Map overlay and
spatial aggregation in sp

Edzer Pebesma*

June 3, 2022

Abstract

Numerical "map overlay" combines spatial features from one map layer with the attribute (numerical) properties of another. This vignette explains the R method "over", which provides a consistent way to retrieve indices or attributes from a given spatial object (map layer) at the locations of another. Using this, the R generic "aggregate" is extended for spatial data, so that any spatial properties can be used to define an aggregation predicate, and any R function can be used as aggregation function.

Contents

1 Introduction

According to the free e-book by Davidson (2008),

---
*Institute for Geoinformatics, University of Muenster, Weseler Strasse 253, 48151 Münster, Germany. edzer.pebesma@uni-muenster.de

> *An overlay is a clear sheet of plastic or semi-transparent paper. It is used to display supplemental map and tactical information related to military operations. It is often used as a supplement to orders given in the field. Information is plotted on the overlay at the same scale as on the map, aerial photograph, or other graphic being used. When the overlay is placed over the graphic, the details plotted on the overlay are shown in their true position.*

This suggests that *map overlay* is concerned with combining two, or possibly more, map layers by putting them on top of each other. This kind of overlay can be obtained in R e.g. by plotting one map layer, and plotting a second map layer on top of it. If the second one contains polygons, transparent colours can be used to avoid hiding of the first layer. When using the `spplot` command, the `sp.layout` argument can be used to combine multiple layers.

O'Sullivan and Unwin (2003) argue in chapter 10 (Putting maps together: map overlay) that map overlay has to do with the combination of two (or more) maps. They mainly focus on the combination of the selection criteria stemming from several map layers, e.g. finding the deciduous forest area that is less than 5 km from the nearest road. They call this *boolean overlays.*

One could look at this problem as a polygon-polygon overlay, where we are looking for the intersection of the polygons with the deciduous forest with the polygons delineating the area less than 5 km from a road. Other possibilities are to represent one or both coverages as grid maps, and find the grid cells for which both criteria are valid (grid-grid overlay). A third possibility would be that one of the criteria is represented by a polygon, and the other by a grid (polygon-grid overlay, or grid-polygon overlay). In the end, as O'Sullivan and Unwin argue, we can overlay any spatial type (points, lines, polygons, pixels/grids) with any other. In addition, we can address spatial attributes (as the case of grid data), or only the geometry (as in the case of the polygon-polygon intersection).

This vignette will explain how the `over` method in package `sp` can be used to compute map overlays, meaning that instead of overlaying maps visually, the digital information that comes from combining two digital map layers is retrieved. From there, methods to *aggregate* (compute summary statistics; Heuvelink and Pebesma, 1999) over a spatial domain will be developed and demonstrated. Pebesma (2012) describes overlay and aggregation for spatio-temporal data.

2   Geometry overlays

We will use the word *geometry* to denote the purely spatial characteristics, meaning that attributes (qualities, properties of something at a particular location) are ignored. With *location* we denote a point, line, polygon or grid cell. Section 3 will discuss how to retrieve and possibly aggregate or summarize attributes found there.

Given two geometries, `A` and `B`, the following equivalent commands

```
> A %over% B
> over(A, B)
```

retrieve the geometry (location) indices of `B` at the locations of `A`. More in particular, an integer vector of length `length(A)` is returned, with `NA` values for locations in `A` not matching with locations in `B` (e.g. those points outside a set of polygons).

Selecting points of `A` *inside* or *on* some geometry B (e.g. a set of polygons) B is done by

```
> A[B,]
```

which is short for

```
> A[!is.na(over(A,B)),]
```

We will now illustrate this with toy data created by

```
> library(sp)
> x = c(0.5, 0.5, 1.0, 1.5)
> y = c(1.5, 0.5, 0.5, 0.5)
> xy = cbind(x,y)
> dimnames(xy)[[1]] = c("a", "b", "c", "d")
> pts = SpatialPoints(xy)
> xpol = c(0,1,1,0,0)
> ypol = c(0,0,1,1,0)
> pol = SpatialPolygons(list(
+         Polygons(list(Polygon(cbind(xpol-1.05,ypol))), ID="x1"),
+         Polygons(list(Polygon(cbind(xpol,ypol))), ID="x2"),
+         Polygons(list(Polygon(cbind(xpol,ypol - 1.0))), ID="x3"),
+         Polygons(list(Polygon(cbind(xpol + 1.0, ypol))), ID="x4"),
+         Polygons(list(Polygon(cbind(xpol+.4, ypol+.1))), ID="x5")
+         ))
```

and shown in figure 1.

Now, the polygons `pol` in which points `pts` lie are

```
> over(pts, pol)

 a  b  c  d
NA  5  5  4
```

As points `b` and `c` touch two overlapping polygons, the output from the previous command does not provide all information about the overlaps, only returning the *last* polygon which touched them (polygon 5 in both cases). The complete information can be retrieved as a list:

```
> over(pts, pol, returnList = TRUE)
```
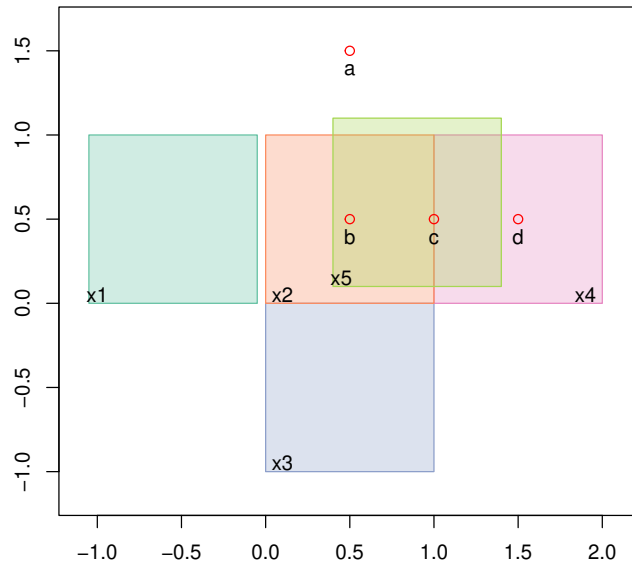
Figure 1: Toy data: points (a-d), and (overlapping) polygons (x1-x5)

```
$a
integer(0)

$b
[1] 2 5

$c
[1] 2 4 5

$d
[1] 4
```

This shows that `over` returns true if geometries in one element touch geometries in another: they do not have to fully overlap (see section 5.2 to constrain the selection criteria). The points falling in or touching any of the polygons are selected by:

```
> pts[pol]

SpatialPoints:
     x    y
```

```
b 0.5 0.5
c 1.0 0.5
d 1.5 0.5
Coordinate Reference System (CRS) arguments: NA
```

The reverse, identical sequence of commands for selecting polygons `pol` that have (one or more) points of `pts` in them is done by

```
> over(pol, pts)

x1 x2 x3 x4 x5
NA  2 NA  3  2

> over(pol, pts, returnList = TRUE)

$x1
integer(0)

$x2
[1] 2 3

$x3
integer(0)

$x4
[1] 3 4

$x5
[1] 2 3

> row.names(pol[pts])

[1] "x2" "x4" "x5"
```

over can also be used to query polygons in a single object overlay each other:

```
> over(pol, pol, returnList = TRUE)

$x1
x1
 1

$x2
x2 x3 x4 x5
 2  3  4  5

$x3
x2 x3 x4
```

```
 2  3  4

$x4
x2 x3 x4 x5
 2  3  4  5

$x5
x2 x4 x5
 2  4  5
```

The output tells us that `x1` does not intersect with any polygons other than itself, within the `pol` object. `x2` intersects with itself and `x3`, `x4` and `x5`, and so on. Note that the *types* of overlap queried by `over` include any intersecting points or edges under the DE-9IM standard. More generic types of spatial overlap can be queried using functions from the rgeos package, as illustrated by the help page launched with `?rgeos::gRelate`.

Constraining polygon-polygon intersections to e.g. *overlap* using `over` is explained in section 5.2.

3 Using `over` to extract attributes

This section shows how `over(x,y)` is used to extract attribute values of argument `y` at locations of `x`. The return value is either an (aggregated) data frame, or a list.

We now create an example `SpatialPointsDataFrame` and a `SpatialPolygonsDataFrame` using the toy data created earlier:

```
> zdf = data.frame(z1 = 1:4, z2=4:1, f = c("a", "a", "b", "b"),
+          row.names = c("a", "b", "c", "d"))
> zdf

  z1 z2 f
a  1  4 a
b  2  3 a
c  3  2 b
d  4  1 b

> ptsdf = SpatialPointsDataFrame(pts, zdf)
> zpl = data.frame(z = c(10, 15, 25, 3, 0), zz=1:5,
+          f = c("z", "q", "r", "z", "q"), row.names = c("x1", "x2", "x3", "x4", "x5"))
> zpl

    z zz f
x1 10  1 z
x2 15  2 q
x3 25  3 r
x4  3  4 z
x5  0  5 q
```

```
> poldf = SpatialPolygonsDataFrame(pol, zpl)
```

In the simplest example

```
> over(pts, poldf)

    z zz    f
a NA NA <NA>
b 15  2    q
c 15  2    q
d  3  4    z
```

a `data.frame` is created with each row corresponding to the first element of the `poldf` attributes at locations in `pts`.

As an alternative, we can pass a user-defined function to process the table (selecting those columns to which the function makes sense):

```
> over(pts, poldf[1:2], fn = mean)

      z         zz
a   NA         NA
b 7.5 3.500000
c 6.0 3.666667
d 3.0 4.000000
```

To obtain the complete list of table entries at each point of `pts`, we use the `returnList` argument:

```
> over(pts, poldf, returnList = TRUE)

$a
[1] z   zz f
<0 rows> (or 0-length row.names)

$b
    z zz f
x2 15  2 q
x5  0  5 q

$c
    z zz f
x2 15  2 q
x4  3  4 z
x5  0  5 q

$d
   z zz f
x4 3  4 z
```

The same actions can be done when the arguments are reversed:

```
> over(pol, ptsdf)

   z1 z2    f
x1 NA NA <NA>
x2  2  3    a
x3 NA NA <NA>
x4  3  2    b
x5  2  3    a

> over(pol, ptsdf[1:2], fn = mean)

    z1  z2
x1  NA  NA
x2 2.5 2.5
x3  NA  NA
x4 3.5 1.5
x5 2.5 2.5
```

4 Lines, and Polygon-Polygon overlays require `rgeos`

Package `sp` provides many of the `over` methods, but not all. Package `rgeos` can compute geometry intersections, i.e. for any set of (points, lines, polygons) to determine whether they have one ore more points in common. This means that the `over` methods provided by package `sp` can be completed by `rgeos` for *any* `over` methods where a `SpatialLines` object is involved (either as `x` or `y`), or where `x` and `y` are both of class `SpatialPolygons` (table 1). For this purpose, objects of class `SpatialPixels` or `SpatialGrid` are converted to `SpatialPolygons`. A toy example combines polygons with lines, created by

```
> l1 = Lines(Line(coordinates(pts)), "L1")
> l2 = Lines(Line(rbind(c(1,1.5), c(1.5,1.5))), "L2")
> L = SpatialLines(list(l1,l2))
```

and shown in figure 2.

The set of `over` operations on the polygons, lines and points is shown below (note that lists and vectors are named in this case):

```
> library(rgeos)
> over(pol, pol)

x1.x1 x2.x2 x3.x2 x4.x2 x5.x2
    1     2     2     2     2

> over(pol, pol,returnList = TRUE)
```

	y: Points	y: Lines	y: Polygons	y: Pixels	y: Grid
x: Points	s	r	s	s	s
x: Lines	r	r	r	r:y	r:y
x: Polygons	s	r	r	s:y	s:y
x: Pixels	s:x	r:x	s:x	s:x	s:x
x: Grid	s:x	r:x	s:x	s:x	s:x

Table 1:   over methods implemented for different x and y arguments. s: provided by **sp**; r: provided by **rgeos**. s:x or s:y indicates that the x or y argument is converted to grid cell center points; r:x or r:y indicate grids or pixels are converted to polygons equal to the grid cell.

```
$x1
x1
 1

$x2
x2 x3 x4 x5
 2  3  4  5

$x3
x2 x3 x4
 2  3  4

$x4
x2 x3 x4 x5
 2  3  4  5

$x5
x2 x4 x5
 2  4  5

> over(pol, L)

x1.NA x2.L1 x3.NA x4.L1 x5.L1
   NA     1    NA     1     1

> over(L, pol)

L1.x2 L2.NA
    2    NA

> over(L, pol, returnList = TRUE)

$L1
x2 x4 x5
 2  4  5
```
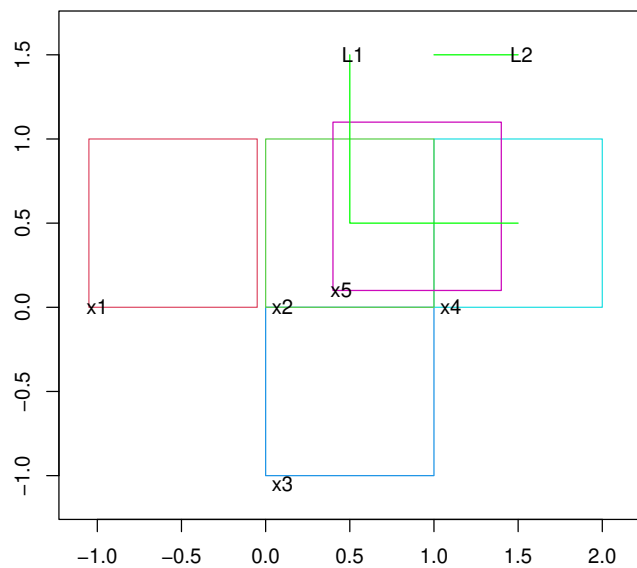
Figure 2: Toy data: two lines and (overlapping) polygons (x1-x5)

```
$L2
named integer(0)

> over(L, L)

L1 L2
 1  2

> over(pts, L)

a b c d
1 1 1 1

> over(L, pts)

 L1.a L2.NA
    1    NA
```

Another example overlays a line with a grid, shown in figure 3.

```
> data(meuse.grid)
> gridded(meuse.grid) = ~x+y
> Pt = list(x = c(178274.9,181639.6), y = c(329760.4,333343.7))
> sl = SpatialLines(list(Lines(Line(cbind(Pt$x,Pt$y)), "L1")))
> image(meuse.grid)
> xo = over(sl, geometry(meuse.grid), returnList = TRUE)
> image(meuse.grid[xo[[1]], ],col=grey(0.5),add=T)
> lines(sl)
```
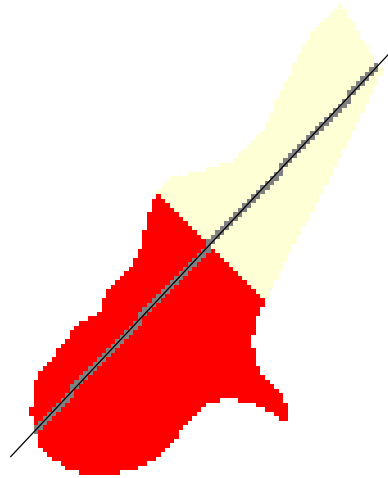

Figure 3: Overlay of line with grid, identifying cells crossed (or touched) by the line

5 Ordering and constraining of `rgeos`-based intersects

Consider the following "identical" 3×3 grid, represented as `SpatialGrid`, `SpatialPolygons` and `SpatialPixels`:

```
> g = SpatialGrid(GridTopology(c(0,0), c(1,1), c(3,3)))
> p = as(g, "SpatialPolygons")
> px = as(g, "SpatialPixels")
> plot(g)
> text(coordinates(g), labels = 1:9)
```

```
+-------+-------+-------+
|       |       |       |
|   1   |   2   |   3   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   4   |   5   |   6   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------+-------+-------+
```

We can match these geometries with themselves by

```
> over(g,g)
```

```
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
```

```
> over(p,p)
```

```
g1.g1 g2.g1 g3.g2 g4.g1 g5.g1 g6.g2 g7.g4 g8.g4 g9.g5
    1     1     2     1     1     2     4     4     5
```

```
> over(p,g)
```

12

```
g1 g2 g3 g4 g5 g6 g7 g8 g9
 1  2  3  4  5  6  7  8  9

> over(g,p)

1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
```

and see that most give a 1:1 match, except for polygons-polygons (p,p).

When we ask for the full set of matches, we see

```
> over(px[5], g, returnList = TRUE)

$`1`
[1] 5

> over(p[c(1,5)], p, returnList = TRUE)

$g1
g1 g2 g4 g5
 1  2  4  5


$g5
g1 g2 g3 g4 g5 g6 g7 g8 g9
 1  2  3  4  5  6  7  8  9
```

and note that the implementation lets grids/pixels not match (intersect) with neighbour grid cells, but that polygons do.

There are two issues we'd like to improve here: the order in which matching features (here: polygons) are returned, and the possibility to limit this by the dimension of the intersection (point/line/area). Both will be explained now.

5.1 Ordering of matches

By default, polygon-polygon features are matched by `rgeos::gIntersects`, which just returns *any* match in *any* order (feature order, it seems). Although it is slower, we can however improve on this by switching to `rgeos::gRelate`, and see

```
> over(px[5], g, returnList = TRUE, minDimension = 0)

$`1`
[1] 5

> over(p[c(1,5)], p, returnList = TRUE, minDimension = 0)

$g1
[1] 1 2 4 5

$g5
[1] 5 2 4 6 8 1 3 7 9
```

When `minDimension = 0` is specified, the matching geometries are being returned based on a nested ordering. First, ordering is done by dimensionality of the intersection, as returned by the `rgeos` function `gRelate` (which uses the DE-9IM model). This means that features that have an area overlapping (dim=2) are listed before features that have a line in common (dim=1), and that line in common features are listed before features that only have a point in common (dim=0).

Remaining ties, indicating cases when there are multiple different intersections of the same dimension, are ordered such that matched feature *interiors* are given higher priority than matched feature boundaries.

Note that the ordering also determines which feature is matched when `returnList=FALSE`, as in this case the first element of the ordered set is taken:

```
> over(p, p, minDimension = 0)

g1 g2 g3 g4 g5 g6 g7 g8 g9
 1  2  3  4  5  6  7  8  9
```

Consider the following example where a point is *on* x1 and *in* x2:

```
> x2 = x1 = cbind(c(0,1,1,0,0), c(0,0,1,1,0))
> x1[,1] = x1[,1]+0.5
> x1[,2] = x1[,2]+0.25
> sp = SpatialPolygons(list(
+  Polygons(list(Polygon(x1)), "x1"),
+  Polygons(list(Polygon(x2)), "x2")))
> pt = SpatialPoints(cbind(0.5,0.5)) # on border of x1
> row.names(pt) = "pt1"
> plot(sp, axes = TRUE)
> text(c(0.05, 0.55, 0.55), c(0.9, 1.15, 0.5), c("x1","x2", "pt"))
> plot(pt, add=TRUE, col='red', pch=16)
```
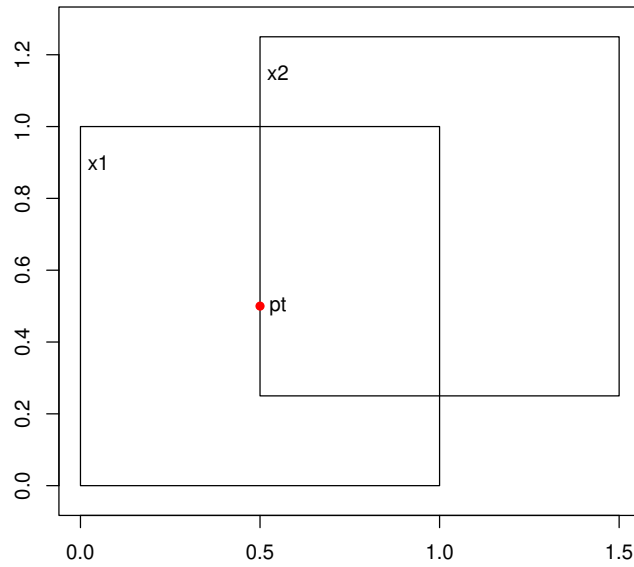
When matching the point `pt` with the two polygons, the sp method (default) gives no preference of the second polygon that (fully) contains the point; the rgeos method however does:

```
> over(pt,sp)

pt1
  2

> over(pt,sp,minDimension=0)

pt1
  2

> over(pt,sp,returnList=TRUE)

$pt1
[1] 1 2

> rgeos::overGeomGeom(pt,sp)

pt1.x1
     1

> rgeos::overGeomGeom(pt,sp,returnList=TRUE)
```

```
$pt1
x1 x2
 1  2

> rgeos::overGeomGeom(pt,sp,returnList=TRUE,minDimension=0)

$pt1
[1] 2 1
```

5.2 Constraining dimensionality of intersection

In some cases for feature selection it may be desired to constrain matching to features that have an area overlap, or that have an area overlap *or* line in common. This can be done using the parameter `minDimension`:

```
> over(p[5], p, returnList=TRUE, minDimension=0)

$g5
[1] 5 2 4 6 8 1 3 7 9

> over(p[5], p, returnList=TRUE, minDimension=1)

$g5
[1] 5 2 4 6 8

> over(p[5], p, returnList=TRUE, minDimension=2)

$g5
[1] 5

> rgeos::overGeomGeom(pt, pt, minDimension=2) # empty

[1] NA

> rgeos::overGeomGeom(pt, pt, minDimension=1) # empty

[1] NA

> rgeos::overGeomGeom(pt, pt, minDimension=0)

pt1
  1
```

6 Aggregation

In the following example, the values of a fine grid with 40 m x 40 m cells are aggregated to a course grid with 400 m x 400 m cells.

```
> data(meuse.grid)
> gridded(meuse.grid) = ~x+y
> off = gridparameters(meuse.grid)$cellcentre.offset + 20
> gt = GridTopology(off, c(400,400), c(8,11))
> SG = SpatialGrid(gt)
> agg = aggregate(meuse.grid[3], SG, mean)
```

Figure 5 shows the result of this aggregation (agg, in colors) and the points
(+) of the original grid (meuse.grid). Function aggregate aggregates its first
argument over the geometries of the second argument, and returns a geometry
with attributes. The default aggregation function (mean) can be overridden.
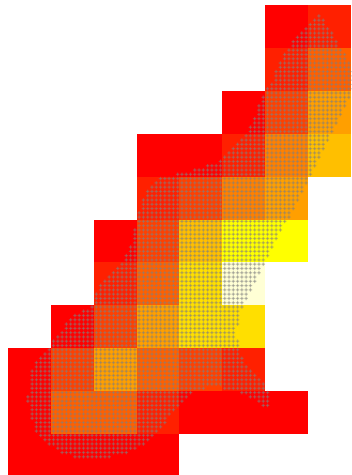

Figure 4:   aggregation over meuse.grid distance values to a 400 m x 400 m grid

An example of the aggregated values of meuse.grid along (or under) the
line shown in Figure 2 are

```
> sl.agg = aggregate(meuse.grid[,1:3], sl, mean)
> class(sl.agg)

[1] "SpatialLinesDataFrame"
attr(,"package")
[1] "sp"
```

17

```
> as.data.frame(sl.agg)


     part.a    part.b       dist
1 0.4904459 0.5095541 0.3100566
```

Function `aggregate` returns a spatial object of the same class of `sl` (`SpatialLines`), and `as.data.frame` shows the attribute table as a `data.frame`.

6.1   Constraining the dimension of intersection

Building on the simple example of section 5.2, we can see what happens if we aggregate polygons *without* specifying *how* polygons intersect[1], the result of which is shown in Figure 5.

```
> g = SpatialGrid(GridTopology(c(5,5), c(10,10), c(3,3)))
> p = as(g, "SpatialPolygons")
> p$z = c(1,0,1,0,1,0,1,0,1)
> cc = coordinates(g)
> p$ag1 = aggregate(p, p, mean)[[1]]
> p$ag1a = aggregate(p, p, mean, minDimension = 0)[[1]]
> p$ag2 = aggregate(p, p, mean, minDimension = 1)[[1]]
> p$ag3 = aggregate(p, p, mean, minDimension = 2)[[1]]
> p$ag4 = aggregate(p, p, areaWeighted=TRUE)[[1]]
```

The option `areaWeighted=TRUE` aggregates area-weighted, giving zero weight to polygons that only have a point or line in common with the target polygon; `minDimension` is passed to `over` to constrain the intersecting polygons used.

The following example furher illustrates the difference between selection using `minDimension`, and area weighting for aggregating the 0-1 checker board of figure 5 by the green square polygon (`sq`) shown in the last panel of that figure:

```
> round(c(
+   aggDefault = aggregate(p, sq, mean)[[1]],
+   aggMinDim0 = aggregate(p, sq, mean, minDimension = 0)[[1]],
+   aggMinDim1 = aggregate(p, sq, mean, minDimension = 1)[[1]],
+   aggMinDim2 = aggregate(p, sq, mean, minDimension = 2)[[1]],
+   areaWeighted = aggregate(p, sq, areaWeighted=TRUE)[[1]]), 3)

  aggDefault  aggMinDim0  aggMinDim1  aggMinDim2 areaWeighted
       0.556       0.556       0.556       0.556        0.722
```

References

- O'Sullivan, D., Unwin, D. (2003) Geographical Information Analysis. Wiley, NJ.

---

[1]sp versions 1.2-1, rgeos versions 0.3-13

18

Figure 5: Effect of aggregating checker board `SpatialPolygons` by themselves, for different values of `minDimension` and `areaWeighted`; the green square example is given in the text.

- Davidson, R., 2008. Reading topographic maps. Free e-book from: http://www.map-reading.com/

- Heuvelink, G.B.M., and E.J. Pebesma, 1999. Spatial aggregation and soil process modelling. Geoderma 89, 1-2, 47-65.

- Pebesma, E., 2012. Spatio-temporal overlay and aggregation. Package vignette for package spacetime, https://cran.r-project.org/web/packages/spacetime/vignettes/sto.pdf