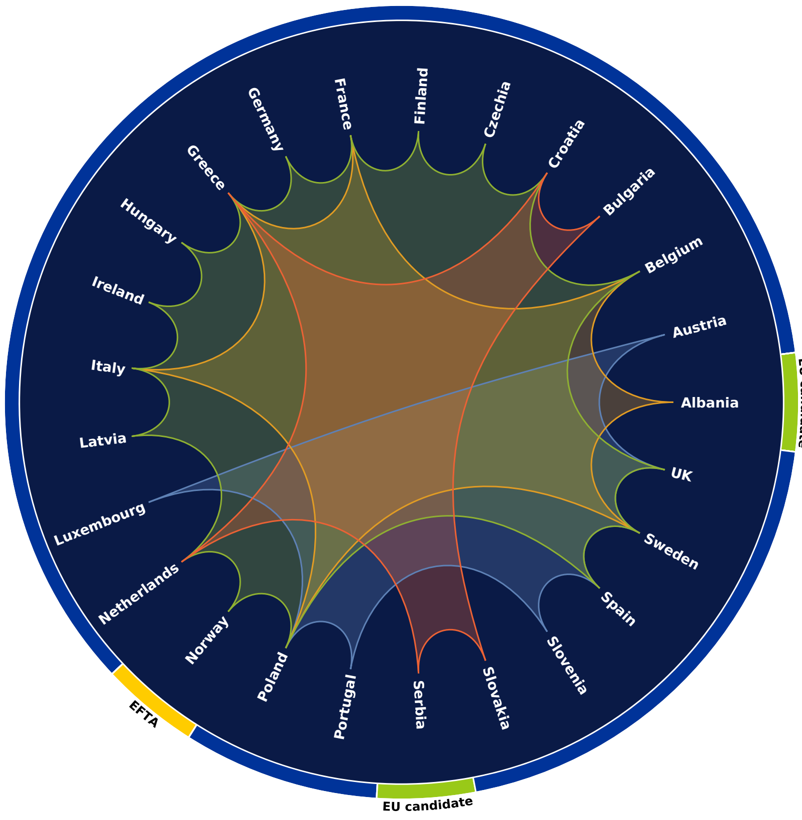
# HyperColl

*We have developed a visually appealing representation for showing multiple shared connections between a large number of entities in a single graph. The library is in particular useful for visualizing collaborative projects.*

The main structure of the representation generated by Hypercoll is very simple: the entities are ordered in some way around a circle. The circle bounds a disk, on which we introduce the standard hyperbolic metric. Each individual connection is drawn as a geodesic, so that each group that shares connections of the same type corresponds to a hyperbolic polygon. A few parameters can be used to adjust the visual appearance of the polygons (shading, shape, etc.).

Here is an example:



In this example, each hyperbolic polygon represents a cooperation project supported by the Creative Europe program of the European Union, which connects certain partner countries. The projects are

- 🟦 Innovation Network of European Showcases
- 🟧 ATLAS OF TRANSITIONS. New geographies for a cross-cultural Europe
- 🟩 Opera Vision
- 🟥 BORDERLINE OFFENSIVE: laughing in the face of fear

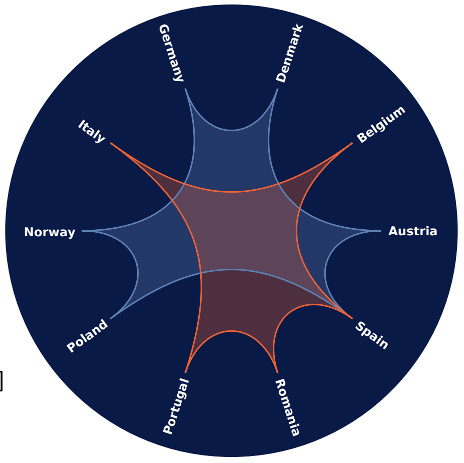
We have simply ordered the countries alphabetically as one goes around the circle counterclockwise; sometimes it makes more sense to group the entities together according to other criteria. To provide additional information, one can use the outermost ring. In our example, we have marked whether the listed countries are already EU members or have a different status.

In our experience, diagrams generated by Hypercoll look best when there is a sufficiently large number of points on the outer circle that get connected. It is just to exhibit the structure as simply as possible that we explain it with only small sets below.

## 1 The HYPERCOLL Module

We provide an easy to use python module for generating HyperColl diagrams.
For example the diagram to the right, showing the projects,

🟦 I Will Be Everything...
🟧 Connecting Emerging Literary Artists



is generated by the python script,

```
from hypercoll import *

all_countries = ['Austria', 'Belgium', 'Denmark', 'Germany',
'Italy', 'Norway', 'Poland', 'Portugal', 'Romania', 'Spain']
hc = HyperColl(all_countires)

p1 = ['Austria', 'Denmark', 'Germany', 'Norway', 'Poland', 'Spain']
p2 = ['Belgium', 'Italy', 'Portugal', 'Romania', 'Spain']
projects = [p1,p2]

hc.hypercoll(projects,150,0.0,0.6,
             color_list=[[0.368417, 0.506779, 0.709798], [0.922526, 0.385626, 0.209179]])
```

All methods and their parameters will be explained in the following.

### 1.1 License

### 1.2 Dependencies

The HYPERCOLL module depends on PyCairo.

### 1.3 Loading the module

First the HYPERCOLL module needs to be loaded. This is for example done using,

```
import hypercoll
```

The module contains only one class called `HyperColl`.

### 1.4 `HyperColl.__init__`

In the next step one has to create an instant of the class HyperColl. The `__init__` method of the `HyperColl` class is defined as follows

```
def __init__(self, labels, filename="out.pdf", width = 595, height = 595):
```

the argumets are

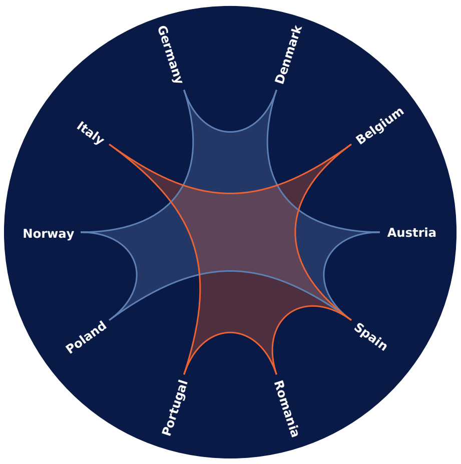
- `labels:` A list of unique strings. They are used both to label the positions around the main diagram and to refere to these positions. The first label will be placed at zero angle (on the right hand side of the diagram).
- `filename:` Output file name.
- `width, height:` width and height of the page containing the diagram in points. A point is $\frac{1}{72}$inch or $0.3528$mm. The default value of $595$pt coresponds to the width of A4 paper.

### 1.5 `HyperColl.hypercoll`

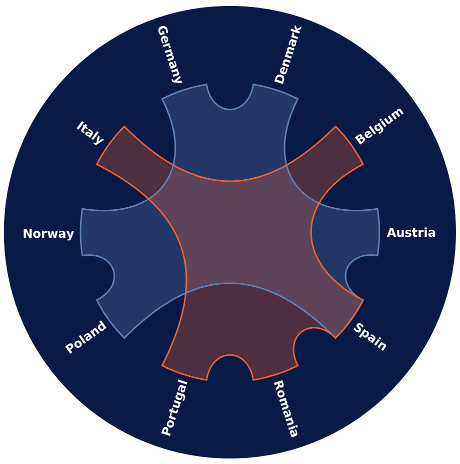
After the class has been created we can draw the main diagram. This is done with the method `hypercoll`. It is defined as

```
def hypercoll(self, colls, r, span, curvature,
              color_list=[[0.368417, 0.506779, 0.709798], [0.880722, 0.611041, 0.142051],
              [0.560181, 0.691569, 0.194885], [0.922526, 0.385626, 0.209179],
              [0.528488, 0.470624, 0.701351], [0.772079, 0.431554, 0.102387],
              [0.363898, 0.618501, 0.782349], [1, 0.75, 0], [0.647624,0.37816, 0.614037],
              [0.571589, 0.586483, 0.], [0.915, 0.3325, 0.2125],
              [0.400822, 0.522007, 0.85], [0.972829, 0.621644, 0.073362],
              [0.736783, 0.358, 0.503027], [0.280264, 0.715, 0.429209]],
              fill_opacity=0.3, edge_order=None, edge_opacity=1.0, edge_width=1.0,
              labels=True,hspace=5,
              font_face = "Sans", font_slant=cairo.FONT_SLANT_NORMAL,
              font_weight=cairo.FONT_WEIGHT_BOLD, font_size=8,
              label_color=[1,1,1], label_opacity=1,
              background_disk=True,
              background_color=[0.0392157, 0.101961, 0.27451],
              background_circle=True,
              background_circle_edge_width=1.0,
              background_circle_color=[0.0392157, 0.101961, 0.27451],
              fill=True):
```

- `colls:` A list of list of labels. Each list defines positions which will be connect using a hyperbolic patch. For example `[['A','B','C'],['D','E']]` will create two patches, the first one connecting `A`,`B` and `C` and the second one connecting `D` and `E`.
- `r:` the radius of the diagram including the labels in points.
- $0 \leq$ `span` $\leq 1$: determines the span of the patch alongside the label. If a patch connects only two entities `span` must be larger than zero.

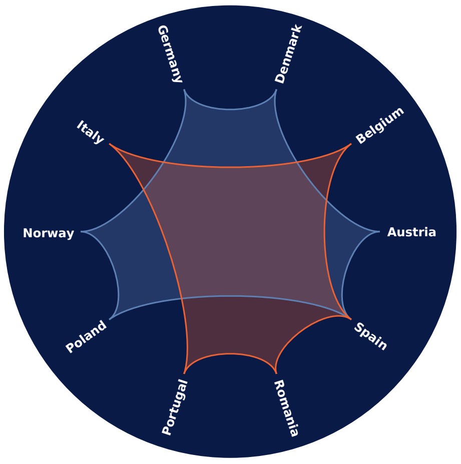


span=0.0

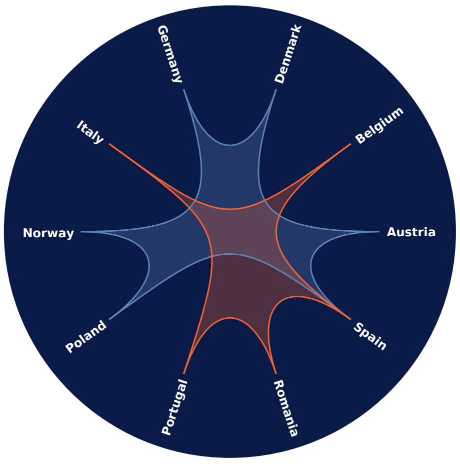


span=0.5

- $0 \leq$ `curvature` $\leq 1$: Determines the curvature of the edges connecting different labels.



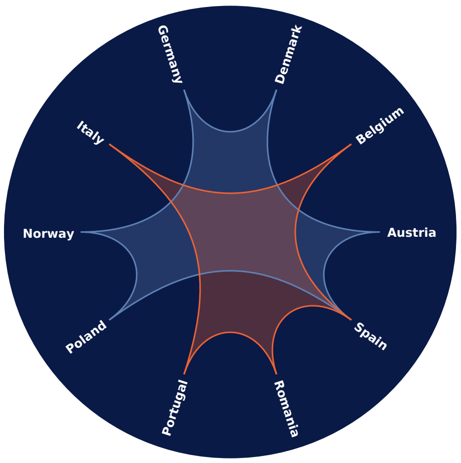
curvature=0.3



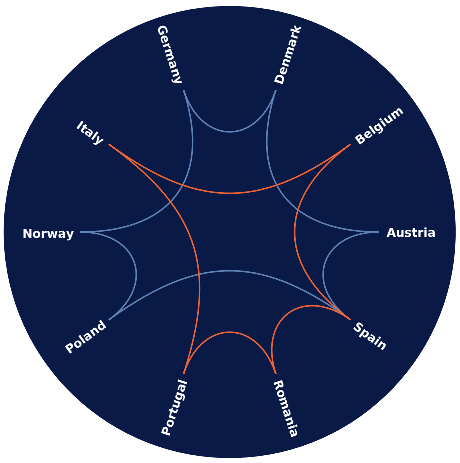
curvature=0.7

- `color_list`: A list of colors used periodically to color the patches and their edges. The default values are taken from Wolfram Mathematica. We found them to work very well on a dark background.
- `fill_opacity, edge_opacity`: The opacity of the patch's filling and its edge.
- `edge_order`: A numeric list which sets the order in which the edges of the patches are drawn. For example `edge_prder=[2,1]` means that the edge of the second patch should be drawn first. By default the edges are drawn as if `edge_order=[0,1,2,...,len(colls)-1]`. The option is mostly relevant when two or more edges overlap.
- `edge_width`: Width of the edge in points.
- `labels` $\in$ {`True`,`False`} If True, the labels are shown. If False, the labels still serve as a reference to the positions around the circle but the labels are not placed at those positions.
- `hspace`: Determines the extra space in points between the inner and out radius of the circular patches and both sides of the labels.
- `font_face,font_slant,font_weight,font_size` Parameter passed to Cairo for rendering the labels.
- `label_color,label_opacity`: Color of labels and their opacity.
- `background_disk`, `background_color`: When `background_disk` is set to `True` a disk with the color `background_color` is drawn in the background of the graphics.
- `background_circle`, `background_circle_edge_width`, `background_circle_color`: When `background_circle` is set to `True` a circle with the color `background_circle_color` and with of `background_circle_edge_width` is drawn around the main graphics.

- `fill` $\in \{$`True`$,$`False`$\}$: If `True` the patches will be filled.



fill=True



fill = False

All colors are to be given as RGB values between 0 and 1. The opacity values range from 0 to 1 too.

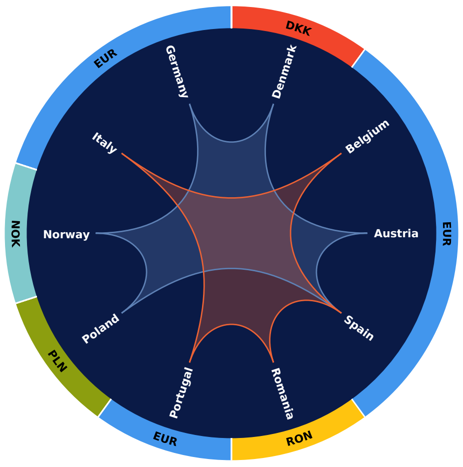
## 1.6 `HyperColl.sectors`

```
  def sectors(self, r1, r2, dic, labels={},
              font_face = "Sans",
              font_slant=cairo.FONT_SLANT_NORMAL,
              font_weight=cairo.FONT_WEIGHT_BOLD,
              font_size=18,
              foreground_color=[1,1,1],opacity=1,
              background_color=[0.0392157, 0.101961, 0.27451],
              boundry_color=[1,1,1],boundy_width=1):
```

The function `sectors` is used for rendering sectors containing additional informations about the labels around main diagram. First one associates a property to every label. Then, the rest of the options are then associated with these properties. For example imagine we would like to add information about the currencies of the involved countries to our diagram. This can be done for example with the following code,

```
hc.sectors(149,165,
           {'Austria':'EUR', 'Belgium':'EUR', 'Denmark':'DKK',
            'Germany':'EUR', 'Italy':'EUR', 'Norway':'NOK', 'Poland':'PLN',
            'Portugal':'EUR', 'Romania':'RON', 'Spain':'EUR'},
            background_color={'EUR':[0.26, 0.59, 0.93],
            'DKK':  [0.95, 0.27, 0.17], 'RON':[1.0, 0.77, 0.059] ,
            'PLN':[0.55, 0.62, 0.059] , 'NOK':[0.50, 0.79, 0.80] },
            foreground_color=[0,0,0], font_size=8,
            boundry_color=[1,1,1],
            boundy_width=1
)
```

to obtain



Some arguments of the `sector` function accept either a constant value or a dictionary as input. If a constant is passed, like `foreground_color=[0,0,0]`, it applies to all properties. In contrast a dictionary can be used to associate different options to different properties, e.g. `background_color={'EUR':[0.26, 0.59, 0.93], 'DKK':[0.95, 0.27, 0.17], ···}`. Such arguments are marked with an asterisk in the following.

- `r1,r2`: inner and outer radius of the ring. The inner radius of the first ring is usually the same as the radius of the main diagram (parameter `r` provided to `hypercoll`).
- `dic`: A dictionary associating a property to every `label` the class was initialized with.
- `labels`: A dictionary giving each property defined in `dic` a label. If not empty, the labels are rendered in the corresponding sectors.
- `font_face*,font_slant*,font_weight*,font_size*` Parameter passed to Cairo for rendering the labels.
- `foreground_color*, opacity*`: Color and opacity used to render the `labels` associated to the properties from `dic` in the corresponding sectors.
- `background_color*`: Color used to fill the background of the sectors.
- `boundry_color`: The color used when rendering the dividing rules between sectors.
- `boundry_with`: The width of the dividing rules in points.