HabplanR Vignette

Max Dolton Jones

2024-10-24

Introduction to HabplanR

The *HabplanR* package was developed to complement 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 today's landscape management problems; however, the use of Habplan is limited, requiring in-depth knowledge of not only the management issue at hand, but also about tedious file-formatting and input requirements for the program to run efficiently. *HabplanR* broadens applicability of Habplan and make file-formatting and data input easier for more efficient program use.

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

- Use forest condition data to create Habplan-appropriate input files;
- Use forest condition 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 analyses focused at either the forest stand, habitat patch, or entire landscape level.

The following vignette provides a working example of using the *HabplanR* package. We demonstrate the utility of *HabplanR* using flow objectives derived from Forest Vegetation Simulator (FVS; https://www.fs.usda.gov/fvs/) growth model data. Specifically, we explore two objective flows: harvested tons of pine pulpwood (acres), and habitat area (acres; based on predetermined Habitat Suitability Index [HSI]). We use these flows as proxies for economic and ecological goals, respectively, highlighting the range of objectives that Habplan can accommodate. There can be any number of objectives used within a multi-objective problem, and this vignette serves as an example on how to incorporate and adjust two objectives, which can be expanded across several user-defined objective functions.

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("NCASI/Habplan_Memberonly/HabplanR",  build_vignettes = TRUE,
force = TRUE)

#The installation might ask if any package updates are needed. If this is
prompted,
#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):

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

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. If there are any issues installing/using the spdep package, there may be an issue relating to the terra package, and/or the package raster may be needed. See https://github.com/r-spatial/spdep/issues/78 for further information.

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

library(readr) #v.2.1.2
library(dplyr) #v.1.0.10
library(ggplot2) #v.3.3.6
library(broom) #v.1.0.5
library(spdep) #v.1.2-7
library(terra) #v.1.6-17
library(tidyterra) #v.0.3.2
library(landscapemetrics) #v.1.5.5
```

The *HabplanR* package works by saving and loading files from the working directory. It is important to set the working directory to the location where all of the input files are located, and where output files will be saved. This working directory will stay consistent for the entire vignette.

All of the necessary files and software for this vignette can be found at https://github.com/NCASI/Habplan_Memberonly. Download both the "HabplanR" and "Hablpan_software" folders, and then copy all of the files from the Habplan_Software folder into the HabplanR folder. The HabplanR folder should then be set as the working directory for the purpose of this vignette.

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

Within the working directory that we just assigned, we need the following files:

- Stand_info.csv (Stand IDs, acreage, and description)
- fvs_results-new.csv (Stand data from projected growth model)

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 of interest. 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

IMPORTANT The publicly available version of Habplan only supports data with up to 500 polygons (stands). For larger data needs, please contact NCASI to obtain access.

```
#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 × 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 FVS. We will call these data std.data.

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

#Take a look at the data
head(std.data)
```

```
#> # A tibble: 6 × 18
#>   RegimeKey `Regime Activity`     StandID  Year   Age    BA   TPA
Pine_Pulp_Tons
#>       <dbl> <chr>                 <chr>   <dbl> <dbl> <dbl> <dbl>
<dbl>
#> 1         4 Thin 1 at 120 BA to … 15588_1  2017     9  5.53  266.
1.35
#> 2         4 Thin 1 at 120 BA to … 15588_1  2020    12  8.82  265.
1.83
#> 3         4 Thin 1 at 120 BA to … 15588_1  2023    15  8.17  169.
2.10
#> 4         4 Thin 1 at 120 BA to … 15588_1  2026    18 10.8   168.
2.79
#> 5         4 Thin 1 at 120 BA to … 15588_1  2029    21 14.0   168.
3.43
#> 6         4 Thin 1 at 120 BA to … 15588_1  2032    24 17.6   167.
3.42
#> # ⓘ 10 more variables: Pine_CNS_Tons <dbl>, Pine_Saw_Tons <dbl>,
#> #   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>, HSI <dbl>
```

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)
2. std.info: the information file for each forest stand (above)
3. col: the appropriate data column from std.data (below)
4. 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(std.data)
#>  [1] "RegimeKey"        "Regime Activity"  "StandID"          "Year"
#>  [5] "Age"              "BA"               "TPA"
"Pine_Pulp_Tons"
```

```
#>  [9] "Pine_CNS_Tons"    "Pine_Saw_Tons"    "Hdwd_Pulp_Tons"
"Hdwd_Saw_Tons"
#> [13] "Harv_P_Pulp_Tons" "Harv_P_CNS_Tons"  "Harv_P_Saw_Tons"
"Harv_H_Pulp_Tons"
#> [17] "Harv_H_Saw_Tons"  "HSI"
```

From the column names, we are interested in "Harv_P_Pulp_Tons" (pulp pinewood harvested [tons/acre]) and "HSI" (stands with a higher HSI value than a specified value), which are column number 13 and 18 respectively. We can input a column number into our function. We also need to provide "nyear" which is the number of years (or time periods) that are of interest (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 available 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 not habitat, whereas any values above 0.5 would be habitat and therefore added to the flow file. Below, we have set the HSI threshold value to 0.7 for creating our HSI flow file.

Later in the vignette, we provide an additional function for creating HSI values using growth model data and user-defined formula.

```
#Create flow file for Harv_P_Pulp_Tons
out.flow.1 <- habConvert(std.data = std.data, std.info = std.info, col = 13,
                         nyear = 35)

#Create flow file for HSI
out.flow.2 <- habConvert(std.data = std.data, std.info = std.info, col = 18,
                         nyear = 35, HSI = 0.7)
```

Now we will have two flow files saved in our working directory:

- Harv_P_Pulp_Tons.dat
- HSI.dat

For the HSI flow file, 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. The Harv_P_Pulp_Tons flow file contains data for each time period extracted directly from the FVS growth model.

Function: create project file - *WriteProj*

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

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

Because Habplan inputs allow for customization, we provide code that 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. We need to create four objects in the R environment for the function to create a project file:

- npoly: The number of unique forest stands.
- config: The configuration of flow components and subcomponents (see below and page 46 of Habplan Manual)
- wd: The working directory where flow files and output files are to be stored, and all data and software are maintained. The working directory is user-defined.
- iter: The number of iterations of Habplan (see page 13 of Habplan Manual)

Config is a string of numbers, where each number represents a flow component or subcomponent to be considered during the Habplan run. The first number represents the number of flows (N), followed by N pairs of numbers that represent clearcut and blocksize subcomponents (0 or 1). The final three numbers are Biol1, Biol2, and Spatial Model subcomponents (0 or 1; see section 10, 11, and 12 of Habplan Manual). Our assigned configuration below states that we are considering two flow components only, with no additional subcomponents.

```
#Overarching information for habplan run:
#Get the number of stands for later
npoly <- length(unique(std.data$StandID))
#We can override config by using the following:
config <- "2,0,0,0,0,0,0,0"
#wd is the working directory where data are stored (end with '/' as shown
below)
wd <- "FILEPATH HERE/" #same as original working directory assigned above,
but include
#the "/" at the end as shown here
#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 and harvested pine pulpwood flows that we created above. We want to achieve the highest acreage of appropriate 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 parameters but focus on three main objects that 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.

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 <- paste0(wd,"HSI.dat")
f1.bygone <- ""
f1.time0 <- "10000"
f1.goal0 <- "0.1"
f1.thlo <- "1000"
f1.thhi <- "5000"
f1.goalplus <- ".05"
f1.goalf <- "0.5"
f1.slope <- "0.0"
f1.weightf <- "1.0"
f1.weight0 <- "1.0"
f1.model <- "1,1000;20,2500;30,5000;"
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 <- paste0(wd, "Harv_P_Pulp_Tons.dat")
f2.bygone <- ""
f2.time0 <- "1000"
f2.goal0 <- "0.1"
f2.thlo <- "1000"
f2.thhi <- "5000"
f2.goalplus <- ".05"
f2.goalf <- "0.5"
f2.slope <- "0.0"
f2.weightf <- "1.0"
f2.weight0 <- "1.0"
```

```
f2.model.1 <- "1,1000;20,2500;30,5000;"
f2.title <- "Harv_P_Pulp_Tons.dat"
#Combine f2 components for writing
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>")


#Provide each of these flow component objects to the function and run
writeProj(f1.comp = f1.comp, 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. The Habplan program needs to be stored in the working directory that was assigned earlier, so that it can be run from RStudio below.

IMPORTANT Java must be installed for Habplan to run, otherwise execution of the following code will fail.

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