HabplanR

Max Dolton Jones

2023-06-28

Introduction to HabplanR

The *HabplanR* package was developed to compliment Habplan software. Habplan is a landscape management and harvest scheduling program that performs multi-objective optimization via a Metropolis-Hastings algorithm. The functionality of Habplan is still sought after in todays landscape management problems; however, the use of Habplan is limited, requiring in-depth knowledge of not only the managment issue at hand, but also about tedious file-formatting and input requirements for the program to run efficiently. This is where *HabplanR* comes in!

The main functions of *HabplanR* is to provide important ecological context to the multi-objective optimization performed by Habplan, and to streamline the use of the program. Specifically, the *HabplanR* R package has the following objectives:

- Use forest simulation data to create Habplan-appropriate input files
- Use forest simulation data to develop context-specific Habitat Suitability Index values
- Set Habplan program parameters and configurations within the R environment prior to program initiation
- Visualize Habplan outputs
- Perform spatial analysis focused at either the forest stand, habitat patch or entire landscape level

The following vignette provides a working example of using the *HabplanR* package.

The *HabplanR* package is currently open-access on a GitHub repository. To download and install the package, we first need the devtools package.

```
#Install devtools:
install.packages("devtools") #Ignore if previously installed
#Load devtools into the session:
library(devtools)
```

We can now use the devtools package to install *HabplanR* from GitHub.

```
install_github("maxdoltonjones/HabplanR",  build_vignettes = TRUE, force = TRUE)

#The installation will ask if any package updates are needed, we can skip these
#by running the number three (3):
3
```

The installation of the package should also prompt the installation of all package dependencies (other packages which are needed for *HabplanR* to work). However, if this is not the case, run the following lines (this may take a few minutes or so):

```
install.packages("readr")
install("dplyr")
install.packages("ggplot2")
install.packages("rgdal")
install.packages("broom")
install.packages("spdep")
install.packages("rgeos")
install.packages("terra")
install.packages("tidyterra")
```

Again, if these packages are already installed, the dependencies will be loaded at the same time as *HabplanR* is loaded. However, we can also load all of the dependencies separately.

```
#We include the package versions used to create this vignette

library(readr) #v.2.1.2
library(dplyr) #v.1.1.0.10
library(ggplot2) #v.3.3.6
library(rgdal) #v.1.5-32
library(broom) #v.0.8.0
library(spdep) #v.1.2-7
library(rgeos) #v.0.5-9
library(terra) #v.1.6-17
library(tidyterra) #v.0.3.2
```

The *HabplanR* package works by saving and loading files from the working directory. It is important to set the working directory, and keep this consistent for the entirety of use of the package and functions.

```
setwd("FILEPATH HERE")
```

Now load in the *HabplanR* package.

```
library(HabplanR)
```

We now need to read in the data. First we will import a csv file which contains information about the forest stands we are interested in. The information we are interested in with this file is the stand acreage, which will link to a specific stand id. We will name this std.info

```
#Load in stand information - needed mostly for acreage
std.info <- read_csv("Stand_info.csv")

#Take a look at the data
head(std.info)
#> # A tibble: 6 x 3
#>   std_id    acres description
#>   <chr>     <dbl> <chr>
#> 1 15588_1   57.2  Planted Longleaf Pine Forest
#> 2 15588_10   9.88 Planted Loblolly Pine Forest
#> 3 15588_100 14.3  Planted Longleaf Pine Forest
#> 4 15588_101 13.0  Natural Loblolly Pine Forest
```

```
#> 5 15588_102 19.0  Upland Hardwood Forest
#> 6 15588_103  1.45 Mesic Hardwood Forest
```

The next data we need are the results from a simulation software which provides outputs based on a particular management regime to a stand, or set of stands. For this vignette, we are using data output from Forest Vegetation Simulator (FVS). We will call this data std.data

```
#Next, load in the data resulting from the Forest Vegetation Simulator (FVS)
std.data <- read_csv("fvs_results.csv")

#Take a look at the data
head(std.data)
#> # A tibble: 6 x 36
#>    ...1 Regime~1 Regim~2 StandID  Year   Age    BA   TPA Pine_~3 Pine_~4 Pine_~5
#>   <dbl> <chr>    <chr>   <chr>   <dbl> <dbl> <dbl> <dbl>   <dbl>   <dbl>   <dbl>
#> 1     0 Batch2   Thin 1~ 15588_1  2017     9  5.53  266.    1.35   0           0
#> 2     1 Batch2   Thin 1~ 15588_1  2020    12  8.82  265.    1.83   0           0
#> 3     2 Batch2   Thin 1~ 15588_1  2023    15  8.17  169.    2.10   0           0
#> 4     3 Batch2   Thin 1~ 15588_1  2026    18 10.8   168.    2.79   0           0
#> 5     4 Batch2   Thin 1~ 15588_1  2029    21 14.0   168.    3.43   0.294       0
#> 6     5 Batch2   Thin 1~ 15588_1  2032    24 17.6   167.    3.42   1.27        0
#> # ... with 25 more variables: Hdwd_Pulp_Tons <dbl>, Hdwd_Saw_Tons <dbl>,
#> #   Harv_P_Pulp_Tons <dbl>, Harv_P_CNS_Tons <dbl>, Harv_P_Saw_Tons <dbl>,
#> #   Harv_H_Pulp_Tons <dbl>, Harv_H_Saw_Tons <dbl>, LL_BA_1to4 <dbl>,
#> #   LL_BA_4to8 <dbl>, LL_BA_8to12 <dbl>, LL_BA_12to16 <dbl>,
#> #   LL_BA_16to20 <dbl>, LL_BA_20andup <dbl>, Pine_BA_1to4 <dbl>,
#> #   Pine_BA_4to8 <dbl>, Pine_BA_8to12 <dbl>, Pine_BA_12to16 <dbl>,
#> #   Pine_BA_16to20 <dbl>, Pine_BA_20andup <dbl>, Hdwd_BA_1to4 <dbl>, ...
```

Function: HSI calculation - *HSIcalc*

We are going to work through three examples of using the Basal Area (BA) to calculate a Habitat Suitability Index (HSI). We have provided a function which will run an HSI formula and combine this with our stand data; however, the formula used to calculate needs to be user defined - based on the species of interest.

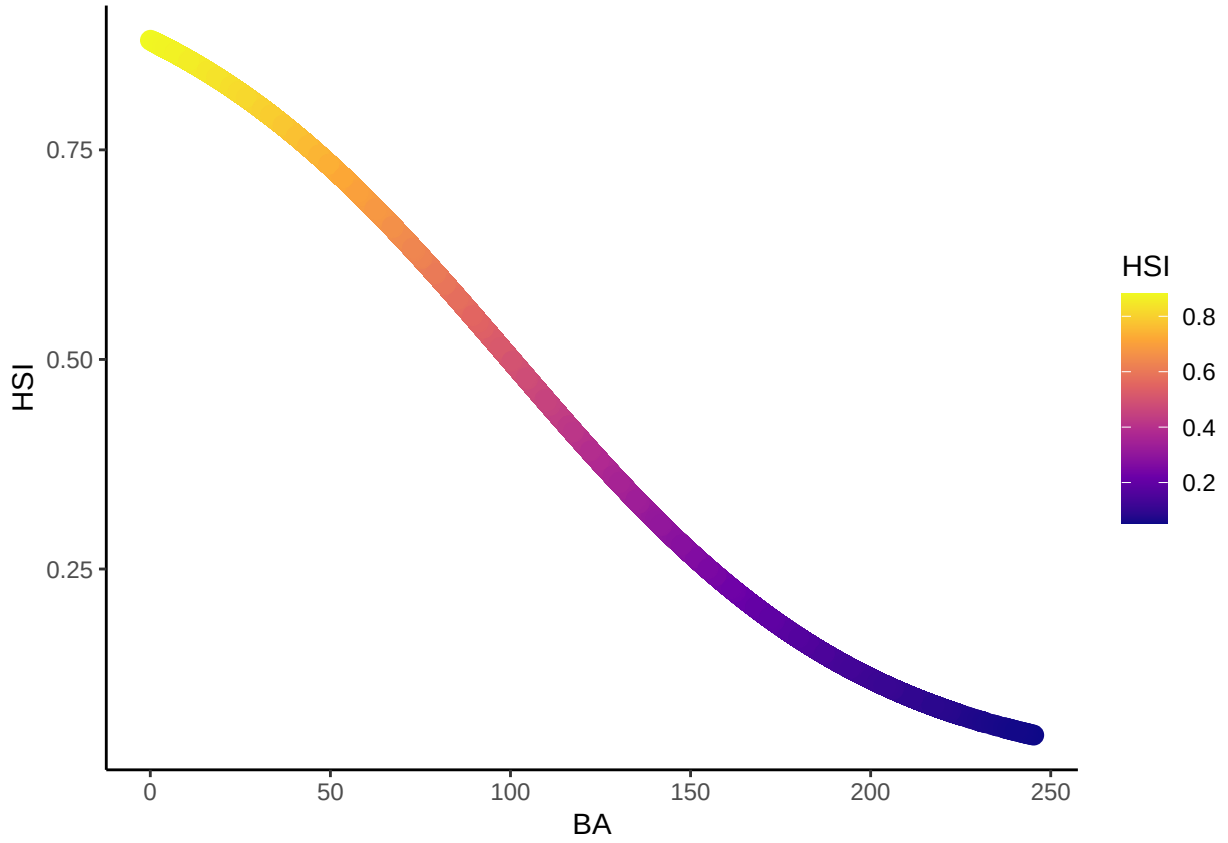
Our first example looks at a negative linear effect of BA on HSI. The equation will need to be provided in form of an R function, as so:

```
#Create function to apply equation for HSI
#Define the slope and y-intercept of the linear equation
hsi.func <- function(x = std.data$BA, m = -0.02, b = 2) {
  return(m * x + b)
  }
```

We can now insert this equation into the *HSIcalc* function to add a column to our stand data for HSI values.

```
#The function will create a new column with the HSI values
new.data <- HSIcalc(std.data, hsi.func)

#Summarise the HSI data
summary(new.data$HSI)
```

```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#> 0.05188 0.40389 0.61333 0.54682 0.73029 0.88072
#Plot data against BA to show relationship
ggplot(data = new.data) +
  geom_point(aes(x = BA, y = HSI, color = HSI),
             size = 3) +
  scale_color_viridis_c(option = "plasma") +
  theme_classic()
```



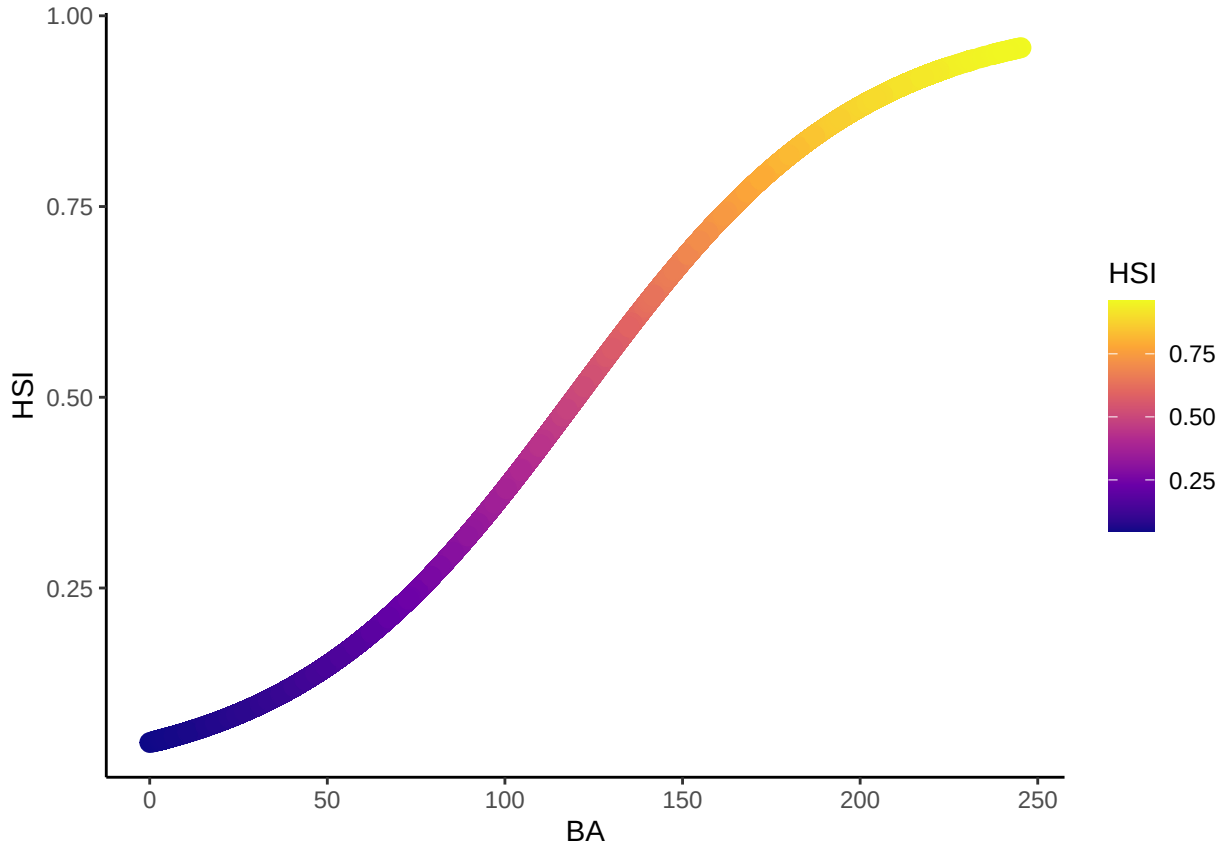
Our second example looks at a positive linear effect of BA on HSI. Again, the equation will need to be provided in form of an R function:

```
#Create function to apply equation for HSI
# Define the slope and y-intercept of the linear equation
hsi.func <- function(x = std.data$BA, m = 0.025, b = -3) {
  return(m * x + b)
  }
```

We can now insert this equation into the *HSIcalc* function to add a column to our stand data for HSI values.

```
#The function will create a new column with the HSI values
new.data <- HSIcalc(std.data, hsi.func)

#Summarise the HSI data
```

```
summary(new.data$HSI)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#> 0.04747 0.14866 0.25413 0.35612 0.49665 0.95819
#Plot data against BA to show relationship
ggplot(data = new.data) +
  geom_point(aes(x = BA, y = HSI, color = HSI),
             size = 3) +
  scale_color_viridis_c(option = "plasma") +
  theme_classic()
```



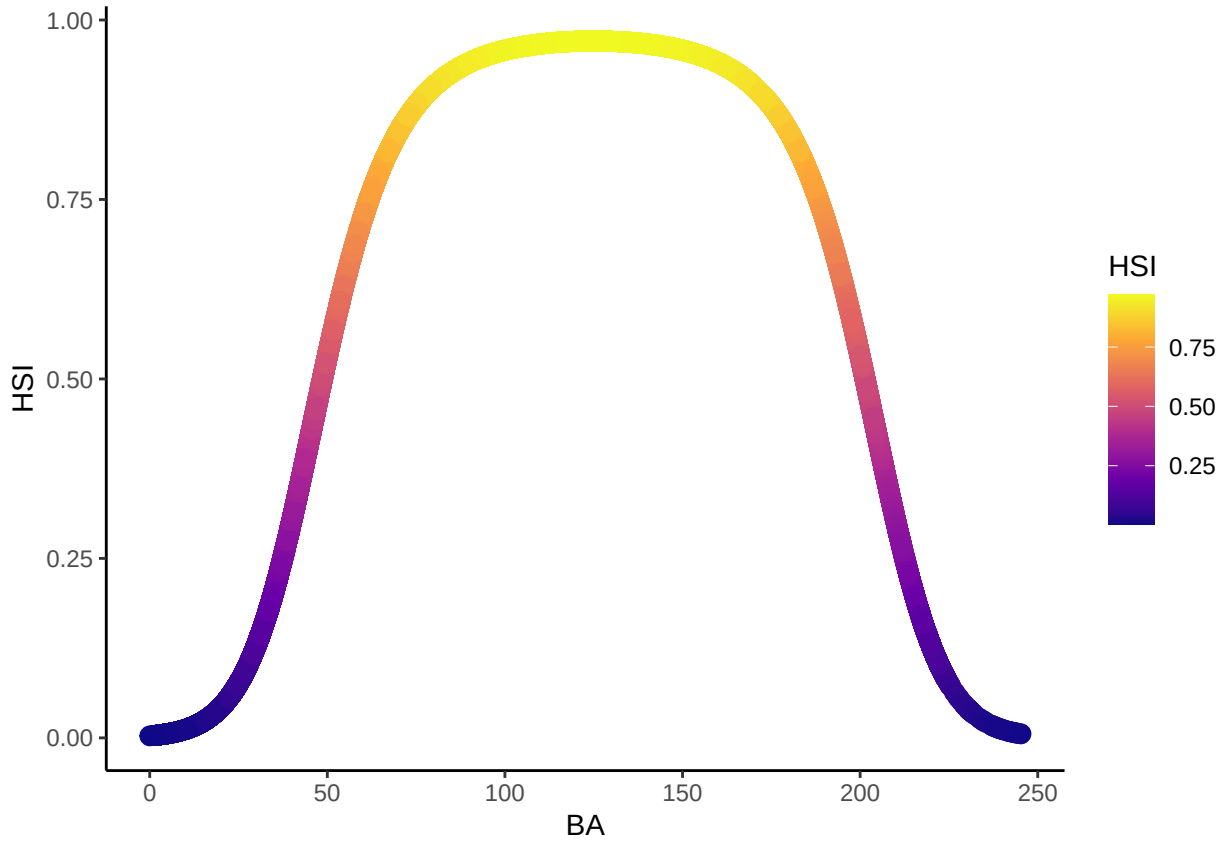
Our third example looks at a quadratic relationship of BA on HSI. Again, the equation will need to be provided in form of an R function:

```
#Create function to apply equation for HSI
#Use a, b, and c to change shape of graph
hsi.func <- function(x = std.data$BA, a = -0.0006, b = 125, c = 3.5) {
  return(a * (x - b)^2 + c)
  }
```

We can now insert this equation into the *HSIcalc* function to add a column to our stand data for HSI values.

```
#The function will create a new column with the HSI values
new.data <- HSIcalc(std.data, hsi.func)
```

```r
#Summarise the HSI data
summary(new.data$HSI)
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> 0.002816 0.393509 0.773965 0.661481 0.937865 0.970688
#Plot data against BA to show relationship
ggplot(data = new.data) +
  geom_point(aes(x = BA, y = HSI, color = HSI),
             size = 3) +
  scale_color_viridis_c(option = "plasma") +
  theme_classic()
```



Function: data conversion - *HabConvert*

We are going to create a flow file (Habplan input file) for the HSI values we have just calculated and inserted into our data frame.

Each line of the flow file has to have the standID, regimeID, years, and flow outputs.

We need to provide the custom function with four arguments: 1. std.data: the flow information in csv format (above) 1. std.info: the information file for each forest stand (above) 1. col: the column of data interested in from std.data (below) 1. nyear: the number of years the simulator is run for (35 in this example)

Look at the column headings to decide which column has flow results.

```
colnames(new.data)
#>  [1] "...1"             "RegimeKey"        "Regime Activity"  "StandID"
#>  [5] "Year"             "Age"              "BA"               "TPA"
#>  [9] "Pine_Pulp_Tons"   "Pine_CNS_Tons"    "Pine_Saw_Tons"    "Hdwd_Pulp_Tons"
#> [13] "Hdwd_Saw_Tons"    "Harv_P_Pulp_Tons" "Harv_P_CNS_Tons"  "Harv_P_Saw_Tons"
#> [17] "Harv_H_Pulp_Tons" "Harv_H_Saw_Tons"  "LL_BA_1to4"       "LL_BA_4to8"
#> [21] "LL_BA_8to12"      "LL_BA_12to16"     "LL_BA_16to20"     "LL_BA_20andup"
#> [25] "Pine_BA_1to4"     "Pine_BA_4to8"     "Pine_BA_8to12"    "Pine_BA_12to16"
#> [29] "Pine_BA_16to20"   "Pine_BA_20andup"  "Hdwd_BA_1to4"     "Hdwd_BA_4to8"
#> [33] "Hdwd_BA_8to12"    "Hdwd_BA_12to16"   "Hdwd_BA_16to20"   "Hdwd_BA_20andup"
#> [37] "HSI"
```

From the column names, we are interested in "HSI", which is column number 37. We can input this into our function. We also need to provide "nyear" which is the number of years (or time periods) that you are interested in (e.g., nyear = 35 could equal 35 years or 35 3-year periods). Lastly, if the column that you are interested in is HSI, then you also need to provide a threshold amount for the habitat to be suitable for your species. The HSI is on a scale between 0-1, so a threshold of 0.5 would mean that anything below 0.5 would be deemed as unsuitable, whereas any values above 0.5 would be suitable and therefore added to the flow file. For our example, we will set the HSI value to 0.7.

```
#Run function
rcw.flow <- habConvert(std.data = new.data, std.info = std.info, col = 37,
                       nyear = 35, HSI = 0.7)
```

Now we will have a flow file of stand acreage for each time period saved in our working directory. If a stand has an HSI over our threshold for that time period, then the flow will show the acreage of the stand. If the HSI was below our threshold, the flow will equal zero.

Function: create project file - *WriteProj*

Another utility of HabplanR is creating project files which can be loaded into habplan to set parameters and other configurations.

The only arguments needed for the actual function to work, are the object names for each flow component which needs to be included.

Since Habplan inputs allow for customization, we provide script which will allow users the same level of input manipulation. Below we give examples on how to set up these arguments, which will be wrapped into our project file using our custom function.

First, setup the main information needed by Habplan prior to flow setup.

```
#Overarching information for habplan run:
#Get the number of polygons for later
npoly <- length(unique(std.data$StandID))
#Get the configuration
configuration <- read.csv("./hbp/configuration.hbp", sep="")
#Remove the unnecessary part of the configuration
config <- gsub('[habplan.config=]', '', configuration$habplan.config.nrow.null)
#We can override config by using the following:
config <- "4,0,0,0,0,0,0,0,0,0,0,0"
```

```
#wd is the working directory where habplan and data are stored
wd <- setwd("~/Habplan/Habplan3/Habplan3/")
#iter is the number of iterations needed for the habplan run
iter <- "10000"
```

Next, we will input the specific information needed for each flow component. This is where specific objectives, weights, and targets are assigned to produce the most suitable outcome based on the forest stands and management aims.

For this vignette, we will work with two flow files, our HSI flow which we have just calculated and created above, and also the harvested pine pulpwood. We want to achieve the highest acreage of HSI, while also being able to harvest as much pine pulpwood as possible.

To maintain a relatively simple workflow for our vignette, we will input all paramters but focus on three main objects which we will edit to achieve our schedule outcomes. The edited objects will be thlo (Lower Threshold), thhi (Upper Threshold), and the model (The modeled targets throughout our study period). The numbers assigned to thlo and thhi determine how much upper and lower flexibility there is in our targets. For example, setting thlo and thhi to 50 and 150 respectively means that our target can not deviate any lower than 50 units, and no higher than 150. If we're wanting to achieve results that maximise output/yield then setting a large thhi may help (a realistic threshold that is).

Below we are setting our model for both flows as 1000 in the first year, 2500 in year 20, and 5000 in year 30 (assigned as 1,1000;20,2500;30,5000;). Our thlo and thhi are set to 1000 and 5000 respectively, allowing broad fluctations from our targets.

```
#Info for f1 component ----
#f1.file is the flow file for the first component
f1.file <- "RCW/RCW/Flows/HSI.dat"
#f1.bygone
f1.bygone <- ""
#f1.time0
f1.time0 <- "10000"
#f1.goal0
f1.goal0 <- "0.1"
#f1.thlo
f1.thlo <- "1000"
#f1.thhi
f1.thhi <- "5000"
#f1.goalplus
f1.goalplus <- ".05"
#f1.goalf
f1.goalf <- "0.5"
#f1.slope
f1.slope <- "0.0"
#f1.weightf
f1.weightf <- "1.0"
#f1.weight0
f1.weight0 <- "1.0"
#f1.model
f1.model <- "1,1000;20,2500;30,5000;"
#f1.title
f1.title <- "Breed.dat"
#Combine f1 components for writing
f1.comp <- c('<flow title="F1 Component">',
```

```
            paste0('<file value="', f1.file, '" />'),
            paste0('<bygone value="', f1.bygone, '" />'),
            paste0('<time0 value="', f1.time0, '" />'),
            paste0('<goal0 value="', f1.goal0, '" />'),
            paste0('<threshLo value="', f1.thlo, '" />'),
            paste0('<threshHi value="', f1.thhi, '" />'),
            paste0('<goalPlus value="', f1.goalplus, '" />'),
            paste0('<goalF value="', f1.goalf, '" />'),
            paste0('<slope value="', f1.slope, '" />'),
            paste0('<weightF value="', f1.weightf, '" />'),
            paste0('<weight0 value="', f1.weight0, '" />'),
            paste0('<model value="', f1.model, '" />'),
            paste0('<title value="', f1.title, '" />'),
            '<bounds height="330" width="366" x="553" y="443" />',
            "</flow>")


#Info for f2 component ----
#f2.file is the flow file for the first component
f2.file <- "RCW/RCW/Flows/Harv_P_Pulp_Tons.dat"
#f2.bygone
f2.bygone <- ""
#f2.time0
f2.time0 <- "1000"
#f2.goal0
f2.goal0 <- "0.1"
#f2.target
#f2.target <- "20000"
#f2.thlo
f2.thlo <- "1000"
#f2.thhi
f2.thhi <- "5000"
#f2.goalplus
f2.goalplus <- ".05"
#f2.goalf
f2.goalf <- "0.5"
#f2.slope
f2.slope <- "0.0"
#f2.weightf
f2.weightf <- "1.0"
#f2.weight0
f2.weight0 <- "1.0"
#f2.model
f2.model.1 <- "1,1000;20,2500;30,5000;"
#f2.next.year - sets up the year where the next target is set
#f2.next.year <- "20"
#f2.next.target - determines what the next target is after the first year
#f2.next.target <- 100000
#f2.title
f2.title <- "Harv_P_Pulp_Tons.dat"

f2.comp <- c('<flow title="F2 Component">',
            paste0('<file value="', f2.file, '" />'),
```

```
            paste0('<bygone value="', f2.bygone, '" />'),
            paste0('<time0 value="', f2.time0, '" />'),
            paste0('<goal0 value="', f2.goal0, '" />'),
            paste0('<threshLo value="', f2.thlo, '" />'),
            paste0('<threshHi value="', f2.thhi, '" />'),
            paste0('<goalPlus value="', f2.goalplus, '" />'),
            paste0('<goalF value="', f2.goalf, '" />'),
            paste0('<slope value="', f2.slope, '" />'),
            paste0('<weightF value="', f2.weightf, '" />'),
            paste0('<weight0 value="', f2.weight0, '" />'),
            #paste0('<model value="1,', f2.target, ';', f2.next.year, ',',
            #      f2.next.target, '" />'),
            paste0('<title value="', f2.title, '" />'),
            '<bounds height="331" width="368" x="1260" y="440" />',
            "</flow>")
```

Before running the function, set a new working directory so that the project file is saved into the correct folder

```
setwd("FILEPATH HERE/project/")
```

```
#Provide each of these flow component objects to the function and run
writeProj(f1.comp = f1.comp, f2.comp= f2.comp)

#Reset working directory
setwd("FILEPATH HERE")
```

Function: create block data - *HabBlock*

We will not be including the block data for the purpose of this vignette due to processing time. However, below we provide instructions for including this component to the Habplan run. Block size constraints are a sub-component of flow components. For a comprehensive guide to the purpose of block size constraints please see the Habplan user manual.

We can now run the function HabBlock from the HabplanR package. This takes the SpatVector shapefile, and stand data/info from above and creates block data (which polygons are adjacent to each other). The block data will save directly to the working directory.

```
#HabBlock will create a block data file in the working directory
HabBlock(std.data = std.data, std.info = std.info, site.shp = site.shp,
         block.title = "block1")

#Since this block size component file is within the working directory already,
#we just need to specify the file name below (b1.file), and input the
#specific parameters for the component.

#Info for f1 block size component (green up) ----
#b1.file is the block data for the first component
b1.file <- "RCW/RCW/Flows/block1.txt"
#b1.notBlock value
b1.notBlock <- "1-4" #which regimes do NOT contribute to block sizes
```

```
#b1.greenUp
b1.greenUp <- "3" #3 year green up period
#b1.min
b1.min <- "1" #minimum allowable block size
#b1.max
b1.max <- "1000" #maximum allowable block size
#b1.goal
b1.goal <- "1" #a value of 1.0 means all blocks must comply
#b1.weight
b1.weight <- "1.0" #good practice to start this at 1
#b1.title
b1.title <- "block1.txt"
#Combine f1 components for writing
block1 <- c('<block title="BK1(1) Component">',
            paste0('<file value="', b1.file, '" />'),
            paste0('<notBlock value="', b1.notBlock, '" />'),
            paste0('<greenUp value="', b1.greenUp, '" />'),
            paste0('<min value="', b1.min, '" />'),
            paste0('<max value="', b1.max, '" />'),
            paste0('<goal value="', b1.goal, '" />'),
            paste0('<weight value="', b1.weight, '" />'),
            paste0('<title value="', b1.title, '" />'),
            '<bounds height="330" width="366" x="553" y="443" />',
            "</block>")
```

You can include as many block size components as there are flow components, which can be included as above but changing the block title and file to match the number of the flow component.

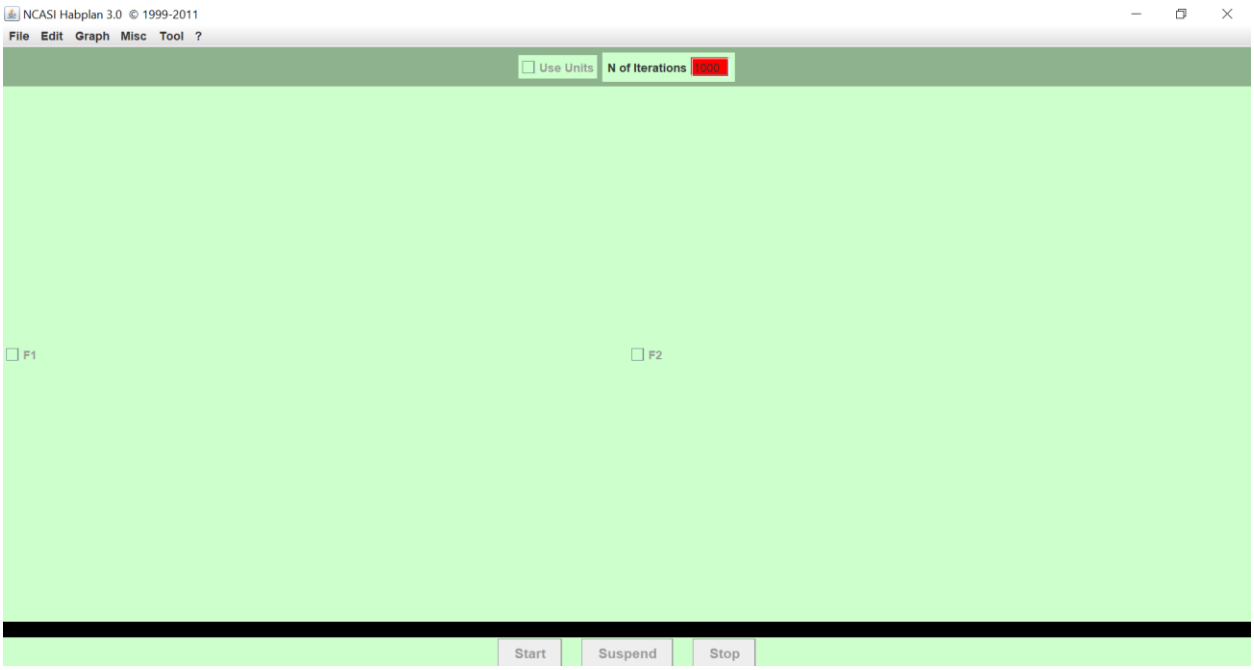
We can then include block size constraints to the project file as so (for demonstrative purposes only).

```
#Provide each of these flow component objects to the function and run
writeProj(f1.comp = f1.comp, block1 = block1, f2.comp= f2.comp)
```

Now we have a project file saved to our working directory. Let's open habplan and see if this has worked.

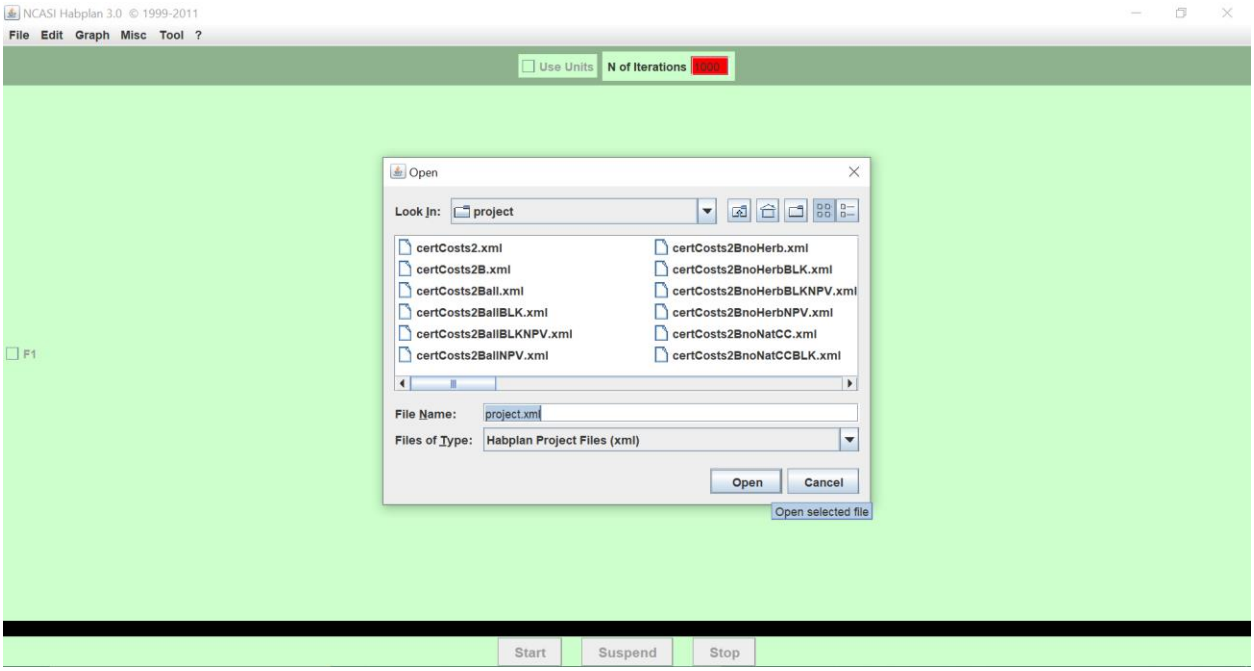
```
#Open Habplan (if run from here, R functionality will cease until Habplan is closed)
shell("h", wait=TRUE)
```

When you first open Habplan from the R script, the following window will open. Note that you will be unable to run anything in your R session when Habplan has been opened via R.

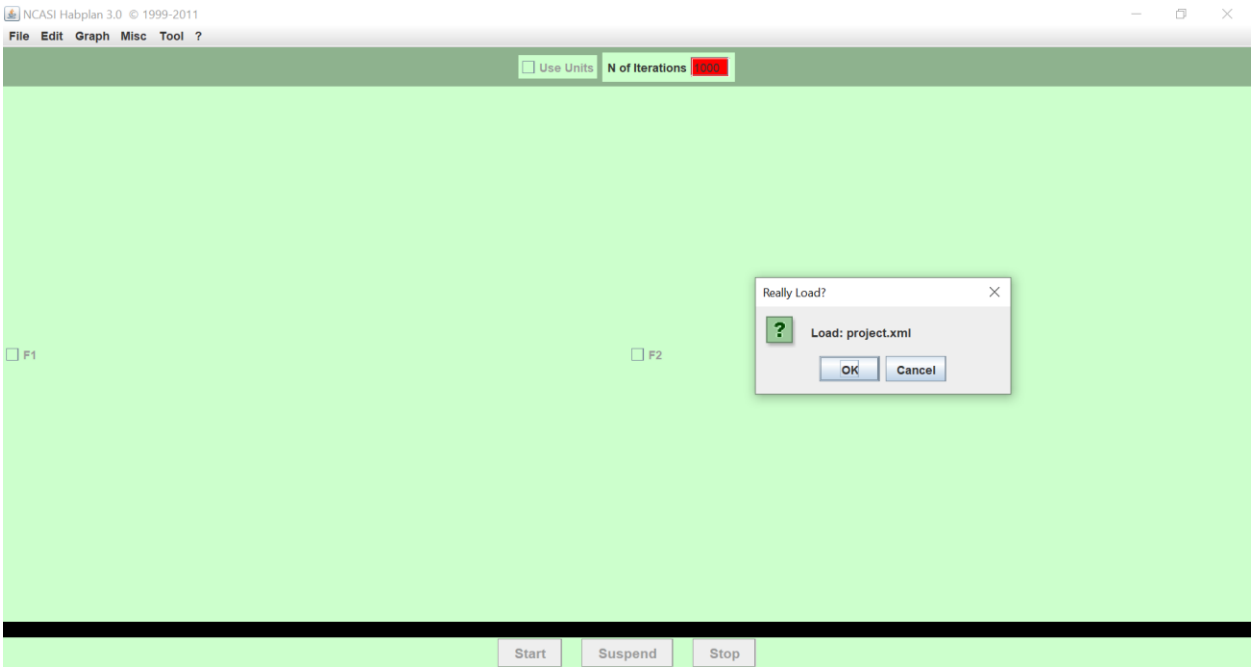


To open the project file we have just created, we can navigate to File > Open…

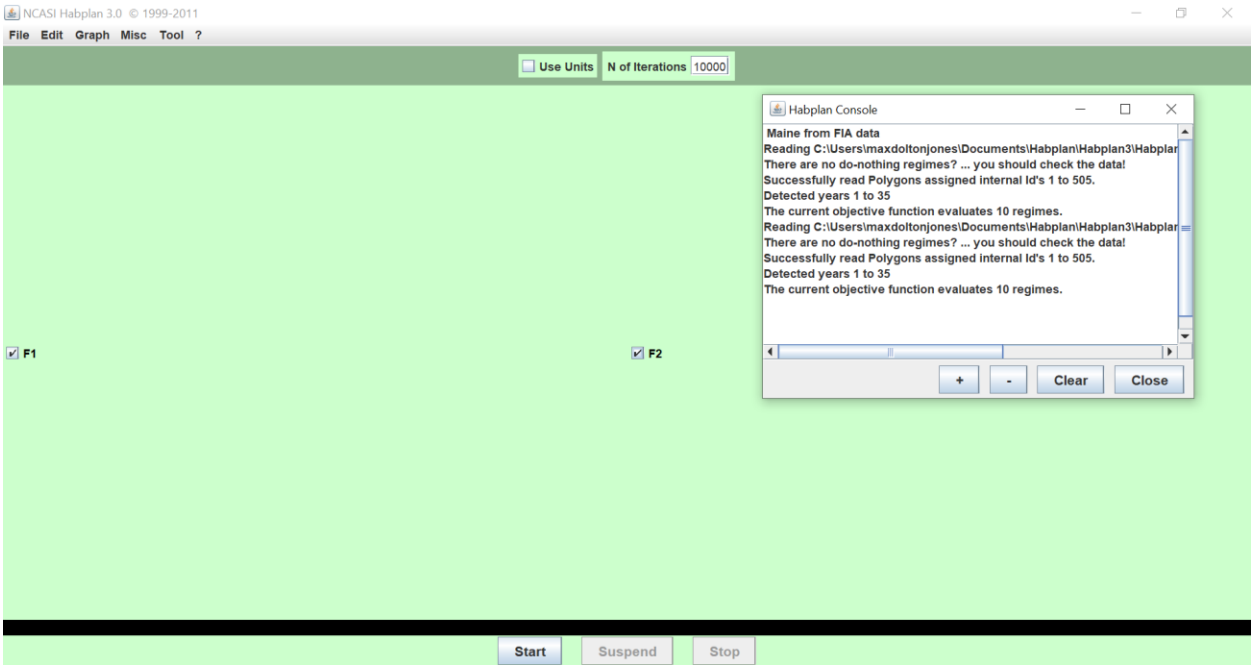
A new window will open to select the appropriate file, which will have saved in the set working directory as project.xml. Habplan will automatically select this file when the window opens, so we can go ahead and click Open.



A new window will then open. Click OK to load the project file.

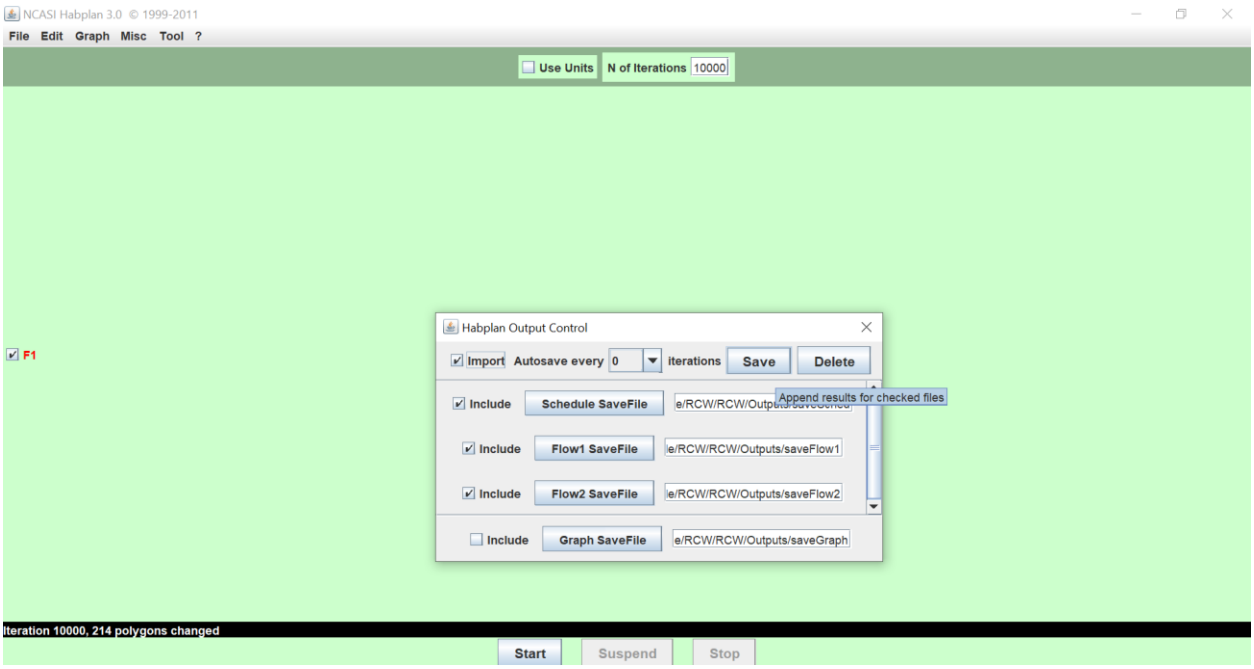


Another new window will now be available titled Habplan Console, where the program will provide updates and any errors discovered. We will also have the opportunity to select the flow components on the main window, named F1 and F2 in our example. Including other flows in the project file will add new component check boxes. Go ahead and select the two check boxes on the main window so that they appear as below. Once they have been selected, click Start to begin Habplan.

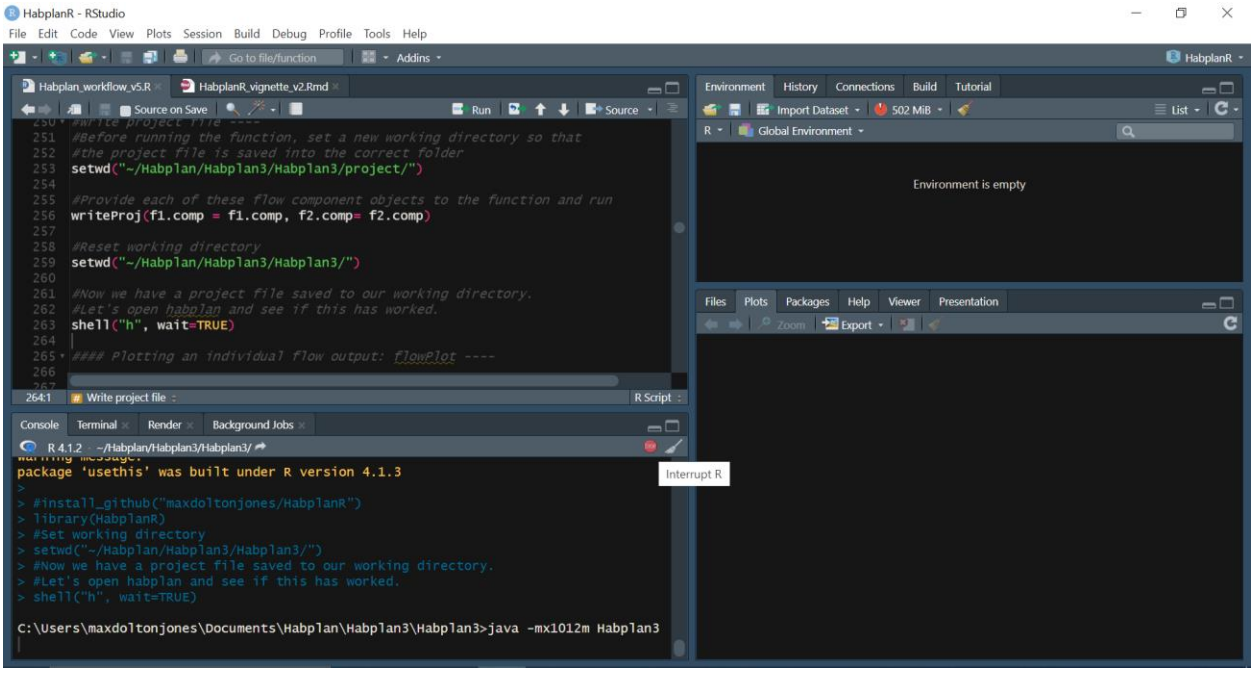


The progress of the Habplan run can be seen in the bottom left corner of the main Habplan window. Once the allocated number of iterations has been completed (which we have set to 10,000 in our script), we can save the output. To do this navigate to File > Output…

A new window will open titled Habplan Output Control. All of the check boxes will be pre-selected, so we can simply click Save to save the output flow files and recommended schedule.



We are now finished with Habplan for the time being. We can either close all Habplan windows, or just go back into RStudio and select the red STOP button on the top right of the console to close Habplan.

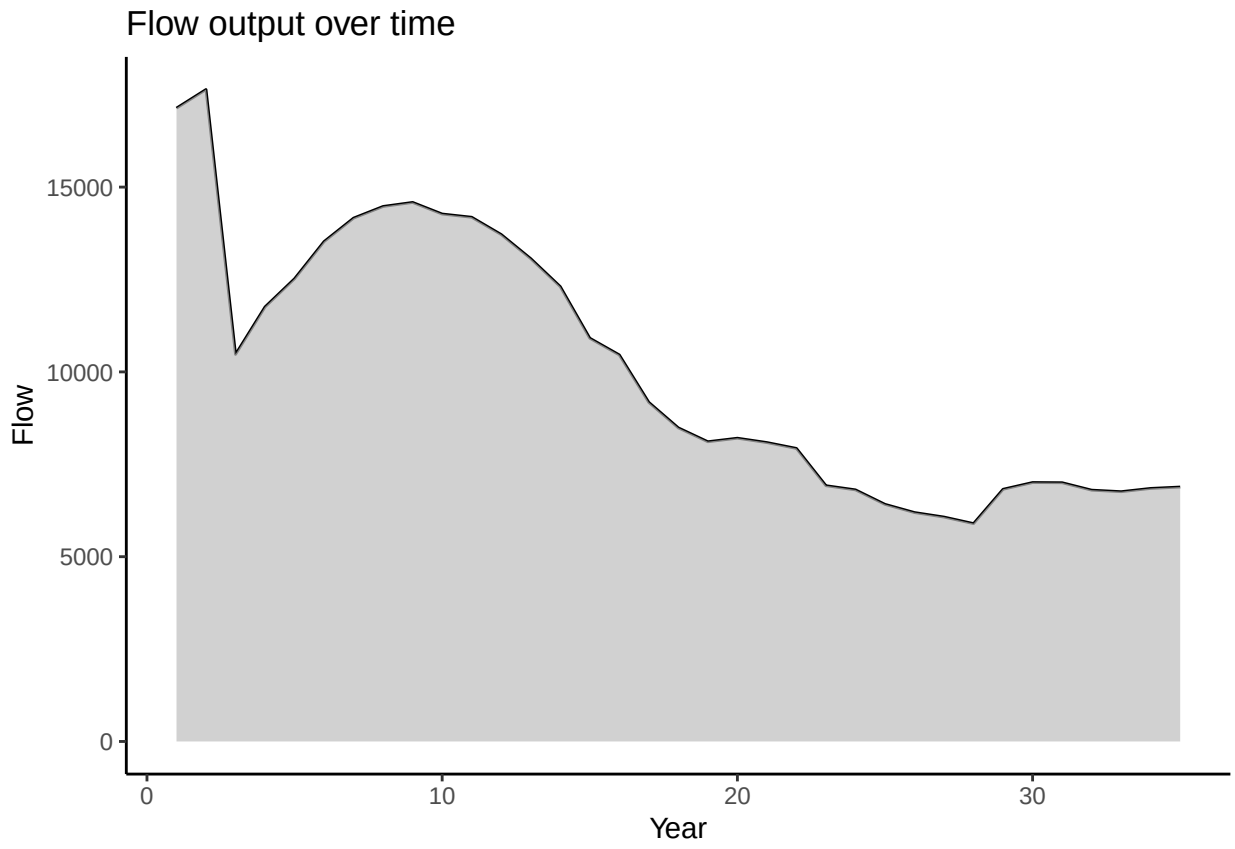


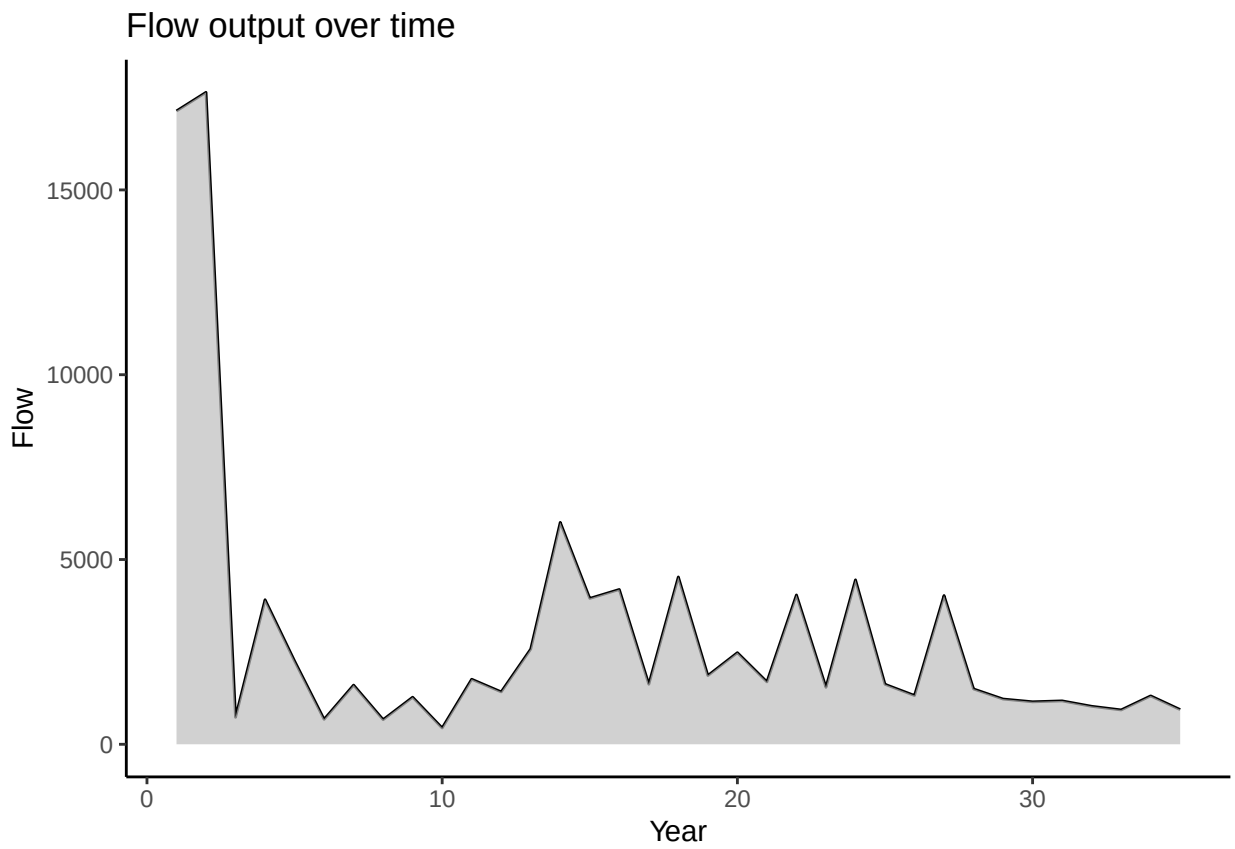
Function: plotting an individual flow output - *FlowPlot*

After running habplan, we will have several flows saved to our working directory. The option still exists to interactively watch the charts in a habplan window. However, we provide a function to visualize each flow individually.

First read in the flow output file, and then feed this file into the function.

```
#Read in one of the flow files
flow1 <- read.csv("./saveFlow1", sep="")
flow2 <- read.csv("./saveFlow2", sep="")

#Input the flow file into the function, and number of years
flowPlot(flow.data = flow1, nyear = 35)
flowPlot(flow.data = flow2, nyear = 35)
```





The first figure above shows the output flow from our HSI component throughout our study period, and the second shows the hard pulpwood yield.

Since we now have a possible solution based on the above output flows, to help make our decision on whether or not this is a viable solution we will run an addition function *HabSpace*. This function takes the output flow information from Habplan and visualizes the flow for each year of interest.

Function: spatial contiguity assessment - *HabSpace*

```
#Read in stand shapefiles
site.shp <- readOGR(dsn = "./shapefiles/Stands2_Shapefile/Stands_final.shp")

#Read in flow file from Habplan run
flow <- read.csv("./example/RCW/RCW/Flows/Breed_final.dat")
```

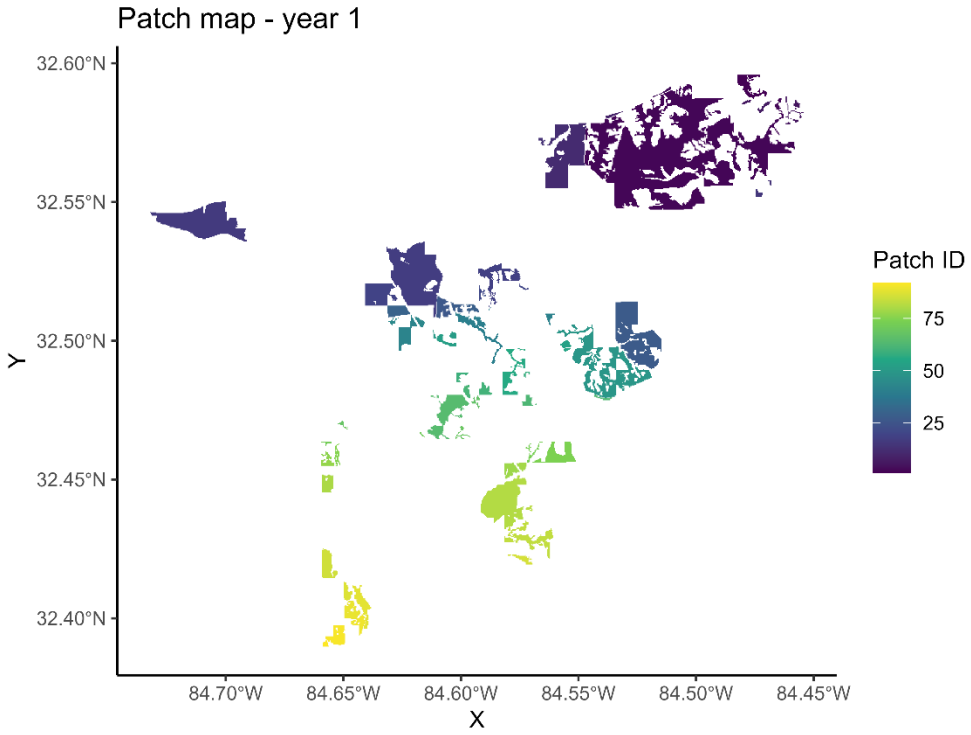
IMPORTANT The shapefile needs a data column called "StdID"

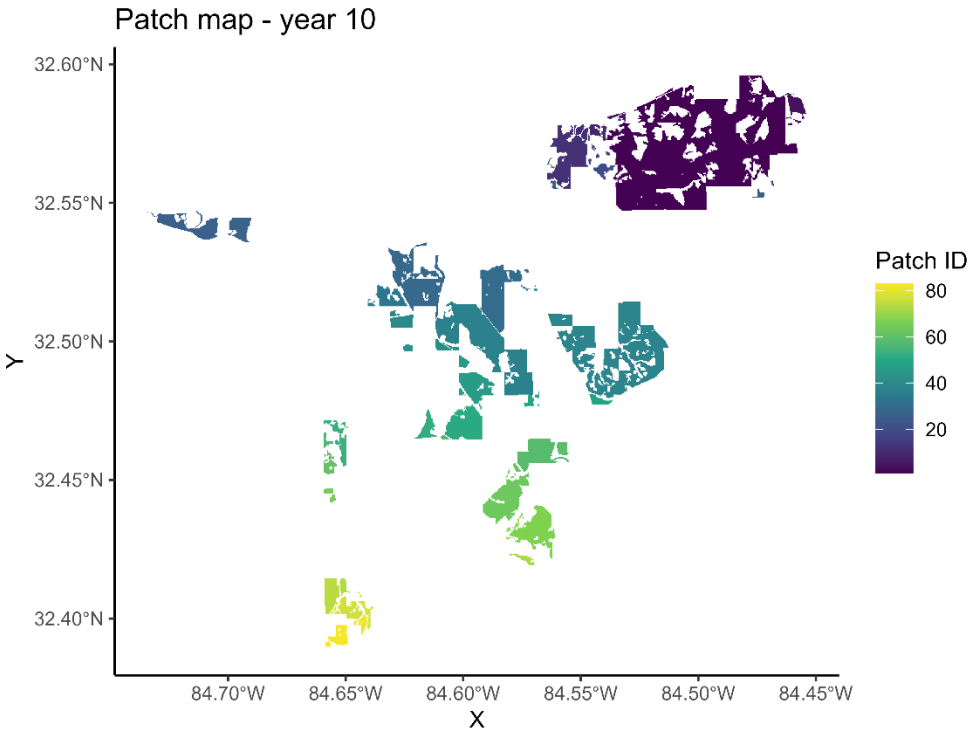
We will add the shapefile of stands, and the flow of choice to the function. As before, we will set the nyear. We now have two more options with this function, mode and dist.

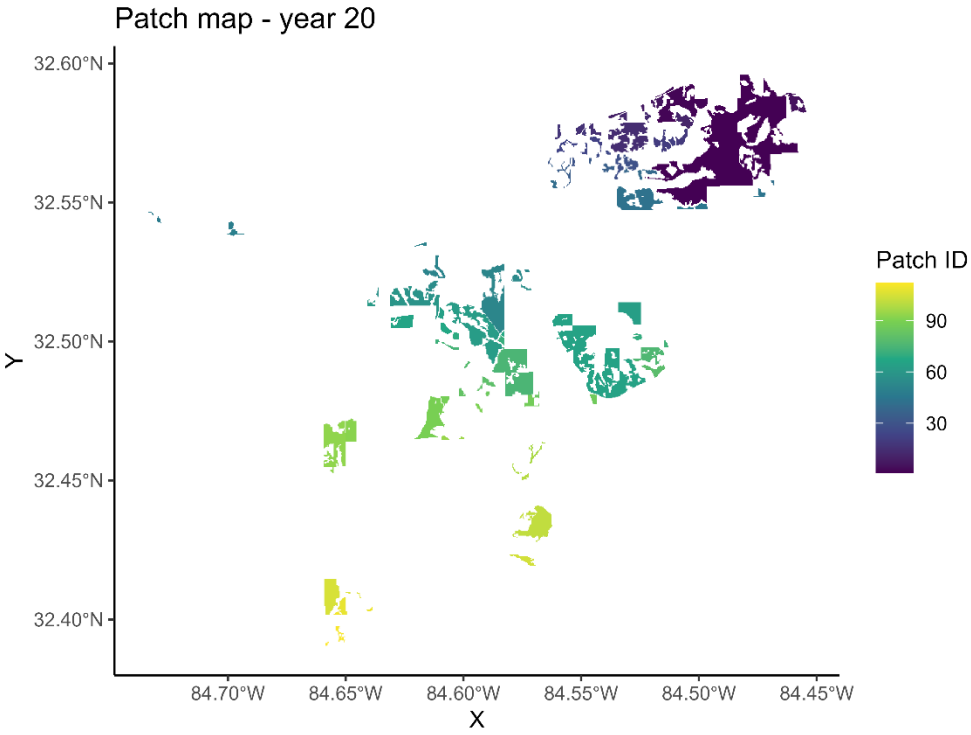
There are two modes to choose from, terrestrial or avian - depending on the species the HSI has been built around. If avian is chosen, the dist setting sets the distance in meters (or the unit of the shapefile) that the species of interest can comfortably travel to reach neighboring habitat patches. If the user has a terrestrial species which may still bridge gaps if the distance is short enough, e.g. crossing a road, then the avian option should be selected, and a smaller dist assigned.

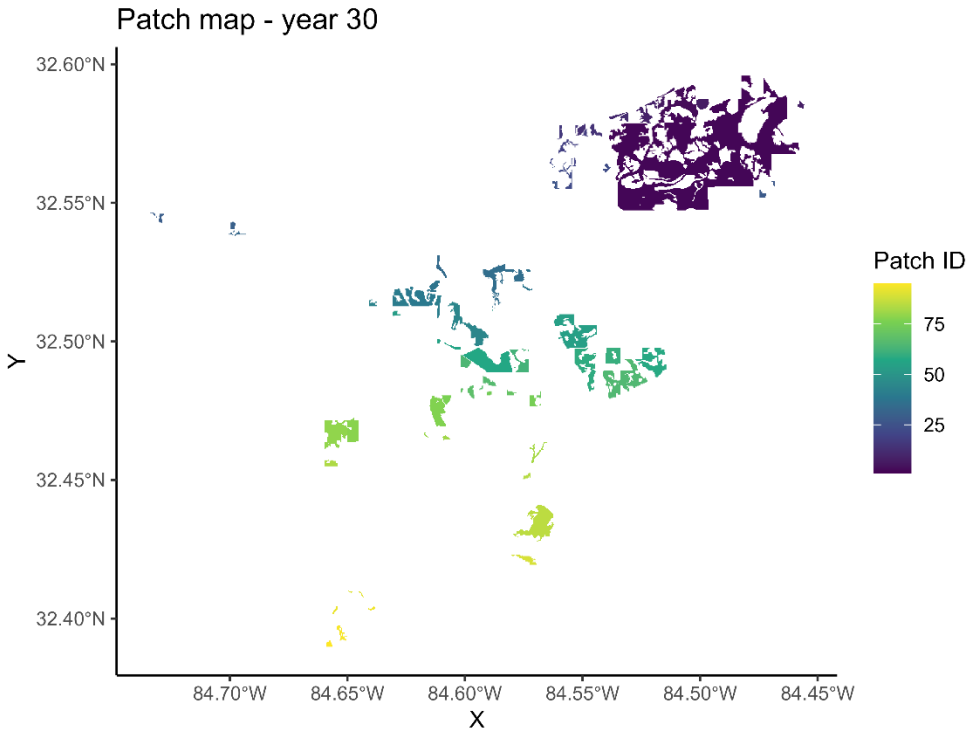
```
#Want to create a shapefile output for each year, to look at change
#of regime for each stand over time.

space.test <- HabSpace(site.shp = site.shp, flow = flow, nyear = 35,
                  mode = "terrestrial", dist = 500, level = "landscape")
```

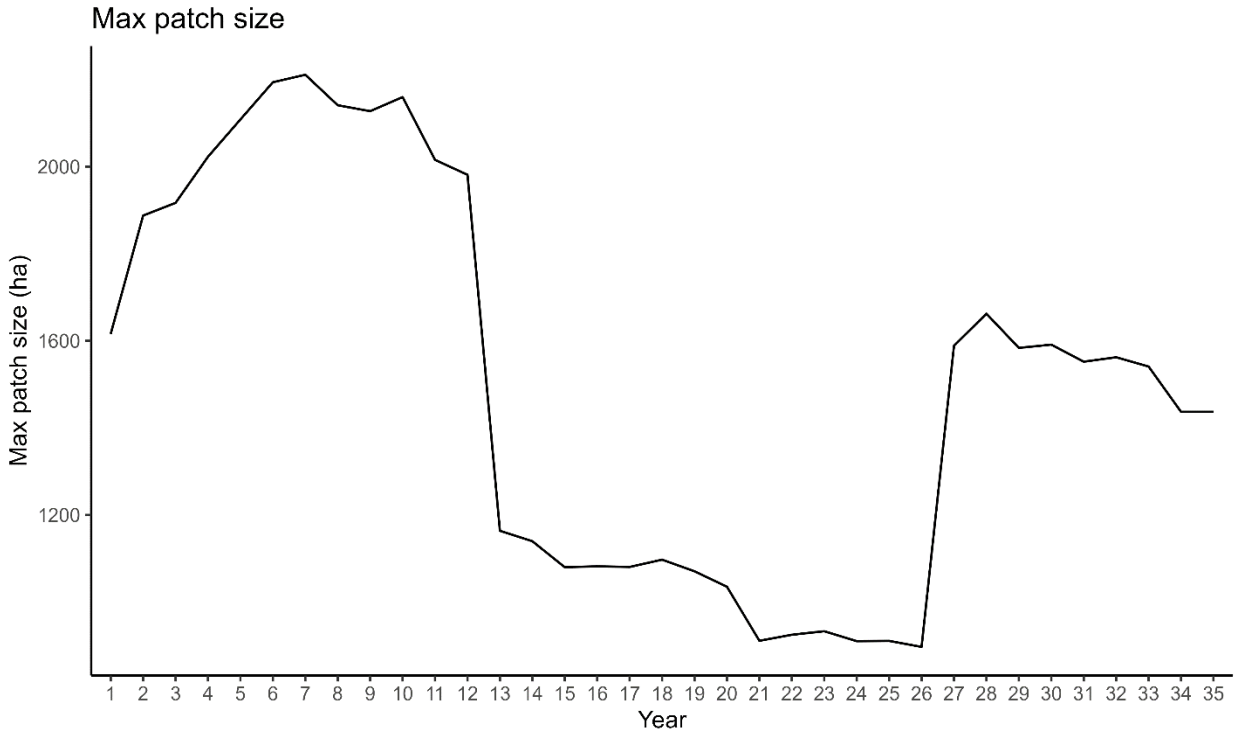
The above function will create maps for each year and save these in the working directory. We have also saved summary graphs, raw data, and summary data for each year.

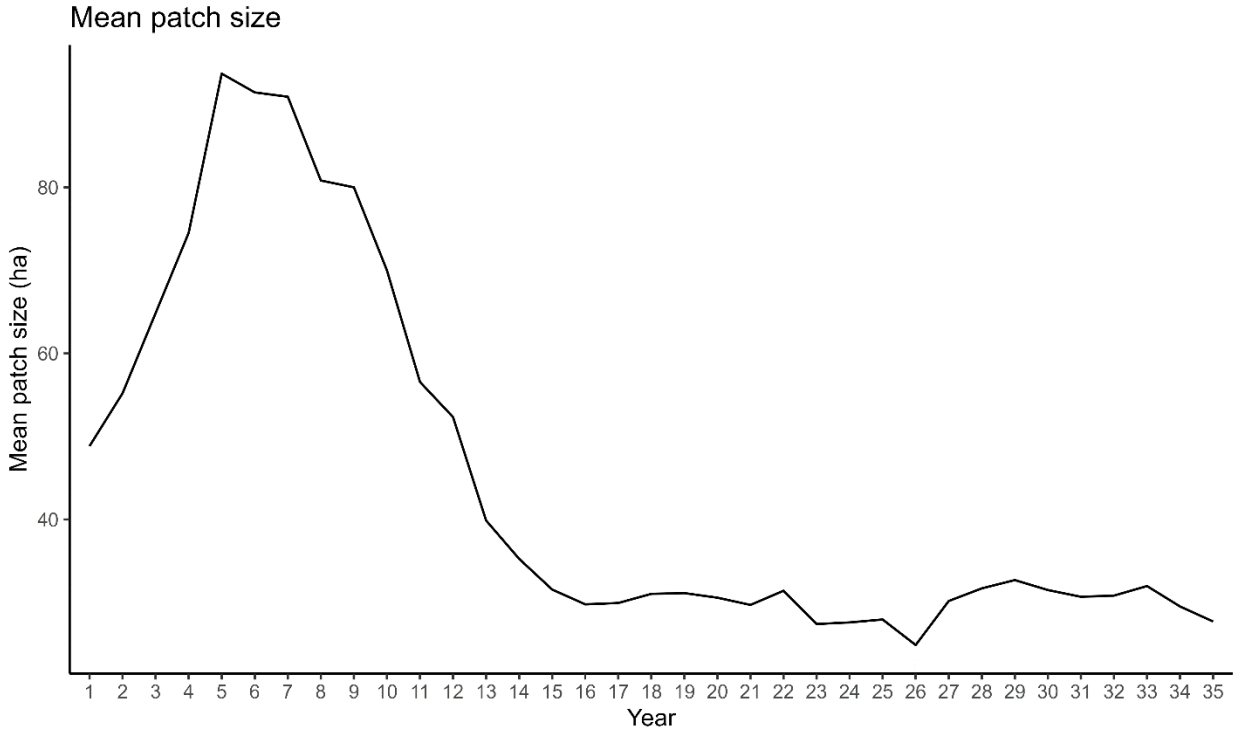

Patch map - year 1
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
75
50
25


Patch map - year 10
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
80
60
40
20


Patch map - year 20
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
90
60
30


Patch map - year 30
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
75
50
25


Max patch size
Max patch size (ha)
2000
1600
1200
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Year


Mean patch size
Mean patch size (ha)
80
60
40
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Year

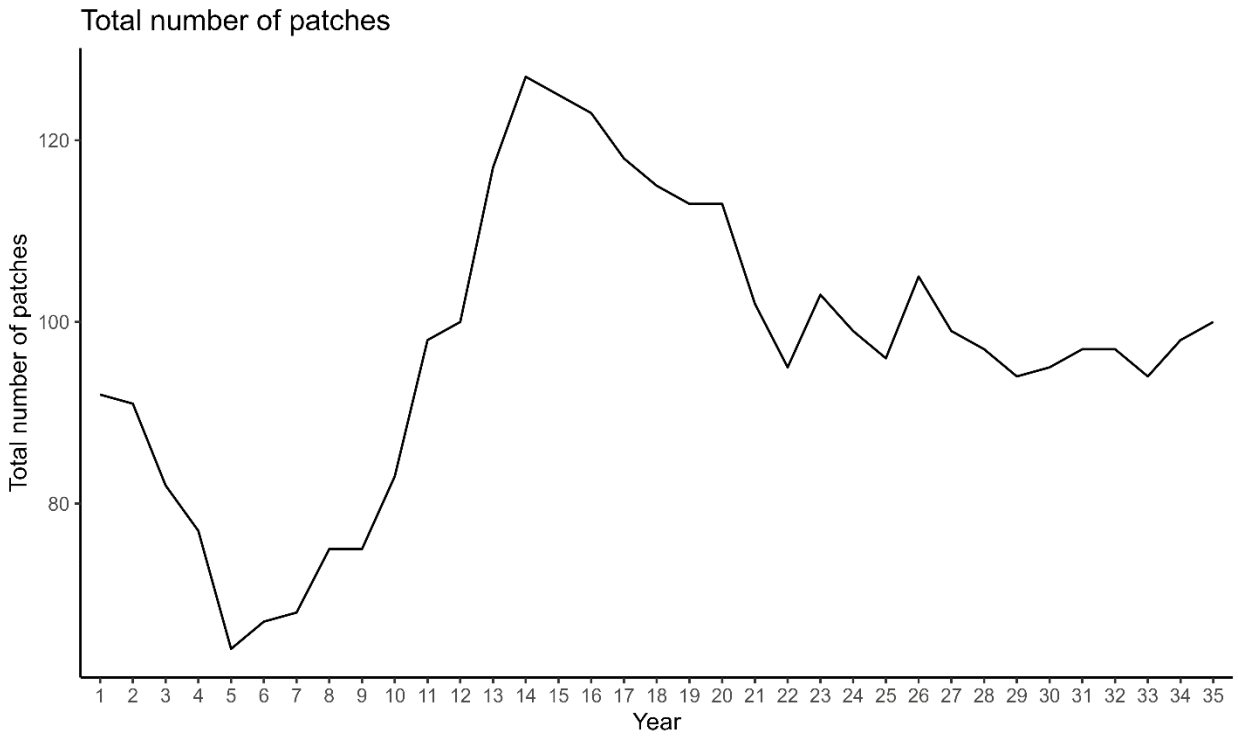


Table 1. Example of landscape-level metrics output from the habSpace function (using the *landscapemetrics* package.

Level	Metric	Value
landscape	ai	97.31649
landscape	area_cv	327.359
landscape	area_mn	60.73622
landscape	area_sd	198.8254
landscape	cai_cv	32.57878
landscape	cai_mn	72.85772
landscape	cai_sd	23.73615
landscape	circle_cv	22.33525
landscape	circle_mn	0.685704
landscape	circle_sd	0.153154

Since we have everything to assess the Habplan run, we are now going to adjust the thlo, thhi, and model objects. For our lower threshold, we are going to decrease this to 500 to prevent too much negative deviation from our target, but instead have a much higher thhi to allow more yield from the target if Habplan can find a schedule which permits that. Additionally, we will change the model targets to 500 in year 1, 2500 in year 20, and 7500 in year 30 (1,500;20,2500;30,7500). We will apply the same changes to both flow components.

```
#Info for f1 component ----
#f1.file is the flow file for the first component
f1.file <- "RCW/RCW/Flows/HSI.dat"
#f1.bygone
f1.bygone <- ""
#f1.time0
f1.time0 <- "10000"
#f1.goal0
f1.goal0 <- "0.1"
#f1.thlo
f1.thlo <- "500"
#f1.thhi
f1.thhi <- "10000"
#f1.goalplus
f1.goalplus <- ".05"
#f1.goalf
f1.goalf <- "0.5"
#f1.slope
f1.slope <- "0.0"
#f1.weightf
f1.weightf <- "1.0"
#f1.weight0
f1.weight0 <- "1.0"
#f1.model
f1.model <- "1,500;20,2500;30,7500;"
#f1.title
f1.title <- "Breed.dat"
#Combine f1 components for writing
f1.comp <- c('<flow title="F1 Component">',
             paste0('<file value="', f1.file, '" />'),
             paste0('<bygone value="', f1.bygone, '" />'),
             paste0('<time0 value="', f1.time0, '" />'),
             paste0('<goal0 value="', f1.goal0, '" />'),
             paste0('<threshLo value="', f1.thlo, '" />'),
             paste0('<threshHi value="', f1.thhi, '" />'),
             paste0('<goalPlus value="', f1.goalplus, '" />'),
             paste0('<goalF value="', f1.goalf, '" />'),
             paste0('<slope value="', f1.slope, '" />'),
             paste0('<weightF value="', f1.weightf, '" />'),
             paste0('<weight0 value="', f1.weight0, '" />'),
             paste0('<model value="', f1.model, '" />'),
             paste0('<title value="', f1.title, '" />'),
             '<bounds height="330" width="366" x="553" y="443" />',
             "</flow>")


#Info for f2 component ----
```

```
#f2.file is the flow file for the first component
f2.file <- "RCW/RCW/Flows/Harv_P_Pulp_Tons.dat"
#f2.bygone
f2.bygone <- ""
#f2.time0
f2.time0 <- "1000"
#f2.goal0
f2.goal0 <- "0.1"
#f2.target
#f2.target <- "20000"
#f2.thlo
f2.thlo <- "500"
#f2.thhi
f2.thhi <- "10000"
#f2.goalplus
f2.goalplus <- ".05"
#f2.goalf
f2.goalf <- "0.5"
#f2.slope
f2.slope <- "0.0"
#f2.weightf
f2.weightf <- "1.0"
#f2.weight0
f2.weight0 <- "1.0"
#f2.model
f2.model.1 <- "1,500;20,2500;30,7500;"
#f2.next.year - sets up the year where the next target is set
#f2.next.year <- "20"
#f2.next.target - determines what the next target is after the first year
#f2.next.target <- 100000
#f2.title
f2.title <- "Harv_P_Pulp_Tons.dat"

f2.comp <- c('<flow title="F2 Component">',
             paste0('<file value="', f2.file, '" />'),
             paste0('<bygone value="', f2.bygone, '" />'),
             paste0('<time0 value="', f2.time0, '" />'),
             paste0('<goal0 value="', f2.goal0, '" />'),
             paste0('<threshLo value="', f2.thlo, '" />'),
             paste0('<threshHi value="', f2.thhi, '" />'),
             paste0('<goalPlus value="', f2.goalplus, '" />'),
             paste0('<goalF value="', f2.goalf, '" />'),
             paste0('<slope value="', f2.slope, '" />'),
             paste0('<weightF value="', f2.weightf, '" />'),
             paste0('<weight0 value="', f2.weight0, '" />'),
             #paste0('<model value="1,', f2.target, ';', f2.next.year, ',',
             #      f2.next.target, '" />'),
             paste0('<title value="', f2.title, '" />'),
             '<bounds height="331" width="368" x="1260" y="440" />',
             "</flow>")
```

Before running the function, set a new working directory so that the project file is saved into the correct folder

```
setwd("FILEPATH HERE/project/")
```

```
#Provide each of these flow component objects to the function and run
writeProj(f1.comp = f1.comp, f2.comp= f2.comp)

#Reset working directory
setwd("FILEPATH HERE")
```

Now we have a project file saved to our working directory. Let's open habplan and see if this has worked. Follow the same instructions above for running Habplan once it has opened.

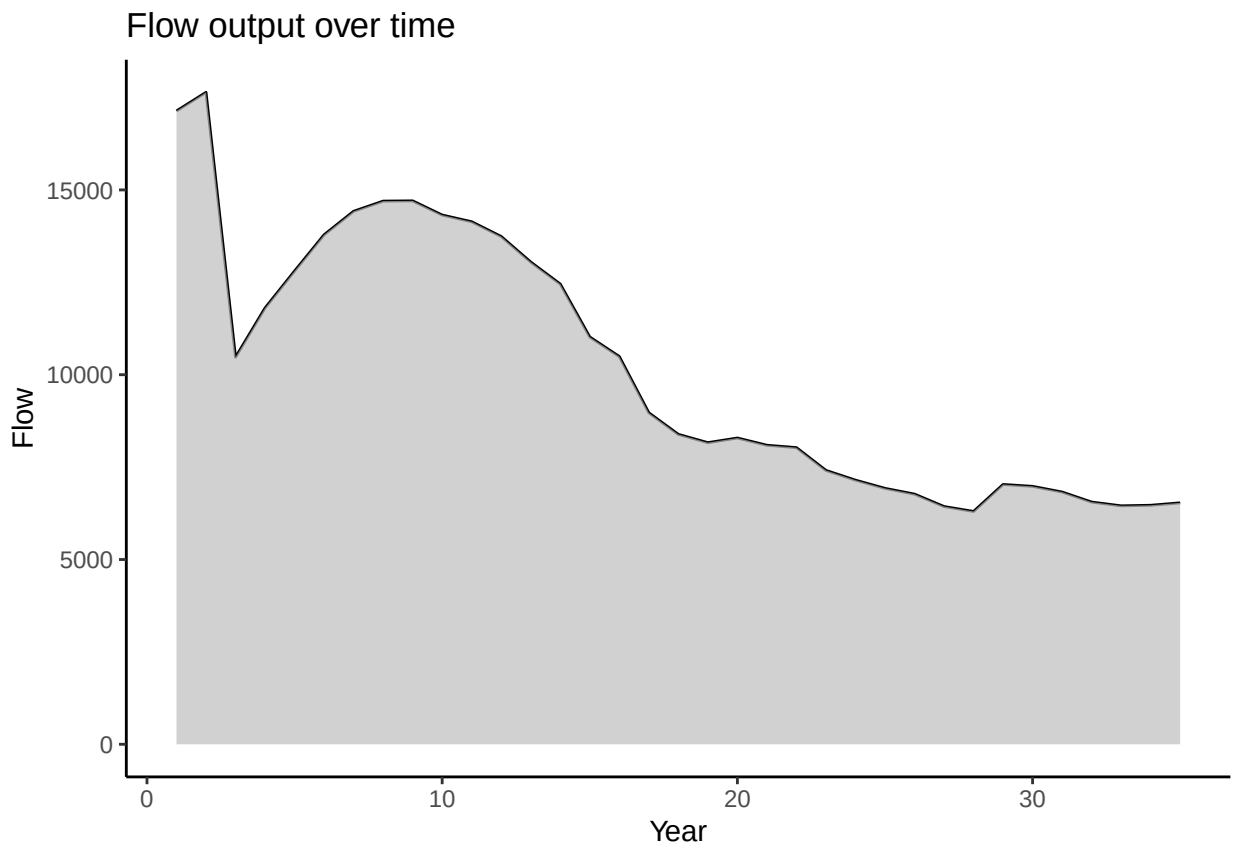
```
#Open Habplan (if run from here, R functionality will cease until Habplan is closed)
shell("h", wait=TRUE)
```

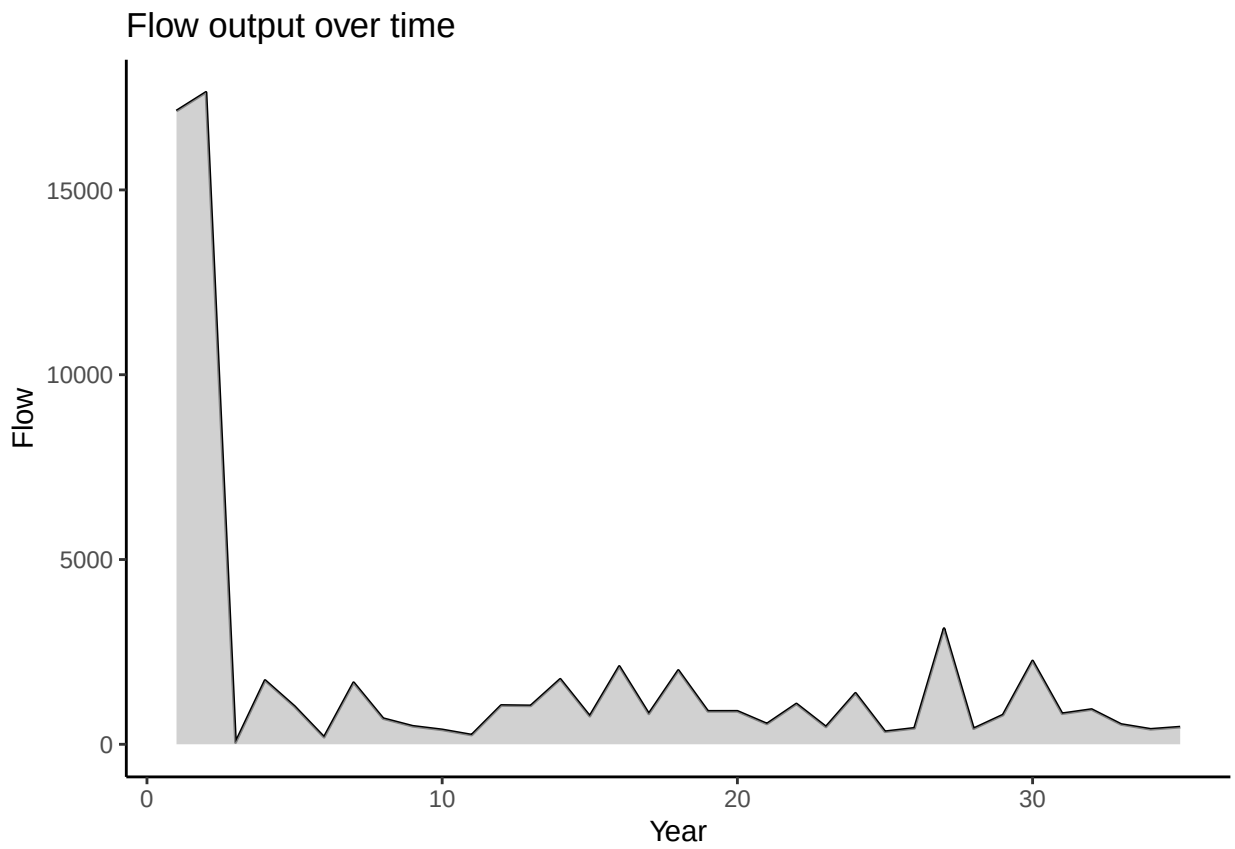
Function: plotting an individual flow output - *FlowPlot*

After running habplan, we will have several flows saved to our working directory. The option still exists to interactively watch the charts in a habplan window. However, we provide a function to visualize each flow individually.

First read in the flow output file, and then feed this file into the function.

```
#Read in one of the flow files
flow1 <- read.csv("./saveFlow1", sep="")
flow2 <- read.csv("./saveFlow2", sep="")

#Input the flow file into the function, and number of years
flowPlot(flow.data = flow1, nyear = 35)
flowPlot(flow.data = flow2, nyear = 35)
```


Flow output over time
15000
10000
5000
0
Flow
0
10
20
30
Year



Again, the first figure above shows the output flow from our HSI component throughout our study period, and the second shows the hard pulpwood yield. From a first glance of these figures, it appears that the outputs have not changed substantially. Let's re-run the HabSpace function to assess the spatial component of the recommended schedule.

Function: spatial contiguity assessment - *HabSpace*

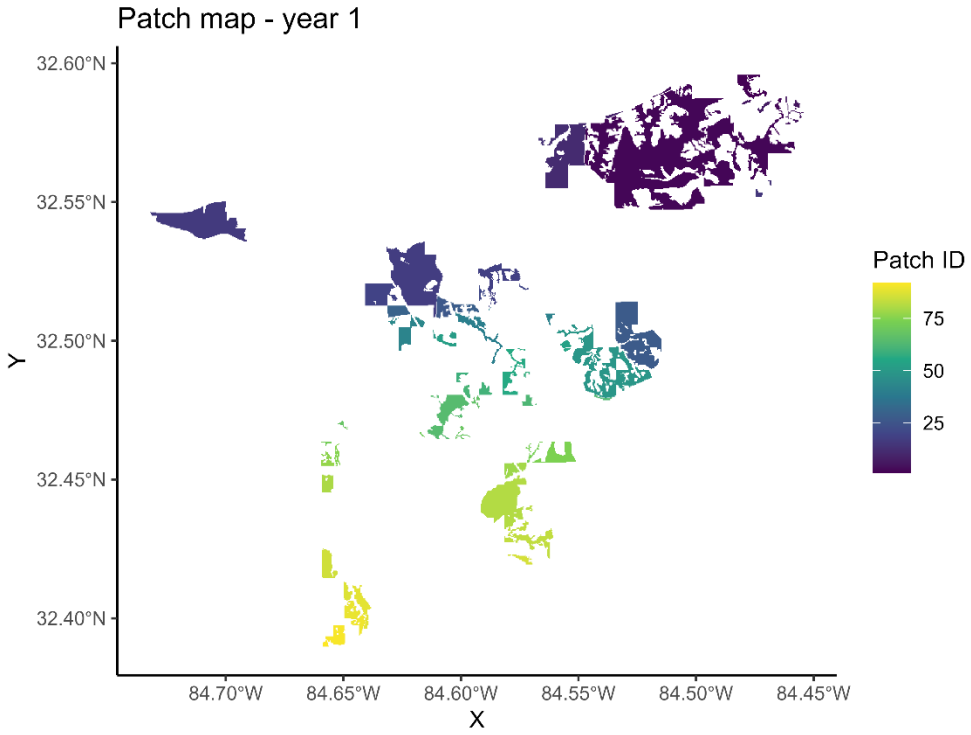
```
#Read in stand shapefiles
site.shp <- readOGR(dsn = "./shapefiles/Stands2_Shapefile/Stands_final.shp")

#Read in flow file from Habplan run
flow <- read.csv("./example/RCW/RCW/Flows/Breed_final.dat")
```

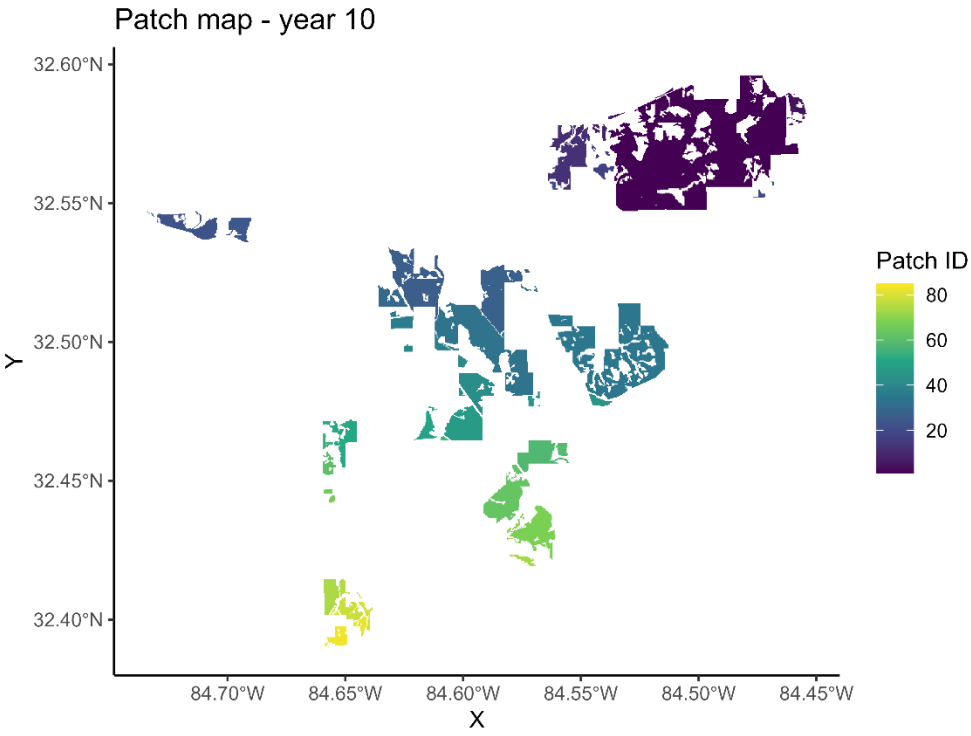
IMPORTANT The shapefile needs a data column called "StdID"

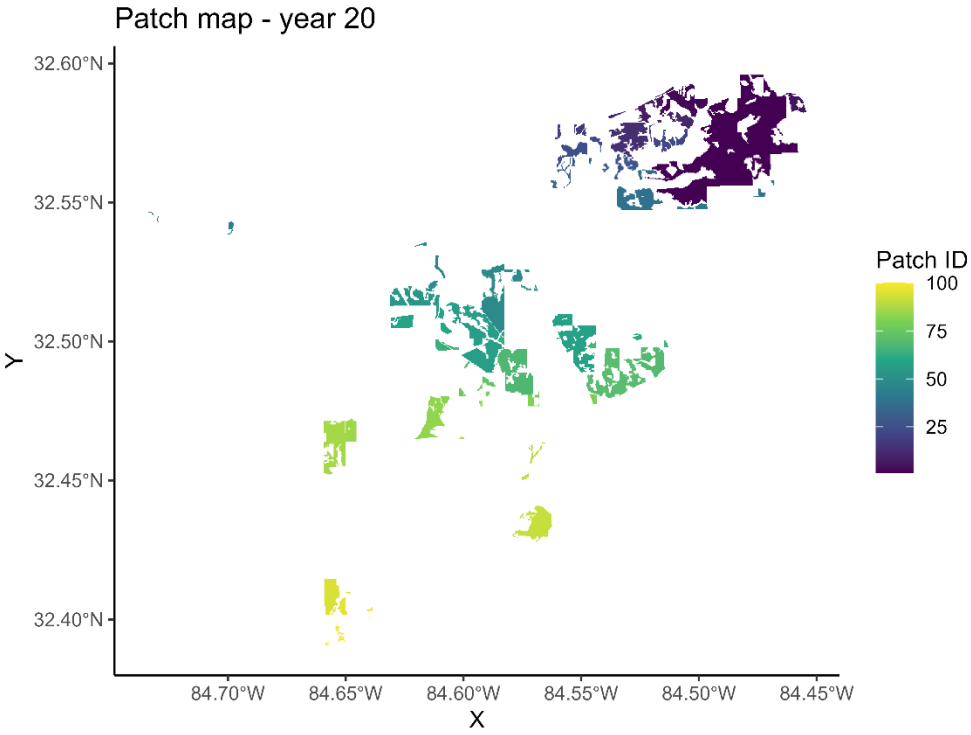
We will add the shapefile of stands, and the flow of choice to the function. As before, we will set the nyear. We now have two more options with this function, mode and dist.

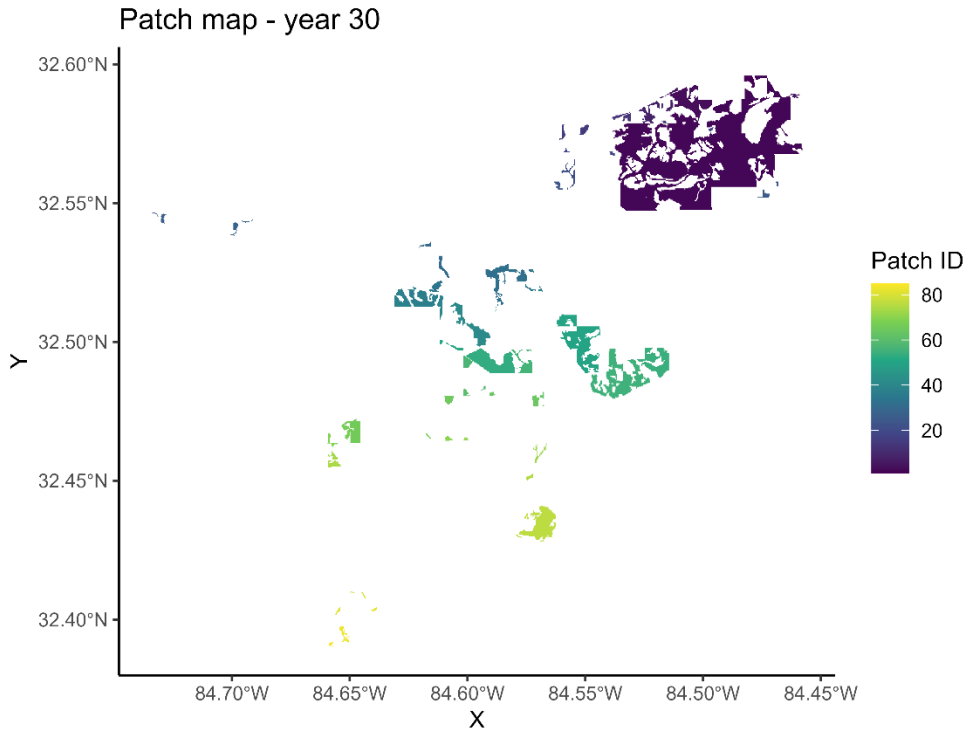
There are two modes to choose from, terrestrial or avian - depending on the species the HSI has been built around. If avian is chosen, the dist setting sets the distance in meters (or the unit of the shapefile) that the species of interest can comfortably travel to reach neighboring habitat patches. If the user has a terrestrial species which may still bridge gaps if the distance is short enough, e.g. crossing a road, then the avian option should be selected, and a smaller dist assigned.

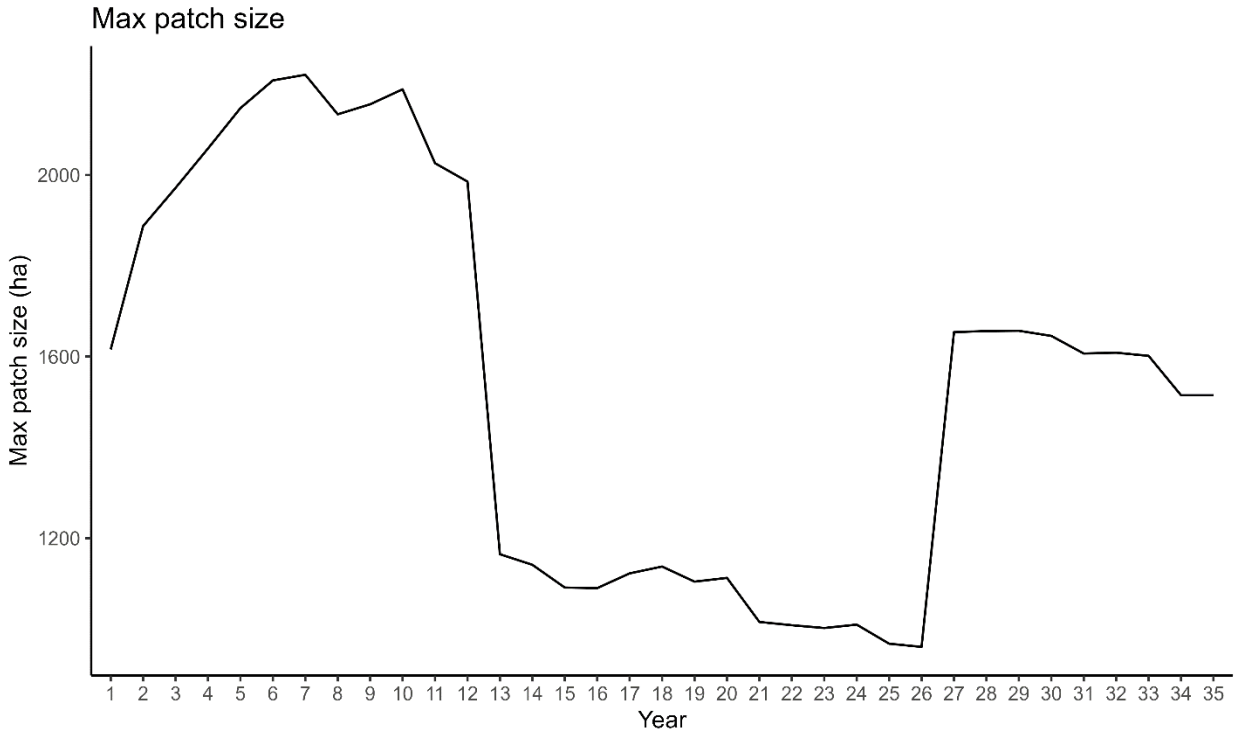
```
#Want to create a shapefile output for each year, to look at change
#of regime for each stand over time.

space.test <- HabSpace(site.shp = site.shp, flow = flow, nyear = 35,
                  mode = "terrestrial", dist = 500, level = "landscape")
```

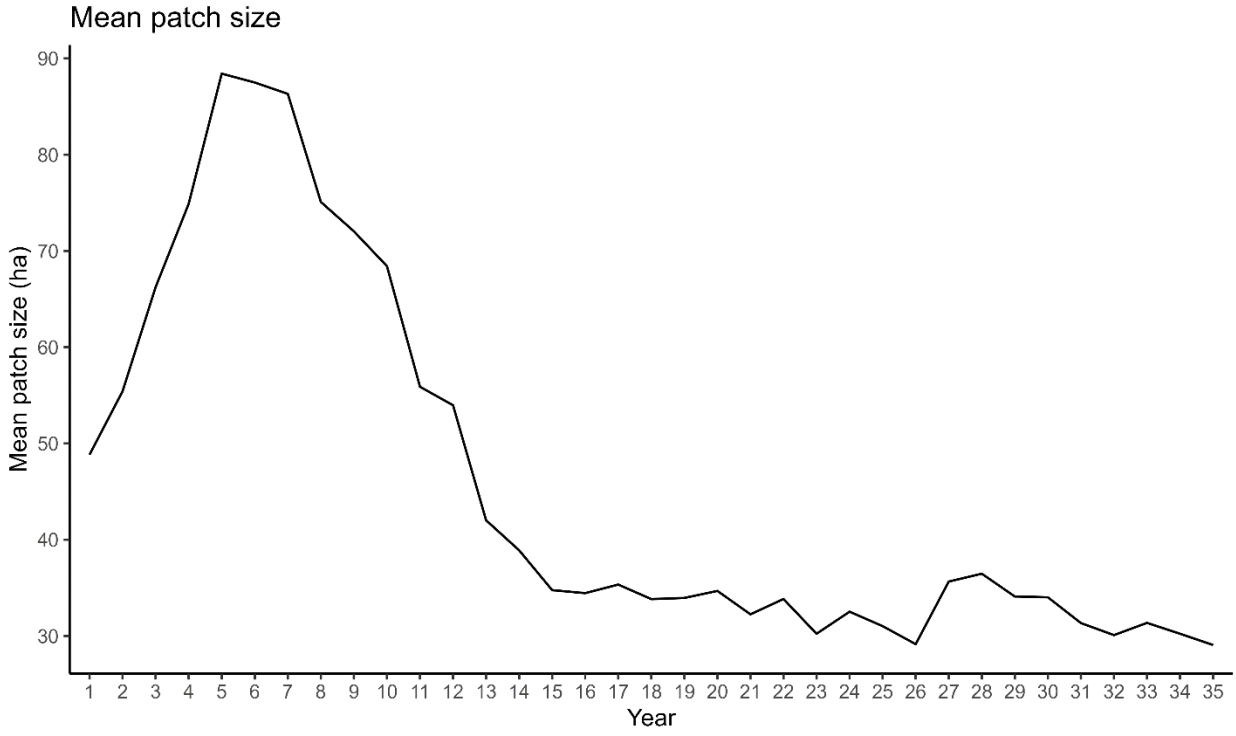

Patch map - year 1
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
75
50
25

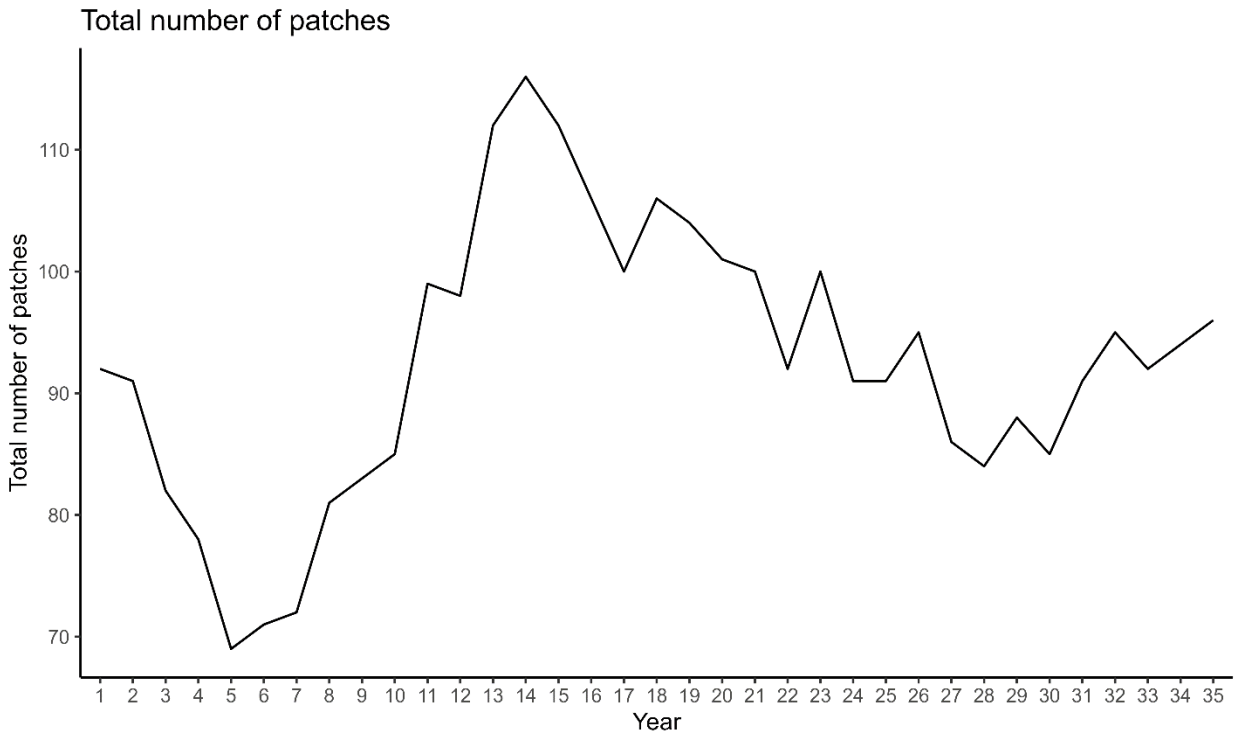

Patch map - year 10
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
80
60
40
20


Patch map - year 20
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
100
75
50
25


Patch map - year 30
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
80
60
40
20


Max patch size
Max patch size (ha)
2000
1600
1200
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Year


Mean patch size
Mean patch size (ha)
90
80
70
60
50
40
30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Year


Total number of patches
Total number of patches
110
100
90
80
70
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Year

For this final run, we are going to apply more dramatic changes to each flow component, with unique values for each one.

For the HSI flow component, we are going to have a much broader thlo set to 5000, but keep the thhi at 10000. However, we are going to set much different targets for the model. We are now aiming for 15000 in year 1, 5000 in year 20, and only 2000 by year 30 (1,15000;20,5000;30,2000).

For the hard pulpwood flow component, we believe we should be able to achieve much higher values. We will therefore set the thlo to 2000, but have the thhi be 100000. With such a large upper threshhold, we can set relatively conservative targets for our model. We will aim for 100 in year 1, 2000 in year 20, and keep this at 2000 for year 30 (1,100;20,2000;30,2000).

```
#Info for f1 component ----
#f1.file is the flow file for the first component
f1.file <- "RCW/RCW/Flows/HSI.dat"
#f1.bygone
f1.bygone <- ""
#f1.time0
f1.time0 <- "10000"
#f1.goal0
f1.goal0 <- "0.1"
#f1.thlo
f1.thlo <- "5000"
#f1.thhi
f1.thhi <- "10000"
#f1.goalplus
f1.goalplus <- ".05"
#f1.goalf
f1.goalf <- "0.5"
#f1.slope
f1.slope <- "0.0"
#f1.weightf
f1.weightf <- "1.0"
#f1.weight0
f1.weight0 <- "1.0"
#f1.model
f1.model <- "1,15000;20,5000;30,2000;"
#f1.title
f1.title <- "Breed.dat"
#Combine f1 components for writing
f1.comp <- c('<flow title="F1 Component">',
             paste0('<file value="', f1.file, '" />'),
             paste0('<bygone value="', f1.bygone, '" />'),
             paste0('<time0 value="', f1.time0, '" />'),
             paste0('<goal0 value="', f1.goal0, '" />'),
             paste0('<threshLo value="', f1.thlo, '" />'),
             paste0('<threshHi value="', f1.thhi, '" />'),
             paste0('<goalPlus value="', f1.goalplus, '" />'),
             paste0('<goalF value="', f1.goalf, '" />'),
             paste0('<slope value="', f1.slope, '" />'),
             paste0('<weightF value="', f1.weightf, '" />'),
             paste0('<weight0 value="', f1.weight0, '" />'),
             paste0('<model value="', f1.model, '" />'),
             paste0('<title value="', f1.title, '" />'),
             '<bounds height="330" width="366" x="553" y="443" />',
```

```
            "</flow>")


#Info for f2 component ----
#f2.file is the flow file for the first component
f2.file <- "RCW/RCW/Flows/Harv_P_Pulp_Tons.dat"
#f2.bygone
f2.bygone <- ""
#f2.time0
f2.time0 <- "1000"
#f2.goal0
f2.goal0 <- "0.1"
#f2.target
#f2.target <- "20000"
#f2.thlo
f2.thlo <- "2000"
#f2.thhi
f2.thhi <- "100000"
#f2.goalplus
f2.goalplus <- ".05"
#f2.goalf
f2.goalf <- "0.5"
#f2.slope
f2.slope <- "0.0"
#f2.weightf
f2.weightf <- "1.0"
#f2.weight0
f2.weight0 <- "1.0"
#f2.model
f2.model.1 <- "1,100;20,2000;30,2000;"
#f2.next.year - sets up the year where the next target is set
#f2.next.year <- "20"
#f2.next.target - determines what the next target is after the first year
#f2.next.target <- 100000
#f2.title
f2.title <- "Harv_P_Pulp_Tons.dat"

f2.comp <- c('<flow title="F2 Component">',
             paste0('<file value="', f2.file, '" />'),
             paste0('<bygone value="', f2.bygone, '" />'),
             paste0('<time0 value="', f2.time0, '" />'),
             paste0('<goal0 value="', f2.goal0, '" />'),
             paste0('<threshLo value="', f2.thlo, '" />'),
             paste0('<threshHi value="', f2.thhi, '" />'),
             paste0('<goalPlus value="', f2.goalplus, '" />'),
             paste0('<goalF value="', f2.goalf, '" />'),
             paste0('<slope value="', f2.slope, '" />'),
             paste0('<weightF value="', f2.weightf, '" />'),
             paste0('<weight0 value="', f2.weight0, '" />'),
             #paste0('<model value="1,', f2.target, ';', f2.next.year, ',',
             #      f2.next.target, '" />'),
             paste0('<title value="', f2.title, '" />'),
             '<bounds height="331" width="368" x="1260" y="440" />',
```

```
              "</flow>")
```

Before running the function, set a new working directory so that the project file is saved into the correct folder

```
setwd("FILEPATH HERE/project/")
```

```
#Provide each of these flow component objects to the function and run
writeProj(f1.comp = f1.comp, f2.comp= f2.comp)

#Reset working directory
setwd("FILEPATH HERE")
```

Now we have a project file saved to our working directory. Let's open habplan and see if this has worked.

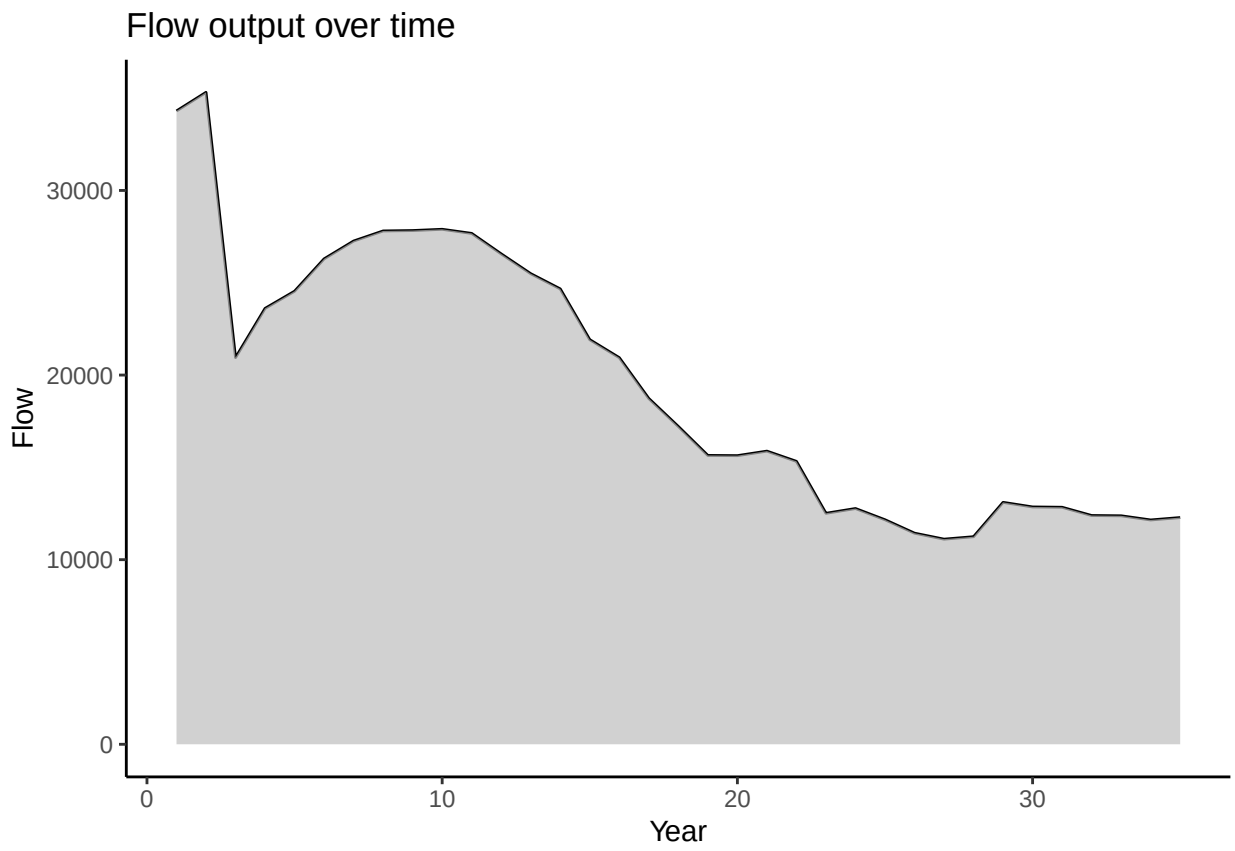
```
#Open Habplan (if run from here, R functionality will cease until Habplan is closed)
shell("h", wait=TRUE)
```

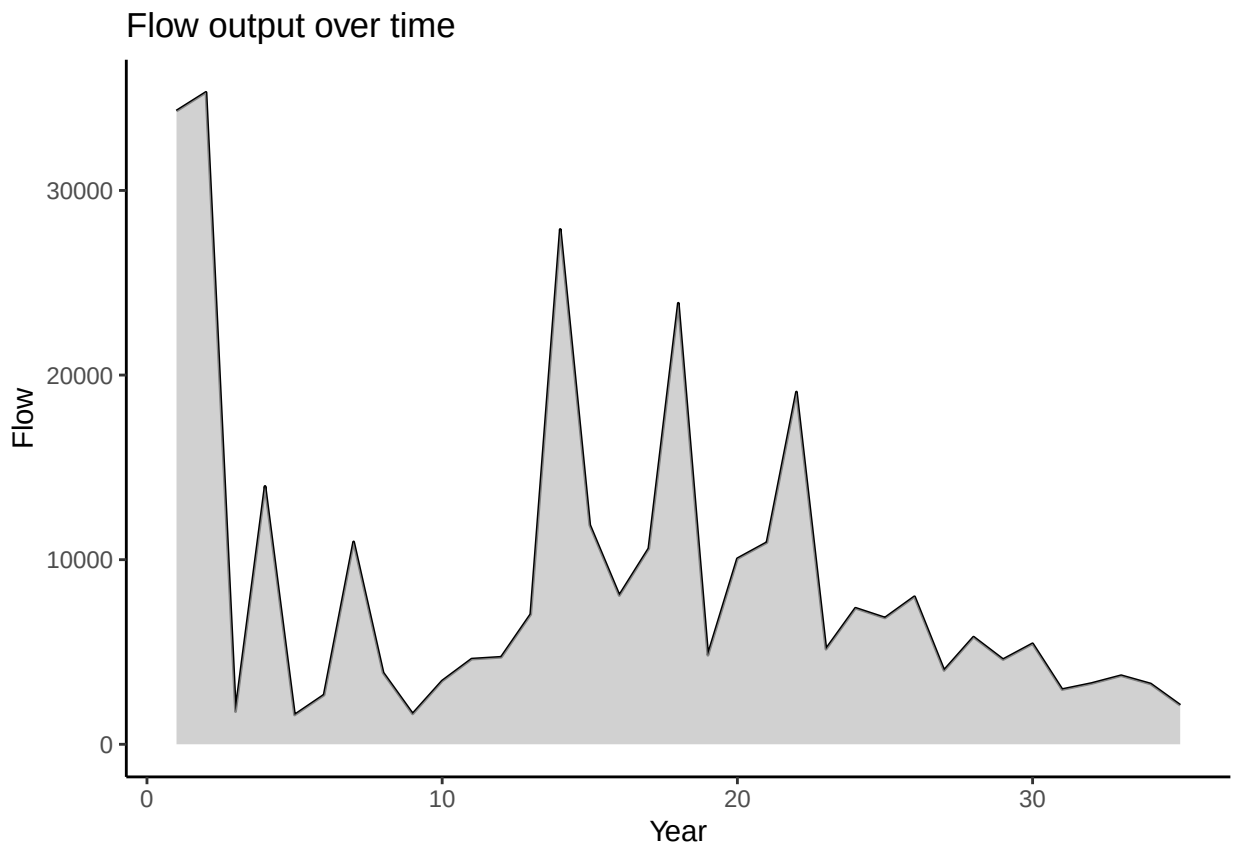
Function: plotting an individual flow output - *FlowPlot*

After running habplan, we will have several flows saved to our working directory. The option still exists to interactively watch the charts in a habplan window. However, we provide a function to visualize each flow individually.

First read in the flow output file, and then feed this file into the function.

```
#Read in one of the flow files
flow1 <- read.csv("./saveFlow1", sep="")
flow2 <- read.csv("./saveFlow2", sep="")

#Input the flow file into the function, and number of years
flowPlot(flow.data = flow1, nyear = 35)
flowPlot(flow.data = flow2, nyear = 35)
```


Flow output over time
30000
20000
10000
0
Flow
0
10
20
30
Year



Again, the first figure above shows the output flow from our HSI component throughout our study period, and the second shows the hard pulpwood yield. These appear to have provided much larger yields for both HSI and hard pulpwood.

We can now run the final iteration of *HabSpace* to assess the spatial component. Even if the flow components are much better, it is still important to ensure that we have not compromised the spatial contiguity and suitability of the landscape for at risk species.

Function: spatial contiguity assessment - *HabSpace*

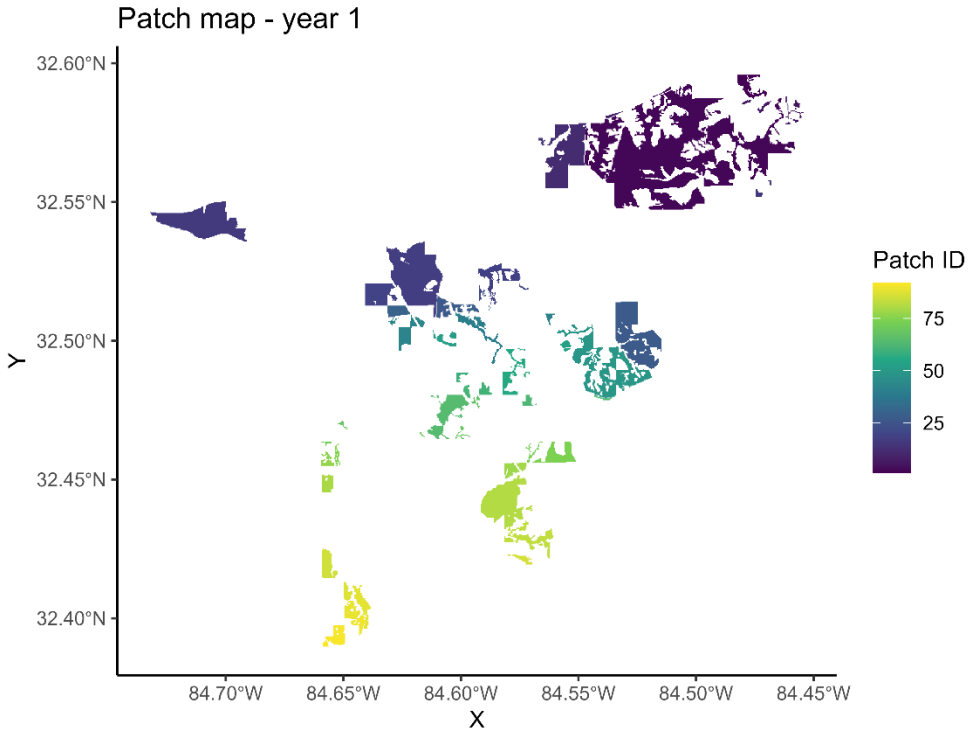
```
#Read in stand shapefiles
site.shp <- readOGR(dsn = "./shapefiles/Stands2_Shapefile/Stands_final.shp")

#Read in flow file from Habplan run
flow <- read.csv("./example/RCW/RCW/Flows/Breed_final.dat")
```

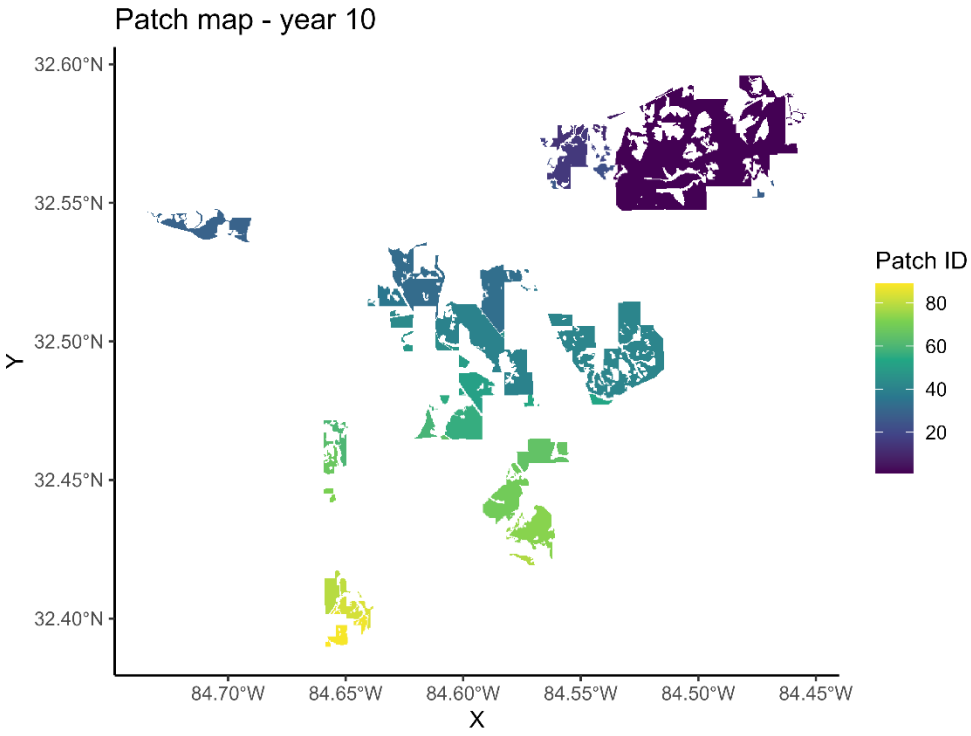
IMPORTANT The shapefile needs a data column called "StdID"

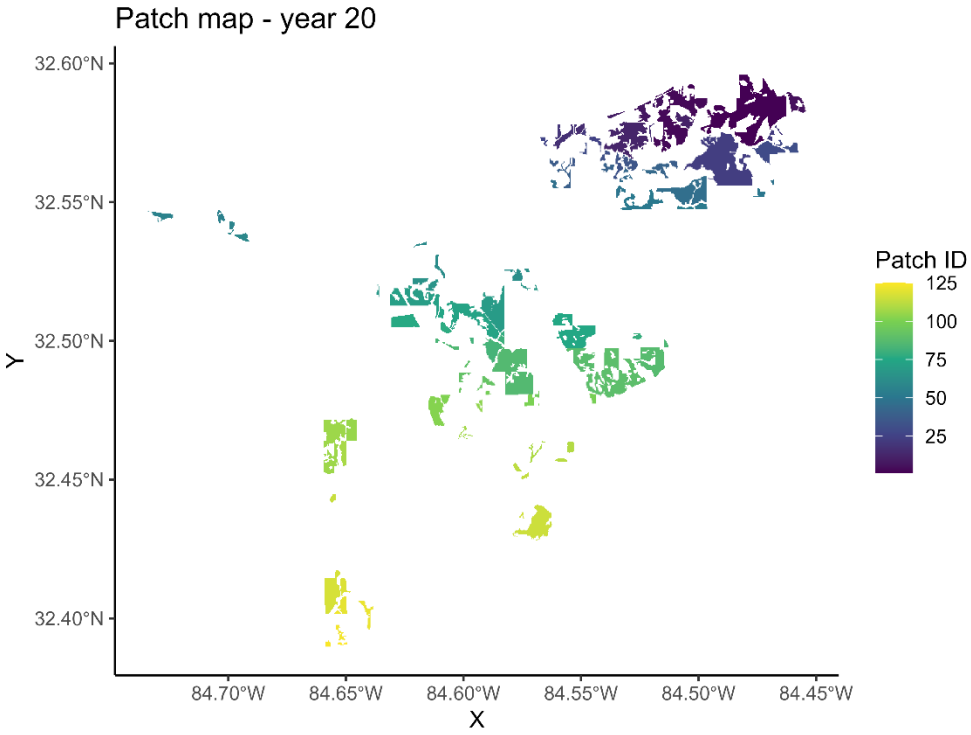
We will add the shapefile of stands, and the flow of choice to the function. As before, we will set the nyear. We now have two more options with this function, mode and dist.

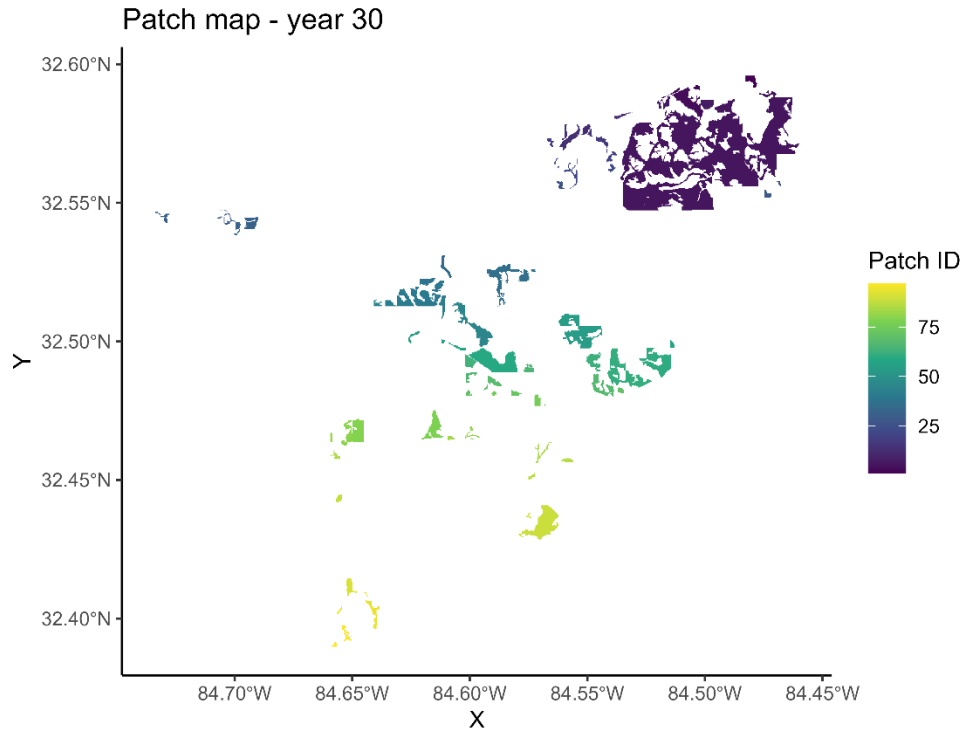
There are two modes to choose from, terrestrial or avian - depending on the species the HSI has been built around. If avian is chosen, the dist setting sets the distance in meters (or the unit of the shapefile) that the species of interest can comfortably travel to reach neighboring habitat patches. If the user has a terrestrial species which may still bridge gaps if the distance is short enough, e.g. crossing a road, then the avian option should be selected, and a smaller dist assigned.

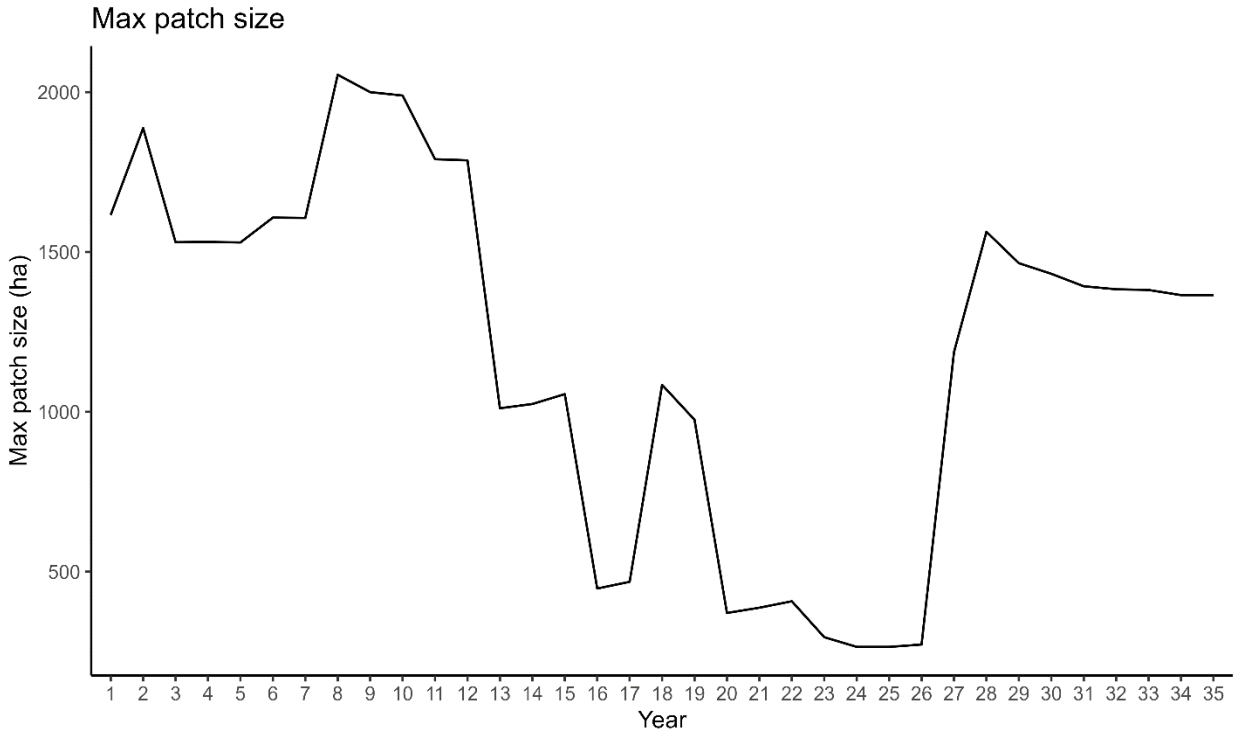
```
#Want to create a shapefile output for each year, to look at change
#of regime for each stand over time.

space.test <- HabSpace(site.shp = site.shp, flow = flow, nyear = 35,
                  mode = "terrestrial", dist = 500, level = "landscape")
```

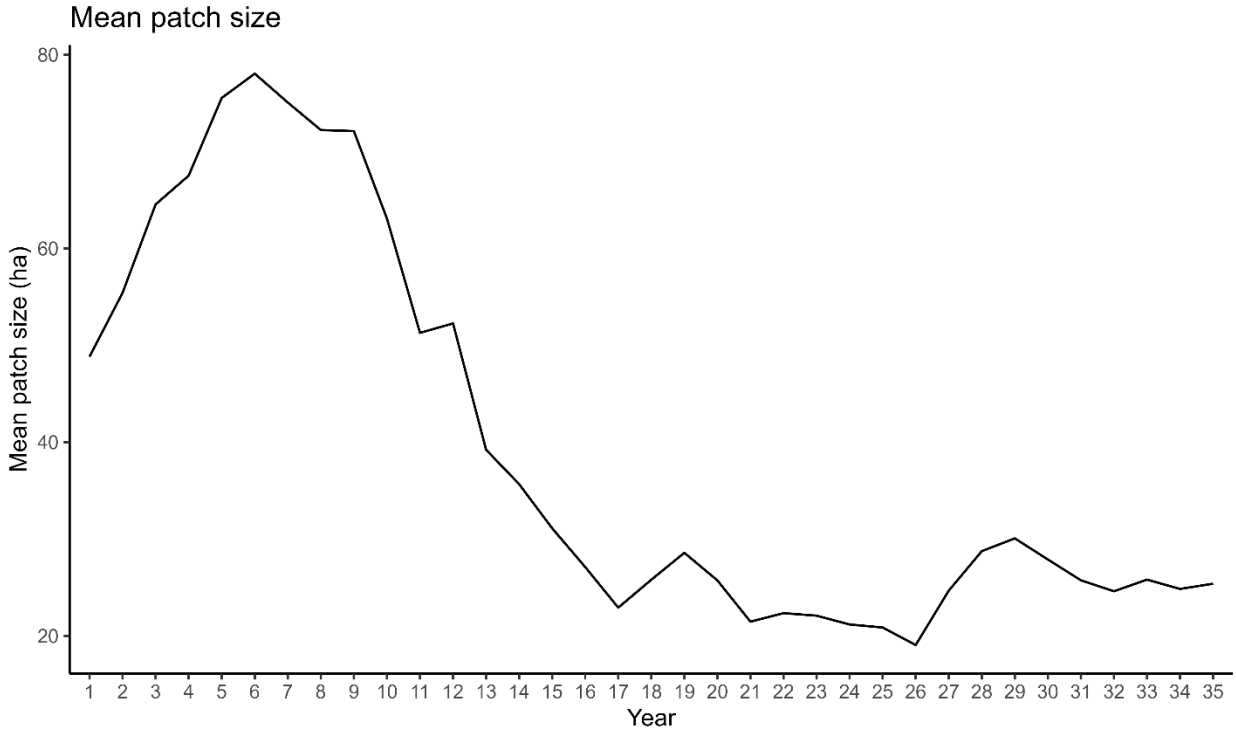

Patch map - year 1
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
75
50
25

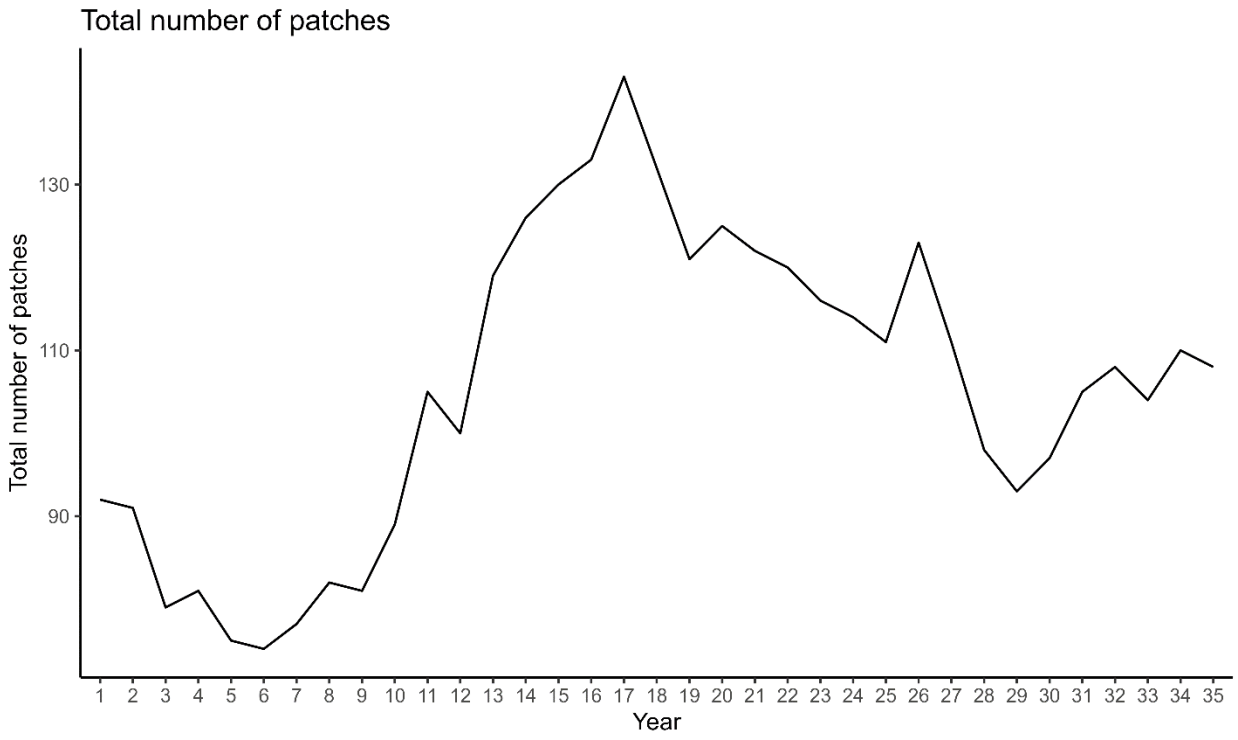

Patch map - year 10
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
80
60
40
20


Patch map - year 20
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
125
100
75
50
25


Patch map - year 30
32.60°N
32.55°N
32.50°N
32.45°N
32.40°N
Y
84.70°W
84.65°W
84.60°W
84.55°W
84.50°W
84.45°W
X
Patch ID
75
50
25


Max patch size
Max patch size (ha)
2000
1500
1000
500
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Year


Mean patch size
Mean patch size (ha)
80
60
40
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Year


Total number of patches
Total number of patches
130
110
90
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Year

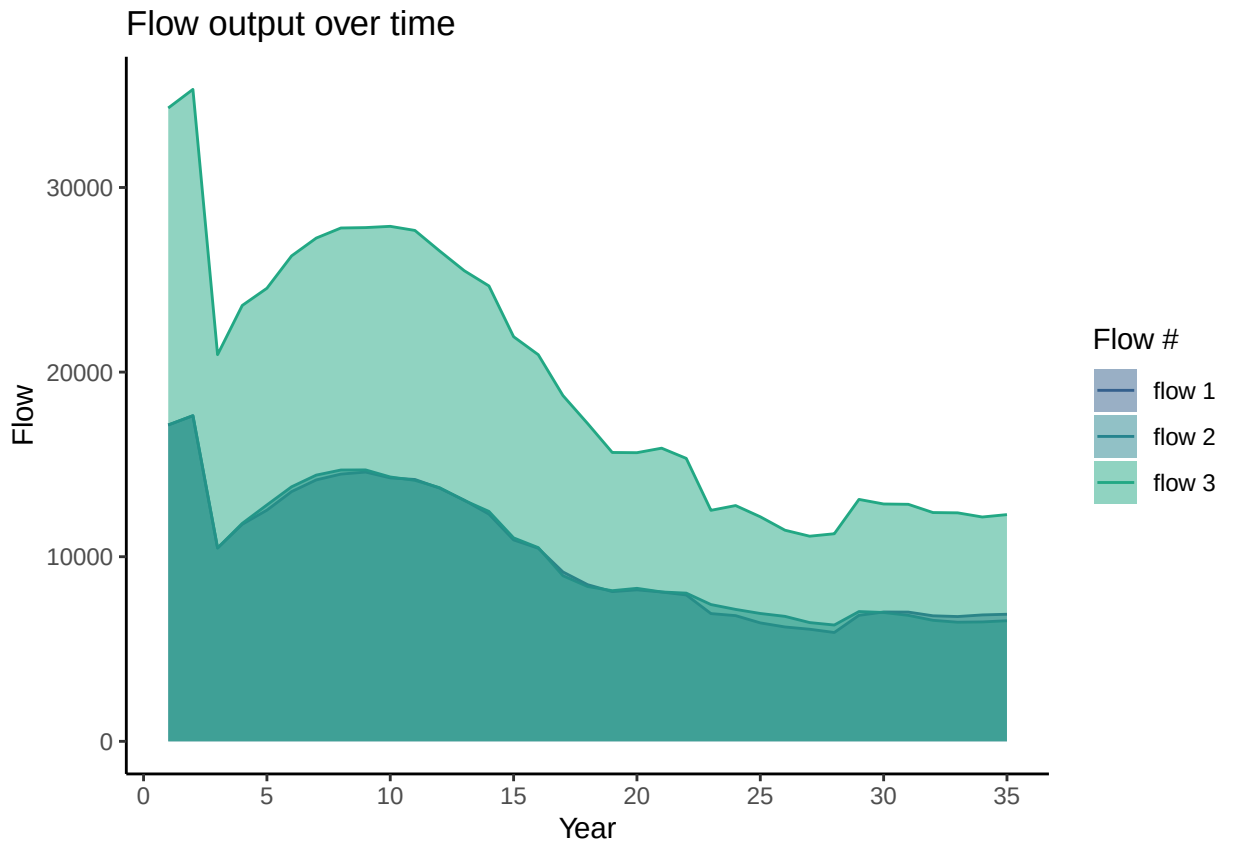
We can compare each run to see how the yields have changed with each change we have made.

Function: multiple flow outputs - *ComPlot*

The above will create a chart for one of the flows. However, it is also useful to be able to visualize the flows in relation to one another. We therefore provide another function: comPlot.

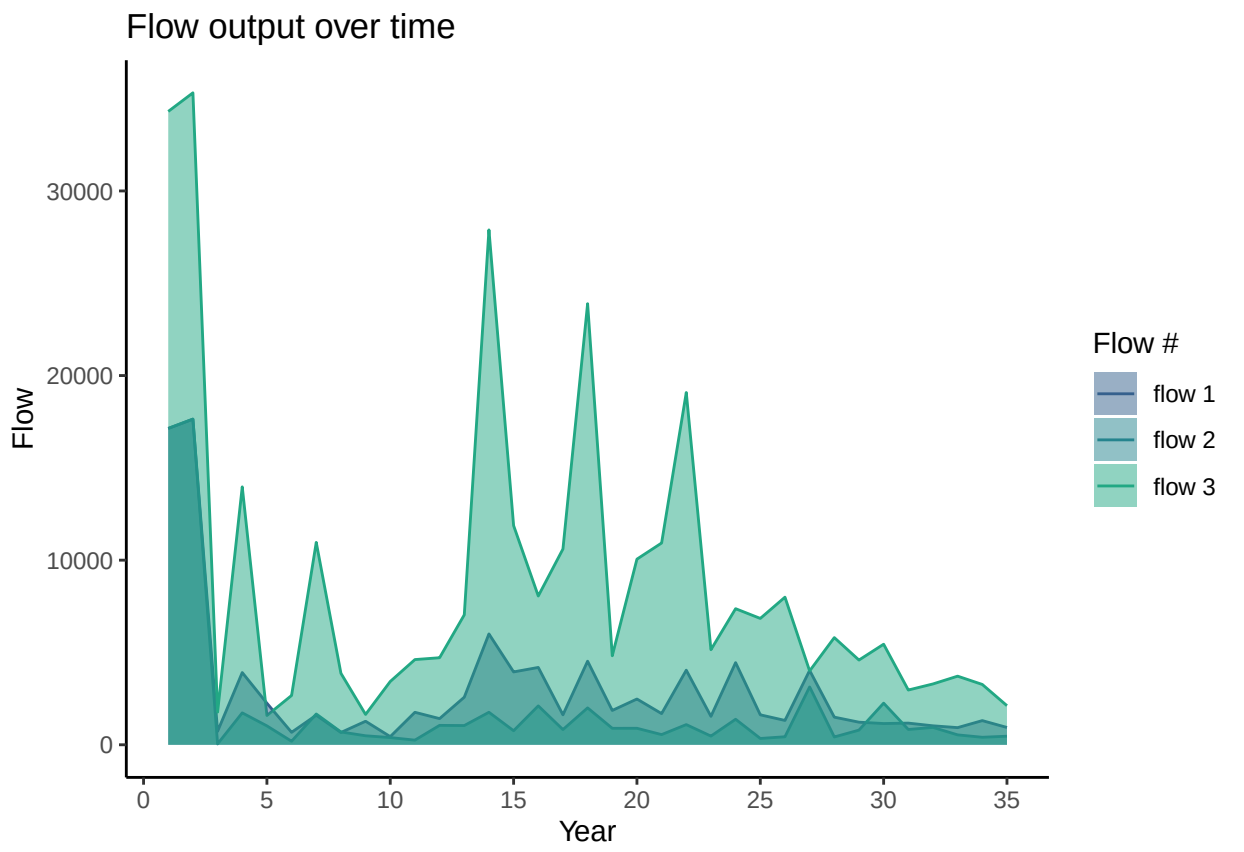
This requires the same input as before, but this time we will introduce some of the other flows outputs. Current maximum of 3 flows.

```
#Read in the example flows:
flow1 <- read.csv("./Run_1/saveFlow1", sep="")
flow2 <- read.csv("./Run_2/saveFlow1", sep="")
flow3 <- read.csv("./Run_3/saveFlow1", sep="")
#flow4 <- read.csv("./saveFlow4", sep="")

#Run function with new flows
comPlot(flow.data.1 = flow1, flow.data.2 = flow2,
        flow.data.3 = flow3, 35)
```




```
#Read in the example flows:
flow1 <- read.csv("./Run_1/saveFlow2", sep="")
flow2 <- read.csv("./Run_2/saveFlow2", sep="")
flow3 <- read.csv("./Run_3/saveFlow2", sep="")
```

```
#flow4 <- read.csv("./saveFlow4", sep="")

#Run function with new flows
comPlot(flow.data.1 = flow1, flow.data.2 = flow2,
        flow.data.3 = flow3, 35)
```



By visualizing the flows in comparison to one another from each run, we can now clearly see the little differences found between the first and second run, but the changes we made to the third run have greatly increased our averge HSI and harvested hard pulpwood - ultimately providing more habitat for our species of interest, but maximizing yield and profits also.

Function: saving the top schedule to shapefile - *StandSched*

From this example, we are happy with the final run and will apply the recommended regime to each stand based on the output schedule from Habplan. Our next function pulls the information from the best schedule saved in the working directory (from Habplan), and attaches this to the stand shapefile.

As long as the file is still in the working directory (saveSched), then all we need to do is input the stand shapefile and run the function.

```
#Read in stand shapefiles
site.shp <- readOGR(dsn = "./shapefiles/Stands2_Shapefile/Stands_final.shp")

#Run function to add schedule to shapefile
standSched(site.shp)
```

A new .shp file can now be found in the working directory titled Site_with_schedule.shp.

End of vignette.