Designing a bird movement algorithm

Load the data

The data has all been saved in an object called test. The distances are between the centroids of the patches. An object called "grat" is a spatialPolygonsDataframe that can be useful for plotting. However there can be problems when merging data directly with this object if some polygons are not included or if the data frame is sorted. Therefore it is preferable to use the sites object that contains the same information with the centroids as x and y coordinates.

```
library(shiny)
library(rgdal)
library(plotly)
library(ggplot2)
library(dismo)
library(dplyr)
load("/home/rstudio/morph/data/test.rob")
map<-gmap(grat,type="satellite")
ls()
```

```
## [1] "clim"  "dist"  "grat"  "map"   "sites" "tides"
```

The dist object

This contains the distances in meters between each patch defined by an rid identifying number. The size if this data frame could be reduced by setting a maximum serach distace around any patch.

```
search_dist<-subset(dist,dist$dist_m<1200)
head(search_dist)
```

```
##     rid rid2   dist_m
## 1   251  251   0.0000
## 17  251  253 871.9961
## 555 106  106   0.0000
## 571 106  119 923.0217
## 781 106  102 870.6276
## 926 106  105 923.0217
```

The sites object

This is a wide data frame with the information added from the database.

```
head(sites)
```

```
##           x       y geom  area_m2       lon      lat        min        q10
## 1 -18134280 7424290    0 805904.0 -162.9030 55.32035 -0.6522903  0.8248394
## 2 -18118072 7435016    0 803687.1 -162.7574 55.37514 -7.4345889 -4.1644136
## 3 -18114830 7435016    0 803687.1 -162.7283 55.37514 -5.4493937 -2.6039047
## 4 -18118072 7422758    0 806221.1 -162.7574 55.31252 -0.7500000 -0.7500000
## 5 -18129417 7424290    0 805904.0 -162.8593 55.32035 -1.5000000 -1.5000000
## 6 -18182903 7375258    0 816097.4 -163.3398 55.06894 -4.7019830 -3.6682689
##         q25    median        mean        q75        q90        max rid
## 1  1.2200000  1.2200000  3.12495249  5.72753334  7.21999979 9.22000027 251
```

```
## 2 -1.8759976 -0.7145216 -1.44510625 -0.41785118 -0.24114663 0.08396664 106
## 3 -0.6711487 -0.2930432 -0.68108442 -0.06607303  0.07590192 0.90033245 120
## 4 -0.5852669 -0.1052567  0.01198997  0.50228344  0.86519202 4.76334715 285
## 5 -0.5131581  0.5526265  0.10859742  0.82338627  0.89244504 0.91000003 264
## 6 -3.2094705 -2.0397470 -2.11956999 -1.10364985 -0.53587186 0.16633394 497
##   psuitable median_biomass mean_biomass median_shootlength
## 1         0          134.5     130.7083               18.0
## 2        10          227.0     212.9032               63.5
## 3        40          214.0     200.9912               56.0
## 4         0          215.0     183.2780               52.0
## 5         0          125.5     131.1867               35.0
## 6         0          199.0     172.3410               96.0
##   mean_shootlength station
## 1         13.16257       1
## 2         65.64624       1
## 3         63.62000       1
## 4         49.13118       1
## 5         46.04217       1
## 6         92.67444       4
```

The tides dataframe

```r
head(tides)
```

```
##   station  name                time       ht
## 1       1 Grant 2016-01-01 01:00:00 1.473570
## 2       1 Grant 2016-01-01 02:00:00 1.312514
## 3       1 Grant 2016-01-01 03:00:00 1.104959
## 4       1 Grant 2016-01-01 04:00:00 0.909518
## 5       1 Grant 2016-01-01 05:00:00 0.710994
## 6       1 Grant 2016-01-01 06:00:00 0.554561
```

The climate data frame

On any given day the birds can find out the maximum and minimum temperature and windspeed.

```r
head(clim)
```

```
##         date avwind tmin tmax
## 1 1984-01-01     92   -6   28
## 2 1984-01-02     67  -72   -6
## 3 1984-01-03     78  -89  -50
## 4 1984-01-04     71  -89   28
## 5 1984-01-05     92    0   28
## 6 1984-01-06    160   22   44
```

Functions

A utility function to make a standardised Posix timestamp from year, month and day. Time can be advanced in seconds, so add 60*60 to move on an hour

```
FMakeTime<-function(year=2016,month=1,day=1,hr=1){
  tm<-sprintf("%04d-%02d-%02d %02d:00:00",year,month,day,hr)
  tm<-as.POSIXct(tm)
  tm
}
tm<-FMakeTime()
tm
```

```
## [1] "2016-01-01 01:00:00 UTC"
```

```
tm+60*60
```

```
## [1] "2016-01-01 02:00:00 UTC"
```

Change water depth

This is a key function in this context. It takes the quantile depths as calculated in the database into account. In the present version the quantiles include are q10, q25, q50 (median), q75 and q90. This could be changed by altering the function in the database.

R passes by reference and problems can sometimes arise if objects within a function have the same names as objects in the global environment. It is also a bad idea to change objects in the global environment directly within functions. So I will preface objects passed to a function from another environment with an f to mean the local (function) version.

```
FSuitable<-function(fsites=sites,ftm=tm,ftides=tides,depth=-1,height=1){
  current_tide<-subset(ftides,ftides$time==ftm)
  d<-merge(fsites,current_tide)
  tide<-d$ht
  depth<-depth+tide
  height<-height+tide
  dd<-cbind(d$min,d$q10,d$q25,d$median,d$q75,d$q90,d$max,depth,height)
  f<-function(x)
  {
    q<-c(0,10,25,50,75,90,100)
    qs<-x[1:7]
    depth<-x[8]
    height<-x[9]
    x2<-q[qs>=depth&qs<=height]
    # The supress warnings are needed as the vector may be of zero
    # length. This also leads to results of -inf instead of zero
    suppressWarnings(x2<-max(x2,na.rm=TRUE)-min(x2,na.rm=TRUE))
    if(is.na(x2))x2<-0
    if(x2==-Inf)x2<-0
    x2
  }
  fsites$psuitable<-apply(dd,1,f)
  fsites
}
```

```
tm<-FMakeTime(2016,3,1)
sites<-FSuitable(fsites=sites,ftm=tm,ftides=tides,depth=-1,height=1)
head(sites)
```

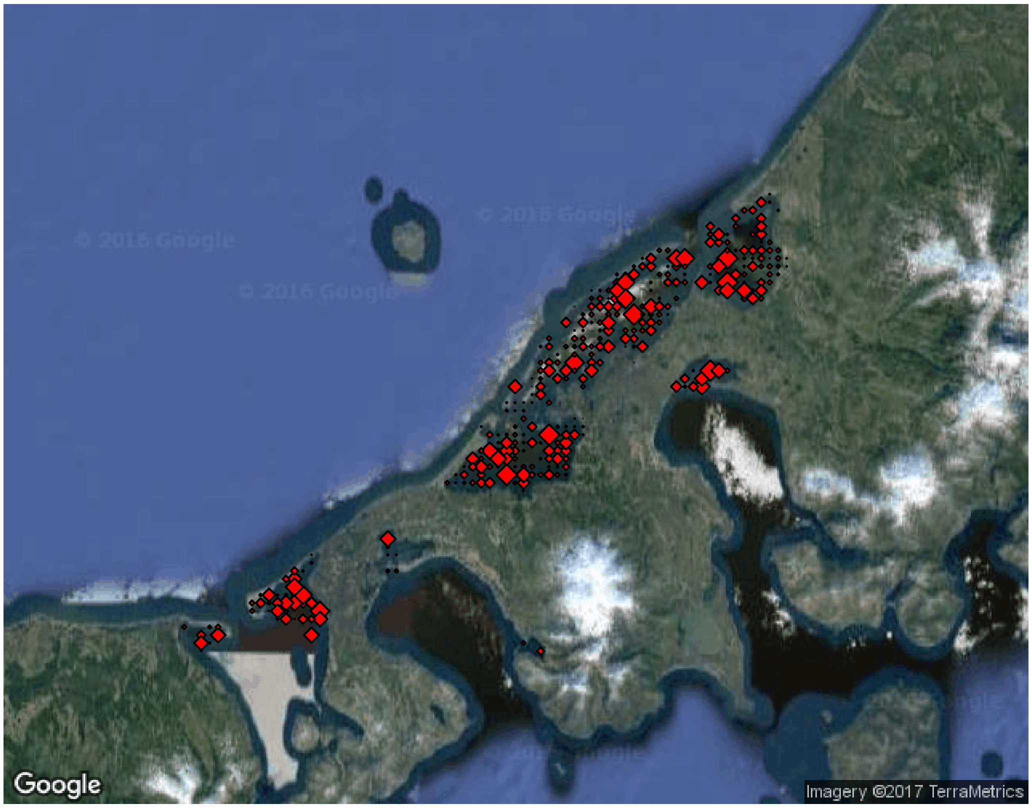
```
##           x       y geom  area_m2       lon      lat        min        q10
## 1 -18134280 7424290    0 805904.0 -162.9030 55.32035 -0.6522903  0.8248394
```

```
## 2 -18118072 7435016    0 803687.1 -162.7574 55.37514 -7.4345889 -4.1644136
## 3 -18114830 7435016    0 803687.1 -162.7283 55.37514 -5.4493937 -2.6039047
## 4 -18118072 7422758    0 806221.1 -162.7574 55.31252 -0.7500000 -0.7500000
## 5 -18129417 7424290    0 805904.0 -162.8593 55.32035 -1.5000000 -1.5000000
## 6 -18182903 7375258    0 816097.4 -163.3398 55.06894 -4.7019830 -3.6682689
##          q25     median        mean         q75         q90        max rid
## 1  1.2200000  1.2200000  3.12495249  5.72753334  7.21999979 9.22000027 251
## 2 -1.8759976 -0.7145216 -1.44510625 -0.41785118 -0.24114663 0.08396664 106
## 3 -0.6711487 -0.2930432 -0.68108442 -0.06607303  0.07590192 0.90033245 120
## 4 -0.5852669 -0.1052567  0.01198997  0.50228344  0.86519202 4.76334715 285
## 5 -0.5131581  0.5526265  0.10859742  0.82338627  0.89244504 0.91000003 264
## 6 -3.2094705 -2.0397470 -2.11956999 -1.10364985 -0.53587186 0.16633394 497
##   psuitable median_biomass mean_biomass median_shootlength
## 1        40          134.5     130.7083               18.0
## 2         0          227.0     212.9032               63.5
## 3        10          214.0     200.9912               56.0
## 4        15          215.0     183.2780               52.0
## 5        50          125.5     131.1867               35.0
## 6        25          199.0     172.3410               96.0
##   mean_shootlength station
## 1         13.16257       1
## 2         65.64624       1
## 3         63.62000       1
## 4         49.13118       1
## 5         46.04217       1
## 6         92.67444       4
```

Example map

```r
plot(map)
```

```r
eelgrass<-(sites$psuitable/100)*(sites$median_biomass/200)
points(sites$x,sites$y,cex=eelgrass,pch=23,bg="red")
```



Add some birds

Birds can arrive with their properties already set through loading from a file. However for testing we'll set up a simple way of adding them.

A range of functions for deriving energy from food and losing it through metabilism will have to be added. However these are comparatively simple functions providing food supply, temperature and activity levels are known.

```
FArriveBirds<-function(ftm=tm,nbirds= 10,fsites=sites)
  {
  wt<-rnorm(nbirds,mean=1.5,sd=0.2) ##Change this later
  ## Add other properties here
  ##
  rid<-sample(fsites$rid,nbirds,replace=TRUE) ## Place them at random
  bid<-1:nbirds ## ID number
  birds<-data.frame(bid,arrive_time=ftm,weight=wt,rid=rid)
  birds
}
```

```
birds<-FArriveBirds(tm,3,sites)
head(birds,3)
```

```
##   bid         arrive_time   weight rid
## 1   1 2016-03-01 01:00:00 1.672112 102
## 2   2 2016-03-01 01:00:00 1.476980 294
## 3   3 2016-03-01 01:00:00 1.317200  23
```

Add birds

```
FAddBirds<-function(ftm=tm,nbirds= 10,fsites=sites,fbirds=birds)
  {
  newbirds<-FArriveBirds(ftm,nbirds,fsites)
  newbirds$bid<-newbirds$bid+max(fbirds$bid)
  rbind(fbirds,newbirds)
}
```

```
birds<-FAddBirds(tm+60*60,3,sites,birds)
birds
```

```
##   bid         arrive_time   weight rid
## 1   1 2016-03-01 01:00:00 1.672112 102
## 2   2 2016-03-01 01:00:00 1.476980 294
## 3   3 2016-03-01 01:00:00 1.317200  23
## 4   4 2016-03-01 02:00:00 1.399493 189
## 5   5 2016-03-01 02:00:00 1.655519  60
## 6   6 2016-03-01 02:00:00 1.479548 211
```

Tell birds where they are

Because sites and birds have one column with the same name that identifies the site all that is needed to provide them with the site properties is to merge the two dataframes using the defaults. Merging is achieved in R through lazy evaluation so this is very fast. Once the birds know which site they are on and the properties of that site the changes to both the birds state and the sites are very easy to implement using simple functions. The most challenging function to implement and to optimise for speed is bird movement. The function should be vectorised rather than looped as this dramatically speeds up the calculations by two to three orders of magnitude. It should be simple to understand in order to test that it does what it should and alter it to allow for new ideas.

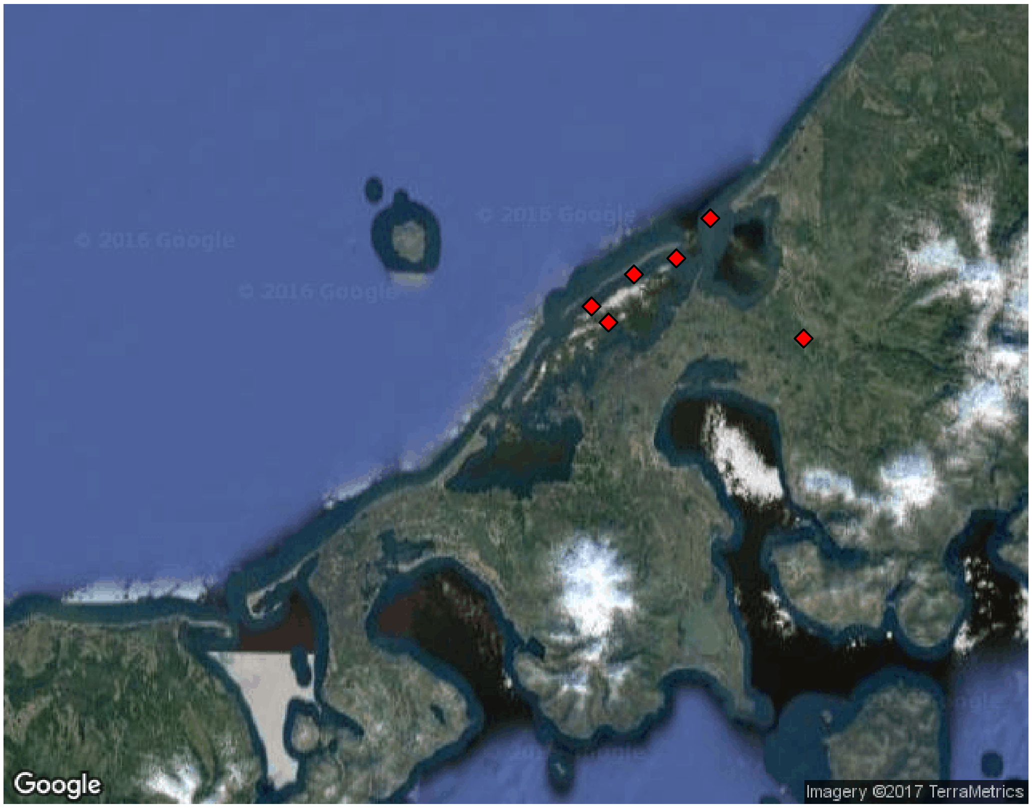
```
birds_sites<-merge(birds,sites)
head(birds_sites)
```

```
##   rid bid         arrive_time   weight         x       y geom  area_m2
## 1  23   3 2016-03-01 01:00:00 1.317200 -18103485 7447274    0 801159.0
## 2  60   5 2016-03-01 02:00:00 1.655519 -18109968 7439613    0 802738.3
## 3 102   1 2016-03-01 01:00:00 1.672112 -18118072 7436548    0 803370.7
## 4 189   4 2016-03-01 02:00:00 1.399493 -18126176 7430419    0 804636.6
## 5 211   6 2016-03-01 02:00:00 1.479548 -18122934 7427355    0 805270.1
## 6 294   2 2016-03-01 01:00:00 1.476980 -18085656 7424290    0 805904.0
##         lon      lat        min        q10        q25     median
## 1 -162.6264 55.43766  1.2200000  1.2200000  1.2200000  1.2249516
## 2 -162.6846 55.39860 -3.9626682 -3.3010278 -2.2829657 -1.2247925
## 3 -162.7574 55.38296 -0.4140983  0.1086742  0.1997935  0.4093660
## 4 -162.8302 55.35167 -0.3304857  0.3098453  0.6447738  0.8313487
## 5 -162.8011 55.33602  0.1975188  0.2303350  0.2579228  0.3444510
## 6 -162.4662 55.32035  4.2199998  4.2199998  4.2199998  4.2199998
##         mean        q75        q90        max psuitable median_biomass
## 1  1.2300283  1.2381527  1.2479258 1.27678299       100             NA
## 2 -1.5439362 -0.8085825 -0.4251881 0.02523129        90            216
## 3  0.5169547  0.7122958  0.9437940 6.21999979        80            154
## 4  0.7954791  1.0463901  1.1886943 1.29853976        90             89
## 5  0.3922216  0.4957069  0.6478341 0.81173319       100            159
```

```
## 6  4.3701480  4.2199998  4.7519135 7.21999979        80             NA
##   mean_biomass median_shootlength mean_shootlength station
## 1          NaN                 17         17.41948       1
## 2    205.24752                 80         87.86142       1
## 3    140.07143                 39         37.06452       1
## 4     89.59005                 28         28.44667       1
## 5    149.72778                 40         38.76111       1
## 6          NaN                  0          0.00000       1
```

```r
plot(map)
```

```r
points(birds_sites$x,birds_sites$y,pch=23,bg="red")
```



Moving birds to best patch

The patches within reach of any other patch are defined in the distance object. If we define a scoring rule for the patches that can be translated into an index we can move the birds onto the best one within range. If we want to prevent the patches filling with birds we could move some first, then re-calculate the desirability by reducing the scores according to the number of birds, then move some more. This keeps the operation vectorised rather than looping through each bird in turn. Moving the birds has the potential to slow the model down dramatically due to the inclusion of the distance matrix.

For the moment we'll try just moving them all together, but the same function could be used to move a subset of birds first and then more later in the same time step.

A scoring rule

Just try something very simple first.

```
FValueSites<-function(fsites=sites)
{
  ## Make a rule for calculating value of site
  # for feeding
  ## Make it the amount of biomass times proportion available
  fsites$value<-fsites$psuitable/100*fsites$mean_biomass
  fsites$value[is.na(fsites$value)]<-0
  fsites
}
```

```
sites<-FValueSites(sites)
head(sites)
```

```
##             x       y geom  area_m2       lon      lat        min        q10
## 1 -18134280 7424290    0 805904.0 -162.9030 55.32035 -0.6522903  0.8248394
## 2 -18118072 7435016    0 803687.1 -162.7574 55.37514 -7.4345889 -4.1644136
## 3 -18114830 7435016    0 803687.1 -162.7283 55.37514 -5.4493937 -2.6039047
## 4 -18118072 7422758    0 806221.1 -162.7574 55.31252 -0.7500000 -0.7500000
## 5 -18129417 7424290    0 805904.0 -162.8593 55.32035 -1.5000000 -1.5000000
## 6 -18182903 7375258    0 816097.4 -163.3398 55.06894 -4.7019830 -3.6682689
##          q25     median        mean         q75         q90        max rid
## 1  1.2200000  1.2200000  3.12495249  5.72753334  7.21999979 9.22000027 251
## 2 -1.8759976 -0.7145216 -1.44510625 -0.41785118 -0.24114663 0.08396664 106
## 3 -0.6711487 -0.2930432 -0.68108442 -0.06607303  0.07590192 0.90033245 120
## 4 -0.5852669 -0.1052567  0.01198997  0.50228344  0.86519202 4.76334715 285
## 5 -0.5131581  0.5526265  0.10859742  0.82338627  0.89244504 0.91000003 264
## 6 -3.2094705 -2.0397470 -2.11956999 -1.10364985 -0.53587186 0.16633394 497
##   psuitable median_biomass mean_biomass median_shootlength
## 1        40          134.5     130.7083               18.0
## 2         0          227.0     212.9032               63.5
## 3        10          214.0     200.9912               56.0
## 4        15          215.0     183.2780               52.0
## 5        50          125.5     131.1867               35.0
## 6        25          199.0     172.3410               96.0
##   mean_shootlength station    value
## 1         13.16257       1 52.28333
## 2         65.64624       1  0.00000
## 3         63.62000       1 20.09912
## 4         49.13118       1 27.49170
## 5         46.04217       1 65.59337
## 6         92.67444       4 43.08524
```

Setting up the possible moves

If we merge just the unique rid's of the patches with birds on with the distances data frame we get an object with all the possible rid2s (destinations) within the search range.

If we then use the rid2s as an index we can find the values of the resource on these patches.

```r
search_dist<-subset(dist,dist$dist_m<1200) ## Reduce the number of options to within a search distance

bird_positions<-data.frame(rid=unique(birds_sites$rid))
bird_moves<-merge(bird_positions,search_dist)
destination_value<-data.frame(rid2=sites$rid,value=sites$value)
bird_moves<-merge(bird_moves,destination_value)
head(bird_moves)
```

```
##   rid2 rid    dist_m    value
## 1   20  23 869.2601  0.00000
## 2   23  23   0.0000  0.00000
## 3   24  23 921.5657  0.00000
## 4   27  23 869.4310 94.14000
## 5   57  60 922.4755 19.11078
## 6   59  60 870.1147 15.03857
```

Rank the moves

Use a tapply to group the moves according to the rid at the place of origin then rank them from each destination.

```
bird_moves<-bird_moves[order(bird_moves$rid),]
set.seed(1)
 f<-function(x)rank(-x,ties.method= "random")
  bird_moves$rank<-unlist(tapply(bird_moves$value,bird_moves$rid,f))
  bird_moves
```

```
##    rid2 rid    dist_m     value rank
## 1    20  23 869.2601   0.00000    2
## 2    23  23   0.0000   0.00000    3
## 3    24  23 921.5657   0.00000    4
## 4    27  23 869.4310  94.14000    1
## 5    57  60 922.4755  19.11078    4
## 6    59  60 870.1147  15.03857    5
## 7    60  60   0.0000 184.72277    2
## 8    61  60 922.4755 208.33898    1
## 14  127  60 870.2856  87.70626    3
## 9    98 102 870.4566  27.33688    3
## 10  101 102 922.8396  44.66742    2
## 11  102 102   0.0000 112.05714    1
## 12  106 102 870.6276   0.00000    5
## 13  115 102 922.8396   0.00000    4
## 15  185 189 871.1407  46.24877    4
## 16  188 189 923.5680   0.00000    5
## 17  189 189   0.0000  80.63105    2
## 18  190 189 923.5680  97.23952    1
## 19  193 189 871.3117  47.54540    3
## 20  198 211 923.9323  63.75671    3
## 21  207 211 871.4828  51.47750    4
## 22  211 211   0.0000 149.72778    1
## 23  212 211 923.9323   0.00000    5
## 24  276 211 871.6539 110.03000    2
## 25  293 294 871.8250   0.00000    2
## 26  294 294   0.0000   0.00000    3
## 27  299 294 924.2966   0.00000    1
```

We could now just take the best move and assign the new rid to all the birds.

```
bird_moves<-subset(bird_moves,bird_moves$rank==1)
bird_moves<-merge(birds,bird_moves)
bird_moves
```

```
##   rid bid         arrive_time   weight rid2   dist_m     value rank
## 1  23   3 2016-03-01 01:00:00 1.317200   27 869.4310  94.14000    1
## 2  60   5 2016-03-01 02:00:00 1.655519   61 922.4755 208.33898    1
## 3 102   1 2016-03-01 01:00:00 1.672112  102   0.0000 112.05714    1
## 4 189   4 2016-03-01 02:00:00 1.399493  190 923.5680  97.23952    1
## 5 211   6 2016-03-01 02:00:00 1.479548  211   0.0000 149.72778    1
## 6 294   2 2016-03-01 01:00:00 1.476980  299 924.2966   0.00000    1
```

Note that the birds now know their new rid (rid2) and the distance they need to move to it. There are some extra columns that need removing to obtain a new birds data frame which is identical to the original, but with an updated rid for the site on which they are on.

```
bird_moves$rid<-bird_moves$rid2
keep_columns<-1:dim(birds)[2]
birds<-bird_moves[,keep_columns]
birds
```

```
##   rid bid         arrive_time   weight
## 1  27   3 2016-03-01 01:00:00 1.317200
## 2  61   5 2016-03-01 02:00:00 1.655519
## 3 102   1 2016-03-01 01:00:00 1.672112
## 4 190   4 2016-03-01 02:00:00 1.399493
## 5 211   6 2016-03-01 02:00:00 1.479548
## 6 299   2 2016-03-01 01:00:00 1.476980
```

Make a bird move function

This work flow is quite simple to follow and so should be robust. It now needs testing with more birds over greater distances. The steps can be rolled up into a function first.

```
FMoveBirds<-function(fbirds=birds,fsites=sites,fdist=dist,search_distance=1200)
{
  ##Set the search distance
  fdist<-subset(fdist,fdist$dist_m<search_distance)
  # Merge the bird data frame to sites to get the values at the sites
  birds_sites<-merge(fbirds,fsites)
  # Take only the sites with birds
  bird_positions<-data.frame(rid=unique(birds_sites$rid))
  # Get all the possible moves from these sites by merging
  bird_moves<-merge(bird_positions,fdist)
  # Find the value of the index used for choosing the site at the destinations
  destination_value<-data.frame(rid2=fsites$rid,value=fsites$value)
  # Add this to the object used for evaluating the moves
  bird_moves<-merge(bird_moves,destination_value)
  ## the next two lines not really needed, Used in testing
  ## They order the object and set a seed for the random choice
  bird_moves<-bird_moves[order(bird_moves$rid),]
  set.seed(1)
  ###
  # A function to rank the values. Ties are assigned at random.
    f<-function(x)rank(-x,ties.method= "random")
  ## Now rank all the moves, grouping by point of origin.
  bird_moves$rank<-unlist(tapply(bird_moves$value,bird_moves$rid,f))
  # Take only the best
  bird_moves<-subset(bird_moves,bird_moves$rank==1)
  # Merge the movements with the birds data frame so that the birds know where they are going to.
  bird_moves<-merge(fbirds,bird_moves)
  # Assign the new rids to the birds
  bird_moves$rid<-bird_moves$rid2
  # Get rid of the extra columns in the dataframe to return it to the old state
  keep_columns<-1:dim(fbirds)[2]
  fbirds<-bird_moves[,keep_columns]
  fbirds
}
```

```r
FMoveBirds(birds,sites,dist)
```

```
##   rid bid         arrive_time   weight
## 1  27   3 2016-03-01 01:00:00 1.317200
## 2  61   5 2016-03-01 02:00:00 1.655519
## 3 102   1 2016-03-01 01:00:00 1.672112
## 4 203   4 2016-03-01 02:00:00 1.399493
## 5 211   6 2016-03-01 02:00:00 1.479548
## 6 301   2 2016-03-01 01:00:00 1.476980
```

Making it more serious

Try adding 40000 birds

```r
birds<-FAddBirds(tm+60*60,40000,sites,birds)
system.time(birds<-FMoveBirds(birds,sites,dist,search_distance=1200))
```

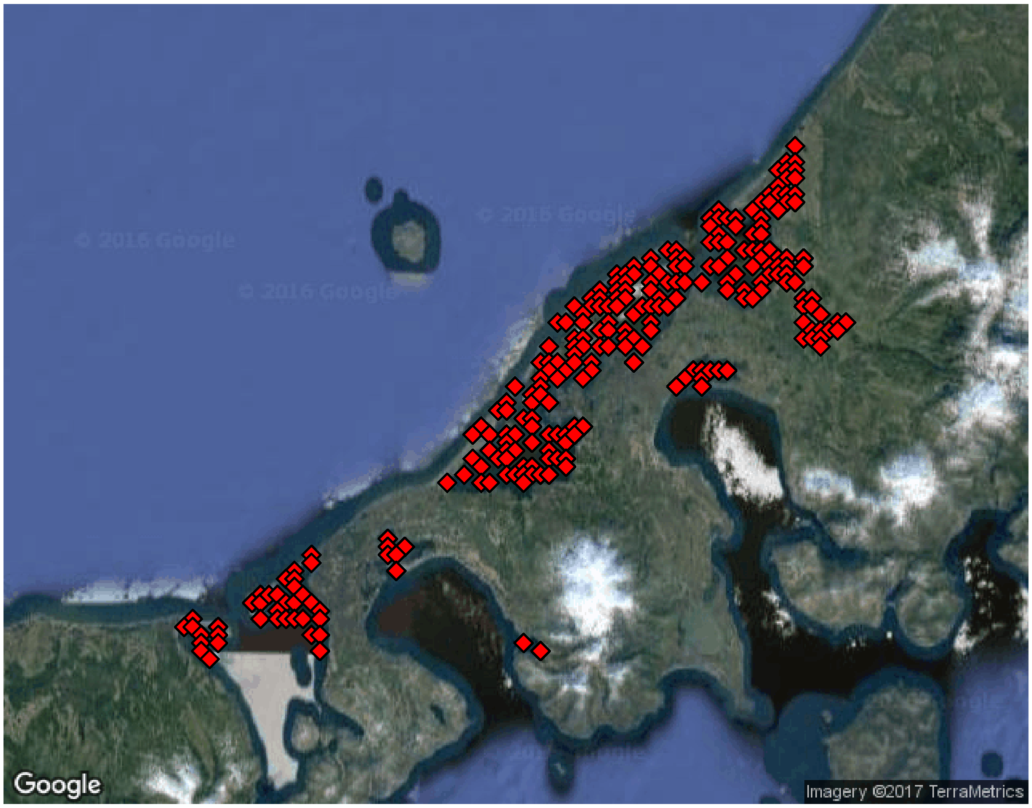
```
##    user  system elapsed
##   0.549   0.000   0.550
```

```r
birds %>%
group_by(rid) %>%
summarise(n())
```

```
## # A tibble: 257 × 2
##      rid `n()`
##    <int> <int>
## 1      1    68
## 2      3   153
## 3      4    77
## 4      7   143
## 5      8    63
## 6      9   142
## 7     10    58
## 8     12    75
## 9     13    62
## 10    14    83
## # ... with 247 more rows
```

```r
birds_sites<-merge(birds,sites)
plot(map)
```

```r
points(birds_sites$x,birds_sites$y,pch=23,bg="red")
```



Now widen search distance to 50 km

```r
system.time(birds<-FMoveBirds(birds,sites,dist,search_distance=50000))
```

```
##    user  system elapsed
##   1.805   0.000   1.809
```

Now as most of the map is within range the birds all go to the two best sites available.

I'm very happy with the calculation time for this. Because the operation is site based rather than bird based and only looks at possible options for sites which are occupied it will speed up once all the birds are in the same place. So it is feasible to iterate the optimality criteria to include crowding and so move the birds several times in each time step to make adjustments while keeping within the target of less than 2 seconds per time step to ensure that a six month model run completes in less than three hours.

Watch the speed up in a second iteration once the number of sites has reduced to two.

```r
system.time(birds<-FMoveBirds(birds,sites,dist,search_distance=2000))
```

```
##    user  system elapsed
##   0.256   0.000   0.256
```

```r
birds_sites<-merge(birds,sites)
plot(map)
```

```
points(birds_sites$x,birds_sites$y,pch=23,bg="red")
```

Time with half a million birds

```
birds<-FAddBirds(tm+60*60,500000,sites,birds)
system.time(birds<-FMoveBirds(birds,sites,dist,search_distance=1200))
```

```
##    user  system elapsed
##   5.352   0.084   5.445
```

Slowing down, but still a reasonable time. However there should never be any need to use such a large number of individuals. In fact there is probably no need to ever use many more individuals than there are sites if each individual represents a super individual.