PRIORITIZR WORKSHOP MANUAL

Jeffrey O. Hanson (with modifications by Richard Schuster)

2019-11-15

Contents

Chapter 1

Welcome!

Here you will find the manual for the prioritizr module of the *Introduction to Marxan & MarZone & Prioritizr - Training Course* held at University of Victoria, Victoria, Canada. **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 be "what have you tried so far?". We can't help you if you haven't tried anything.

2.2 Setting up your computer

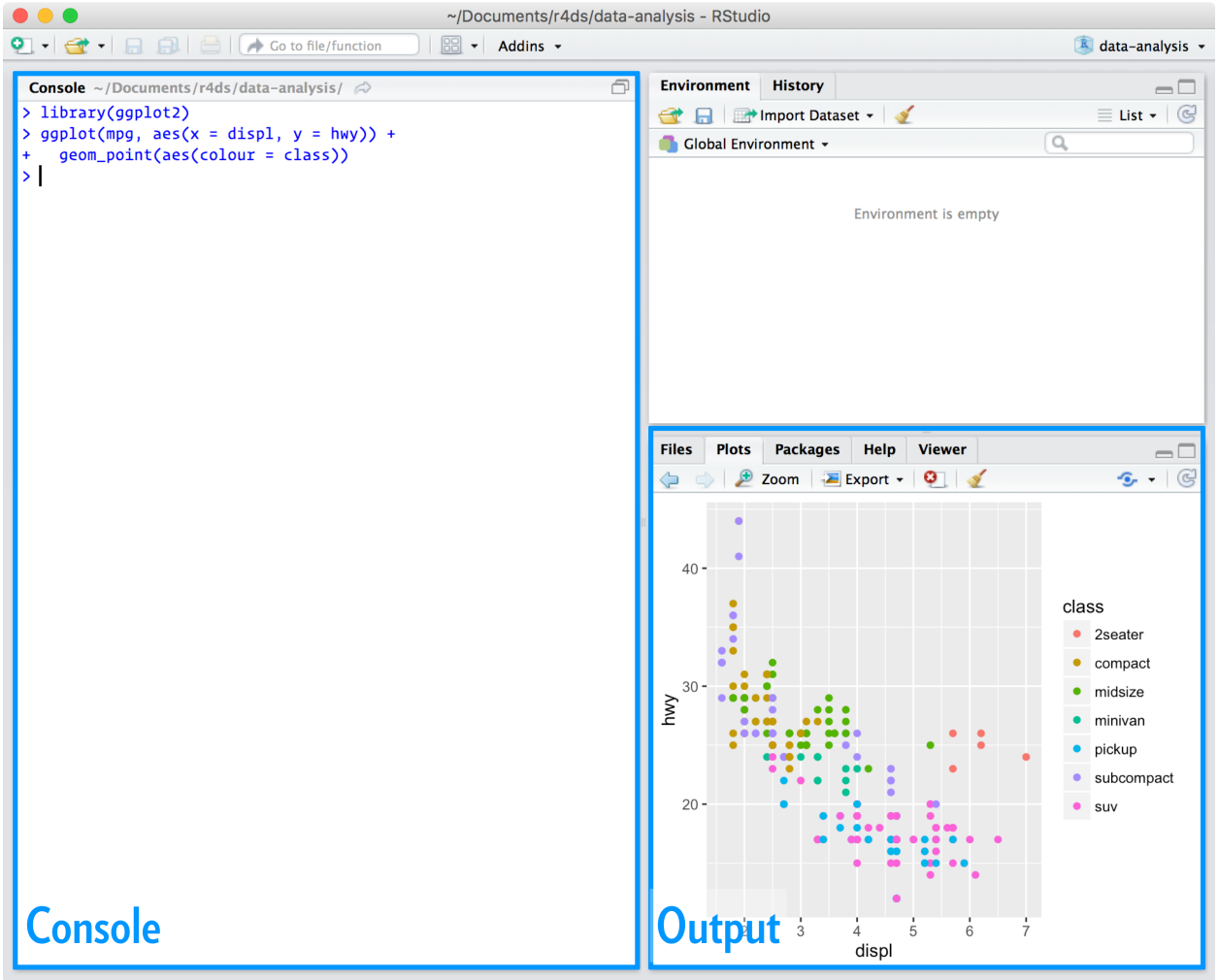
You will need to have both R and RStudio installed on your computer to complete this workshop. Although it is not imperative that you have the latest version of RStudio installed, **you will need the latest version of R installed (i.e. version 3.6.1)**. Please note that you might need administrative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

2.2.1 R

The R statistical computing environment can be downloaded from the Comprehensive R Archive Network (CRAN). Specifically, 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 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* and press the enter key to run 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 fifteen thousand R packages available on CRAN. Each of these R packages are developed to perform a specific task, 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 almost always not "can I use R to …?" but "what R package can I use to …?". During this workshop, we will use several R packages. To install these R packages, please enter the code below in the *Console* part of the RStudio interface and press enter. Note that you will require an Internet connection and the installation process may take some time to complete.

```
install.packages(c("sf", "tidyverse", "sp", "rgeos", "rgdal", "raster",
                   "units", "prioritizr", "prioritizrdata", "Rsymphony",
                   "mapview", "assertthat", "velox", "remotes",
                   "gridExtra", "data.table", "readxl", "BiocManager"))
BiocManager::install("lpsymphony", version = "3.9")
```

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

Redo Marxan analysis

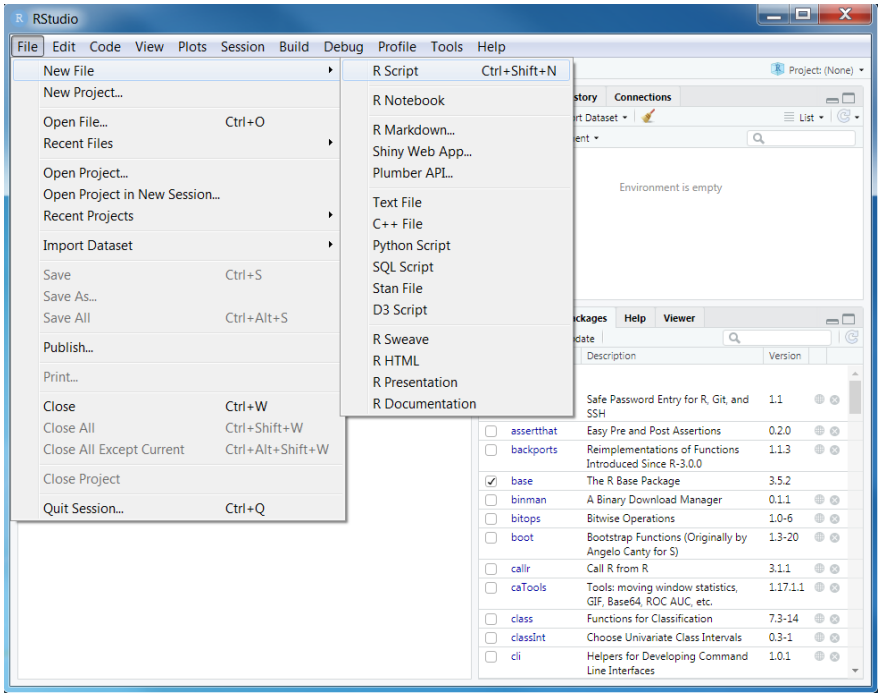
3.1 Introduction

Before we begin to prioritize areas for protected area establishment using the full feature set of *prioritizr*, we will re-do the Marxan analysis from Tuesday in *prioritizr*. This exercise is meant to show you how you can use your current Marxan files in prioritizr, if you choose to do so. Once we have run the example using input.dat, as well as the individual .dat files, we will also work on preparing the data you worked with on Monday to be used in prioritzr. Once that's complete, we will run our first *prioritizr* analysis, using the notation typical for prioritizr analysis.

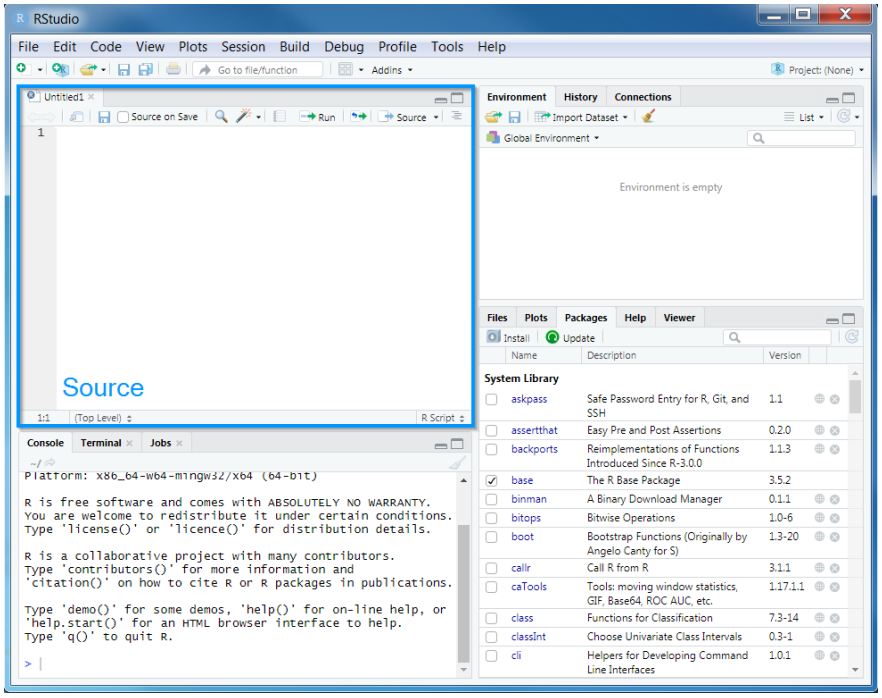
The data for this exercise were provided by PacMara and BCMCA.

3.2 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, then *New File*, and then *R Script*.



After creating a new script, you will notice that a new *Source* panel has appeared. In the *Source* panel, you can type and edit code before you run it. 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 R

script. 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`)!

3.3 Attaching packages

Now we will set up 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)
library(data.table)
library(readxl)
```

3.4 Base analysis on input.dat

Now we will redo the Marxan analyis you have done on Tuesday, but using *prioritzr*. To do so we need the *Marxan database* you used on Tuesday, as well we the *raw data* you used on Monday. The files for both are already included in the R Studio project you received for this exercise. Now please open the *PacMara_workshop.Rproj* file by double clicking it. You are now ready to start with the exercise.

First, we are going to use the information from the input.dat file to run the analysis you completed on Tuesday, using *prioritzr*. To do so, all you need to do is point to input.dat and tell *prioritizr* where to find it. Once that's done we can generate the problem and solve it.

```
input_file <- "Marxan_database/input.dat"

p1 <- marxan_problem(input_file)

s1 <- solve(p1)
```

Next, we are going to have a look at the solution and explore the output by first displaying a couple of rows from the output data, then counting the number of planning units in the solution and calcualating the proportion of planning units in the solution.

```
head(s1)
```

```
##   id    cost status locked_in locked_out solution_1
## 1  1 2000000      0     FALSE      FALSE          1
## 2  2 2000000      0     FALSE      FALSE          1
## 3  3 2000000      0     FALSE      FALSE          1
## 4  4 2000000      0     FALSE      FALSE          1
## 5  5 2000000      0     FALSE      FALSE          1
## 6  6 2000000      0     FALSE      FALSE          1
```

```
# count number of planning units in solution
sum(s1$solution_1)
```

```
## [1] 3262
```

```
# proportion of planning units in solution
mean(s1$solution_1)
```

```
## [1] 0.2683669
```

Next we are going to explore how well the features are represented in the solution.

```
# calculate feature representation
r1 <- feature_representation(p1, s1[, "solution_1", drop = FALSE])
print(r1)
```

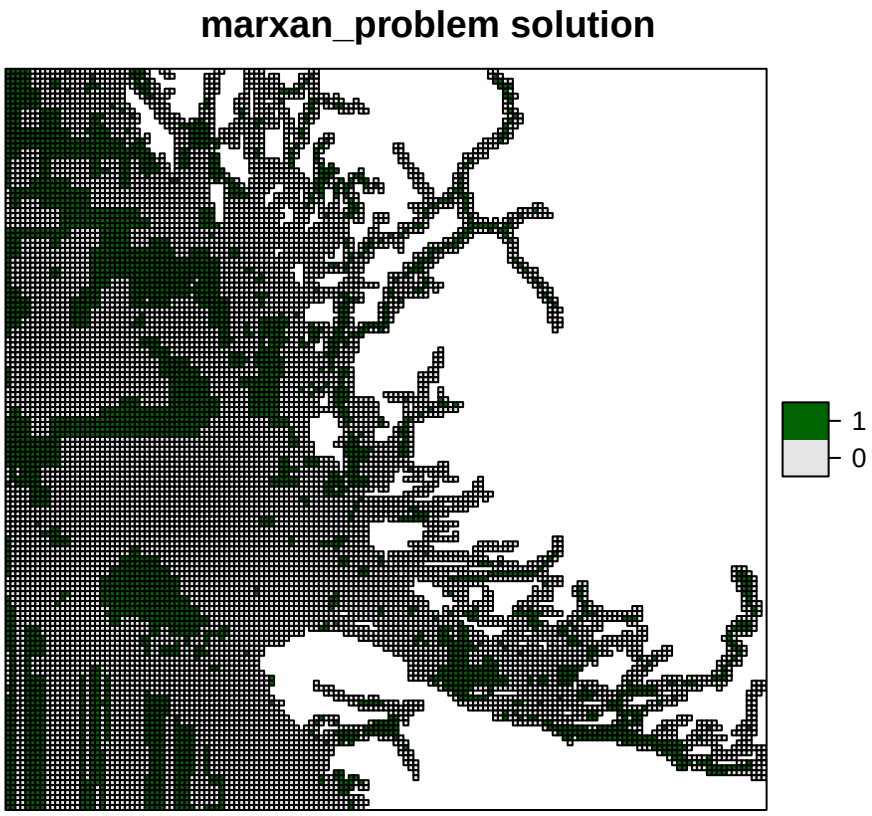
```
## # A tibble: 19 x 3
##    feature      absolute_held relative_held
##    <chr>                <dbl>         <dbl>
##  1 0-20 Hard        213510000         0.300
##  2 0-20 Muddy       145420000         0.302
##  3 0-20 Sandy        45810000         0.301
##  4 20-50 Hard       842380000         0.300
##  5 20-50 Muddy      120580000         0.300
##  6 20-50 Sandy       93400000         0.300
##  7 200+ Hard        136480000         0.300
##  8 200+ Muddy       870960000         0.300
##  9 200+ Sandy      1075960000         0.300
## 10 200+ UnId       2025510000         0.300
## 11 50-200 Hard     2151230000         0.300
## 12 50-200 Muddy    1080500000         0.300
## 13 50-200 Sandy    3347250000         0.300
## 14 50-200 UnId        3260000         0.337
## 15 iba             1991070000         0.300
## 16 kelp              51940000         0.305
## 17 killer whale    1596950000         0.386
## 18 sealions         697060000         0.494
## 19 seaotters       1596000000         0.3
```

Finally, we are going to visualize the solution by converting the solution to a spatial object.

```
pulayer <- readOGR("Marxan_database/pulayer/pulayer_BC_marine.shp", stringsAsFactors = F
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/home/travis/build/prioritizr/PacMara_workshop/Marxan_database/pulayer/pulay
## with 12155 features
## It has 1 fields
## Integer64 fields read as strings:  PUID
```

```
pulayer1 <- pulayer
pulayer1$solution_1 <- s1$solution_1
pulayer1$solution_1 <- factor(pulayer1$solution_1)
spplot(pulayer1, "solution_1", col.regions = c("grey90", "darkgreen"),
       main = "marxan_problem solution")
```



Now, think about the following questions.

1. Are the results from Marxan and prioritizr the same/similar?
2. If you see differences, why do you think those differences occur?
3. Can you think of ways to reduce difference/improve outcomes?

3.5 Base analysis using individual .dat files

Now, lets redo the analysis, but instead of using input.dat, we will use the individual .dat files to create the problem. You will see that the syntax for the problem formulation is very similar, but instead of supplying one value to the *marxan_problem* function, we now specify *pu*, *spec*, *puvsp* and *bound*. If you want to learn more about the *marxan_problem* function, just type in `?marxan_problem` and you can have a look at the function help page.

```
pu <- fread("Marxan_database/input/pu.dat", data.table = FALSE)
spec <- fread("Marxan_database/input/spec.dat", data.table = FALSE)
puvsp <- fread("Marxan_database/input/puvsp.dat", data.table = FALSE)
bound <- fread("Marxan_database/input/bound.dat", data.table = FALSE)

p2 <- marxan_problem(x = pu, spec = spec, puvspr = puvsp)

s2 <- solve(p2)

# count number of planning units in solution
sum(s2$solution_1)
```

```
## [1] 3261
```

```
# proportion of planning units in solution
mean(s2$solution_1)
```

```
## [1] 0.2682847
```

```
# calculate feature representation
r2 <- feature_representation(p2, s2[, "solution_1", drop = FALSE])
print(r2)
```

```
## # A tibble: 19 x 3
##    feature      absolute_held relative_held
##    <chr>                <dbl>         <dbl>
##  1 0-20 Hard        213530000         0.300
##  2 0-20 Muddy       145420000         0.302
##  3 0-20 Sandy        45940000         0.302
##  4 20-50 Hard       842510000         0.300
##  5 20-50 Muddy      120590000         0.300
##  6 20-50 Sandy       93640000         0.301
##  7 200+ Hard        136630000         0.300
##  8 200+ Muddy       871080000         0.300
##  9 200+ Sandy      1075970000         0.300
```

```
## 10 200+ UnId       2025510000         0.300
## 11 50-200 Hard     2150800000         0.300
## 12 50-200 Muddy    1081840000         0.301
## 13 50-200 Sandy    3347430000         0.300
## 14 50-200 UnId        3260000         0.337
## 15 iba             1992240000         0.300
## 16 kelp              51570000         0.303
## 17 killer whale    1594570000         0.385
## 18 sealions         702660000         0.498
## 19 seaotters       1596000000         0.3
```

3.6 Recreate the Marxan analysis starting from the raw data

Now that we have solved a problem that was formatted the way Marxan needs data, lets go ahead and start from the raw data, as you did on Monday.

We will first process the data, so we can use it in *prioriztr* and then we will create the problem and solve it in the 'standard' *prioritizr* way.

Starting with the raw data from folder *Marxan_Data* we will go ahead and create our problem. First, lets load all the data we need in terms of features:

```
# We first load this Excel file to extract feature names later
feat_ids <- read_xlsx("Marxan_Data/conservation_feats_ids.xlsx")

# now lets load all the rasters we need
iba <- raster("Marxan_Data/iba_bc.tif")
kelp <- raster("Marxan_Data/kelp_bc.tif")
killerw <- raster("Marxan_Data/killerwhale.tif")
seal <- raster("Marxan_Data/sealions.tif")
seao <- raster("Marxan_Data/seaotter.tif")

benthic <- raster("Marxan_Data/benthic14cl.tif")

# benthic and the rest of the rasters are not exactly in the same format (same number
# so we need to go ahead and make sure benthic has the same format as the other raster
benthic <- resample(benthic, iba, method="ngb")
```

Now that we have loaded all the feature data into R, we need to go ahead and create the benthic classes, you have used to setup the Marxan problem before. This is specific to the way the benthic raster is setup and will differ from case to case in real world examples you might explore in the future.

In this specfic case, we know that benthic has a total of 14 classes, so the R code below does split the benthic raster up into 14 rasters, based on cell values, and at the end puts them together in a stack.

```
benthic_values <- values(benthic)
ben_list <- list()
for(ii in 1:14){
  tmp_r <- benthic
  tmp_r_val <- benthic_values
  tmp_r_val <- ifelse(tmp_r_val == ii, 1, NA)
  values(tmp_r) <- tmp_r_val

  ben_list[[ii]] <- tmp_r

  rm(tmp_r, tmp_r_val)

}
ben_stack <- stack(ben_list)
```

Now that all rasters have been created, we can combine them in a stack and give them the names from the Excel file we read in earlier.

```
features <- stack(ben_stack, iba, kelp, killerw, seal, seao)
names(features) <- feat_ids$New_Name
```

Next, we load in the fishcost layer and also resample it to fit the rest of the raster layers.

```
fishcost <- raster("Marxan_Data/fishcost.tif")
fishcost <- resample(fishcost, iba, method="ngb")
```

Now its time to setup the *prioritizr* problem. As a first step, we are reading in the pulayer from the *Marxan_database*. I'm doing this to show you a nice way to setup the *prioritizr* problem, using a shapefile directly in the `problem` function call. We need to extract the fishcost data to that pulayer, before we can use this information in the `problem` formulation.

```
pulayer <- readOGR("Marxan_database/pulayer/pulayer_BC_marine.shp", stringsAsFactors = F
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/home/travis/build/prioritizr/PacMara_workshop/Marxan_database/pulayer/pulay
## with 12155 features
## It has 1 fields
## Integer64 fields read as strings:  PUID
```

```
pulayer$cost <- as.vector(fast_extract(fishcost, pulayer))
```

Now for the actual `problem` formulation. You will see that we use the pulayer as one of the inputs to the `problem` function. As pulayer is a shapefile, we need to tell *prioritzr* which attribute to use as the cost column. We also include the features raster stack directly in the `problem` function. We also set the objective function (minumum set), the relative targets (0.3 or 30% of each feature), and the decision type (binary for integer linear programming).

When that's done we can solve the problem.

```
p3 <- problem(pulayer, cost_column = "cost", features = features, run_checks = FALSE) %>
  add_min_set_objective() %>%
  add_relative_targets(0.3) %>%
  add_binary_decisions()

s3 <- solve(p3)
```

As we have done before, we will now go ahead and extract summary statistics as well as plot the results. We just worked through an entire *prioritzr* problem, from reading and processing raw data to setting up and solving a `problem`, to extracting statistics and spatial visualization of results.

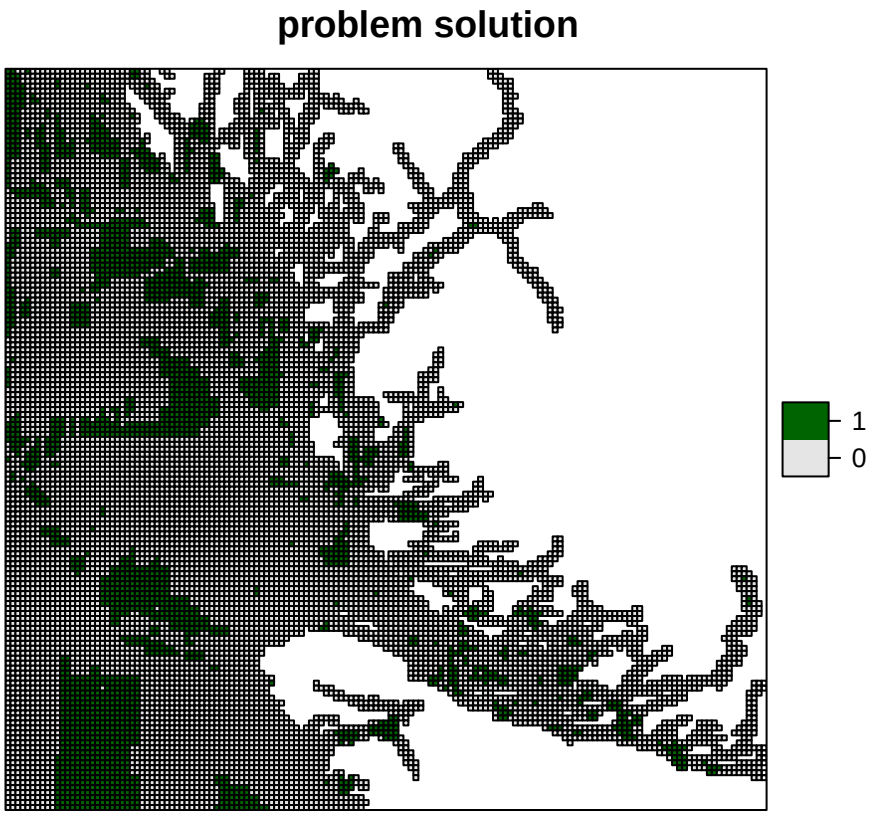
```
# count number of planning units in solution
sum(s3$solution_1, na.rm = TRUE)
```

```
## [1] 2332
```

```
# proportion of planning units in solution
mean(s3$solution_1, na.rm = TRUE)
```

```
## [1] 0.1919816
```

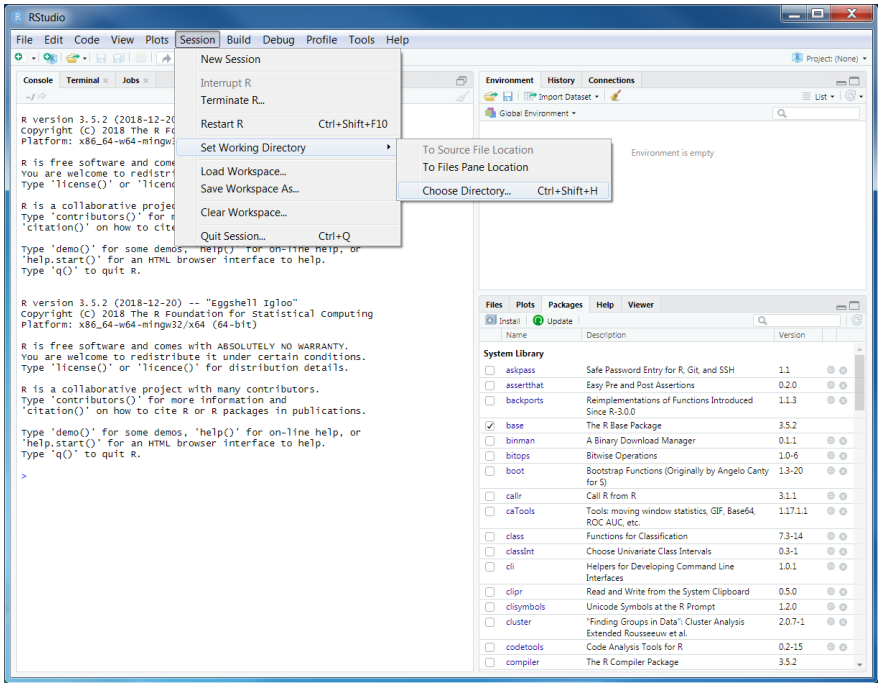
```
s3$solution_1 <- factor(s3$solution_1)
spplot(s3, "solution_1", col.regions = c("grey90", "darkgreen"),
       main = "problem solution")
```


problem solution
1
0

Chapter 4

Data

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/PacMara_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
```

4.1 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/PacMara_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")
```

4.2 Planning unit data

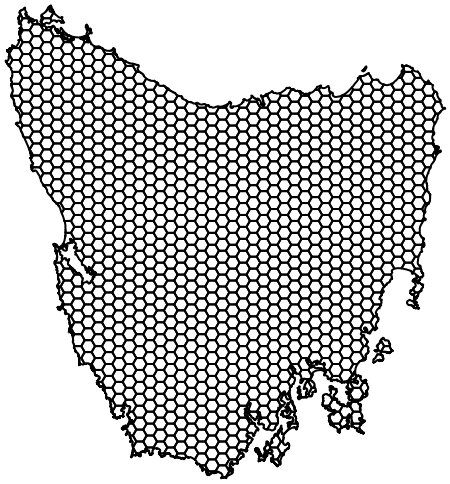
The planning unit data contains spatial data describing the geometry for each planning unit and attribute data with information about each planning unit (e.g. cost values). Let's investigate the `pu_data` object. The attribute data contains 5 columns with contain the following information:

- `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 are too degraded.

```
# 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 +ellp
## 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
```

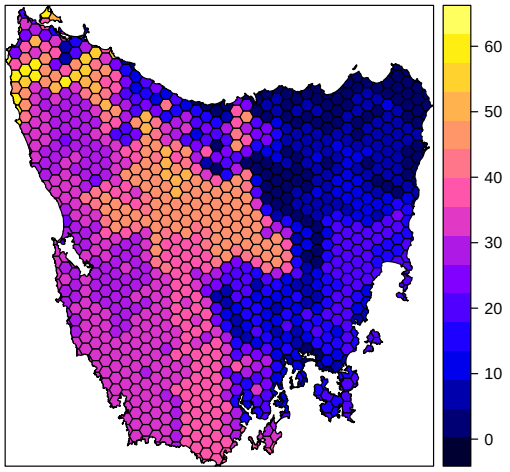
```
# print the smallest cost value
min(pu_data$cost)
```

```
## [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")
```

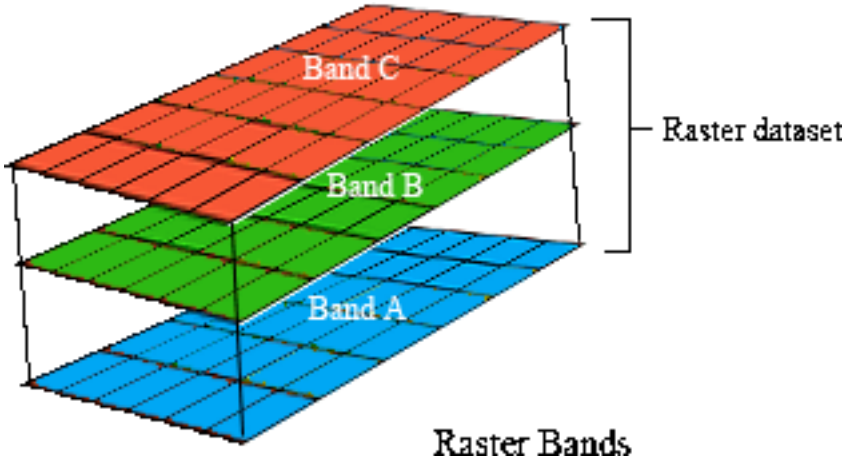
Now, you can try and answer some questions about the planning unit data.

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 highly degraded (hint: `sum(x)`)?
6. What is the proportion of planning units are highly degraded (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 very 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).

4.3 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 square grid comprising square grid cells that are each the same size. In our case, the raster data contains multiple layers (also called "bands") and each layer has 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 rows, 320 columns, and 109760 cells. Within each layer, each cell corresponds to a 1 by 1 km square. The values associated with each grid cell indicate the (one) presence or (zero) absence of a given vegetation class in the cell.

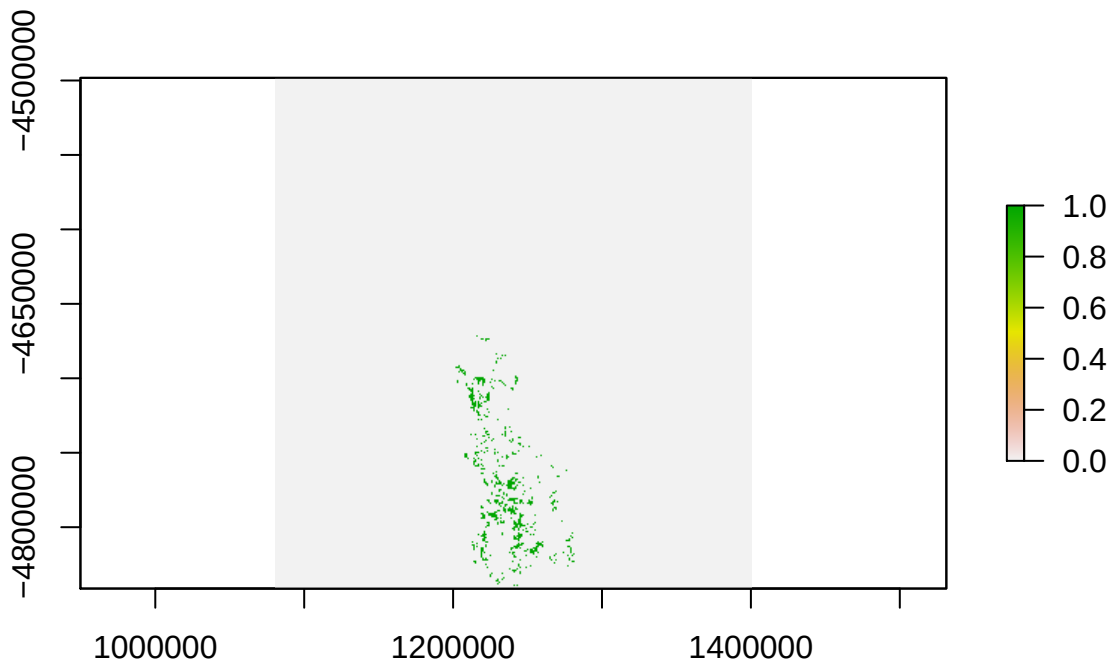


Let's explore the vegetation data.

```
# 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/PacMara_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?
4. The planning unit data and the vegetation data should have the same coordinate reference system. Can you check if they are the same?

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 they are 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 formulation of the reserve selection problem. This formulation means that we want a solution that will meet the targets for our biodiversity features for minimum cost. Here, we will set 5% targets for each vegetation class and use the data in the `cost` column to specify acquisition costs. Unlike Marxan, we do not have to calibrate species penalty factors (SPFs) to ensure that our target are met—prioritizr should always return solutions to minimum set problems where all the targets are met. 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 easier to install. The Gurobi solver is much faster than the lpsymphony solver (see here for 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 +ellp
## 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
```

```
# 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
# indicating if a planning unit is (1) selected 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 +ellp
## variables   : 6
## names       :   id,              cost, status, locked_in, locked_out, solution_1
## min values  :    1, 0.192488262910798,      0,         0,          0,          0
## max values  : 1130,  61.9272727272727,      2,         1,          1,          1
```

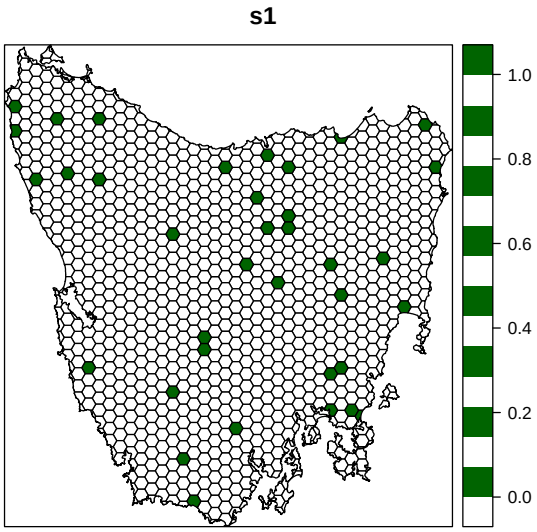
```
# calculate number of planning units selected in the prioritization
sum(s1$solution_1)
```

```
## [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 planing 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"])`)?

5.3 Adding complexity

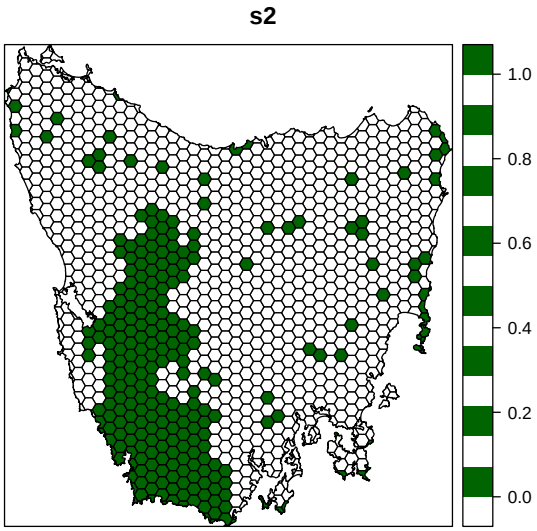
Our first prioritization suffers many limitations, so let's add additional constraints to the problem to make it more useful. First, let's lock in planing units that are already by covered

protected areas. If some vegetation communities are already secured inside existing protected areas, then we might not need to add as many new protected areas to the existing protected area system to meet their targets. Since our planning unit data (`pu_da`) already contains this information in the `locked_in` column, we can use this column name to specify which planning units should be locked in.

```
# 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")
```

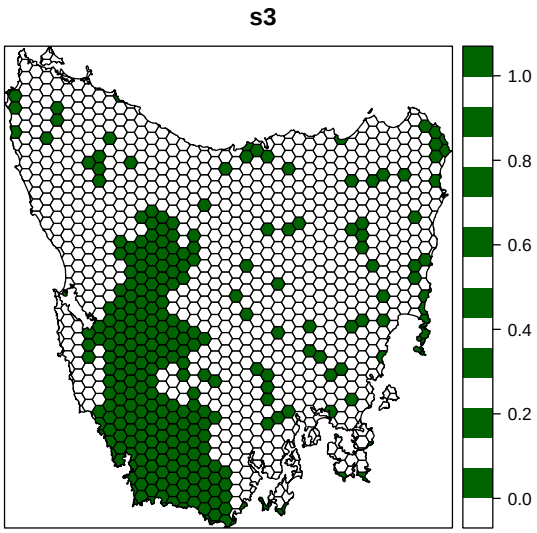


Let's pretend that we talked to an expert on the vegetation communities in our study system and they recommended that a 20% target was needed for each vegetation class. So, armed with this information, let's set the targets to 20%.

```
# 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. Similar to before, this data is present in our planning unit data so we can use the `locked_out` column name to achieve this.

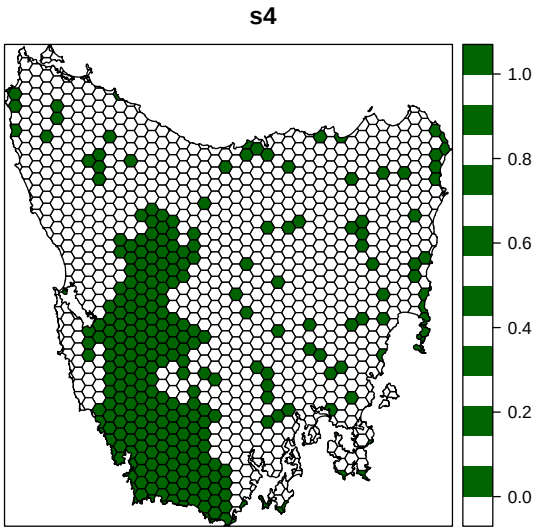
```
# make prioritization problem
p4 <- problem(pu_data, veg_data, cost_column = "cost") %>%
      add_min_set_objective() %>%
      add_relative_targets(0.2) %>%
      add_locked_in_constraints("locked_in") %>%
      add_locked_out_constraints("locked_out") %>%
      add_binary_decisions() %>%
      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 out planning units [132 locked units]
##                    Locked in planning units [257 locked units]>
##   penalties:      <none>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```r
# 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 will 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. Since our planning unit data is in a spatial format (i.e. vector or raster data), prioritizr can automatically calculate the boundary data for us.

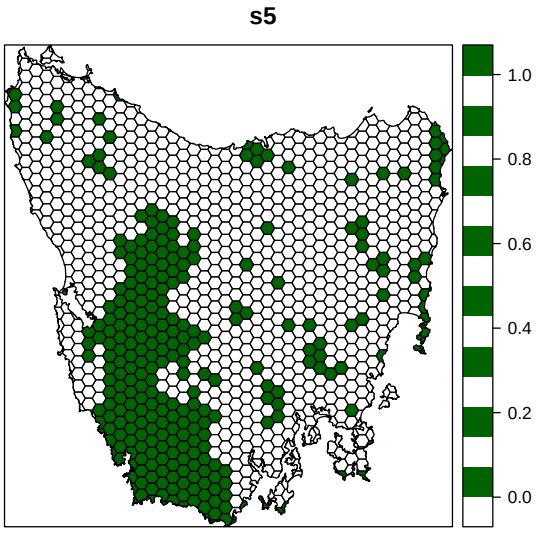
```
# make prioritization problem
p5 <- problem(pu_data, veg_data, cost_column = "cost") %>%
      add_min_set_objective() %>%
      add_boundary_penalties(penalty = 0.0005) %>%
      add_relative_targets(0.2) %>%
      add_locked_in_constraints("locked_in") %>%
      add_locked_out_constraints("locked_out") %>%
      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 [132 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 +ellp
## variables   : 6
## names       :   id,              cost, status, locked_in, locked_out, solution_1
## 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 0.0005. 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 is 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?

5.5 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 $904 million AUD to ensure that each vegetation community is adequately represented in the protected area system. But what if the funds available for establishing new protected areas 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 prioritization by maximizing the number of adequately represented features whilst keeping within a pre-specified budget.

```
# funds for additional land acquisition (same units as cost data)
funds <- 100

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

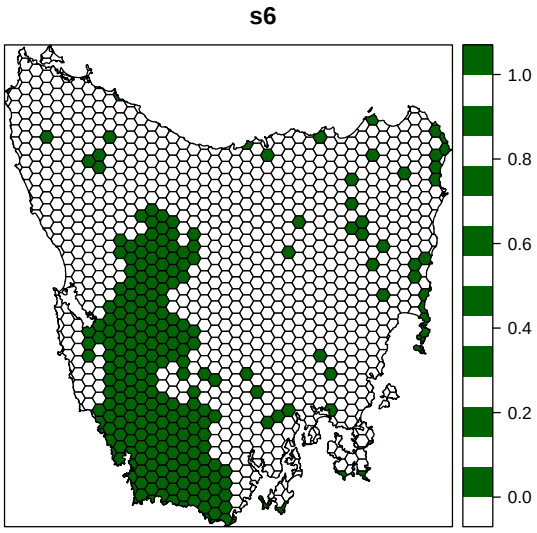
```
## [1] 8575.56
```

```
# 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 [132 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] 28
```

```
# find out which features have their targets met when we add weights,
# note that NA is for vegetation.61
print(r6$feature[r6$relative_held >= 0.2])
```

```
##  [1] "vegetation.1"  "vegetation.2"  "vegetation.3"  "vegetation.4"
##  [5] "vegetation.5"  "vegetation.6"  "vegetation.7"  "vegetation.8"
##  [9] "vegetation.11" "vegetation.12" "vegetation.13" "vegetation.14"
## [13] "vegetation.15" "vegetation.17" "vegetation.25" "vegetation.28"
## [17] "vegetation.29" "vegetation.30" "vegetation.32" "vegetation.33"
## [21] "vegetation.34" "vegetation.35" "vegetation.36" "vegetation.37"
```

```
## [25] "vegetation.38" "vegetation.39" "vegetation.40" "vegetation.45"
## [29] 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.

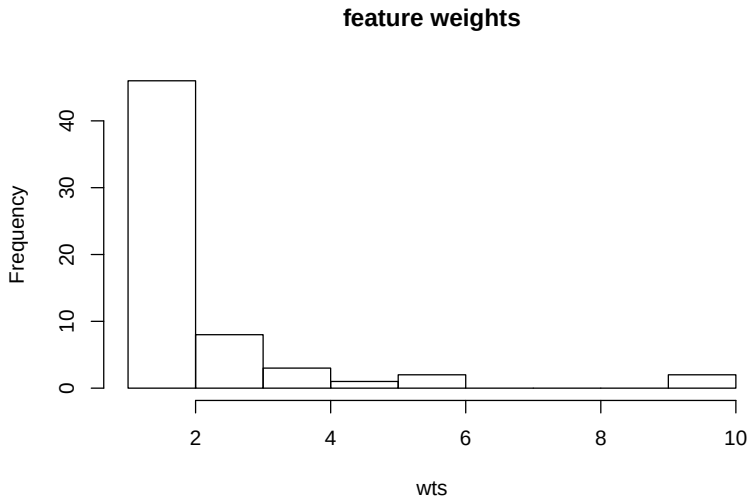
```
# calculate weights as the log inverse number of grid cells that each vegetation
# class occupies, rescaled between 1 and 100
wts <- 1 / cellStats(veg_data, "sum")
wts <- rescale(wts, to = c(1, 10))

# 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"
```

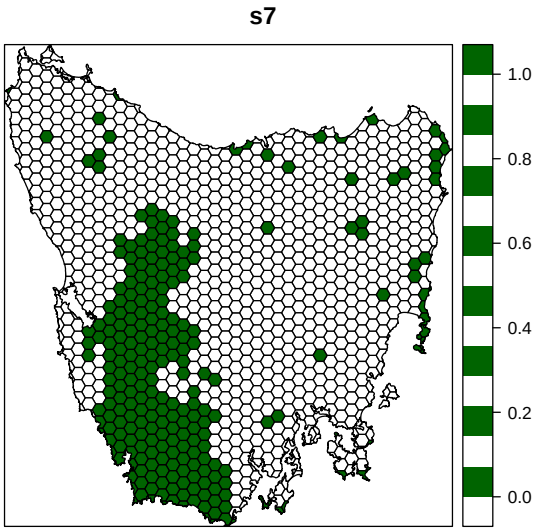
```
# plot histogram of weights
hist(wts, main = "feature weights")
```



```r
# 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 out planning units [132 locked units]
##                    Locked in planning units [257 locked units]>
##   penalties:      <Feature weights [weights (min: 1, max: 10)]>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```r
# 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] 26
```

```
# find out which features have their targets met when we add weights,
# note that NA is for vegetation.61
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.54" "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? If so, what is the difference?

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 spatially distinct 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 +ellp
## variables   : 11
## names       :   id,              cost, status, locked_in, locked_out, solution_1, sol
## min values  :    1, 0.192488262910798,      0,         0,          0,          0,
## max values  : 1130,  61.9272727272727,      2,         1,          1,          1,
```

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

```
## [1] 2169.162
```

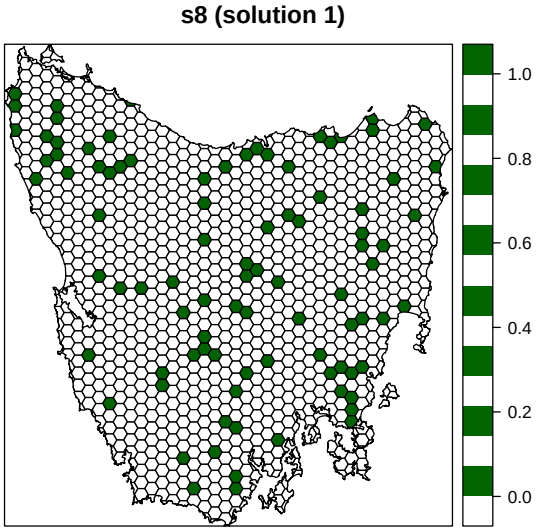
```
# calculate the cost of the second solution
sum(s8$solution_2 * s8$cost)
```

```
## [1] 2127.815
```

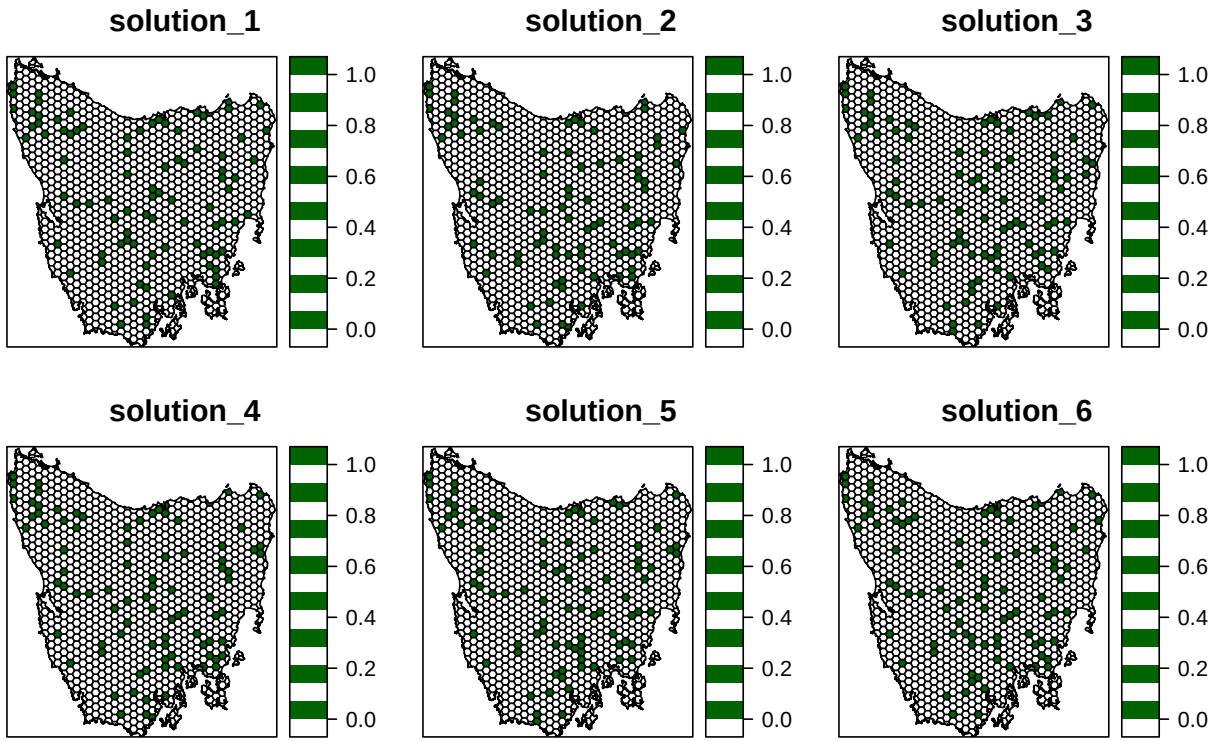
```
# calculate the proportion of planning units with the same solution values
# in the first and second solutions
mean(s8$solution_1 == s8$solution_2)
```

```
## [1] 0.9566372
```

```
# 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)))
```



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 Redo Marxan analysis

6.1.1 Base analysis on input.dat

1. Subjective.
2. I would say, the most obvious differences are in the bottom left corner, where the Marxan results are rather diffuse and selection frequencies are low. This is actually a very good example of some 'issues' people have described with prioritizr. Its essentially related to a problem not having enough feature and cost heterogeneity for a decision support tool such as Marxan or prioritzr to find reasonable solutions.
3. Heterogenous cost structures help. So do features that are more complex or have more overlap with each other.

6.2 Data

6.2.1 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.

6.2.2 Vegetation data

1. Central-north Tasmania
2. `cellStats(veg_data[[12]], "mean")`
3. `names(veg_data)[which.max(cellStats(veg_data, "sum"))]`
4. Yes, they are the same.

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)`

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 an obvious difference in the spatial location of the planning units which they do not share.
4. Yes. Both solutions contain adequately represent these features:
   `r6$feature[r6$relative_held > 0.2 & r7$relative_held > 0.2]`
   The sixth (`s6`) is adequately represents these features too:
   `r6$feature[r6$relative_held > 0.2 & !r7$relative_held > 0.2]`
   The seventh (`s7`) is adequately represents these features too:
   `r7$feature[r7$relative_held > 0.2 & !r6$relative_held > 0.2]`

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. We also thank Garrett Grolemund and Hadley Wickham for creating one of the Rstudio screenshots used in this manual that was originally a part of their *R for Data Science* book.

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] readxl_1.3.1       data.table_1.12.6  gridExtra_2.3      assertthat_0.2.1
##  [5] scales_1.0.0       units_0.6-5        mapview_2.7.0      rgeos_0.5-2
##  [9] rgdal_1.4-7        prioritizr_4.1.4.2 proto_1.0.0        raster_3.0-7
## [13] sp_1.3-2           forcats_0.4.0      stringr_1.4.0      dplyr_0.8.3
## [17] purrr_0.3.3        readr_1.3.1        tidyr_1.0.0        tibble_2.1.3
## [21] 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    lubridate_1.7.4
##  [5] webshot_0.5.1      httr_1.4.1         tools_3.6.1        backports_1.1.5
##  [9] utf8_1.1.4         R6_2.4.1           KernSmooth_2.23-15 DBI_1.0.0
## [13] lazyeval_0.2.2     colorspace_1.4-1   withr_2.1.2        tidyselect_0.2.5
## [17] leaflet_2.0.2      compiler_3.6.1     cli_1.1.0          rvest_0.3.5
## [21] xml2_1.2.2         bookdown_0.15.1    classInt_0.4-2     digest_0.6.22
## [25] rmarkdown_1.17     base64enc_0.1-3    pkgconfig_2.0.3    htmltools_0.4.0
## [29] lpsymphony_1.12.0  fastmap_1.0.1      htmlwidgets_1.5.1  rlang_0.4.1
## [33] rstudioapi_0.10    shiny_1.4.0        generics_0.0.2     jsonlite_1.6
## [37] crosstalk_1.0.0    Rsymphony_0.1-28   magrittr_1.5       Matrix_1.2-17
## [41] fansi_0.4.0        Rcpp_1.0.3         munsell_0.5.0      lifecycle_0.1.0
## [45] stringi_1.4.3      yaml_2.2.0         plyr_1.8.4         grid_3.6.1
## [49] parallel_3.6.1     promises_1.1.0     crayon_1.3.4       lattice_0.20-38
## [53] haven_2.2.0        hms_0.5.2          zeallot_0.1.0      knitr_1.26
## [57] pillar_1.4.2       uuid_0.1-2         velox_0.2.0        codetools_0.2-16
## [61] stats4_3.6.1       glue_1.3.1         evaluate_0.14      modelr_0.1.5
## [65] png_0.1-7          vctrs_0.2.0        httpuv_1.5.2       cellranger_1.1.0
## [69] gtable_0.3.0       xfun_0.11          mime_0.7           xtable_1.8-4
## [73] broom_0.5.2        e1071_1.7-2        later_1.0.0        class_7.3-15
## [77] viridisLite_0.3.0
```

Chapter 9

References

Bibliography

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.