

PRIORITIZR WORKSHOP MANUAL

Jeffrey O. Hanson

2019-10-06

Contents

1	Welcome!	5
2	Introduction	7
3	Data	11
4	Gap analysis	27
5	Spatial prioritizations	35
6	Answers	53
7	Acknowledgements	59
8	Session information	61

Chapter 1

Welcome!

Here you will find the manual for the prioritizr module of the *Spatial Conservation Prioritization: Concepts, Methods and Application workshop* held at CIBIO-InBIO, Vairão, Portugal. Before you arrive at the workshop, you should make sure that you have correctly **set up your computer for the workshop** and you have **downloaded the data from [here](#)**. We cannot guarantee a reliable Internet connection during the workshop, and so you may be unable to complete the workshop if you have not set up your computer beforehand.

Chapter 2

Introduction

2.1 Overview

The aim of this workshop is to help you get started with using the `prioritizr` R package for systematic conservation planning. It is not designed to give you a comprehensive overview and you will not become an expert after completing this workshop. Instead, we want to help you understand the core principles of conservation planning and guide you through some of the common tasks involved with developing prioritizations. In other words, we want to give you the knowledge base and confidence needed to start applying systematic conservation planning to your own work.

You are not alone in this workshop. If you are having trouble, please put your hand up and one of the instructors will help you as soon as they can. You can also ask the people sitting next to you for help too. **Most importantly, the code needed to answer the questions in this workshop are almost always located in the same section as the question. So if you are stuck, try rereading the example code and see if you can modify it to answer the question.** Please note that the first thing an instructor will ask you will (probably) be “what have you tried so far?”. We can’t help you if you haven’t tried anything.

2.2 Setting up your computer

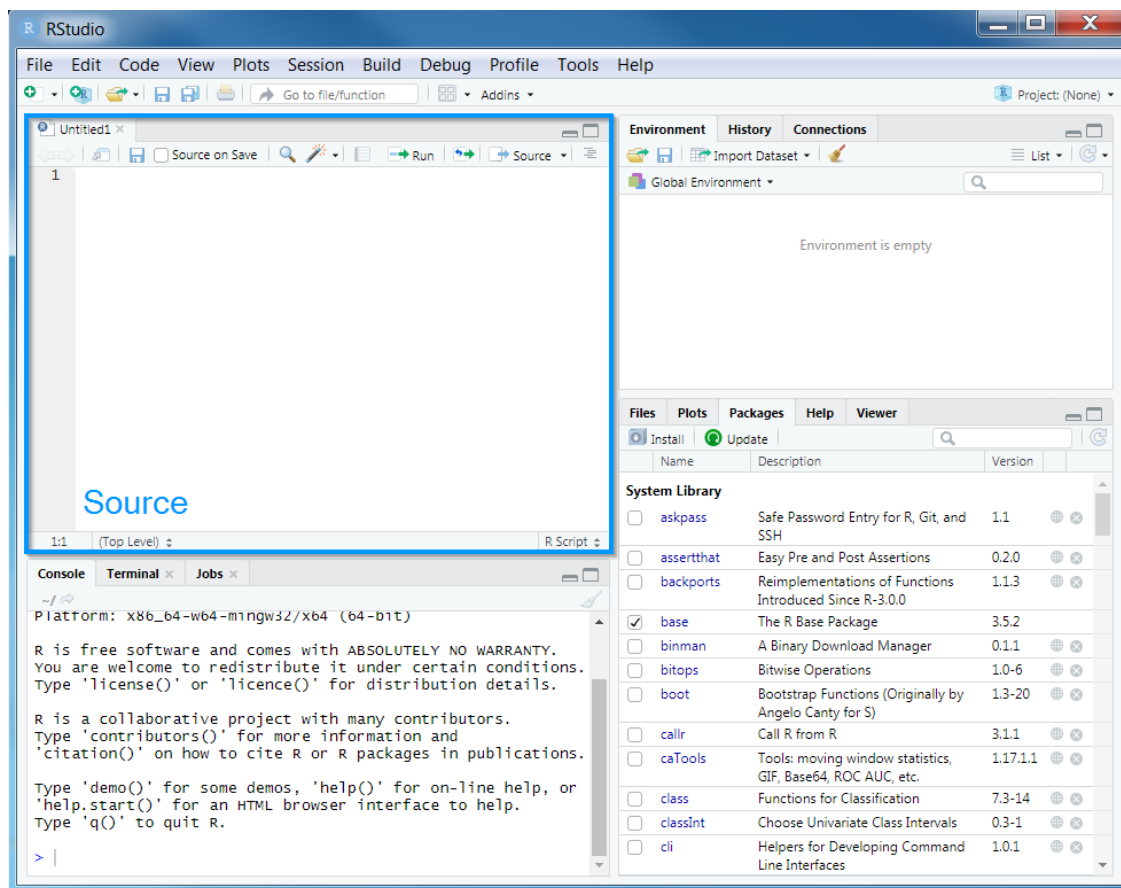
You will need to have both [R](#) and [RStudio](#) installed on your computer to complete this workshop. Although it is not imperative that you have the latest version of RStudio installed, **you will need the latest version of R installed (i.e. version 3.6.1)**. Please note that you need administrative permissions to complete install these programs. After installing them, you will also need to install various R packages too.

2.2.1 R

The [R statistical computing environment](#) can be downloaded from the Comprehensive R Archive Network (CRAN). You can download the latest version of R (version 3.6.1) from here: <https://cloud.r-project.org>. Please note that you will need to download the correct file for your operating system (i.e. Linux, Mac OSX, Windows).

2.2.2 RStudio

[RStudio](#) is an integrated development environment (IDE). In other words, it is a program that is designed to make your R programming experience more enjoyable. During this workshop, you will interact with R through RStudio—meaning that you will open RStudio when you want to code in R. You can download the latest version of RStudio here: <http://www.rstudio.com/download>. When you start RStudio, you will see two main parts of the interface:



You can type R code into the *Console* part of the interface and press the enter key to run the code.

2.2.3 R packages

An R package is a collection of R code and documentation that can be installed to enhance the standard R environment with additional functionality. Currently, there are over ten thousand R packages available on CRAN. Each of these R packages (mostly) aim to serve a specific need, such as [reading Excel spreadsheets](#), [downloading satellite imagery data](#), [downloading and cleaning protected area data](#), or [fitting environmental niche models](#). In fact, R has such a diverse ecosystem of R packages, that the question is (generally) not “can I use R to do ...?” but “what R package can I use to ...?”. During this workshop, we will use various R packages. To install these R packages, please run enter the code below in the *Console* part of the RStudio interface and press enter. Please note that you will require an internet connection to install the packages and the installation process may take a while to complete.

```
install.packages(c("sf", "tidyverse", "sp", "rgeos", "rgdal", "raster", "units",  
                  "prioritizr", "prioritizrdata", "Rsymphony", "mapview",  
                  "assertthat", "velox", "remotes", "gridExtra"))  
remotes::install_bioc("lpsymphony")
```

2.3 Further reading

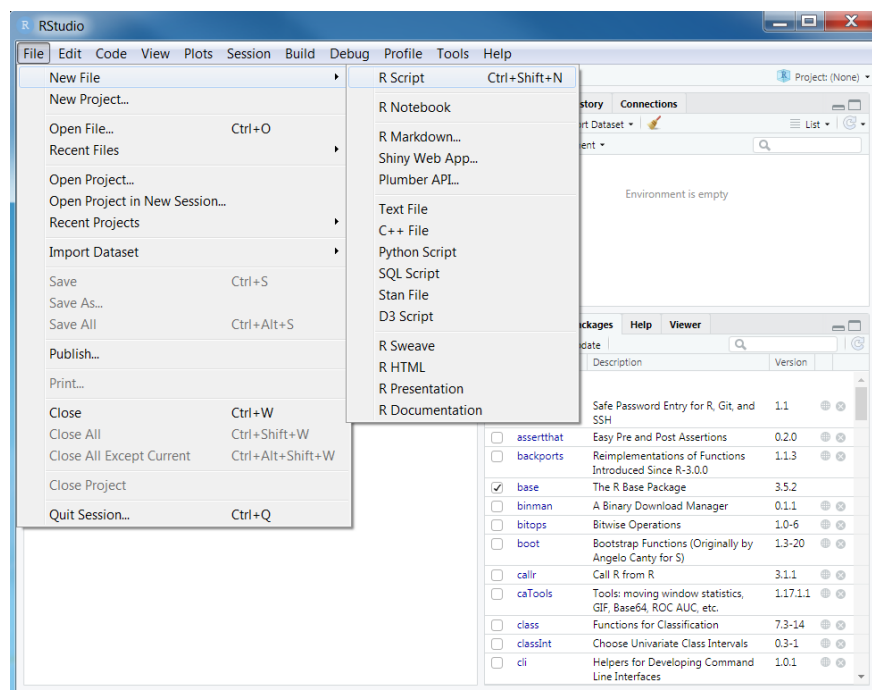
There is a wealth of resources available for learning how to use R. Although not required for this workshop, I would highly recommend that you read [R for Data Science](#) by Garrett Grolemund and Hadley Wickham. **This veritable trove of R goodness is freely available online.** If you spend a week going through this book then you will save months debugging and rerunning incorrect code. I would urge any and all ecologists – especially those working on Masters or PhD degrees – to read this book. I even bought this book as a Christmas present for my sister—and, yes, she was happy to receive it! For intermediate users looking to skill-up, I would recommend the [The Art of R Programming: A Tour of Statistical Software Design](#) by Norman Matloff and [Advanced R](#) by Hadley Wickham. Finally, if you wish to learn more about using R as a geospatial information system (GIS), I would recommend [Geocomputation with R](#) by Robin Lovelace, Jakub Nowosad, and Jannes Muenchow which is also freely available online. I also recommend [Applied Spatial Data Analysis](#) by Roger S. Bivand, Edzer Pebesma, and Virgilio Gómez-Rubio too.

Chapter 3

Data

3.1 Starting out

We will start by opening RStudio. Ideally, you will have already installed both R and Rstudio before the workshop. If you have not done this already, then please see the [Setting up your computer](#) section. **During this workshop, please do not copy and paste code from the workshop manual into RStudio. Instead, please write it out yourself in an R script.** When programming, you will spend a lot of time fixing coding mistakes – that is, debugging your code – so it is best to get used to making mistakes now when you have people here to help you. You can create a new R script by clicking on *File* in the RStudio menu bar and then *R Script*.



After creating a new script, you will notice that a new *Source* panel has appeared. In this new *Source* panel, you can type and edit code. You can run code in the *Source* panel by placing the cursor (i.e. the blinking line) on the desired line of code and pressing **Control + Enter** on your keyboard (or **CMD + Enter** if you are using an Apple computer). You can save the code in the *Source* panel by pressing **Control + s** on your keyboard (or **CMD + s** if you are using an Apple computer).



You can also make notes and write your answers to the workshop questions inside the *Source* panel. When writing notes and answers, add a **#** symbol so that the text following the **#** symbol is treated as a comment and not code. This means that you don't have to worry about highlighting specific parts of the script to avoid errors.

```
# this is a comment and R will ignore this text if you run it
# R will run the code below because it does not start with a # symbol
print("this is not a comment")
```

```
## [1] "this is not a comment"
```

```
# you can also add comments to the same line of R code too
print("this is also not a comment") # but this is a comment
```

```
## [1] "this is also not a comment"
```

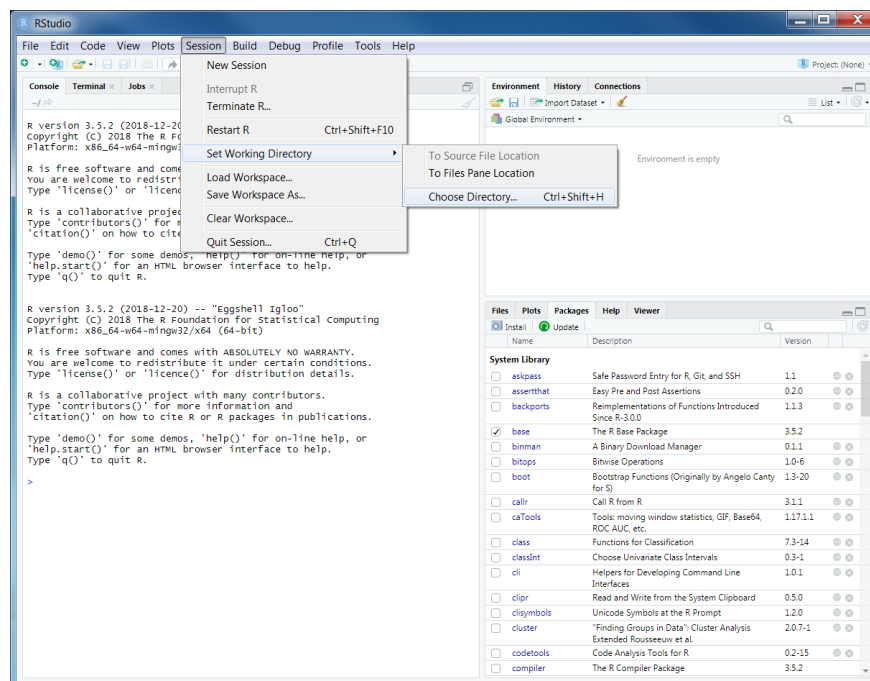
Remember to save your script regularly to ensure that you don't lose anything in the event that RStudio crashes (e.g. using **Control + s** or **CMD + s**)!

3.2 Attaching packages

Now we will setup our R session for the workshop. Specifically, enter the following R code to attach the R packages used in this workshop.

```
# load packages
library(tidyverse)
library(prioritizr)
library(rgdal)
library(raster)
library(rgeos)
library(mapview)
library(units)
library(scales)
library(assertthat)
library(gridExtra)
```

You should have already downloaded the data for the prioritizr module of this workshop. If you have not already done so, you can download it from here: <https://github.com/prioritizr/cibio-workshop/raw/master/data.zip>. After downloading the data, you can unzip the data into a new folder. Next, you will need to set the working directory to this new folder. To achieve this, click on the *Session* button on the RStudio menu bar, then click *Set working directory*, and then *Choose Directory*.



Now navigate to the folder where you unzipped the data and select *Open*. You can verify that you have correctly set the working directory using the following R code. You should

see the output `TRUE` in the *Console* panel.

```
file.exists("data/pu.shp")
```

```
## [1] TRUE
```

3.3 Data import

Now that we have downloaded the dataset, we will need to import it into our R session. Specifically, this data was obtained from the “Introduction to Marxan” course and was originally a subset of a larger spatial prioritization project performed under contract to Australia’s Department of Environment and Water Resources. It contains vector-based planning unit data (`pu.shp`) and the raster-based data describing the spatial distributions of 62 vegetation classes (`vegetation.tif`) in Tasmania, Australia. Please note this dataset is only provided for teaching purposes and should not be used for any real-world conservation planning. We can import the data into our R session using the following code.

```
# import planning unit data
pu_data <- readOGR("data/pu.shp")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/home/travis/build/prioritizr/cibio-workshop/data/pu.shp", layer: "pu"
## with 1130 features
## It has 5 fields
```

```
# format columns in planning unit data
pu_data$locked_in <- as.logical(pu_data$locked_in)
pu_data$locked_out <- as.logical(pu_data$locked_out)

# import vegetation data
veg_data <- stack("data/vegetation.tif")
```

3.4 Planning unit data

The planning unit data contains spatial data describing the geometry for each planning unit and attribute data with information about each planning unit (e.g. cost values). Let’s investigate the `pu_data` object. The attribute data contains 5 columns with contain the following information:

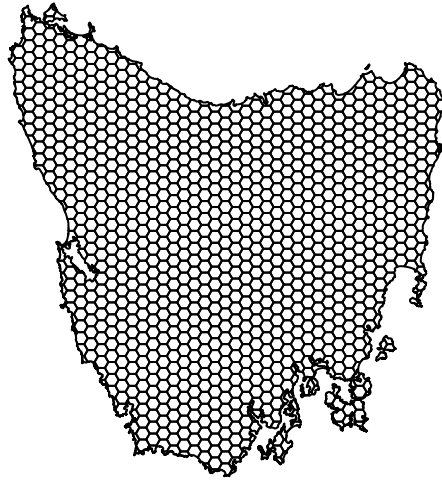
- `id`: unique identifiers for each planning unit

- **cost**: acquisition cost values for each planning unit (millions of Australian dollars).
- **status**: status information for each planning unit (only relevant with Marxan)
- **locked_in**: logical values (i.e. TRUE/FALSE) indicating if planning units are covered by protected areas or not.
- **locked_out**: logical values (i.e. TRUE/FALSE) indicating if planning units cannot be managed as a protected area because they contain too much anthropologically altered land.

```
# print a short summary of the data
print(pu_data)
```

```
## class      : SpatialPolygonsDataFrame
## features   : 1130
## extent     : 1080623, 1399989, -4840595, -4497092 (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps=
## variables  : 5
## names      : id, cost, status, locked_in, locked_out
## min values : 1, 0.192488262910798, 0, 0, 0
## max values : 1130, 61.9272727272727, 2, 1, 1
```

```
# plot the planning unit data
plot(pu_data)
```



```
# plot an interactive map of the planning unit data
mapview(pu_data)
```

```
# print the structure of object
str(pu_data, max.level = 2)
```

```
## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data      :'data.frame':  1130 obs. of  5 variables:
##   ..@ polygons  :List of 1130
##   ..@ plotOrder : int  [1:1130] 217 973 506 645 705 975 253 271 704 889 ...
##   ..@ bbox      : num  [1:2, 1:2] 1080623 -4840595 1399989 -4497092
##   .. ..- attr(*, "dimnames")=List of 2
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

```
# print the class of the object
class(pu_data)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
```



```
## [1] "sp"
```

```
# print the slots of the object
slotNames(pu_data)
```

```
## [1] "data"          "polygons"      "plotOrder"     "bbox"          "proj4string"
```

```
# print the geometry for the 80th planning unit
pu_data@polygons[[80]]
```

```
## An object of class "Polygons"
## Slot "Polygons":
## [[1]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 1289177 -4558185
##
## Slot "area":
## [1] 1060361
##
## Slot "hole":
## [1] FALSE
##
## Slot "ringDir":
## [1] 1
##
## Slot "coords":
##          [,1]      [,2]
## [1,] 1288123 -4558431
## [2,] 1287877 -4558005
## [3,] 1288177 -4558019
## [4,] 1288278 -4558054
## [5,] 1288834 -4558038
## [6,] 1289026 -4557929
## [7,] 1289168 -4557928
## [8,] 1289350 -4557790
## [9,] 1289517 -4557744
## [10,] 1289618 -4557773
## [11,] 1289836 -4557965
## [12,] 1290000 -4557984
## [13,] 1290025 -4557987
## [14,] 1290144 -4558168
## [15,] 1290460 -4558431
## [16,] 1288123 -4558431
```

```
##
##
##
## Slot "plotOrder":
## [1] 1
##
## Slot "labpt":
## [1] 1289177 -4558185
##
## Slot "ID":
## [1] "79"
##
## Slot "area":
## [1] 1060361
```

```
# print the coordinate reference system
print(pu_data@proj4string)
```

```
## CRS arguments:
## +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0
## +ellps=GRS80 +units=m +no_defs
```

```
# print number of planning units (geometries) in the data
nrow(pu_data)
```

```
## [1] 1130
```

```
# print the first six rows in the attribute data
head(pu_data@data)
```

```
##   id      cost status locked_in locked_out
## 0  1 60.24638      0    FALSE      TRUE
## 1  2 19.86301      0    FALSE     FALSE
## 2  3 59.68051      0    FALSE      TRUE
## 3  4 32.41614      0    FALSE     FALSE
## 4  5 26.17706      0    FALSE     FALSE
## 5  6 51.26218      0    FALSE      TRUE
```

```
# print the first six values in the cost column of the attribute data
head(pu_data$cost)
```

```
## [1] 60.24638 19.86301 59.68051 32.41614 26.17706 51.26218
```

```
# print the highest cost value  
max(pu_data$cost)
```

```
## [1] 61.92727
```

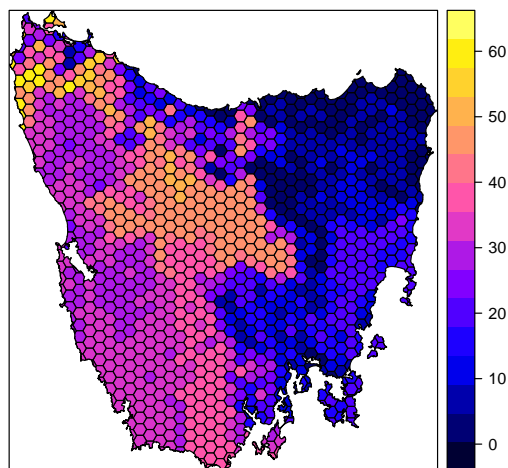
```
# print the smallest cost value  
min(pu_data$cost)
```

```
## [1] 0.1924883
```

```
# print average cost value  
mean(pu_data$cost)
```

```
## [1] 25.13536
```

```
# plot a map of the planning unit cost data  
spplot(pu_data, "cost")
```



```
# plot an interactive map of the planning unit cost data  
mapview(pu_data, zcol = "cost")
```

Now, you can try and answer some questions about the planning unit data.

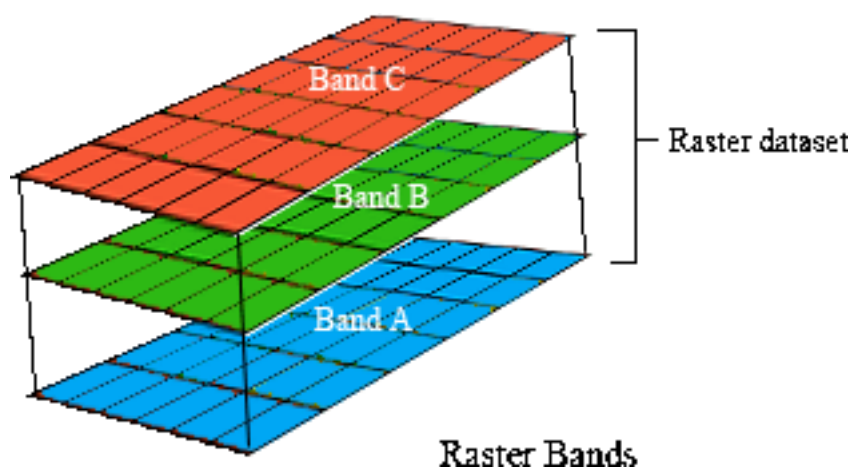


1. How many planning units are in the planning unit data?
2. What is the highest cost value?
3. How many planning units are covered by the protected areas (hint: `sum(x)`)?

4. What is the proportion of the planning units that are covered by the protected areas (hint: `mean(x)`)?
5. How many planning units are dominated by anthropologically altered land (hint: `sum(x)`)?
6. What is the proportion of planning units dominated by anthropologically altered land (hint: `mean(x)`)?
7. Can you verify that all values in the `locked_in` and `locked_out` columns are zero or one (hint: `min(x)` and `max(x)`)?
8. Can you verify that none of the planning units are missing cost values (hint: `all(is.finite(x))`)?
9. Can you verify that none of the planning units have duplicated identifiers? (hint: `sum(duplicated(x))`)?
10. Is there a spatial pattern in the planning unit cost values (hint: use `spplot` to make a map).
11. Is there a spatial pattern in where most planning units are covered by protected areas (hint: use `spplot` to make a map).

3.5 Vegetation data

The vegetation data describes the spatial distribution of 62 vegetation classes in the study area. This data is in a raster format and so the data are organized using a regular spatial grid with square grid cells. In our case, our raster data contains multiple layers (also called “bands”) and each layer corresponds to a spatial grid with exactly the same area and has exactly the same dimensionality (i.e. number of rows, columns, and cells). In this dataset, there are 62 different regular spatial grids layered on top of each other – with each layer corresponding to a different vegetation class – and each of these layers contains a grid with 343 columns, 320 rows, and 109760 cells. Within each layer, each cell corresponds to a 1 by 1 km square. The values associated with each grid cell contain values (i.e. one or zero) indicating the presence or absence of a given vegetation class in the cell.

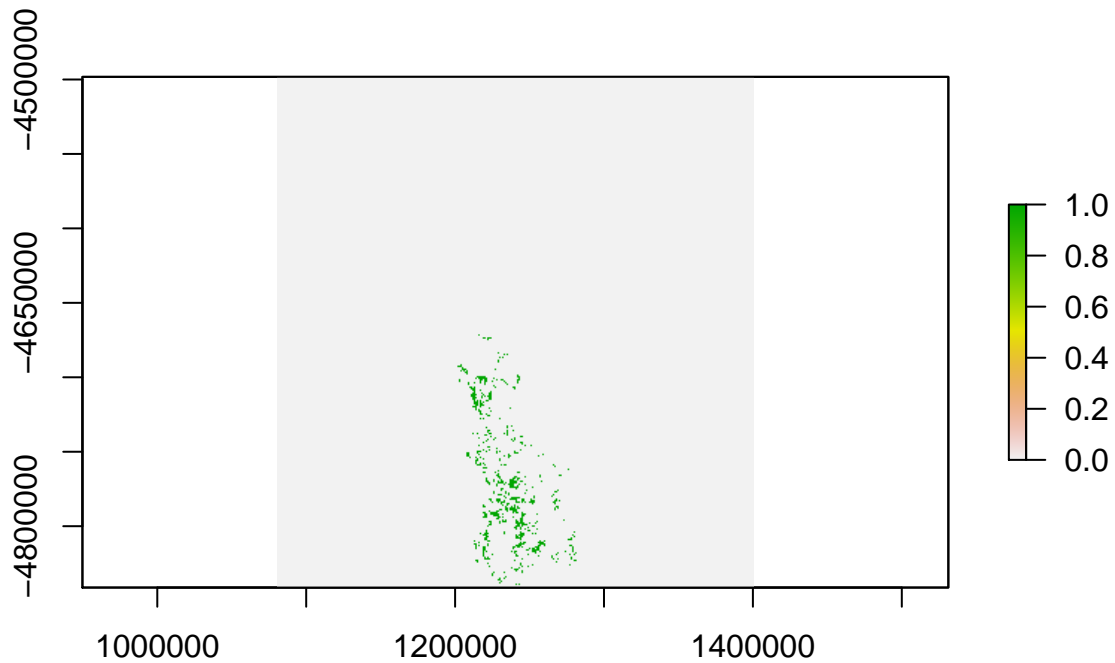


Let's explore the vegetation data.

```
# print a short summary of the data
print(veg_data)
```

```
## class      : RasterStack
## dimensions : 343, 320, 109760, 62  (nrow, ncol, ncell, nlayers)
## resolution : 1000, 1000  (x, y)
## extent     : 1080496, 1400496, -4841217, -4498217  (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps
## names      : vegetation.1, vegetation.2, vegetation.3, vegetation.4, vegetation.5, ve
## min values :          0,          0,          0,          0,          0,
## max values :          1,          1,          1,          1,          1,
```

```
# plot a map of the 36th vegetation class  
plot(veg_data[[36]])
```



```
# plot an interactive map of the 36th vegetation class  
mapview(veg_data[[36]])
```

```
# print number of rows in the data  
nrow(veg_data)
```

```
## [1] 343
```

```
# print number of columns in the data  
ncol(veg_data)
```

```
## [1] 320
```

```
# print number of cells in the data
ncell(veg_data)
```

```
## [1] 109760
```

```
# print number of layers in the data
nlayers(veg_data)
```

```
## [1] 62
```

```
# print resolution on the x-axis
xres(veg_data)
```

```
## [1] 1000
```

```
# print resolution on the y-axis
yres(veg_data)
```

```
## [1] 1000
```

```
# print spatial extent of the grid, i.e. coordinates for corners
extent(veg_data)
```

```
## class      : Extent
## xmin       : 1080496
## xmax       : 1400496
## ymin       : -4841217
## ymax       : -4498217
```

```
# print the coordinate reference system
print(veg_data@crs)
```

```
## CRS arguments:
## +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0
## +ellps=GRS80 +units=m +no_defs
```

```
# print a summary of the first layer in the stack
print(veg_data[[1]])
```

```
## class      : RasterLayer
## band       : 1 (of 62 bands)
```

```
## dimensions : 343, 320, 109760 (nrow, ncol, ncell)
## resolution : 1000, 1000 (x, y)
## extent      : 1080496, 1400496, -4841217, -4498217 (xmin, xmax, ymin, ymax)
## crs         : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps
## source      : /home/travis/build/prioritizr/cibio-workshop/data/vegetation.tif
## names       : vegetation.1
## values      : 0, 1 (min, max)
```

```
# print the value in the 800th cell in the first layer of the stack
print(veg_data[[1]][800])
```

```
##
## 0
```

```
# print the value of the cell located in the 30th row and the 60th column of
# the first layer
print(veg_data[[1]][30, 60])
```

```
##
## 0
```

```
# calculate the sum of all the cell values in the first layer
cellStats(veg_data[[1]], "sum")
```

```
## [1] 36
```

```
# calculate the maximum value of all the cell values in the first layer
cellStats(veg_data[[1]], "max")
```

```
## [1] 1
```

```
# calculate the minimum value of all the cell values in the first layer
cellStats(veg_data[[1]], "min")
```

```
## [1] 0
```

```
# calculate the mean value of all the cell values in the first layer
cellStats(veg_data[[1]], "mean")
```

```
## [1] 0.0003279883
```



```
# calculate the maximum value in each layer
as_tibble(data.frame(max = cellStats(veg_data, "max")))
```

```
## # A tibble: 62 x 1
##       max
##   <dbl>
## 1     1
## 2     1
## 3     1
## 4     1
## 5     1
## 6     1
## 7     1
## 8     1
## 9     1
## 10    1
## # ... with 52 more rows
```

Now, you can try and answer some questions about the vegetation data.



1. What part of the study area is the 51st vegetation class found in (hint: make a map)?
2. What proportion of cells contain the 12th vegetation class?
3. Which vegetation class is present in the greatest number of cells?

Chapter 4

Gap analysis

4.1 Introduction

Before we begin to prioritize areas for protected area establishment, we should first understand how well existing protected areas are conserving our biodiversity features (i.e. native vegetation classes in Tasmania, Australia). This step is critical: we cannot develop plans to improve conservation of biodiversity if we don't understand how well existing policies are currently conserving biodiversity! To achieve this, we can perform a “gap analysis”. A gap analysis involves calculating how well each of our biodiversity features (i.e. vegetation classes in this exercise) are represented (covered) by protected areas. Next, we compare current representation by protected areas of each feature (e.g. 5% of their spatial distribution covered by protected areas) to a target threshold (e.g. 20% of their spatial distribution covered by protected areas). This target threshold denotes the minimum amount (e.g. minimum proportion of spatial distribution) that we need of each feature to be represented in the protected area system. Ideally, targets should be based on an estimate of how much area or habitat is needed for ecosystem function or species persistence. In practice, targets are generally set using simple rules of thumb (e.g. 10% or 20%), policy (17%; <https://www.cbd.int/sp/targets/rationale/target-11>) or standard practices [e.g. setting targets for species based on range-size; Butchart et al., 2015, Rodrigues et al., 2004].

4.2 Feature abundance

Now we will perform some preliminary calculations for the gap analysis. First, we will calculate how much of each vegetation feature occurs inside each planning unit (i.e. the abundance of the features). To achieve this, we will use the `problem` function to create an empty conservation planning problem that only contains the planning unit and biodiversity data. We will then use the `feature_abundances` function to calculate the total amount of each feature in each planning unit.

```
# create prioritizr problem with only the data
p0 <- problem(pu_data, veg_data, cost_column = "cost")

# print empty problem,
# we can see that only the cost and feature data are defined
print(p0)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:           min: 0.19249, max: 61.92727
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:      none
##   targets:        none
##   decisions:      default
##   constraints:    <none>
##   penalties:      <none>
##   portfolio:      default
##   solver:         default
```

```
# calculate amount of each feature in each planning unit
abundance_data <- feature_abundances(p0)

# print abundance data
print(abundance_data)
```

```
## # A tibble: 62 x 3
##   feature          absolute_abundance relative_abundance
##   <chr>              <dbl>              <dbl>
## 1 vegetation.1         33                1
## 2 vegetation.2        173                1
## 3 vegetation.3         24                1
## 4 vegetation.4         31                1
## 5 vegetation.5         23                1
## 6 vegetation.6         22                1
## 7 vegetation.7         15                1
## 8 vegetation.8         45                1
## 9 vegetation.9        384                1
## 10 vegetation.10       14                1
## # ... with 52 more rows
```

```
# note that only the first ten rows are printed,
# this is because the abundance_data object is a tibble (i.e. tbl_df) object
# and not a standard data.frame object
print(class(abundance_data))
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

```
# we can print all of the rows in abundance_data like this
print(abundance_data, n = Inf)
```

```
## # A tibble: 62 x 3
##   feature      absolute_abundance relative_abundance
##   <chr>          <dbl>          <dbl>
## 1 vegetation.1          33              1
## 2 vegetation.2         173              1
## 3 vegetation.3          24              1
## 4 vegetation.4          31              1
## 5 vegetation.5          23              1
## 6 vegetation.6          22              1
## 7 vegetation.7          15              1
## 8 vegetation.8          45              1
## 9 vegetation.9         384              1
## 10 vegetation.10         14              1
## 11 vegetation.11         39              1
## 12 vegetation.12         26              1
## 13 vegetation.13         20              1
## 14 vegetation.14        123              1
## 15 vegetation.15         18              1
## 16 vegetation.16         11              1
## 17 vegetation.17         24              1
## 18 vegetation.18         19              1
## 19 vegetation.19         24              1
## 20 vegetation.20        895              1
## 21 vegetation.21        258              1
## 22 vegetation.22          8              1
## 23 vegetation.23         10              1
## 24 vegetation.24         21              1
## 25 vegetation.25         13              1
## 26 vegetation.26          9              1
## 27 vegetation.27         15              1
## 28 vegetation.28        660              1
## 29 vegetation.29         30              1
## 30 vegetation.30         26              1
```

## 31	vegetation.31	52	1
## 32	vegetation.32	30	1
## 33	vegetation.33	312	1
## 34	vegetation.34	36	1
## 35	vegetation.35	173	1
## 36	vegetation.36	714	1
## 37	vegetation.37	26	1
## 38	vegetation.38	17	1
## 39	vegetation.39	18	1
## 40	vegetation.40	28	1
## 41	vegetation.41	59	1
## 42	vegetation.42	9	1
## 43	vegetation.43	80	1
## 44	vegetation.44	139	1
## 45	vegetation.45	40	1
## 46	vegetation.46	25	1
## 47	vegetation.47	24	1
## 48	vegetation.48	224	1
## 49	vegetation.49	4	1
## 50	vegetation.50	41	1
## 51	vegetation.51	223	1
## 52	vegetation.52	2	1
## 53	vegetation.53	4	1
## 54	vegetation.54	5	1
## 55	vegetation.55	7	1
## 56	vegetation.56	8	1
## 57	vegetation.57	18	1
## 58	vegetation.58	4	1
## 59	vegetation.59	36	1
## 60	vegetation.60	2	1
## 61	vegetation.61	0	NaN
## 62	vegetation.62	1	1

The `abundance_data` object contains three columns. The `feature` column contains the name of each feature (derived from `names(veg_data)`), the `absolute_abundance` column contains the total amount of each feature in all the planning units, and the `relative_abundance` column contains the total amount of each feature in the planning units expressed as a proportion of the total amount in the underlying raster data. Since all the raster cells containing vegetation overlap with the planning units, all of the values in the `relative_abundance` column are equal to one (meaning 100%)—except for the 61st feature which has a value on `NaN` because it does not occur in the study area at all (i.e. all of its raster values are zeros). Now let's add a new column with the feature abundances expressed in aerial units (i.e. km²).

```
# add new column with feature abundances in km^2
abundance_data$absolute_abundance_km2 <-
  (abundance_data$absolute_abundance * prod(res(veg_data))) %>%
  set_units(m^2) %>%
  set_units(km^2)

# print abundance data
print(abundance_data)
```

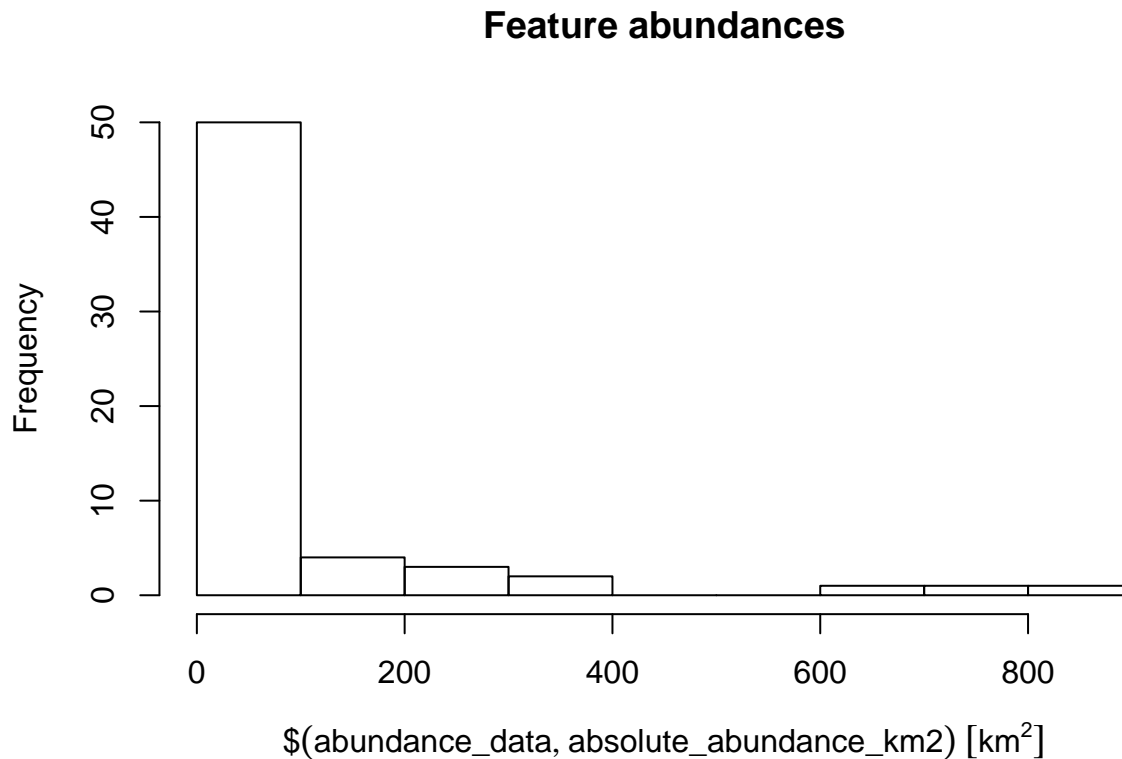
```
## # A tibble: 62 x 4
##   feature          absolute_abundance relative_abundan~ absolute_abundance_k~
##   <chr>              <dbl>              <dbl>              [km^2]
## 1 vegetation.1           33                1                33
## 2 vegetation.2          173                1               173
## 3 vegetation.3           24                1                24
## 4 vegetation.4           31                1                31
## 5 vegetation.5           23                1                23
## 6 vegetation.6           22                1                22
## 7 vegetation.7           15                1                15
## 8 vegetation.8           45                1                45
## 9 vegetation.9          384                1               384
## 10 vegetation.10         14                1                14
## # ... with 52 more rows
```

Now let's explore the abundance data.

```
# calculate the average abundance of the features
mean(abundance_data$absolute_abundance_km2)
```

```
## 86.67742 [km^2]
```

```
# plot histogram of the features' abundances
hist(abundance_data$absolute_abundance_km2, main = "Feature abundances")
```



```
# find the abundance of the feature with the largest abundance
max(abundance_data$absolute_abundance_km2)
```

```
## 895 [km^2]
```

```
# find the name of the feature with the largest abundance
abundance_data$feature[which.max(abundance_data$absolute_abundance_km2)]
```

```
## [1] "vegetation.20"
```

Now, try to answer the following questions.



1. What is the median abundance of the features (hint: `median`)?
2. What is the abundance of the feature with smallest abundance?
3. What is the name of the feature with smallest abundance?
4. What is the total abundance of all features in the planning units summed together?
5. How many features have a total abundance greater than 100 km² (hint: `sum(abundance_values > set_units(threshold_value, km^2))`)?

4.3 Feature representation by protected areas

After calculating the total amount of each feature in the planning units (i.e. the features' abundances), we will now calculate the amount of each feature in the planning units that are covered by protected areas (i.e. feature representation by protected areas). We can complete this task using the `feature_representation` function. This function requires (i) a conservation problem object with the planning unit and biodiversity data and also (ii) an object representing a solution to the problem (i.e. an object in the same format as the planning unit data with zeros and ones indicating if the planning units are contained in the prioritization problem or not).

```
# create column in planning unit data with binary values (zeros and ones)
# indicating if a planning unit is covered by protected areas or not
pu_data$pa_status <- as.numeric(pu_data$locked_in)

# calculate feature representation by protected areas
repr_data <- feature_representation(p0, pu_data[, "pa_status"])

# print feature representation data
print(repr_data)
```

```
## # A tibble: 62 x 3
##   feature      absolute_held relative_held
##   <chr>          <dbl>         <dbl>
## 1 vegetation.1           1         0.0303
## 2 vegetation.2          14         0.0809
## 3 vegetation.3           2         0.0833
## 4 vegetation.4           1         0.0323
## 5 vegetation.5           0           0
## 6 vegetation.6           0           0
## 7 vegetation.7           0           0
## 8 vegetation.8           6         0.133
## 9 vegetation.9          20         0.0521
## 10 vegetation.10         0           0
## # ... with 52 more rows
```

Similar to the abundance data before, the `repr_data` object contains three columns. The `feature` column contains the name of each feature, the `absolute_held` column shows the total amount of each feature held in the solution (i.e. the planning units covered by protected areas), and the `relative_held` column shows the proportion of each feature held in the solution (i.e. the proportion of each feature's spatial distribution held in protected areas). Since the `absolute_held` values correspond to the number of grid cells in the `veg_data` object with overlap with protected areas, let's convert them to aerial-based units (i.e. km²) so we can report them.

```
# add new column with the areas represented in km^2
repr_data$absolute_held_km2 <-
  (repr_data$absolute_held * prod(res(veg_data))) %>%
  set_units(m^2) %>%
  set_units(km^2)

# print representation data
print(repr_data)
```

```
## # A tibble: 62 x 4
##   feature      absolute_held relative_held absolute_held_km2
##   <chr>          <dbl>         <dbl>         [km^2]
## 1 vegetation.1          1      0.0303           1
## 2 vegetation.2         14      0.0809          14
## 3 vegetation.3          2      0.0833           2
## 4 vegetation.4          1      0.0323           1
## 5 vegetation.5          0          0           0
## 6 vegetation.6          0          0           0
## 7 vegetation.7          0          0           0
## 8 vegetation.8          6      0.133            6
## 9 vegetation.9         20      0.0521          20
## 10 vegetation.10        0          0           0
## # ... with 52 more rows
```

Now let's investigate how well the species are represented.



1. What is the average proportion of the features held in protected areas (hint: `mean(x, na.rm = TRUE)`)?
2. What is the average amount of land in km^2 that features are represented by protected areas?
3. What is the name of the feature with the greatest proportionate coverage by protected areas?
4. What is the name of the feature with the greatest aerial coverage by protected areas?
5. Do questions two and three have the same answer? If not, why could this be?
6. Is there a relationship between the total abundance of a feature and how well it is represented by protected areas (hint: `plot(abundances ~ relative_held)`)?
7. Are any features entirely missing from protected areas (hint: `sum(x == 0)`)?
8. If we set a target of 10% coverage by protected areas, how many features fail to meet this target (hint: `sum(relative_held >= target)`)?
9. If we set a target of 20% coverage by protected areas, how many features fail to meet this target?

Chapter 5

Spatial prioritizations

5.1 Introduction

Here we will develop prioritizations to identify priority areas for protected area establishment. Its worth noting that [prioritizr](#), [Marxan](#), and [Zonation](#) are all decision *support* tools. This means that these tools are all designed to *help* you make decisions—they can’t make decisions for you.

5.2 Starting out simple

To start things off, let’s keep things simple. Let’s create a prioritization using the [minimum set problem formulation of the reserve selection problem](#). This problem will have 5% targets for each vegetation class and use the data in the `cost` column to specify acquisition costs. Although we strongly recommend using [Gurobi](#) to solve problems (with `add_gurobi_solver`), we will use the [lpsymphony solver](#) in this workshop since it is much easier to install. The Gurobi solver is the fastest solver that `prioritizr` can use to generate solutions and it is much, much, much faster than the `lpsymphony` solver ([see here for Gurobi installation instructions](#)).

```
# print planning unit data
print(pu_data)
```

```
## class      : SpatialPolygonsDataFrame
## features   : 1130
## extent     : 1080623, 1399989, -4840595, -4497092 (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps=GRS80
## variables  : 6
## names      : id, cost, status, locked_in, locked_out, pa_status
## min values : 1, 0.192488262910798, 0, 0, 0, 0
## max values : 1130, 61.9272727272727, 2, 1, 1, 1
```

```

# make prioritization problem
p1 <- problem(pu_data, veg_data, cost_column = "cost") %>%
  add_min_set_objective() %>%
  add_relative_targets(0.05) %>% # 5% representation targets
  add_binary_decisions() %>%
  add_lpsymphony_solver(verbose = FALSE)

# print problem
print(p1)

```

```

## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:          min: 0.19249, max: 61.92727
##   features:      vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:     Minimum set objective
##   targets:       Relative targets [targets (min: 0.05, max: 0.05)]
##   decisions:     Binary decision
##   constraints:    <none>
##   penalties:     <none>
##   portfolio:     default
##   solver:        Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose

```

```

# solve problem
s1 <- solve(p1)

# print solution,
# the solution_1 column contains the solution values with binary values
# indicating if a planning unit was selected as a (1) priority area or (0) not
print(s1)

```

```

## class      : SpatialPolygonsDataFrame
## features   : 1130
## extent     : 1080623, 1399989, -4840595, -4497092 (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps=
## variables  : 7
## names      : id, cost, status, locked_in, locked_out, pa_status, solu
## min values : 1, 0.192488262910798, 0, 0, 0, 0,
## max values : 1130, 61.9272727272727, 2, 1, 1, 1,

```

```

# calculate number of planning units selected in the prioritization
sum(s1$solution_1)

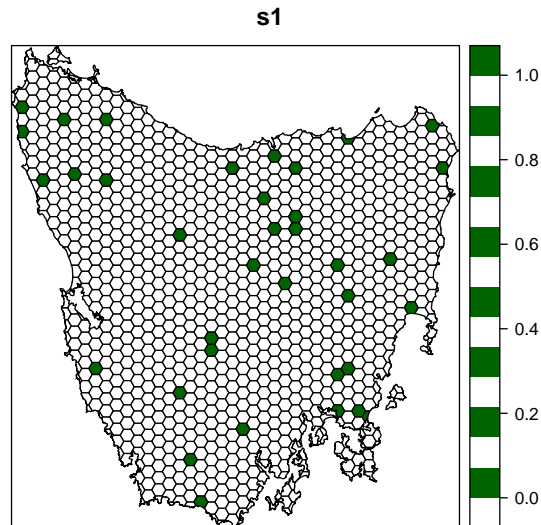
```

```
## [1] 36
```

```
# calculate total cost of the prioritization
sum(s1$solution_1 * s1$cost)
```

```
## [1] 806.2393
```

```
# plot solution
spplot(s1, "solution_1", col.regions = c("white", "darkgreen"), main = "s1")
```



Now let's examine the solution.



1. How many planning units were selected in the prioritization? What proportion of planning units were selected in the prioritization?
2. Is there a pattern in the spatial distribution of the priority areas?
3. Can you verify that all of the targets were met in the prioritization (hint: `feature_representation(p1, s1[, "solution_1"])`)?
4. What are limitations of this prioritization?

5.3 Adding complexity

Our first prioritization suffers many limitations, so let's add additional constraints to the problem to make it more useful. First, let's lock in planning units that are already inside protected areas.

```
# make prioritization problem
p2 <- problem(pu_data, veg_data, cost_column = "cost") %>%
```

```

add_min_set_objective() %>%
add_relative_targets(0.05) %>%
add_locked_in_constraints("locked_in") %>%
add_binary_decisions() %>%
add_lpsymphony_solver(verbose = FALSE)

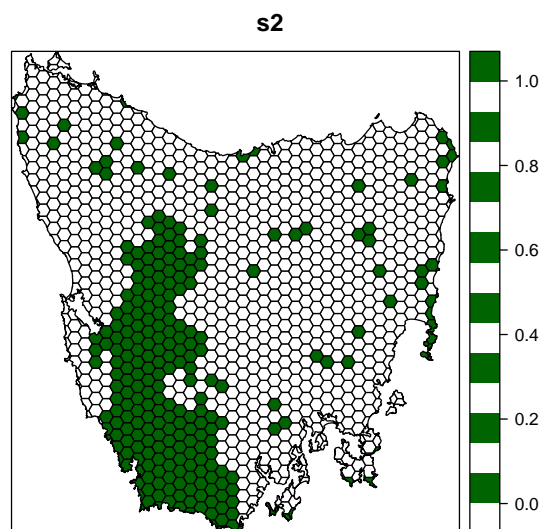
# print problem
print(p2)

## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:           min: 0.19249, max: 61.92727
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:      Minimum set objective
##   targets:        Relative targets [targets (min: 0.05, max: 0.05)]
##   decisions:      Binary decision
##   constraints:    <Locked in planning units [257 locked units]>
##   penalties:     <none>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose

# solve problem
s2 <- solve(p2)

# plot solution
spplot(s2, "solution_1", col.regions = c("white", "darkgreen"), main = "s2")

```



Let's pretend we talked to an expert on the vegetation communities in our study system and

they recommended that a 20% target was needed. So, now let's set the targets to 20% of their total distribution in the study area.

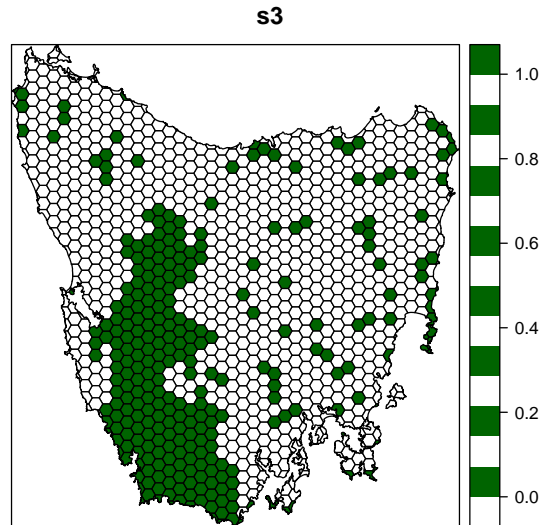
```
# make prioritization problem
p3 <- problem(pu_data, veg_data, cost_column = "cost") %>%
  add_min_set_objective() %>%
  add_relative_targets(0.2) %>%
  add_locked_in_constraints("locked_in") %>%
  add_binary_decisions() %>%
  add_lpsymphony_solver(verbose = FALSE)

# print problem
print(p3)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:           min: 0.19249, max: 61.92727
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:      Minimum set objective
##   targets:        Relative targets [targets (min: 0.2, max: 0.2)]
##   decisions:      Binary decision
##   constraints:    <Locked in planning units [257 locked units]>
##   penalties:      <none>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```
# solve problem
s3 <- solve(p3)

# plot solution
spplot(s3, "solution_1", col.regions = c("white", "darkgreen"), main = "s3")
```



Next, let's lock out highly degraded areas.

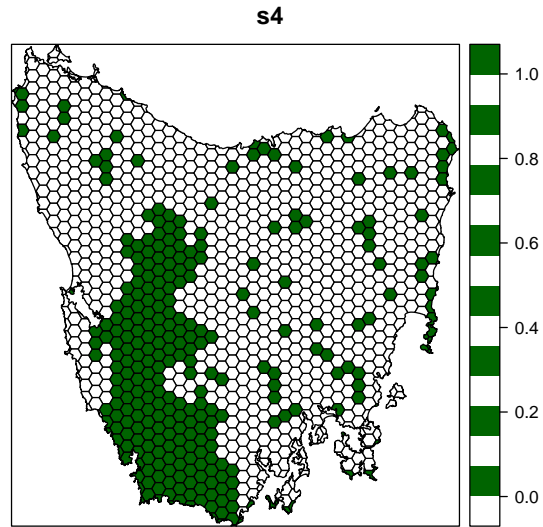
```
# make prioritization problem
p4 <- problem(pu_data, veg_data, cost_column = "cost") %>%
  add_min_set_objective() %>%
  add_relative_targets(0.2) %>%
  add_locked_in_constraints("locked_in") %>%
  add_locked_out_constraints("locked_out") %>%
  add_binary_decisions() %>%
  add_lpsymphony_solver(verbose = FALSE)
```

```
# print problem
print(p4)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:           min: 0.19249, max: 61.92727
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:      Minimum set objective
##   targets:        Relative targets [targets (min: 0.2, max: 0.2)]
##   decisions:      Binary decision
##   constraints:    <Locked in planning units [257 locked units]
##                  Locked out planning units [51 locked units]>
##   penalties:      <none>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```



```
# solve problem  
s4 <- solve(p4)  
  
# plot solution  
spplot(s4, "solution_1", col.regions = c("white", "darkgreen"), main = "s4")
```



Now, let's compare the solutions.



1. What is the cost of the planning units selected in **s2**, **s3**, and **s4**?
2. How many planning units are in **s2**, **s3**, and **s4**?
3. Do the solutions with more planning units have a greater cost? Why or why not?
4. Why does the first solution (**s1**) cost less than the second solution with protected areas locked into the solution (**s2**)?
5. Why does the third solution (**s3**) cost less than the fourth solution solution with highly degraded areas locked out (**s4**)?
6. Since planning units covered by existing protected areas have already been purchased, what is the cost for expanding the protected area system based on on the fourth prioritization (**s4**) (hint: total cost minus the cost of locked in planning units)?
7. What happens if you specify targets that exceed the total amount of vegetation in the study area and try to solve the problem? You can do this by modifying the code to make **p4** with `add_absolute_targets(1000)` instead of `add_relative_targets(0.2)` and generating a new solution.

5.4 Penalizing fragmentation

Plans for protected area systems should facilitate gene flow and dispersal between individual reserves in the system. However, the prioritizations we have made so far have been highly fragmented. Similar to the Marxan decision support tool, we can add penalties to our conservation planning problem to penalize fragmentation (i.e. total exposed boundary length) and we also need to set a useful penalty value when adding such penalties (akin to Marxan's boundary length multiplier value; BLM). If we set our penalty value too low, then we end up with a solution that is identical to the solution with no added penalties. If we set our penalty value too high, then `prioritizr` will take a long time to solve the problem and we will end up with a solution that contains lots of extra planning units that are not needed (since the penalty value is so high that minimizing fragmentation is more important than cost). As a rule of thumb, we generally want penalty values between 0.00001 and 0.01 but finding a useful penalty value requires calibration. The "correct" penalty value depends on the size of the planning units, the main objective values (e.g. cost values), and the effect of fragmentation on biodiversity persistence. Let's create a new problem that is similar to our previous problem (**p4**) – except that it contains boundary length penalties and a slightly higher optimality gap to reduce runtime (default is 0.1) – and solve it.

```

# make prioritization problem
p5 <- problem(pu_data, veg_data, cost_column = "cost") %>%
  add_min_set_objective() %>%
  add_boundary_penalties(penalty = 0.0005) %>%
  add_relative_targets(0.2) %>%
  add_locked_in_constraints("locked_in") %>%
  add_locked_out_constraints("locked_out") %>%
  add_binary_decisions() %>%
  add_lpsymphony_solver(verbose = FALSE, gap = 1)

# print problem
print(p5)

```

```

## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:          min: 0.19249, max: 61.92727
##   features:      vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:     Minimum set objective
##   targets:       Relative targets [targets (min: 0.2, max: 0.2)]
##   decisions:     Binary decision
##   constraints:   <Locked in planning units [257 locked units]
##                 Locked out planning units [51 locked units]>
##   penalties:     <Boundary penalties [edge factor (min: 0.5, max: 0.5), penalty (5e-
##   portfolio:     default
##   solver:        Lpsymphony [first_feasible (0), gap (1), time_limit (-1), verbose (

```

```

# solve problem,
# note this will take around 30 seconds
s5 <- solve(p5)

# print solution
print(s5)

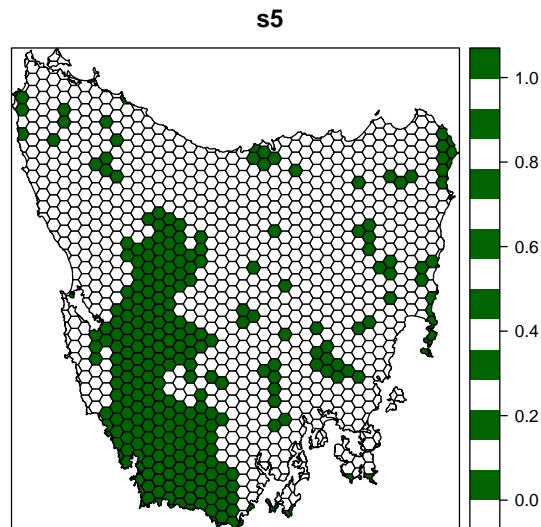
```

```

## class      : SpatialPolygonsDataFrame
## features   : 1130
## extent     : 1080623, 1399989, -4840595, -4497092 (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps=
## variables  : 7
## names      : id, cost, status, locked_in, locked_out, pa_status, solu
## min values : 1, 0.192488262910798, 0, 0, 0, 0,
## max values : 1130, 61.9272727272727, 2, 1, 1, 1,

```

```
# plot solution
spplot(s5, "solution_1", col.regions = c("white", "darkgreen"), main = "s5")
```



Now let's compare the solutions to the problems with (**s5**) and without (**s4**) the boundary length penalties.

?

1. What is the cost the fourth (**s4**) and fifth (**s5**) solutions? Why does the fifth solution (**s5**) cost more than the fourth (**s4**) solution?
2. Try setting the penalty value to 0.000000001 (i.e. $1e-9$) instead of 0.001. What is the cost of the solution now? Is it different from the fourth solution (**s4**) (hint: try plotting the solutions to visualize them)? Is this a useful penalty value? Why?
3. Try setting the penalty value to 0.5. What is the cost of the solution now? Is it different from the fourth solution (**s4**) (hint: try plotting the solutions to visualize them)? Is this a useful penalty value? Why?

Budget limited prioritizations

In the real-world, the funding available for conservation is often very limited. As a consequence, decision makers often need prioritizations where the total cost of priority areas does not exceed a budget. In our fourth prioritization (`s4`), we found that we would need to spend an additional \$909 million AUD to ensure that each vegetation community is adequately represented in the protected area system. But what if the funds available for protecting new land were limited to \$100 million AUD? In this case, we need a “budget limited prioritization”. Budget limited prioritizations aim to maximize some measure of conservation benefit subject to a budget (e.g. [number of species with at least one occurrence in the protected area system](#), or [phylogenetic diversity](#)). Let’s create a budget limited prioritization that aims to adequately represent as many targets as possible whilst remaining within a budget.

```
# set the funds for additional land acquisition
funds <- 100

# calculate the total budget for the prioritization
budget <- funds + sum(s4$cost * s4$locked_in)
print(budget)
```

```
## [1] 8575.56
```

```
# make prioritization problem
p6 <- problem(pu_data, veg_data, cost_column = "cost") %>%
  add_max_features_objective(budget) %>%
  add_relative_targets(0.2) %>%
  add_locked_in_constraints("locked_in") %>%
  add_locked_out_constraints("locked_out") %>%
  add_binary_decisions() %>%
  add_lpsymphony_solver(verbose = FALSE)

# print problem
print(p6)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:          min: 0.19249, max: 61.92727
##   features:      vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:     Maximum representation objective [budget (8575.56009869836)]
##   targets:       Relative targets [targets (min: 0.2, max: 0.2)]
##   decisions:     Binary decision
##   constraints:   <Locked in planning units [257 locked units]
##                 Locked out planning units [51 locked units]>
```

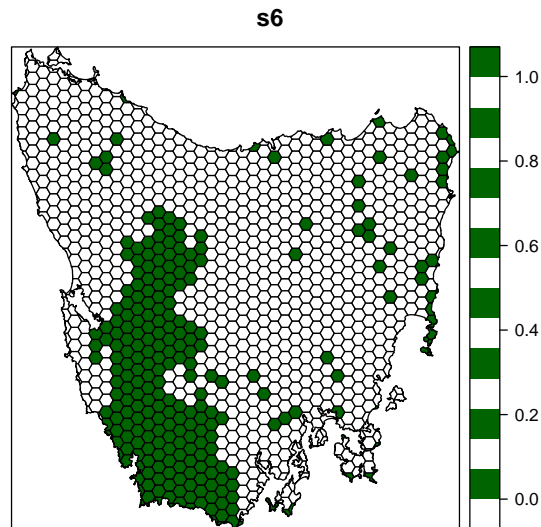
```
## penalties:      <none>
## portfolio:      default
## solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```
# solve problem
```

```
s6 <- solve(p6)
```

```
# plot solution
```

```
spplot(s6, "solution_1", col.regions = c("white", "darkgreen"), main = "s6")
```



```
# calculate feature representation
```

```
r6 <- feature_representation(p6, s6[, "solution_1"])
```

```
# calculate number of features with targets met
```

```
sum(r6$relative_held > 0.2, na.rm = TRUE)
```

```
## [1] 25
```

```
# find out which features have their targets met when we add weights
```

```
print(r6$feature[r6$relative_held > 0.2])
```

```
## [1] "vegetation.1" "vegetation.2" "vegetation.3" "vegetation.4"
## [5] "vegetation.5" "vegetation.6" "vegetation.11" "vegetation.12"
## [9] "vegetation.14" "vegetation.15" "vegetation.17" "vegetation.25"
## [13] "vegetation.28" "vegetation.29" "vegetation.30" "vegetation.32"
## [17] "vegetation.33" "vegetation.34" "vegetation.35" "vegetation.36"
## [21] "vegetation.37" "vegetation.38" "vegetation.39" "vegetation.40"
## [25] "vegetation.45" NA
```

We can also add weights to specify that it is more important to meet the targets for certain features and less important for other features. A common approach for weighting features is to assign a greater importance to features with smaller spatial distributions. The rationale behind this weighting method is that features with smaller spatial distributions are at greater risk of extinction. So, let's calculate some weights for our vegetation communities and see how weighting the features changes our prioritization.

```
# calculate weights as the log inverse number of grid cells that each vegetation
# class occupies, rescaled between 1 and 100
wts <- 1 / cellStats(veg_data, "sum")
wts <- rescale(wts, to = c(1, 10))
```

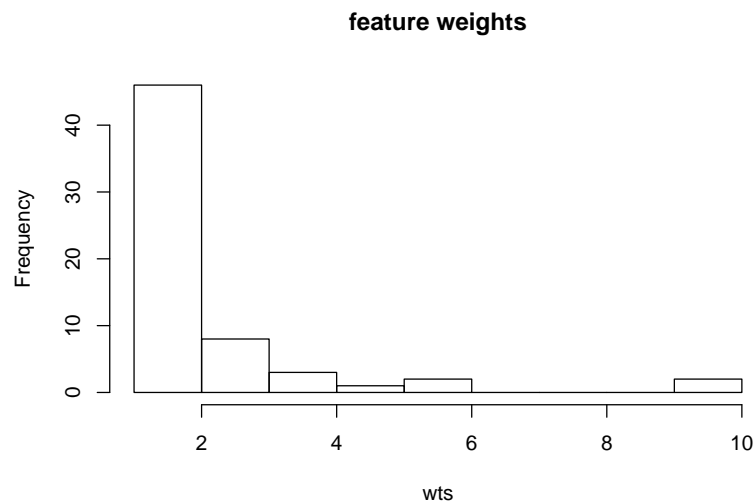
```
# print the name of the feature with smallest weight
names(veg_data)[which.min(wts)]
```

```
## [1] "vegetation.20"
```

```
# print the name of the feature with greatest weight
names(veg_data)[which.max(wts)]
```

```
## [1] "vegetation.52"
```

```
# plot histogram of weights
hist(wts, main = "feature weights")
```



```
# make prioritization problem with weights
p7 <- problem(pu_data, veg_data, cost_column = "cost") %>%
```

```

add_max_features_objective(budget) %>%
add_relative_targets(0.2) %>%
add_feature_weights(wts) %>%
add_locked_in_constraints("locked_in") %>%
add_locked_out_constraints("locked_out") %>%
add_binary_decisions() %>%
add_lpsymphony_solver(verbose = FALSE)

# print problem
print(p7)

```

```

## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:           min: 0.19249, max: 61.92727
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:      Maximum representation objective [budget (8575.56009869836)]
##   targets:        Relative targets [targets (min: 0.2, max: 0.2)]
##   decisions:      Binary decision
##   constraints:    <Locked in planning units [257 locked units]
##                  Locked out planning units [51 locked units]>
##   penalties:      <Feature weights [weights (min: 1, max: 10)]>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose

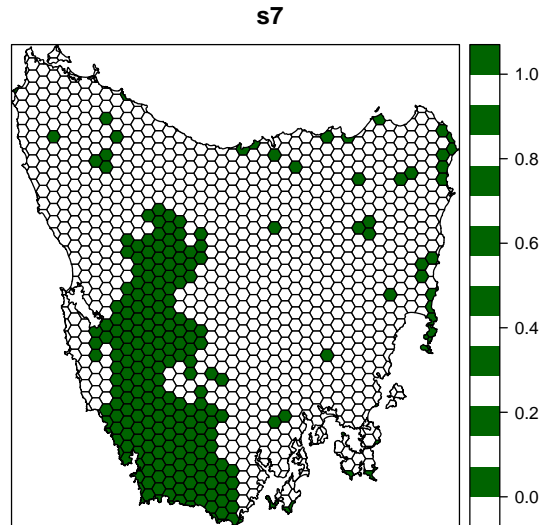
```

```

# solve problem
s7 <- solve(p7)

# plot solution
spplot(s7, "solution_1", col.regions = c("white", "darkgreen"), main = "s7")

```

```
# calculate feature representation
r7 <- feature_representation(p7, s7[, "solution_1"])
```

```
# calculate number of features with targets met
sum(r7$relative_held > 0.2, na.rm = TRUE)
```

```
## [1] 25
```

```
# find out which features have their targets met when we add weights
print(r7$feature[r7$relative_held > 0.2])
```

```
## [1] "vegetation.1" "vegetation.2" "vegetation.4" "vegetation.5"
## [5] "vegetation.6" "vegetation.8" "vegetation.11" "vegetation.28"
## [9] "vegetation.29" "vegetation.30" "vegetation.32" "vegetation.33"
## [13] "vegetation.34" "vegetation.35" "vegetation.36" "vegetation.37"
## [17] "vegetation.38" "vegetation.39" "vegetation.40" "vegetation.45"
## [21] "vegetation.49" "vegetation.50" "vegetation.52" "vegetation.53"
## [25] "vegetation.55" NA
```



1. What is the name of the feature with the smallest weight?
2. What is the cost of the sixth (s6) and seventh (s7) solutions?
3. Does there seem to be a big difference in which planning units were selected in the sixth (s6) and seventh (s7) solutions?
4. Is there a difference between which features are adequately represented in the sixth (s6) and seventh (s7) solutions?

5.6 Solution portfolios

In systematic conservation planning, only rarely do we have data on all of the stakeholder preferences and biodiversity features that we are interested in conserving. As a consequence, it is often useful to generate a portfolio of near optimal solutions to present to decision makers to guide the reserve selection process. Generally we would want many solutions in our portfolio (e.g. 1000) to ensure that our portfolio contains a range of highly different solutions, but here we will generate a portfolio containing just six near-optimal solutions so the code doesn't take too long to run. We will also increase the optimality gap to obtain solutions that are more suboptimal than earlier (the default gap value is 0.1).

```
# make problem with a shuffle portfolio
p8 <- problem(pu_data, veg_data, cost_column = "cost") %>%
  add_max_features_objective(budget) %>%
  add_relative_targets(0.2) %>%
  add_feature_weights(wts) %>%
  add_binary_decisions() %>%
  add_shuffle_portfolio(number_solutions = 6,
                        remove_duplicates = FALSE) %>%
  add_lpsymphony_solver(verbose = TRUE, gap = 10)
```

```
# print problem
print(p8)
```

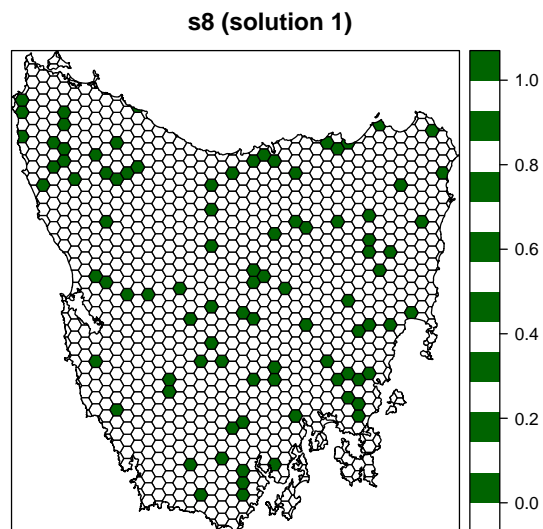
```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (1130 units)
##   cost:           min: 0.19249, max: 61.92727
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (62 features)
##   objective:      Maximum representation objective [budget (8575.56009869836)]
##   targets:        Relative targets [targets (min: 0.2, max: 0.2)]
##   decisions:      Binary decision
##   constraints:    <none>
##   penalties:      <Feature weights [weights (min: 1, max: 10)]>
##   portfolio:      Shuffle portfolio [number_solutions (6), remove_duplicates (0), thr
##   solver:         Lpsymphony [first_feasible (0), gap (10), time_limit (-1), verbose
```

```
# solve problem
# note that this will contain six solutions since we added a portfolio
s8 <- solve(p8)

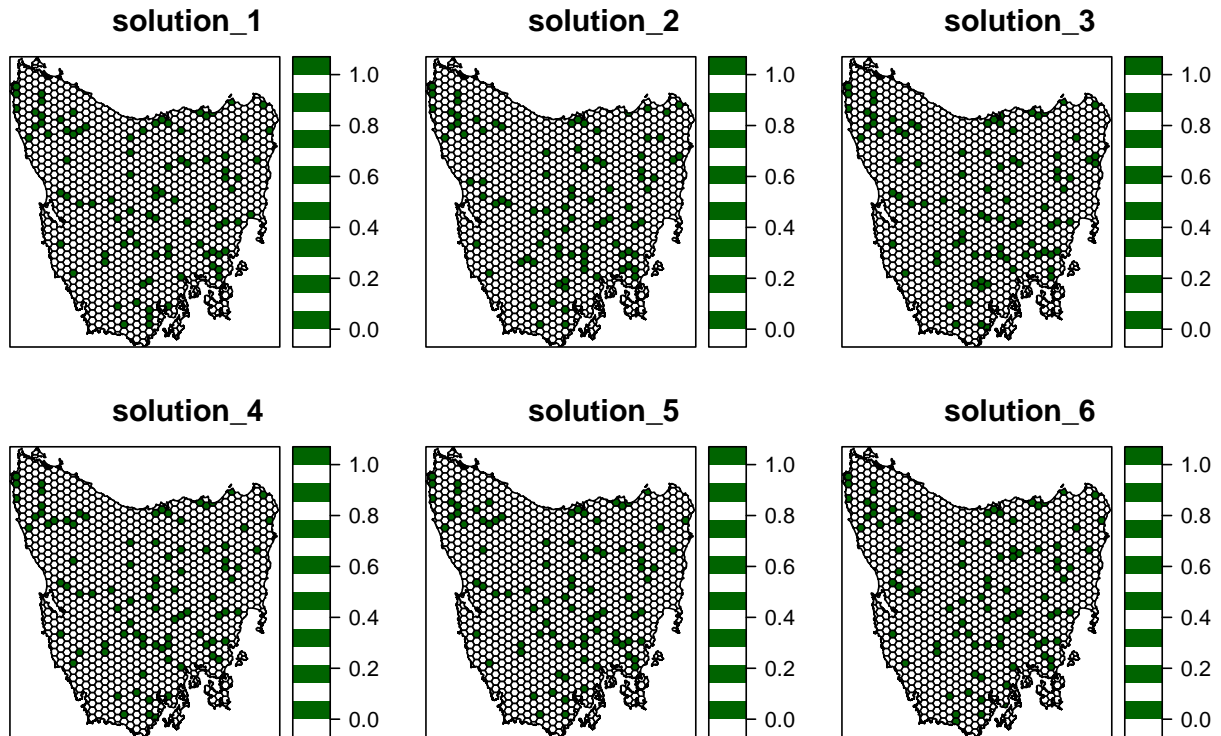
# print solution
print(s8)
```

```
## class      : SpatialPolygonsDataFrame
## features   : 1130
## extent     : 1080623, 1399989, -4840595, -4497092 (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps=GRS80
## variables  : 12
## names      : id, cost, status, locked_in, locked_out, pa_status, solu
## min values : 1, 0.192488262910798, 0, 0, 0, 0,
## max values : 1130, 61.9272727272727, 2, 1, 1, 1,
```

```
# plot first solution
spplot(s8, "solution_1", col.regions = c("white", "darkgreen"),
       main = "s8 (solution 1)")
```



```
# plot all solutions
s8_plots <- lapply(paste0("solution_", seq_len(6)), function(x) {
  spplot(s8, x, main = x, col.regions = c("white", "darkgreen"))
})
do.call(grid.arrange, append(s8_plots, list(ncol = 3)))
```



```
# calculate the cost of the first solution
sum(s8$solution_1 * s8$cost)
```

```
## [1] 2194.232
```

```
# calculate the cost of the second solution
sum(s8$solution_2 * s8$cost)
```

```
## [1] 2142.162
```

```
# calculate the proportion of planning units with the same solution values
# in the first and second solutions
mean(s8$solution_1 == s8$solution_2)
```

```
## [1] 0.9584071
```



1. What is the cost of each of the six solutions in portfolio? Are their costs very different?
2. Are the solutions in the portfolio very different?
3. What could we do to obtain a portfolio with more different solutions?

Chapter 6

Answers

This chapter contains the answers to the questions presented in the earlier chapters. The answers are provided here so you can check if your answers are correct.

6.1 Data

6.1.1 Planning unit data



1. `nrow(pu_data)`
2. `max(pu_data$cost)`
3. `sum(pu_data$locked_in)`
4. `mean(pu_data$locked_in)`
5. `sum(pu_data$locked_out)`
6. `mean(pu_data$locked_out)`
7. `assert_that(min(c(pu_data$locked_in, pu_data$locked_out)) == 0)`

`assert_that(max(c(pu_data$locked_in, pu_data$locked_out)) == 1)`
8. `all(is.finite(pu_data$cost))`
9. `assert_that(sum(duplicated(pu_data$id)) == 0)`
10. Yes, the eastern side of Tasmania is generally much cheaper than the western side.
11. Yes, most planning units covered by protected areas are located in the south-western side of Tasmania.

6.1.2 Vegetation data



1. Central-north Tasmania
2. `cellStats(veg_data[[12]], "mean")`
3. `names(veg_data)[which.max(cellStats(veg_data, "sum"))]`

6.2 Gap analysis

6.2.1 Feature abundance



1. `median(abundance_data$absolute_abundance_km2)`
2. `min(abundance_data$absolute_abundance_km2)`
3. `abundance_data$feature[which.min(abundance_data$absolute_abundance_km2)]`
4. `sum(abundance_data$absolute_abundance_km2)`
5. `sum(abundance_data$absolute_abundance_km2 > set_units(100, km^2))`

6.2.2 Feature representation by protected areas



1. `mean(repr_data$relative_held, na.rm = TRUE)`
2. `mean(repr_data$absolute_held_km2, na.rm = TRUE)`
3. `repr_data$feature[which.max(repr_data$relative_held)]`
4. `repr_data$feature[which.max(repr_data$absolute_held)]`
5. No, just because a vegetation class is widespread does not necessarily mean that it has the greatest overlap with protected areas. In fact, due to biases in the establishment of protected areas this can often be the case.
6. Yes, the largest protected areas tend to have the great representation (broadly speaking).
See `plot(x = abundance_data$absolute_abundance, y = repr_data$relative_held)`
7. `sum(repr_data$absolute_held < 1e-10)` (floating point errors)
8. `sum(repr_data$relative_held > 0.1)`
9. `sum(repr_data$relative_held > 0.2)`

6.3 Spatial prioritizations

6.3.1 Starting out simple



1. `sum(s1$solution_1)`
`mean(s1$solution_1)`
2. Yes, the planning units are generally spread out across most of the study area and they are not biased towards specific areas.
3. `all(feature_representation(p1, s1[, "solution_1"])$relative_held >= 0.2)`
4. This prioritization (i) does not account for existing protected areas, (ii) does not account for the spatial fragmentation of priority areas, or (iii) is not likely conserve enough habitat for each vegetation type (i.e. 5% is pretty small).

6.3.2 Adding complexity



1. `sum(s2$cost * s2$solution_1)`
`sum(s3$cost * s3$solution_1)`
`sum(s4$cost * s4$solution_1)`
2. `sum(s2$solution_1)`
`sum(s3$solution_1)`
`sum(s4$solution_1)`
3. No, just because a solution a solution has more planning units does not mean that it will cost less.
4. This is because the planning units covered by existing protected areas have a non-zero cost and locking in these planning units introduces inefficiencies into the solution. This is very common in real-world conservation prioritizations because existing protected areas are often in places that do little to benefit biodiversity [Fuller et al., 2010].
5. This is because some of the planning units that are highly degraded – based on just the planning unit costs and vegetation data – provide cost-efficient opportunities for meeting the targets and excluding them from the reserve selection process means that other more costly planning units are needed to meet the targets.
6. `sum(s4$cost * s4$solution_1) - sum(s4$cost * s4$locked_in)`
7. We get an error message stating the the problem is infeasible because there is no valid solution—even if we selected all the planning units the study area we would still not meet the targets.

6.3.3 Penalizing fragmentation



1. The cost of the fourth solution is `sum(s4$solution_1 * s4$cost)` and the cost of the fifth solution is `sum(s5$solution_1 * s5$cost)`. The fifth solution (`s5`) costs more than the fourth solution (`s4`) because we have added penalties to the conservation planning problem to indicate that we are willing to accept a slightly more costly solution if it means that we can reduce fragmentation.
2. The solution is now nearly identical to the fourth solution (`s4`) and so has nearly the same cost. This penalty value is too low and is not useful because it does not reduce the fragmentation in our solution.
3. The solution now contains a lot of extra planning units that are not needed to meet our targets. In fact, nearly every planning unit in the study is now selected. This penalty value is too high and is not useful.

6.3.4 Budget limited prioritizations



1. `names(veg_data)[which.min(wts)]`
2. `sum(s6$cost * s6$solution_1)`
`sum(s7$cost * s7$solution_1)`
3. No, the sixth (`s6`) and seventh (`s7`) solutions both share many of the same selected planning units and there does not appear to be specific difference in the spatial location of the planning units which they do not share.
4. Yes. Although both of these solutions are meeting the targets for `r6$feature[r6$relative_held > 0.2 & r7$relative_held > 0.2]` vegetation communities, the sixth (`s6`) is also adequately representing `r6$feature[r6$relative_held > 0.2 & !r7$relative_held > 0.2]` additional community and the seventh (`s7`) solution is adequately representing `r6$feature[!r6$relative_held > 0.2 & r7$relative_held > 0.2]` different additional community.

6.3.5 Solution portfolios



1. No the cost are very similar.
`sum(s8$solution_1 * s8$cost)`
`sum(s8$solution_2 * s8$cost)`
`sum(s8$solution_3 * s8$cost)`
`sum(s8$solution_4 * s8$cost)`
`sum(s8$solution_5 * s8$cost)`
`sum(s8$solution_6 * s8$cost)`

2. No the status of the planning units are very similar in the all of the solutions in the portfolio.

```
mean((s8$solution_1 == s8$solution_2) & (s8$solution_1 == s8$solution_3)
      & (s8$solution_1 == s8$solution_4) & (s8$solution_1 == s8$solution_5)
      & (s8$solution_1 == s8$solution_6))
```

3. We should increase the number of the solutions in the portfolio.

Chapter 7

Acknowledgements

Many thanks to [Icons8](#) for providing the icons used in this manual and to Yihui Xie for developing the [bookdown R package](#) that underpins this manual.

Chapter 8

Session information

```
# print session information
sessionInfo()
```

```
## R version 3.6.1 (2017-01-27)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.6 LTS
##
## Matrix products: default
## BLAS: /home/travis/R-bin/lib/R/lib/libRblas.so
## LAPACK: /home/travis/R-bin/lib/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8      LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8  LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8     LC_NAME=C
##  [9] LC_ADDRESS=C             LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
##  [1] gridExtra_2.3      assertthat_0.2.1  scales_1.0.0      units_0.6-4
##  [5] mapview_2.7.0      rgeos_0.5-2       rgdal_1.4-6       prioritizr_4.1.4
##  [9] proto_1.0.0        raster_3.0-7      sp_1.3-1          forcats_0.4.0
## [13] stringr_1.4.0      dplyr_0.8.3       purrr_0.3.2       readr_1.3.1
## [17] tidyr_1.0.0        tibble_2.1.3      ggplot2_3.2.1     tidyverse_1.2.1
##
```

```
## loaded via a namespace (and not attached):
## [1] nlme_3.1-140      sf_0.8-0          satellite_1.0.1
## [4] lubridate_1.7.4  webshot_0.5.1     httr_1.4.1
## [7] tools_3.6.1      backports_1.1.5   utf8_1.1.4
## [10] R6_2.4.0         KernSmooth_2.23-15 DBI_1.0.0
## [13] lazyeval_0.2.2   colorspace_1.4-1  withr_2.1.2
## [16] tidyselect_0.2.5 leaflet_2.0.2     compiler_3.6.1
## [19] cli_1.1.0        rvest_0.3.4       xml2_1.2.2
## [22] bookdown_0.14.1  classInt_0.4-1    digest_0.6.21
## [25] rmarkdown_1.16   base64enc_0.1-3   pkgconfig_2.0.3
## [28] htmltools_0.4.0  lpsymphony_1.12.0 htmlwidgets_1.5
## [31] rlang_0.4.0      readxl_1.3.1      rstudioapi_0.10
## [34] shiny_1.3.2      generics_0.0.2    jsonlite_1.6
## [37] crosstalk_1.0.0  magrittr_1.5      Matrix_1.2-17
## [40] Rcpp_1.0.2       munsell_0.5.0     fansi_0.4.0
## [43] lifecycle_0.1.0  stringi_1.4.3     yaml_2.2.0
## [46] plyr_1.8.4       grid_3.6.1        parallel_3.6.1
## [49] promises_1.1.0   crayon_1.3.4      lattice_0.20-38
## [52] haven_2.1.1      hms_0.5.1         zeallot_0.1.0
## [55] knitr_1.25       pillar_1.4.2      uuid_0.1-2
## [58] velox_0.2.0      codetools_0.2-16  stats4_3.6.1
## [61] glue_1.3.1       evaluate_0.14     modelr_0.1.5
## [64] png_0.1-7        vctrs_0.2.0       httpuv_1.5.2
## [67] cellranger_1.1.0 gtable_0.3.0      xfun_0.10
## [70] mime_0.7         xtable_1.8-4      broom_0.5.2
## [73] e1071_1.7-2      later_1.0.0       class_7.3-15
## [76] viridisLite_0.3.0
```

Bibliography

Stuart H.M. Butchart, Martin Clarke, Robert J. Smith, Rachel E. Sykes, Jörn P.W. Scharlemann, Mike Harfoot, Graeme M. Buchanan, Ariadne Angulo, Andrew Balmford, Bastian Bertzky, Thomas M. Brooks, Kent E. Carpenter, Mia T. Comeros-Raynal, John Cornell, G. Francesco Ficetola, Lincoln D.C. Fishpool, Richard A. Fuller, Jonas Geldmann, Heather Harwell, Craig Hilton-Taylor, Michael Hoffmann, Ackbar Joolia, Lucas Joppa, Naomi Kingston, Ian May, Amy Milam, Beth Polidoro, Gina Ralph, Nadia Richman, Carlo Rondinini, Daniel B. Segan, Benjamin Skolnik, Mark D. Spalding, Simon N. Stuart, Andy Symes, Joseph Taylor, Piero Visconti, James E.M. Watson, Louisa Wood, and Neil D. Burgess. Shortfalls and solutions for meeting national and global conservation area targets. *Conservation Letters*, 8(5):329–337, 2015.

Richard A Fuller, Eve McDonald-Madden, Kerrie A Wilson, Josie Carwardine, Hedley S Grantham, James EM Watson, Carissa J Klein, David C Green, and Hugh P Possingham. Replacing underperforming protected areas achieves better conservation outcomes. *Nature*, 466(7304):365, 2010.

Ana S. L. Rodrigues, H. Resit Akçakaya, Sandy J. Andelman, Mohamed I. Bakarr, Luigi Boitani, Thomas M. Brooks, Janice S. Chanson, Lincoln D. C. Fishpool, Gustavo A. B. Da Fonseca, Kevin J. Gaston, Michael Hoffmann, Pablo A. Marquet, John D. Pilgrim, Robert L. Pressey, Jan Schipper, Wes Sechrest, Simon N. Stuart, Les G. Underhill, Robert W. Waller, Matthew E. J. Watts, and Xie Yan. Global gap analysis: priority regions for expanding the global protected-area network. *BioScience*, 54(12):1092–1100, 2004.