compute RHS of constraints
b_nature <- c(80,150,20,65,20,45,5,15,0,70,0,35,0,30,120,150,0,60,400,rep(1,44),rep(1,400))

# signs of the constraints
constraint_ineq_nature <- c(">=", "<=",">=", "<=",">=", "<=",">=", "<=", ">=","<=",">=", "<=",">=","<="

#variable types
var_types <- rep("B",n)

## Solve the LP problem
LPsolve <- Rglpk_solve_LP(obj = Objective_nature, mat = sparse_mat_nature, dir = constraint_ineq_nature

## plot the optimal solution
LPsolveSolution <- as.matrix(LPsolve$solution)
#LPsolveSolution
sum(LPsolveSolution)

## display solution as grid
#first populate solution into grid form
LPsolveSolutiongrid <- matrix(LPsolveSolution,nrow = 400,ncol = 9,byrow=TRUE)

Solutiongridvalues <- matrix(NA,nrows*ncols,1)
for(i in 1:(nrows*ncols)){
  Solutiongridvalues[i,1] <-  which(LPsolveSolutiongrid[i,]!=0,arr.ind = T)
}
Solutiongrid <- matrix(Solutiongridvalues,nrow = 20,ncol = 20,byrow=TRUE)

#define colours for ggplot
col = c("green4","green","red4","aquamarine4","yellow","magenta","yellow4","turquoise1","skyblue2")

#get unique values of solution in case htat some landtypes are missing
colourind <- sort(unique(c(Solutiongrid)))

library(ggplot2);
library(reshape2)
Soldf <- data.frame(indv=factor(paste("Y",1:20),
                                levels = rev(paste("Y",1:20))), as.matrix(Solutiongrid))
Soldf <- Soldf[,1:20]
```

```
Soldf <- melt(Soldf, id.var = 'indv')

nature_opt <- LPsolve$optimum
nature_sol <- LPsolve$solution


## coefficients of the objective function
objcoeffs <- rbind(rep(6,400),as.vector(t(Fig3a)),rep(3,400),rep(1,400),as.vector(t(Fig3b)),as.vector(t

Objective <- numeric()
for (j in 1:400) {
  for (i in 1:9){
    Objective <- c(Objective,objcoeffs[i,j])
  }
}
Objective_rec <- as.matrix(Objective)
objcoeffs_rec <- objcoeffs

## coefficients of the constraint matrix
#predifine the constraints that make up the 3600x3600 matrix of constraints
int_agriconstmin_ind <- numeric()
int_agriconstmax_ind <- numeric()
ext_agriconstmin <- numeric()
ext_agriconstmax <- numeric()
residencemin <- numeric()
residencemax <- numeric()
industrymin <- numeric()
industrymax <- numeric()
recreationdaymin <- numeric()
recreationdaymax <- numeric()
recreationnightmin <- numeric()
recreationnightmax <- numeric()
wetareamin <- numeric()
wetareamax <- numeric()
waterrecreationalmin <- numeric()
waterrecreationalmax <- numeric()
waterlimitedmin <- numeric()
waterlimitedmax <- numeric()

# populate the column indices for each constraint
for (i in seq(1, nrows*ncols*nooflandtypes,9)) {
  int_agriconstmin_ind <- cbind(int_agriconstmin_ind,i)
  int_agriconstmax_ind <- cbind(int_agriconstmax_ind,i)
  ext_agriconstmin <- cbind(ext_agriconstmin,i+1)
  ext_agriconstmax <- cbind(ext_agriconstmax,i+1)
  residencemin <- cbind(residencemin,i+2)
  residencemax <- cbind(residencemax,i+2)
  industrymin <- cbind(industrymin,i+3)
  industrymax <- cbind(industrymax,i+3)
  recreationdaymin <- cbind(recreationdaymin,i+4)
  recreationdaymax <- cbind(recreationdaymax,i+4)
  recreationnightmin <- cbind(recreationnightmin,i+5)
  recreationnightmax <- cbind(recreationnightmax,i+5)
```

```
  wetareamin <- cbind(wetareamin,i+6)
  wetareamax <- cbind(wetareamax,i+6)
  waterrecreationalmin <- cbind(waterrecreationalmin,i+7)
  waterrecreationalmax <- cbind(waterrecreationalmax,i+7)
  waterlimitedmin <- cbind(waterlimitedmax,i+8)
  waterlimitedmax <- cbind(waterlimitedmax,i+8)
}

# land (grid) constraint column ind
landconstraint <- as.numeric(c(1:n))

#create a vector of the column indices of all constraints
colinds <- cbind(int_agriconstmin_ind,int_agriconstmax_ind,ext_agriconstmin,
                 ext_agriconstmax,residencemin,residencemax,
                 industrymin,industrymax,recreationdaymin,recreationdaymax,recreationnightmin,
                 recreationnightmax,wetareamin,wetareamax,waterrecreationalmin,
                 waterrecreationalmax, waterlimitedmin,waterlimitedmax,t(landconstraint))

# Compute the row indices of activity constraints
rowinds <- numeric()
for (i in 1:(18)){
  rowinds <- c(rowinds,rep(i,400))
}
#add land constraints (400 blocks maximum)
rowinds <- c(rowinds,rep(19,n))

## add figure 5 constraints
#column indices
Fig5indices <- which(Fig5!=0,arr.ind = T)
Fig5indices <- Fig5indices[order(Fig5indices[,1]), ]
Fig5values <- as.vector(Fig5[Fig5indices])
fixedconstraint_colinds <- numeric()

for (i in 1:length(Fig5values)){
  # the following formula gives the vector index of the grid position (index)
  fixedconstraint_colinds <- c(fixedconstraint_colinds,(Fig5indices[i,1]-1)*20*9 +  Fig5indices[i,2]*9 -
}
fixedconstraint_colinds <- as.matrix(fixedconstraint_colinds)

#populate fixed constraints into vector indices of all other constraints
colinds <- c(colinds,fixedconstraint_colinds)
colinds <- as.matrix(colinds)

#row indices
for (i in 20:(19+length(Fig5values))){
  rowinds <- c(rowinds,i)
}
rowinds <- as.matrix(rowinds)

## add constraints for maximum of one activity per grid block
#blockconstraint <- numeric()
for (i in 64:(63+nrows*ncols)){
  rowinds <- c(rowinds,rep(i,9))
```

```
}
colinds <- c(colinds,1:n)

## create the spasre matrix of constraint coeffs
sparse_mat_rec <- simple_triplet_matrix(i = rowinds, j = colinds, v = c(rep(1,length(colinds))))
realmat <- as.matrix(sparse_mat_rec) #check that sparse matrix makes sense

#compute RHS of constraints
b_rec <- c(80,150,20,65,20,45,5,15,0,70,0,35,0,30,120,150,0,60,400,rep(1,44),rep(1,400))

# signs of the constraints
constaint_ineq_rec <- c(">=", "<=",">=", "<=",">=", "<=",">=", "<=", ">=","<=",">=", "<=",">=","<=",">=

#variable types
var_types <- rep("B",n)

## Solve the LP problem
LPsolve <- Rglpk_solve_LP(obj = Objective_rec, mat = sparse_mat_rec, dir = constaint_ineq_rec, rhs = b_

## plot the optimal solution
LPsolveSolution <- as.matrix(LPsolve$solution)
#LPsolveSolution
#sum(LPsolveSolution)

## display solution as grid
#first populate solution into grid form
LPsolveSolutiongrid <- matrix(LPsolveSolution,nrow = 400,ncol = 9,byrow=TRUE)

Solutiongridvalues <- matrix(NA,nrows*ncols,1)
for(i in 1:(nrows*ncols)){
  Solutiongridvalues[i,1] <-  which(LPsolveSolutiongrid[i,]!=0,arr.ind = T)
}
Solutiongrid <- matrix(Solutiongridvalues,nrow = 20,ncol = 20,byrow=TRUE)

#get unique values of solution in case htat some landtypes are missing
colourind <- sort(unique(c(Solutiongrid)))

col = c("green4","green","red4","aquamarine4","yellow","magenta","yellow4","turquoise1","skyblue2")


library(ggplot2);
library(reshape2)
Soldf <- data.frame(indv=factor(paste("Y",1:20),
                                levels = rev(paste("Y",1:20))), as.matrix(Solutiongrid))
Soldf <- Soldf[,1:20]

Soldf <- melt(Soldf, id.var = 'indv')

rec_opt <- LPsolve$optimum
rec_sol <- LPsolve$solution

## coefficients of the objective function
Objective <- numeric()
```

```
for (i in 1:nrows){
  for (j in 1:ncols){
    for (k in 1:nooflandtypes){
      if(Fig2c[i,j]==k){
        Objective <- c(Objective,as.matrix(cost_matrix[k,]))
      }
    }
  }
}
Objective_cost <- as.matrix(Objective)

## coefficients of the constraint matrix
#predifine the constraints that make up the 3600x3600 matrix of constraints
int_agriconstmin_ind <- numeric()
int_agriconstmax_ind <- numeric()
ext_agriconstmin <- numeric()
ext_agriconstmax <- numeric()
residencemin <- numeric()
residencemax <- numeric()
industrymin <- numeric()
industrymax <- numeric()
recreationdaymin <- numeric()
recreationdaymax <- numeric()
recreationnightmin <- numeric()
recreationnightmax <- numeric()
wetareamin <- numeric()
wetareamax <- numeric()
waterrecreationalmin <- numeric()
waterrecreationalmax <- numeric()
waterlimitedmin <- numeric()
waterlimitedmax <- numeric()

# populate the column indices for each constraint
for (i in seq(1, nrows*ncols*nooflandtypes,9)) {
  int_agriconstmin_ind <- cbind(int_agriconstmin_ind,i)
  int_agriconstmax_ind <- cbind(int_agriconstmax_ind,i)
  ext_agriconstmin <- cbind(ext_agriconstmin,i+1)
  ext_agriconstmax <- cbind(ext_agriconstmax,i+1)
  residencemin <- cbind(residencemin,i+2)
  residencemax <- cbind(residencemax,i+2)
  industrymin <- cbind(industrymin,i+3)
  industrymax <- cbind(industrymax,i+3)
  recreationdaymin <- cbind(recreationdaymin,i+4)
  recreationdaymax <- cbind(recreationdaymax,i+4)
  recreationnightmin <- cbind(recreationnightmin,i+5)
  recreationnightmax <- cbind(recreationnightmax,i+5)
  wetareamin <- cbind(wetareamin,i+6)
  wetareamax <- cbind(wetareamax,i+6)
  waterrecreationalmin <- cbind(waterrecreationalmin,i+7)
  waterrecreationalmax <- cbind(waterrecreationalmax,i+7)
  waterlimitedmin <- cbind(waterlimitedmax,i+8)
  waterlimitedmax <- cbind(waterlimitedmax,i+8)
}
```

```
# land (grid) constraint column ind
landconstraint <- as.numeric(c(1:n))

#create a vector of the column indices of all constraints
colinds <- cbind(int_agriconstmin_ind,int_agriconstmax_ind,ext_agriconstmin,
                 ext_agriconstmax,residencemin,residencemax,
                 industrymin,industrymax,recreationdaymin,recreationdaymax,recreationnightmin,
                 recreationnightmax,wetareamin,wetareamax,waterrecreationalmin,
                 waterrecreationalmax, waterlimitedmin,waterlimitedmax,t(landconstraint))


# Compute the row indices of activity constraints
rowinds <- numeric()
for (i in 1:(18)){
  rowinds <- c(rowinds,rep(i,400))
}
#add land constraints (400 blocks maximum)
rowinds <- c(rowinds,rep(19,n))

## add figure 5 constraints
#column indices
Fig5indices <- which(Fig5!=0,arr.ind = T)
Fig5indices <- Fig5indices[order(Fig5indices[,1]), ]
Fig5values <- as.vector(Fig5[Fig5indices])
fixedconstraint_colinds <- numeric()

for (i in 1:length(Fig5values)){
  # the following formula gives the vector index of the grid position (index)
  fixedconstraint_colinds <- c(fixedconstraint_colinds,(Fig5indices[i,1]-1)*20*9 +  Fig5indices[i,2]*9 -
}
fixedconstraint_colinds <- as.matrix(fixedconstraint_colinds)

#populate fixed constraints into vector indices of all other constraints
colinds <- c(colinds,fixedconstraint_colinds)
colinds <- as.matrix(colinds)

#row indices
for (i in 20:(19+length(Fig5values))){
  rowinds <- c(rowinds,i)
}
rowinds <- as.matrix(rowinds)

## add constraints for maximum of one activity per grid block
#blockconstraint <- numeric()
for (i in 64:(63+nrows*ncols)){
  rowinds <- c(rowinds,rep(i,9))
}
colinds <- c(colinds,1:n)

## add constraints for land types that are infeasible to convert
for (i in 1:nrows){
  for (j in 1:ncols){
    for (k in 1:nooflandtypes){
```

```
      if(Fig2c[i,j]=="NA"){
        colinds <- c(colinds,(i-1)*20*9 +  (j-1)*9 + k)
      }
    }
  }
}

## create the sparse matrix of constraint coeffs
sparse_mat_cost <- simple_triplet_matrix(i = rowinds, j = colinds, v = c(rep(1,length(colinds))))
#realmat <- as.matrix(sparse_mat) #check that sparse matrix makes sense

#compute RHS of constraints
b_cost <- c(80,150,20,65,20,45,5,15,0,70,0,35,0,30,120,150,0,60,400,rep(1,44),rep(1,400))

# signs of the constraints
constaint_ineq_cost <- c(">=", "<=",">=", "<=",">=", "<=",">=", "<=", ">=","<=",">=", "<=",">=","<=",">=

#variable types
var_types <- rep("B",n)

## Solve the LP problem
LPsolve <- Rglpk_solve_LP(obj = Objective_cost, mat = sparse_mat_cost, dir = constaint_ineq_cost, rhs =
costs_opt <- LPsolve$optimum
options(scipen = 999)
cost_sol <- LPsolve$solution


################################################################################
## Once we have data frrom 3 individual optimizations, perform MOGP
## Payoff table
solutions <- rbind(nature_sol,rec_sol,cost_sol)
objs <- cbind(Objective_nature,Objective_rec,Objective_cost)
payoff_tab <- matrix(0,3,3)

for (i in 1:3){
  payoff_tab[i,1] <- as.vector(solutions[i,])%*%objs[,1]
  payoff_tab[i,2] <- as.vector(solutions[i,])%*%objs[,2]
  payoff_tab[i,3] <- as.vector(solutions[i,])%*%objs[,3]
}

# set infeasible solutions to 0
payoff_tab[payoff_tab<=-100000] <- 0
#payoff_tab[payoff_tab<0] <- payoff_tab[payoff_tab<0] + 1000000

## goals
#specify priority of each individual objective
priority_levels <- rep(0.5,3)
#compute the range between optimum value and minimum value for each objective in payoff table
goal_range <- diag(payoff_tab) - apply(payoff_tab,2,min)
#use priority and range to get value of goal for each obj
goals <- apply(payoff_tab,2,min) + priority_levels*goal_range
#goals <- diag(payoff_tab)
```

```r
payoff_tab_dat <- data.frame(payoff_tab)
colnames(payoff_tab_dat) <- c("Obj 1","Obj 2","Obj 3")
row.names(payoff_tab_dat) <- c("Obj 1","Obj 2","Obj 3")

#knitr::kable(payoff_tab_dat,caption="Payoff table for the 3 objectives. Rows represent the objective b

## recast goals as constraints
#add objective constraints that must satisfy
rowinds <- c(rowinds,rep(464,3601),rep(465,3601),rep(466,3601))

constraint_coeffs <- c(rep(1,length(colinds)),Objective_nature,1,Objective_rec,1,Objective_cost,1)
colinds <- c(colinds,1:3601,1:3600,3602,1:3600,3603)

#add zero for each weight onto end of each constraint
sparse_mat <- simple_triplet_matrix(i = rowinds, j = colinds, v = constraint_coeffs,nrow = 466,ncol = 3

#compute RHS of constraints
b <- c(80,150,20,65,20,45,5,15,0,70,0,35,0,30,120,150,0,60,400,rep(1,44),rep(1,400),goals)

# signs of the constraints
constaint_ineq <- c(">=", "<=",">=", "<=",">=", "<=",">=", "<=", ">=","<=",">=", "<=",">=","<=",">=", "

#variable types
var_types <- rep("B",n+3)

## Solve the LP problem
LPsolve <- Rglpk_solve_LP(obj = c(rep(0,3600),priority_levels/sum(priority_levels)), mat = sparse_mat, 

## plot the optimal solution
LPsolveSolution <- as.matrix(LPsolve$solution[1:3600])

## display solution as grid
#first populate solution into grid form
LPsolveSolutiongrid <- matrix(LPsolveSolution,nrow = 400,ncol = 9,byrow=TRUE)

Solutiongridvalues <- matrix(NA,nrows*ncols,1)
for(i in 1:(nrows*ncols)){
  Solutiongridvalues[i,1] <-  which(LPsolveSolutiongrid[i,]!=0,arr.ind = T)
}

Solutiongrid <- matrix(Solutiongridvalues,nrow = 20,ncol = 20,byrow=TRUE)

library(ggplot2)
library(reshape2)
library(gplots)
Soldf <- data.frame(indv=factor(paste("Y",1:20),levels = rev(paste("Y",1:20))), as.matrix(Solutiongrid)
Soldf <- melt(Soldf, id.var = 'indv')

#get unique values of solution in case htat some landtypes are missing
colourind <- sort(unique(c(Solutiongrid)))

col = c("green4","green","red4","aquamarine4","yellow","magenta","yellow4","turquoise1","skyblue2")
col <- col[colourind]
```

```
p <- ggplot(Soldf, aes(variable, indv,fill=as.factor(value)))+theme(axis.text.x=element_text(angle=90,v
p + guides(fill=guide_legend(title="Land-use Type")) + labs(x = "X",y = "Y")
```

7.6 Create Figures

```
## Optimisation Project
#
# Reuired:
# 1. Rdata file containing land uses for figure ... (a)
#
# Problem Specification:
#
# Author : N Murphy
#
## Version Control
# ~/R/Opt/Project/Land-use-planning

## 1. Clear environemnt and remove all plots
rm(list=ls()) # clear environment
#dev.off()

## 2. load required libraries


#-+-+-+-+-+-+-+
## Figure 2 c |
#-+-+-+-+-+-+-+
r1 <- c(1,1,8,rep(1,4),8,rep(1,4),rep(8,3),rep(1,5))
r2 <- c(1,1,8,8,rep(1,3),rep(8,4),rep(7,3),rep(8,3),rep(1,3))
r3 <- c(1,1,1,rep(8,2),rep(1,2),8,rep(1,2),rep(8,2),7,rep(8,7))
r4 <- c(1,1,1,1,rep(8,2),1,8,rep(1,3),rep(8,7),rep(1,2))
r5 <- c(rep(1,5),8,1,rep(8,3),rep(1,3),rep(8,7))
r6 <- c(rep(8,4),1,rep(8,3),7,rep(8,5),rep(1,6))
r7 <- c(rep(1,3),rep(8,4),rep(7,3),8,8,rep(1,5),rep(8,3))
r8 <- c(rep(1,4),6,1,rep(8,9),1,8,8,1,1)
r9 <- c(rep(3,5),rep(8,3),rep(1,6),8,8,8,1,1,1)
r10 <- c(rep(5,2),3,rep(8,7),1,1,1,8,8,1,8,8,8,8)
r11 <- c(5,rep(8,13),rep(1,6))
r12 <- c(8,8,1,rep(8,9),1,8,8,rep(1,5))
r13 <- c(rep(1,3),rep(8,6),1,1,8,1,1,rep(8,6))
r14 <- c(1,rep(8,8),1,1,8,8,rep(1,7))
r15 <- c(8,8,1,1,rep(8,4),5,3,1,1,8,8,8,8,rep(1,4))
r16 <- c(rep(1,5),8,8,8,5,rep(3,3),8,1,1,rep(8,5))
r17 <- c(1,1,rep(8,5),rep(3,5),8,rep(1,4),8,rep(1,2))
r18 <- c(8,8,rep(1,5),rep(3,4),5,rep(8,8))
r19 <- c(8,rep(1,3),8,rep(3,6),rep(4,2),8,rep(1,6))
r20 <- c(rep(8,5),rep(3,3),rep(4,5),8,8,rep(1,5))

Fig2c <- rbind(r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15,r16,r17,r18,r19,r20)
```

```r
library(ggplot2);
library(reshape2)
col = c("green4","red4","aquamarine4","yellow","magenta","yellow4","turquoise1")

Fig2cdf <- data.frame(indv=factor(paste("Y",1:20),
                                  levels = rev(paste("Y",1:20))), as.matrix(Fig2c))
Fig2cdf1 <- Fig2cdf[,1:20]

Fig2cdf2 <- melt(Fig2cdf, id.var = 'indv')

Fig2cdf2[Fig2cdf2[,3]==1,3] <- 2
#Fig2cdf2[,3] <- Fig2cdf2[,3]

p <- ggplot(Fig2cdf2, aes(variable, indv,fill=as.factor(value))) + geom_tile(color = "gray") +   scale_
p + guides(fill=guide_legend(title="Land-use Type")) + labs(x = "X",y = "Y")
#ggsave("figures/Fig2c.png")

save(Fig2c, file = "data/spatialDSS_inputs_fig2c.RData")


#-+-+-+-+-+-+-+
## Figure 3 a |
#-+-+-+-+-+-+-+

load("data/spatialDSS_inputs_fig3a.RData")
Fig3a <- nature_values

jpeg('figures/Fig3a.jpg',width=600,height=600)
# heatmap.2(as.matrix(Fig3a), Rowv=FALSE, Colv=FALSE, dendrogram="none", main="Figure 3 a", col=pal, tr
#           sepcolor="black",
#           colsep=1:ncol(Fig3a),
#           rowsep=1:nrow(Fig3a),
#           key=FALSE)
# dev.off()

library(ggplot2);
library(reshape2)

Fig3adf <- data.frame(indv=factor(paste("Y",1:20),
                                  levels = rev(paste("Y",1:20))), as.matrix(Fig3a))
#Fig3adf1 <- Fig3adf[,1:20]

Fig3adf2 <- melt(Fig3adf, id.var = 'indv')

#Fig3adf2[Fig3adf2[,3]==1,3] <- 2

p <- ggplot(Fig3adf2, aes(variable, indv)) + geom_tile(aes(fill = value),color = "gray") + scale_colour_
p + guides(fill=guide_legend(title="Value")) + labs(x = "X",y = "Y")


#-+-+-+-+-+-+-+
## Figure 3 b |
#-+-+-+-+-+-+-+
```

```
rm(list=ls()) # clear environment

c1 <- c(4,4,4,4,4,10,4,4,4,10,10,10,4,4,4,4,4,4,4,4)
c2 <- c(4,4,4,4,4,10,4,4,4,10,10,10,4,4,4,4,4,4,4,4)
c3 <- c(4,4,4,4,4,10,4,4,4,4,10,10,10,4,4,4,4,4,4,4)

c4 <- c(4,4,4,4,4,10,10,10,4,10,10,10,10,10,4,4,4,4,4,4)

c5 <- c(4,4,4,4,4,4,10,10,4,10,10,10,10,10,10,4,4,4,4,4)

c6 <- c(4,4,4,4,4,10,10,10,10,10,10,10,10,10,10,10,10,10,4,4,4)
c7 <- c(4,4,4,4,4,10,10,10,10,10,10,10,10,10,10,10,10,10,4,4,4)

c8 <- c(10,10,10,10,10,10,4,10,10,10,10,10,10,10,10,10,10,4,4,4,4)

c9 <- c(4,10,4,4,10,4,4,10,4,10,10,10,10,10,10,10,4,4,4,4)

c10 <- c(4,10,4,4,10,10,4,10,4,10,10,10,10,10,4,4,4,4,4,4)

c11 <- c(4,10,10,4,4,10,10,10,4,4,10,10,10,10,4,4,4,4,4,4)

c12 <- c(4,4,10,10,4,10,10,10,4,4,10,10,10,10,4,4,4,4,4,4)

c13 <- c(4,4,4,10,4,10,4,10,4,4,10,4,4,4,4,4,4,4,4,4)

c14 <- c(4,4,10,10,10,10,4,10,4,10,10,4,4,4,4,4,4,4,4,4)

c15 <- c(4,10,10,10,10,4,4,10,10,10,4,4,4,4,4,4,4,4,4,4)

c16 <- c(4,10,10,10,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c17 <- c(4,10,10,10,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c18 <- c(4,4,10,10,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c19 <- as.vector(rep(4,20))

c20 <- as.matrix(rep(4,20))

Fig3b <- as.matrix(cbind(c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16,c17,c18,c19,c20))
# display fig 3b
library(gplots)
pal <- colorRampPalette(c(rgb(0.96,0.96,1), rgb(0.1,0.1,0.9)), space = "rgb")

jpeg('figures/Fig3b.jpg',width=600,height=600)
# heatmap.2(as.matrix(Fig3b), Rowv=FALSE, Colv=FALSE, dendrogram="none", main="Figure 3 b", col=pal, tr
#           sepcolor="black",
#           colsep=1:ncol(Fig3b),
#           rowsep=1:nrow(Fig3b),
#           labRow = c(1:20),
#           labCol = c(1:20),
#           key=FALSE)
dev.off()
```

```r
# save file
Fig3b <- as.data.frame(Fig3b)
saveRDS(Fig3b, file = "data/spatialDSS_inputs_fig3b.Rda")
save(Fig3b, file = "data/spatialDSS_inputs_fig3b.RData")

Fig3adf <- data.frame(indv=factor(paste("Y",1:20),
                                  levels = rev(paste("Y",1:20))), as.matrix(Fig3a))
#Fig3adf1 <- Fig3adf[,1:20]

Fig3adf2 <- melt(Fig3adf, id.var = 'indv')

Fig3adf2[Fig3adf2[,3]==1,3] <- 2

p <- ggplot(Fig3adf2, aes(variable, indv)) + geom_tile(aes(fill = value),color = "gray") + scale_colour
p + guides(fill=guide_legend(title="Value")) + labs(x = "X",y = "Y")


#-+-+-+-+-+-+-+
## Figure 3 c |
#-+-+-+-+-+-+-+
rm(list=ls()) # clear environment

c1 <- rep(4,20)
c2 <- rep(4,20)
c3 <- c(4,4,4,4,4,4,4,4,4,10,10,10,10,10,4,4,4,4,4,4)

c4 <- c(4,4,4,4,4,4,4,4,4,10,4,4,4,4,4,4,10,4,4,4,4,4)

c5 <- c(4,4,4,4,4,4,4,4,4,10,4,4,4,4,4,4,4,10,10,4,4,4)

c6 <- c(4,4,4,4,4,4,4,4,10,4,4,4,4,4,4,4,4,4,4,10,4,4)

c7 <- c(4,4,4,4,4,4,4,10,4,4,4,4,4,4,4,4,4,4,4,10,4,4)

c8 <- c(4,4,4,4,4,4,4,10,4,4,4,4,4,4,4,4,4,4,4,10,4,4,4)

c9 <- c(4,4,4,4,4,4,4,4,10,10,4,4,4,4,4,4,10,10,4,4,4,4,4)

c10 <- c(4,4,4,4,4,4,4,4,4,10,4,4,4,4,10,10,4,4,4,4,4,4,4)

c11 <- c(4,4,4,4,4,4,4,4,4,4,4,10,4,4,10,4,4,4,4,4,4,4,4)

c12 <- c(4,4,4,4,4,4,4,4,4,4,4,4,10,10,4,4,4,4,4,4,4,4,4)

c13 <- c(4,4,10,10,10,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c14 <- c(10,10,4,4,4,4,4,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c15 <- c(4,4,4,4,4,4,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c16 <- c(10,4,4,4,4,4,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c17 <- c(10,4,4,4,4,4,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)
```

```r
c18 <- c(4,10,4,4,4,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c19 <- c(4,4,10,10,10,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4)

c20 <- rep(4,20)

Fig3c <- cbind(c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16,c17,c18,c19,c20)
# display fig 3b
library(gplots)

pal <- colorRampPalette(c(rgb(0.96,0.96,1), rgb(0.1,0.1,0.9)), space = "rgb")
jpeg('figures/Fig3c.jpg',width=600,height=400)
# heatmap.2(as.matrix(Fig3c), Rowv=FALSE, Colv=FALSE, dendrogram="none", main="Figure 3 c", col=pal, tr
#           sepcolor="black",
#           colsep=1:ncol(Fig3c),
#           rowsep=1:nrow(Fig3c),
#           labRow = c(1:20),
#           labCol = c(1:20),
#           key=FALSE)
dev.off()

# save matrix
Fig3c <- as.data.frame(Fig3c)
save(Fig3c, file = "data/spatialDSS_inputs_fig3c.RData")


# -+-+-+-+-+-+
## Figure 5  |
# -+-+-+-+-+-+
r1 <- rep(0,20)
r2 <- c(rep(0,11),rep(7,3),rep(0,6))
r3 <- c(rep(0,12),7,rep(0,7))
r4 <- rep(0,20)
r5 <- rep(0,20)
r6 <- c(rep(0,8),7,rep(0,11))
r7 <- c(rep(0,7),rep(7,3),rep(0,10))
r8 <- c(rep(0,4),6,rep(0,15))
r9 <- c(rep(3,5),rep(0,15))
r10 <- c(rep(0,2),3,rep(0,17))
r11r14 <- matrix(0,4,20)
r15 <- c(rep(0,9),3,rep(0,10))
r16 <- c(rep(0,9),rep(3,3),rep(0,8))
r17 <- c(rep(0,7),rep(3,5),rep(0,8))
r18 <- c(rep(0,7),rep(3,4),rep(0,9))
r19 <- c(rep(0,5),rep(3,6),rep(4,2),rep(0,7))
r20 <- c(rep(0,5),rep(3,3),rep(4,5),rep(0,7))

Fig5 <- rbind(r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11r14,r15,r16,r17,r18,r19,r20)


# display fig 5
library(gplots)
pal <- colorRampPalette(c(rgb(0.96,0.96,1), rgb(0.1,0.1,0.9)), space = "rgb")
```

```r
col = c("white","red4","cyan4","magenta","yellow4")

jpeg('figures/Fig5.jpg',width=600,height=400)
# heatmap.2(as.matrix(Fig5),scale="none",breaks = c(0,2,3,5,6,9),col=col ,Rowv=FALSE, Colv=FALSE, dendr
#           sepcolor="black",
#           colsep=1:ncol(Fig5),
#           rowsep=1:nrow(Fig5),
#           labRow = c(1:20),
#           labCol = c(1:20),
#           key=FALSE)
dev.off()

# save matrix
Fig5 <- as.data.frame(Fig5)
save(Fig5, file = "data/spatialDSS_inputs_fig5.RData")

## Save land use types as data
landtypes <- c("Intensive Agriculture","Extensive Agriculture","Residence","Industry","Recreation (day
save(landtypes,file="data/landtypes.RData")

## EOF
```