Tests and ideas with new package

Derek Corcoran

2020-02-26

1 Load packages

I will load a bunch of packages including *ggraph* to ilustrate the network, in particular the Phillips Problem.

```
library(dispRsal)
library(kableExtra)
library(gdistance)
library(ggraph)
library(igraph)
library(lpSolve)
library(magrittr)
library(raster)
library(tidyverse)
```

2 The Phillips problem

This problem starts with a toy model of a species. We have modeled the distribution of that species and we have projected that distribution through 2 different time-slices T0 and T1, this is conceptualized by the following rasterstack called *Phillips*, as seen in Figure 1

```
plot(Phillips)
```

Fot that same area we have a costlayer (*PhilCost*), we but the two middle cells are unavailable for the species, because there is an obstacle there (eg: a city or a glaciar) as seen in Figure 2

The idea is to create a solution where the species has always at least two cells (*nchains*) available at each time-slice, but those cells have to be within reach for the species between time-slices. Let's say for example that for this species, the distance the species can travel is 111 km between time-slice 0 to time-slice 1, in this case the species would only be able to move to one adyacent cell between time-slices, this will be defined by the following variables:

```
layers_habitat = Phillips
layer_cost = PhilCost
Dist = 111000
nchains = 2
```

3 Step 1

First we eliminate the NA sites created by the unavailable areas shown in figure 2

```
layers_habitat <- layers_habitat * !is.na(layer_cost)
```

This will modify the raster shown in figure 1 1 into the one we see in figure 3, notice that we lost 2 cells that use to be available when we first looked at the data.

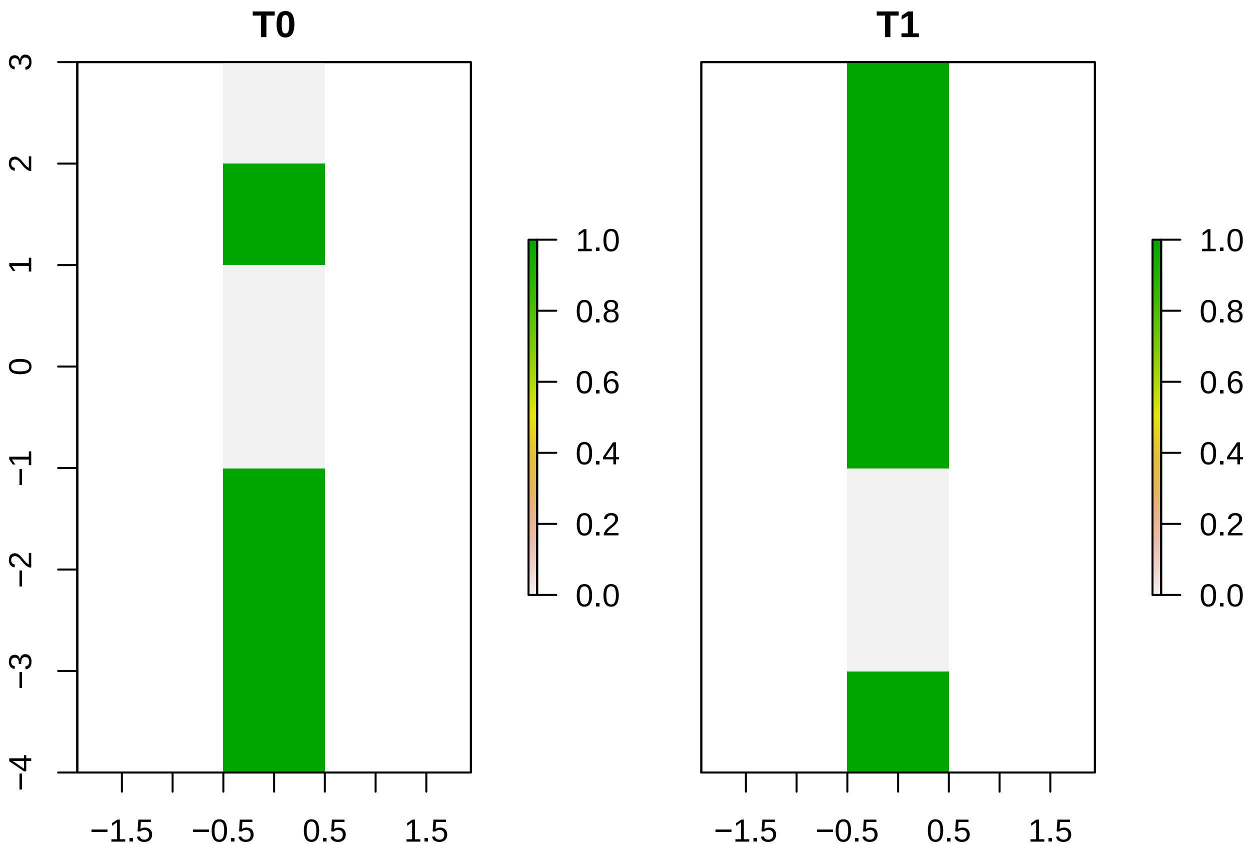

Figure 1: Predicted distribution of a species for two time slices, green means the species is predicted to be present, and grey that the species is predicted to be absent

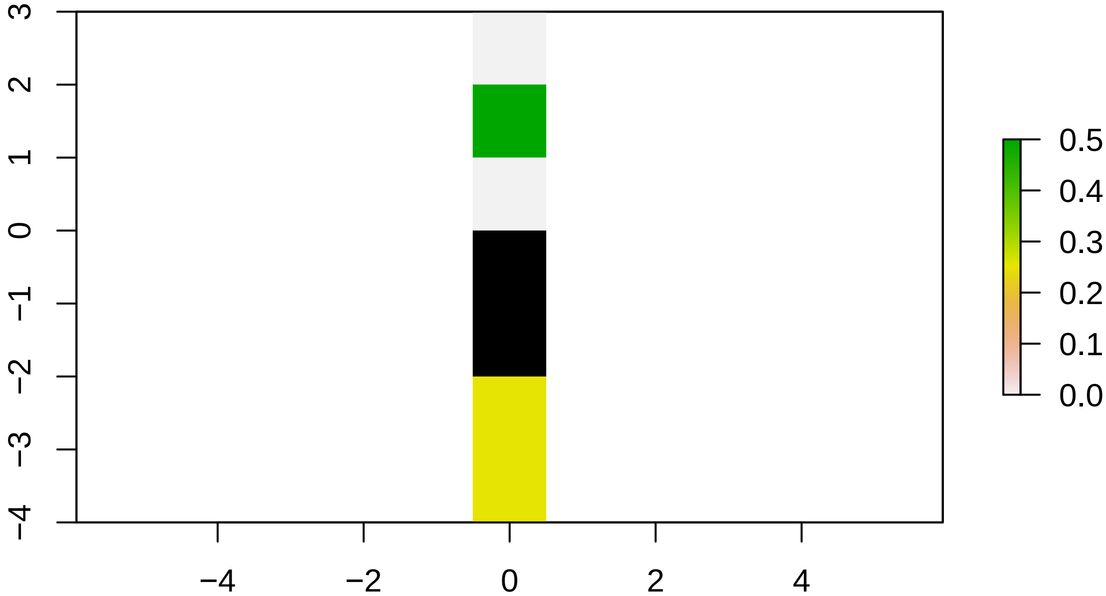

Figure 2: Raster with the cost of buying each cell, black cells are unavailable cells

```r
layers_habitat <- layers_habitat * !is.na(layer_cost)
```

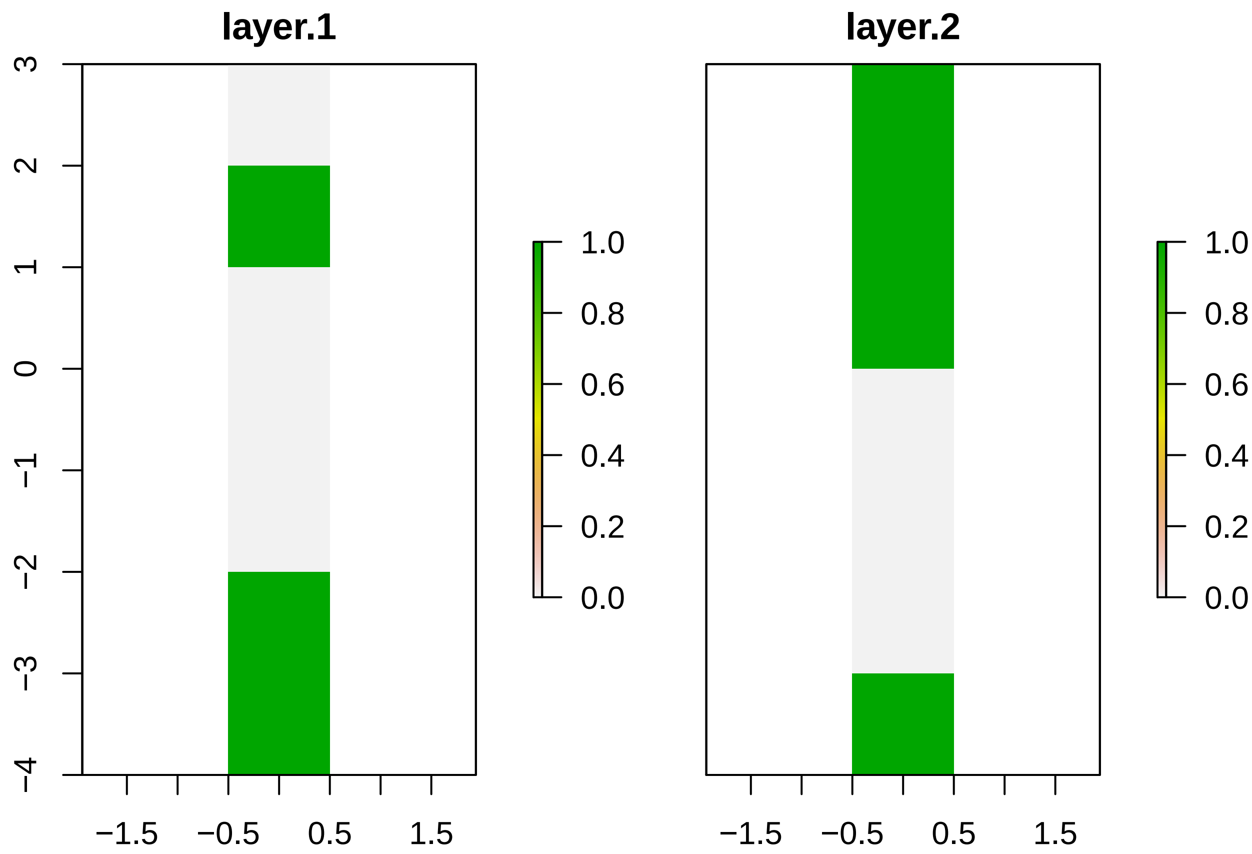

Figure 3: Habitat layers modified so that we take out the non available values

Then we create a data frame of all the nodes, as we see in table 1, a new *Cell_id* was generated that is unique to every node in each time-slice, but the raster_id was kept in order to be able to get the results back to the raster.

```r
df_habitat <- map_dfr(layers_habitat = layers_habitat, .x = 1:nlayers(layers_habitat),
    .f = ~layers_habitat[[.x]] %>% values() %>% tibble(habitat = .) %>% transmute(cell_id = row_number(
        time = .x, habitat = habitat)) %>% # give the same cell in different time slices different IDs
mutate(raster_id = cell_id, cell_id = cell_id + max(cell_id) * time)
```

4 Generate a DF that keeps the ID of the raster cell and the new cell ID

In order to be able to compare the cell_id with raster_id in further tables a table was created with this use in mind, this is seen in table 2:

Table 1: Nodes with a new cell id that is unique for every time slice, the raster id is also kept so that we can get the results back to the raster

cell_id	time	habitat	raster_id
8	1	0	1
9	1	1	2
10	1	0	3
11	1	0	4
12	1	0	5
13	1	1	6
14	1	1	7
15	2	1	1
16	2	1	2
17	2	1	3
18	2	0	4
19	2	0	5
20	2	0	6
21	2	1	7

Table 2: Table of cell ID and Raster ID equivalencies

node_to	raster_id
8	1
9	2
10	3
11	4
12	5
13	6
14	7
15	1
16	2
17	3
18	4
19	5
20	6
21	7

```
df_IDs <- df_habitat %>% rename(node_to = cell_id) %>% dplyr::select(node_to, raster_id)
```

4.0.1 Generate a DF with the costs of each cell

Using the cost layer seen in figure 2 we generate a table with the cost of each node as seen in table table 3, this could be made smaller by only using the values of each cell and then using 2 to reference the cost

```
df_cost <- layer_cost %>% values() %>% tibble(cost = .) %>% mutate(cell_id = row_number())
df_cost <- map_dfr(df_cost = df_cost, .x = 1:nlayers(layers_habitat), .f = ~df_cost %>%
    transmute(cell_id = cell_id + max(cell_id) * .x, edge_cost = cost))
```

Table 3: Cost of every node

cell_id	edge_cost
8	0.00
9	0.50
10	0.00
11	NA
12	NA
13	0.25
14	0.25
15	0.00
16	0.50
17	0.00
18	NA
19	NA
20	0.25
21	0.25

5 Getting the distances between variables

For that we use the function `edge_distance_limit` this will give use table 4, in which we have the pair of nodes and the distance between them in thre raster, this has already filtered out any distances greater than `Dist`

```
connections <- edge_distance_limit(layer_cost = PhilCost, layers_habitat = Phillips,
    Dist = Dist)
```

6 get all valid edge connections,

In this step we do 3 different processes

First we filter only the pairs of nodes that hava a capacity of one:

```
edges_timesteps <- map_dfr(df_habitat = df_habitat, .x = 2:nlayers(layers_habitat),
    .f = ~expand_grid(node_from = df_habitat %>% filter(time == .x - 1 & habitat ==
        1) %>% pull(cell_id), node_to = df_habitat %>% filter(time == .x & habitat ==
        1) %>% pull(cell_id), timeslice_to = .x)) %>% mutate(edge_id = row_number(),
    edge_capacity = 1, edge_cost = NA)
```

Then we add the cost by adding the costs of both nodes that participate in the edge:

```
for (i in 1:nrow(edges_timesteps)) {
    # Check if the raster ID is equal in node_to and node_from
    if (df_IDs$raster_id[df_IDs$node_to == edges_timesteps[i, ]$node_to] == df_IDs$raster_id[df_IDs$node
        edges_timesteps[i, ]$node_from]) {
        ## If that is the case just add the cost once
        edges_timesteps$edge_cost[i] <- df_cost$edge_cost[df_cost$cell_id == edges_timesteps[i,
            ]$node_from]
    }
    # Check if the raster ID is Different in node_to and node_from
    if (df_IDs$raster_id[df_IDs$node_to == edges_timesteps[i, ]$node_to] != df_IDs$raster_id[df_IDs$node
        edges_timesteps[i, ]$node_from]) {
        ## If that is the case just add the cost twice
```

Table 4: Edgelist of pairs of nodes among times including distance in meters

dist	node_from	node_to
0.0	8	15
110575.7	8	16
110575.7	9	15
0.0	9	16
110574.6	9	17
110574.6	10	16
0.0	10	17
0.0	13	20
110577.3	13	21
110577.3	14	20
0.0	14	21
0.0	15	22
110575.7	15	23
110575.7	16	22
0.0	16	23
110574.6	16	24
110574.6	17	23
0.0	17	24
0.0	20	27
110577.3	20	28
110577.3	21	27
0.0	21	28

```
        edges_timesteps$edge_cost[i] <- df_cost$edge_cost[df_cost$cell_id == edges_timesteps[i,
            ]$node_from] + df_cost$edge_cost[df_cost$cell_id == edges_timesteps[i,
            ]$node_to]
    }
}
```

And finally, we filter those nodes according to the ones that are at distances permited by `Dist` using table 4, we finally get table 5.

```
edges_timesteps <- left_join(edges_timesteps, connections) %>% dplyr::filter(!is.na(dist)) %>%
    dplyr::select(-dist)
```

Table 5: Edgelist of pairs of nodes where habitat is 1 and distance is bellow 11,000

node_from	node_to	timeslice_to	edge_id	edge_capacity	edge_cost
9	15	2	1	1	0.50
9	16	2	2	1	0.50
9	17	2	3	1	0.50
13	21	2	8	1	0.50
14	21	2	12	1	0.25

7 add source and target edges

```
edges_source <- tibble(
  # lowest ids available
  # FIXME: temp workaround
  edge_id = min(edges_timesteps[['edge_id']]),
  node_from = 0,
  # find all timeslice 1 nodes
  node_to = edges_timesteps %>%
    filter(timeslice_to == 2) %>%
    pull(node_from) %>%
    unique(),
  timeslice_to = 1,
  edge_capacity = nchains,
  edge_cost = 1) %>%
  # unique ids!
  mutate(edge_id = edge_id - row_number())

edges_target <- tibble(
  # FIXME: temp workaround
  edge_id = max(edges_timesteps[['edge_id']]),
  node_from = edges_timesteps %>%
    filter(timeslice_to == max(edges_timesteps[['timeslice_to']])) %>%
    pull(node_to) %>%
    unique(),
  node_to = max(edges_timesteps[['node_to']]) + 1,
  timeslice_to = max(edges_timesteps[['timeslice_to']]) + 1,
  edge_capacity = nchains,
  edge_cost = 1) %>%
  # unique ids!
  mutate(edge_id = edge_id + row_number())

edges_formatted <- edges_timesteps %>%
  bind_rows(edges_source, edges_target) %>%
  arrange(edge_id)
```

8 sovler

9 create constraints matrix

```
constraints_matrix <- create_constraints_matrix(edges = edges_formatted, total_flow = nchains)
```

10 run lpSolve to find best solution

```
solution <- lp(direction = "min", objective.in = edges_formatted[["edge_cost"]],
    const.mat = constraints_matrix[["lhs"]], const.dir = constraints_matrix[["dir"]],
    const.rhs = constraints_matrix[["rhs"]])
```

Table 6: Edgelist of pairs of nodes where habitat is 1 and distance is bellow 11,000, including source and target

node_from	node_to	timeslice_to	edge_id	edge_capacity	edge_cost
0	14	1	-2	2	1.00
0	13	1	-1	2	1.00
0	9	1	0	2	1.00
9	15	2	1	1	0.50
9	16	2	2	1	0.50
9	17	2	3	1	0.50
13	21	2	8	1	0.50
14	21	2	12	1	0.25
15	22	3	13	2	1.00
16	22	3	14	2	1.00
17	22	3	15	2	1.00
21	22	3	16	2	1.00

Table 7: Constraints matrix

-2	-1	0	1	2	3	8	12	13	14	15	16	x	x
1	0	0	0	0	0	0	0	0	0	0	0	<=	2
0	1	0	0	0	0	0	0	0	0	0	0	<=	2
0	0	1	0	0	0	0	0	0	0	0	0	<=	2
0	0	0	1	0	0	0	0	0	0	0	0	<=	1
0	0	0	0	1	0	0	0	0	0	0	0	<=	1
0	0	0	0	0	1	0	0	0	0	0	0	<=	1
0	0	0	0	0	0	1	0	0	0	0	0	<=	1
0	0	0	0	0	0	0	1	0	0	0	0	<=	1
0	0	0	0	0	0	0	0	1	0	0	0	<=	2
0	0	0	0	0	0	0	0	0	1	0	0	<=	2
0	0	0	0	0	0	0	0	0	0	1	0	<=	2
0	0	0	0	0	0	0	0	0	0	0	1	<=	2
0	0	1	-1	-1	-1	0	0	0	0	0	0	==	0
0	1	0	0	0	0	-1	0	0	0	0	0	==	0
1	0	0	0	0	0	0	-1	0	0	0	0	==	0
0	0	0	1	0	0	0	0	-1	0	0	0	==	0
0	0	0	0	1	0	0	0	0	-1	0	0	==	0
0	0	0	0	0	1	0	0	0	0	-1	0	==	0
0	0	0	0	0	0	1	1	0	0	0	-1	==	0
0	0	1	0	0	0	0	0	0	0	0	0	<=	1
0	1	0	0	0	0	0	0	0	0	0	0	<=	1
1	0	0	0	0	0	0	0	0	0	0	0	<=	1
0	0	0	1	0	0	0	0	0	0	0	0	<=	1
0	0	0	0	1	0	0	0	0	0	0	0	<=	1
0	0	0	0	0	1	0	0	0	0	0	0	<=	1
0	0	0	0	0	0	1	1	0	0	0	0	<=	1
-1	-1	-1	0	0	0	0	0	0	0	0	0	==	-2
0	0	0	0	0	0	0	0	1	1	1	1	==	2

11 visualize

12 combine with edge information

```
edges_solved <- bind_cols(edges_formatted, solution = solution[["solution"]]) %>%
    left_join(df_IDs) %>% dplyr::filter(solution != 0)


TestStack <- list()


for (i in 1:nlayers(layers_habitat)) {
    Test <- layer_cost
    values(Test) <- 0
    values(Test)[edges_solved %>% dplyr::filter(timeslice_to == (i)) %>% pull(raster_id) %>%
        unique()] <- edges_solved %>% dplyr::filter(timeslice_to == (i)) %>% group_by(raster_id) %>%
        summarise(flow = sum(solution)) %>% pull(flow)
    TestStack[[i]] <- Test
}

TestStack <- do.call("stack", TestStack)

names(TestStack) <- paste0("T", 1:nlayers(TestStack))
```

Table 8: Solution

node_from	node_to	timeslice_to	edge_id	edge_capacity	edge_cost	solution	raster_id
0	14	1	-2	2	1.00	1	7
0	9	1	0	2	1.00	1	2
9	15	2	1	1	0.50	1	1
14	21	2	12	1	0.25	1	7
15	22	3	13	2	1.00	1	NA
21	22	3	16	2	1.00	1	NA

12.1 Quadriatic solution for the same

```
##
## Using solver 'cccp' with parameters:
##          Value
## trace        0
## abstol   1e-06
## feastol  1e-05
## stepadj    0.9
## maxiters   100
## reltol   1e-06
## beta       0.5
```

```
edges_solved <- bind_cols(edges_formatted, solution = res$x) %>% left_join(df_IDs) %>%
    dplyr::filter(solution != 0) %>% group_by(node_to, raster_id, timeslice_to) %>%
    summarise(solution = sum(solution))
```

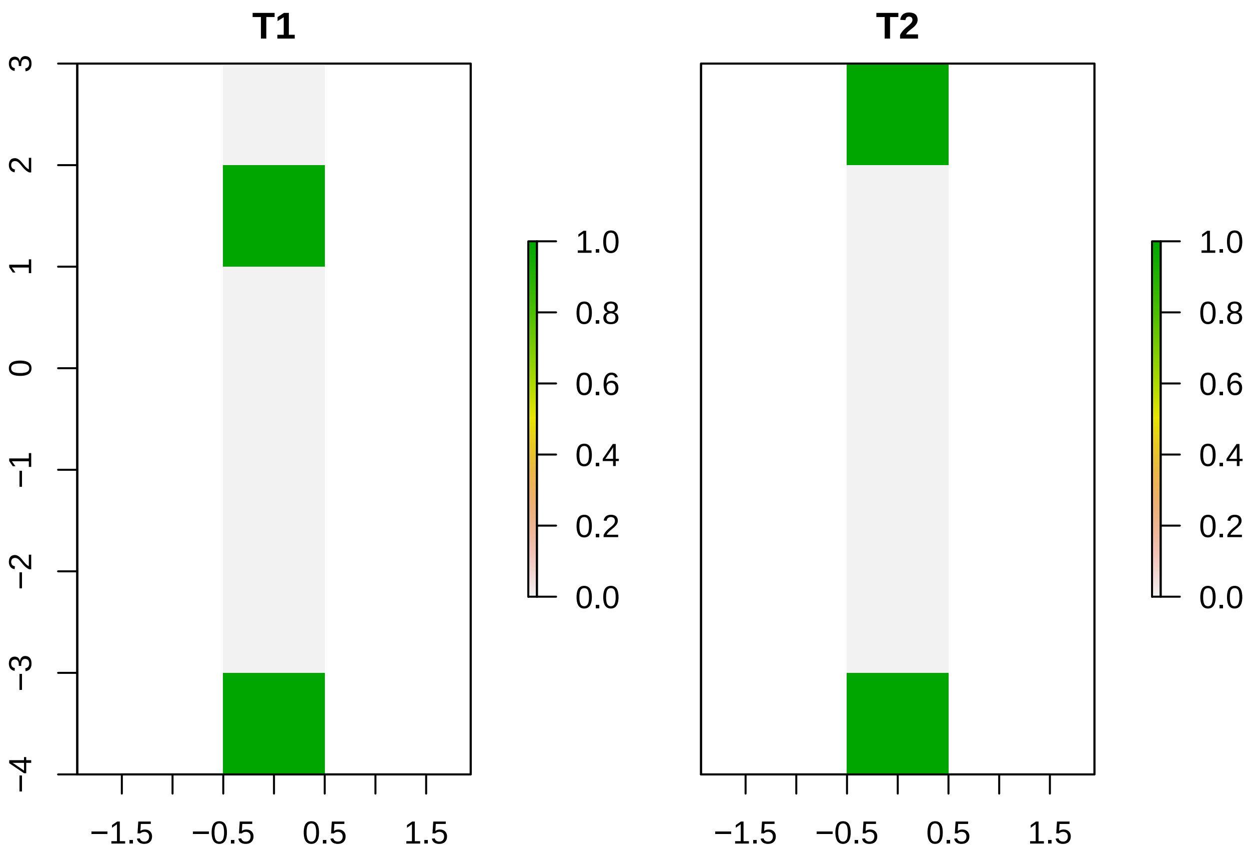

Figure 4: Time slice by time slice solution

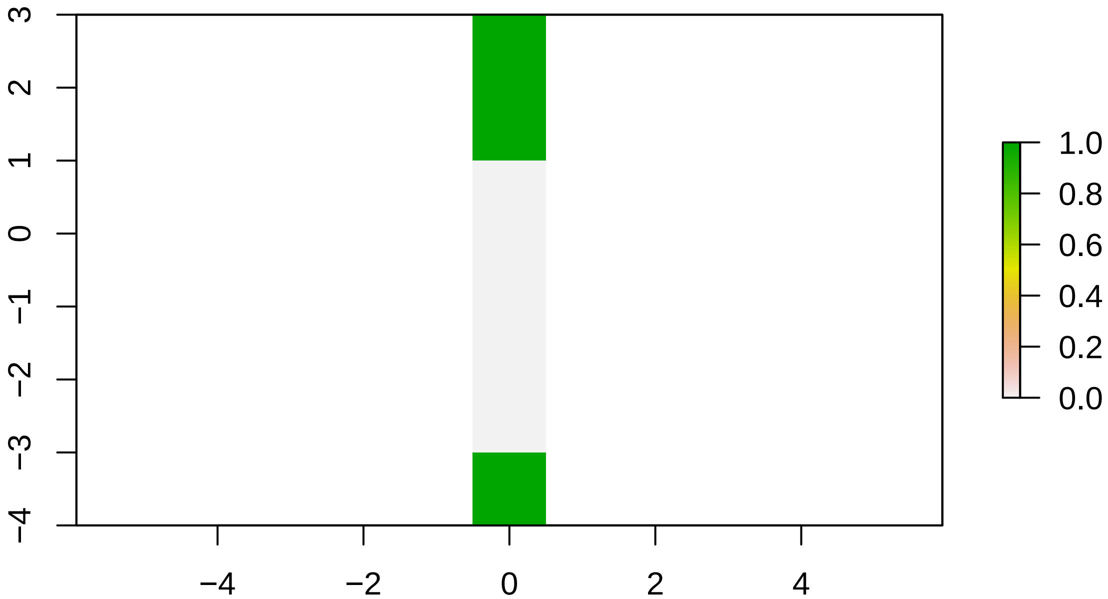

Figure 5: Global solution

```r
TestStack <- list()


for (i in 1:nlayers(layers_habitat)) {
    Test <- layer_cost
    values(Test) <- 0
    values(Test)[edges_solved %>% dplyr::filter(timeslice_to == (i)) %>% pull(raster_id) %>%
        unique()] <- edges_solved %>% dplyr::filter(timeslice_to == (i)) %>% group_by(raster_id) %>%
        summarise(flow = sum(solution)) %>% pull(flow)
    TestStack[[i]] <- Test
}

TestStack <- do.call("stack", TestStack)

names(TestStack) <- paste0("T", 1:nlayers(TestStack))
```

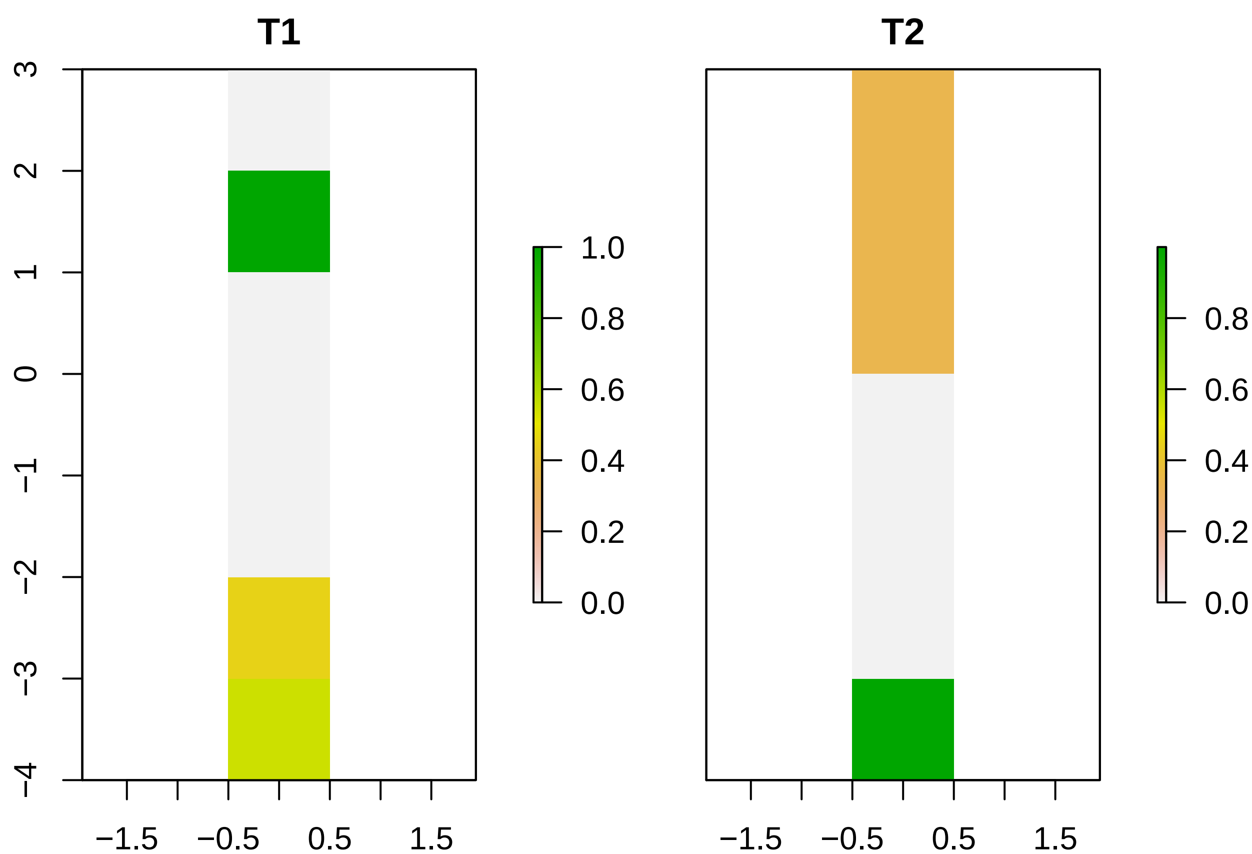

Figure 6: Time slice by time slice quadratic solution

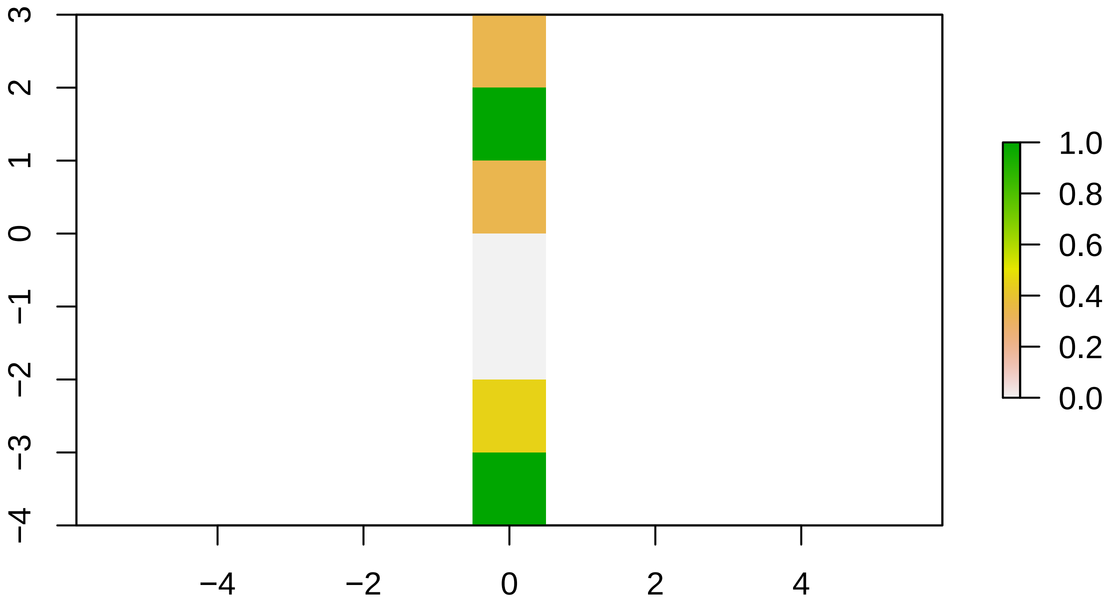

Figure 7: Global quadratic solution

Table 9: Solution

node_to	raster_id	timeslice_to	solution
9	2	1	1.0000000
13	6	1	0.4545454
14	7	1	0.5454546
15	1	2	0.3333333
16	2	2	0.3333333
17	3	2	0.3333333
21	7	2	1.0000000
22	NA	3	2.0000000