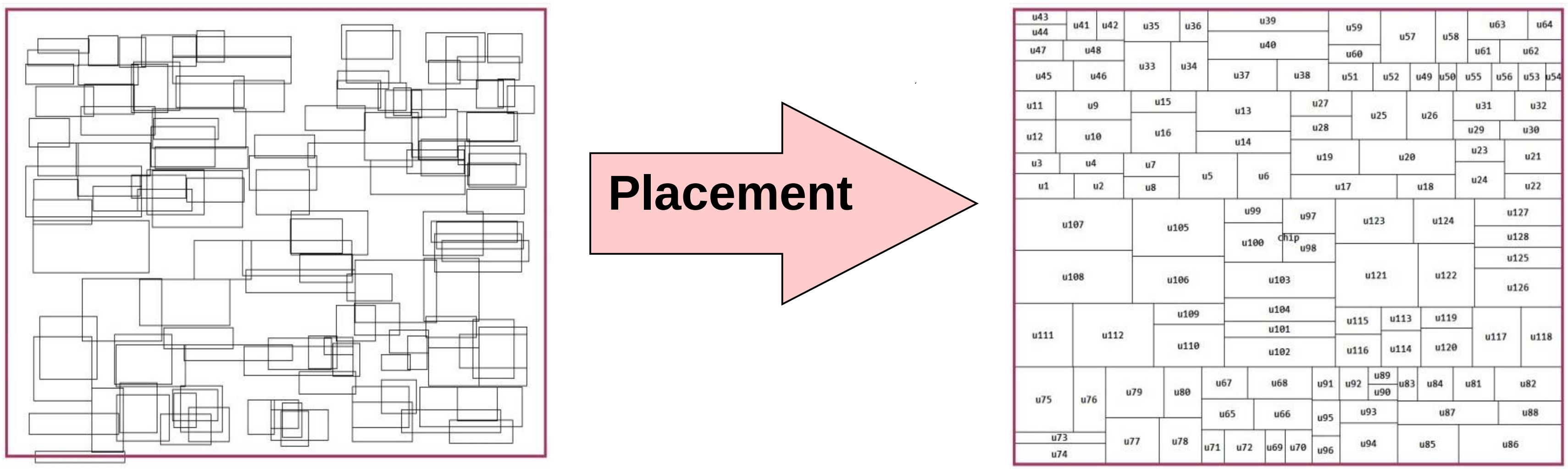
Final Project

FLOORPLAN MODELING AND DESIGN



EDA Modules

- You will need the **eda** modules for the project
- EDA = Electronic Design Automation
- Download eda.zip from: https://samyzaf.com/braude/DSAL/projects/final/eda.zip
- After unzipping this archive you will find the following files:
  - **point.py**
  - **line.py**
  - **rectangle.py**
  - **graphics.py**
  - **unit.py**
  - **grid.py**
- Throw these files into your project directory (make it simple like `c:\project`). Your project should start with these files, but you will have to add new ones of course.

RELATION

- A relation is any function **rel(x,y)** of two variables which returns a Boolean value (True/False)
- Example:

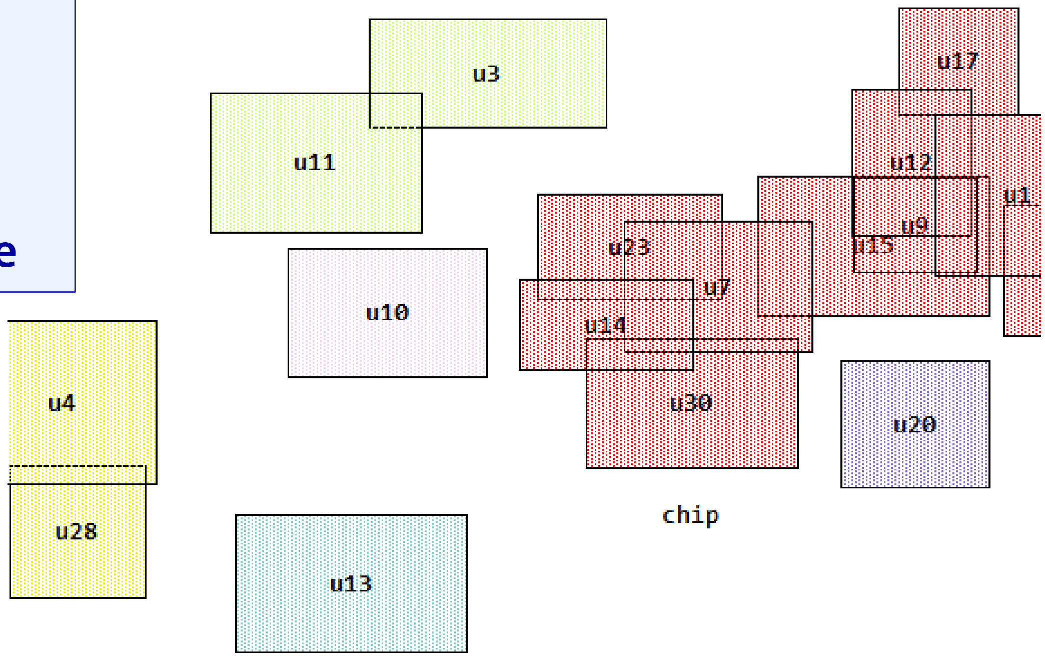
```
def rel(x,y):
    return (x-y)%7 == 0

rel(8,15)
=> True
rel(8,16)
=> False
```

SYMMETRIC RELATION

- A relation is called **symmetric** if `rel(x,y) = rel(y,x)` holds for every x and y in the domain of the relation
- Example: the relation “unit x intersects unit y” is a symmetric relation:

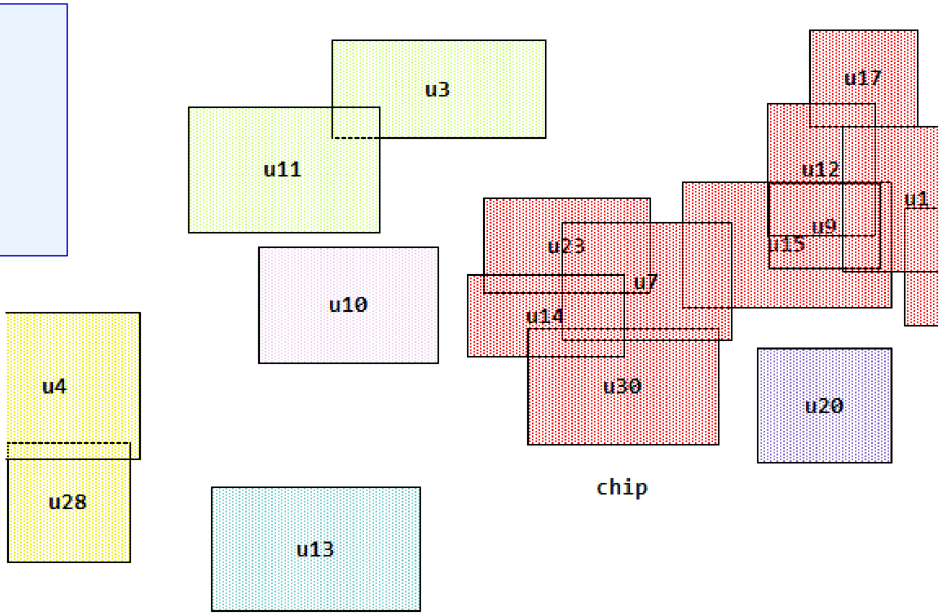
```
intersect(u3, u11) = True
intersect(u11, u3) = True
intersect(u3, u10) = False
intersect(u10, u3) = False
intersect(u14, u30) = True
intersect(u28, u13) = False
```



CONNECTION

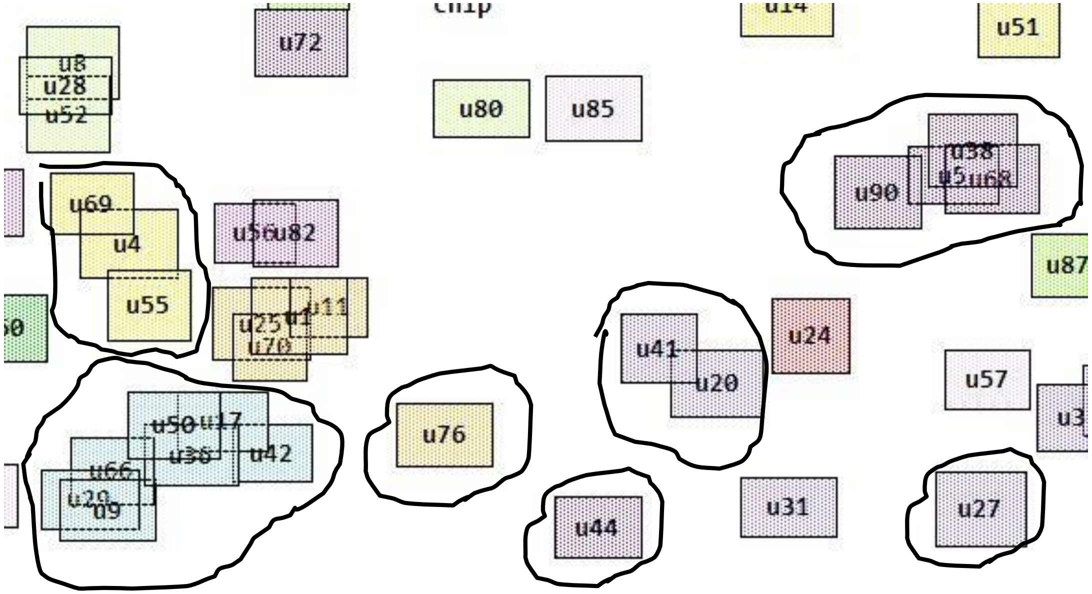
- Two elements a and b are said to be **connected** by a relation **rel** if there is a sequence of elements x1, x2, …, xn, such that: `rel(a,x1)` and `rel(x1,x2)` and `rel(x2,x3)` and … `rel(xn,b)`
- Example: In the following diagram we see the element u17 is connected to the element `u30` (`rel = intersecting_units`)

```
intersect(u17, u12) = True
intersect(u12, u15) = True
intersect(u15, u7)  = True
intersect(u7, u30)  = True
```



CLUSTERS

- A cluster is a maximal subset of connected elements
- That is:
  - Every two elements in the cluster must be connected
  - Every element outside the cluster is disconnected from the cluster
- Example:
  The `unit_intersection` relation on eda units can generate clusters:



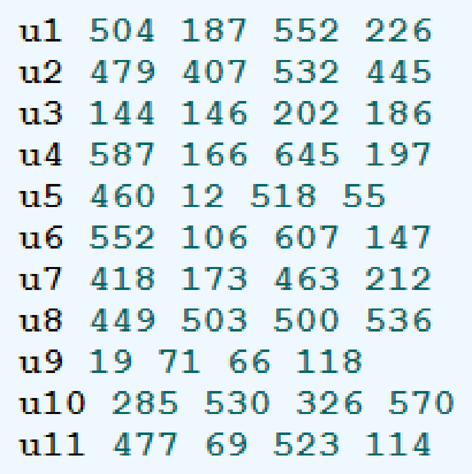
PROBLEM 1

- Write an algorithm for finding all the clusters of a given symmetric relation on a domain (list of elements)
- You need to write a Python function:

```
def get_connectivity_clusters(domain, relation):
    # Partitions a domain of objects into clusters
    # Arguments:
    #    domain   - collection of objects to be partitioned
    #    relation - symmetric relation (as a function)
    #               i.e. relation(a,b) is True <=> a and b are connected
    # Return value – clusters
```

PROBLEM 2

- Download the following location files:
  https://samyzaf.com/braude/DSAL/projects/final/cells1.loc
  https://samyzaf.com/braude/DSAL/projects/final/cells2.loc
- Location file describes a list of units. Each line consists of the unit name and its coordinates:

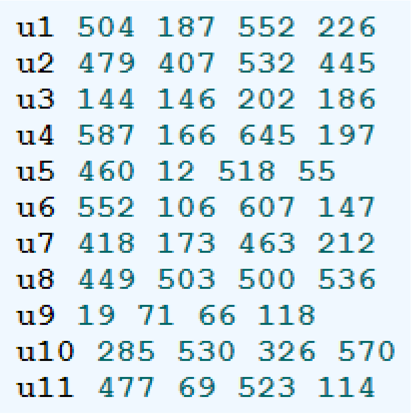


```
u1 504 187 552 226
u2 479 407 532 445
u3 144 146 202 186
u4 587 166 645 197
u5 460 12 518 55
u6 552 106 607 147
u7 418 173 463 212
u8 449 503 500 536
u9 19 71 66 118
u10 285 530 326 570
u11 477 69 523 114
```

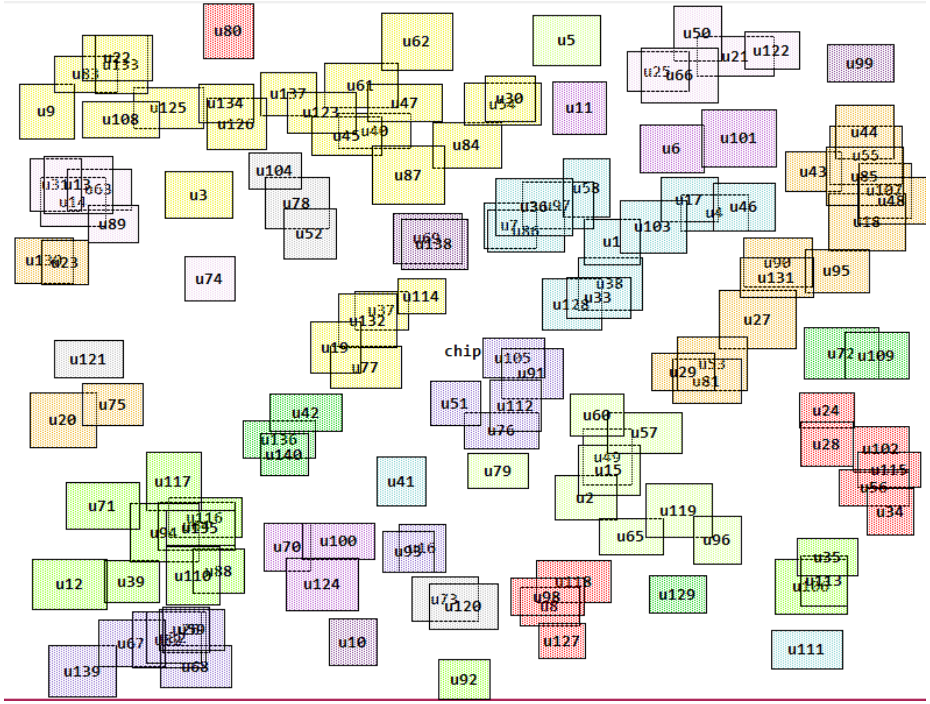
- Your will need to write Python functions for reading and writing location files (loc type files)

PROBLEM 2 (continued)

- Your mission is to apply your clustering algorithm in order to find the units clusters in these location files
- Try to use your drawing capabilities in order to color each cluster in a separate color so that it will be easier for you to debug your program

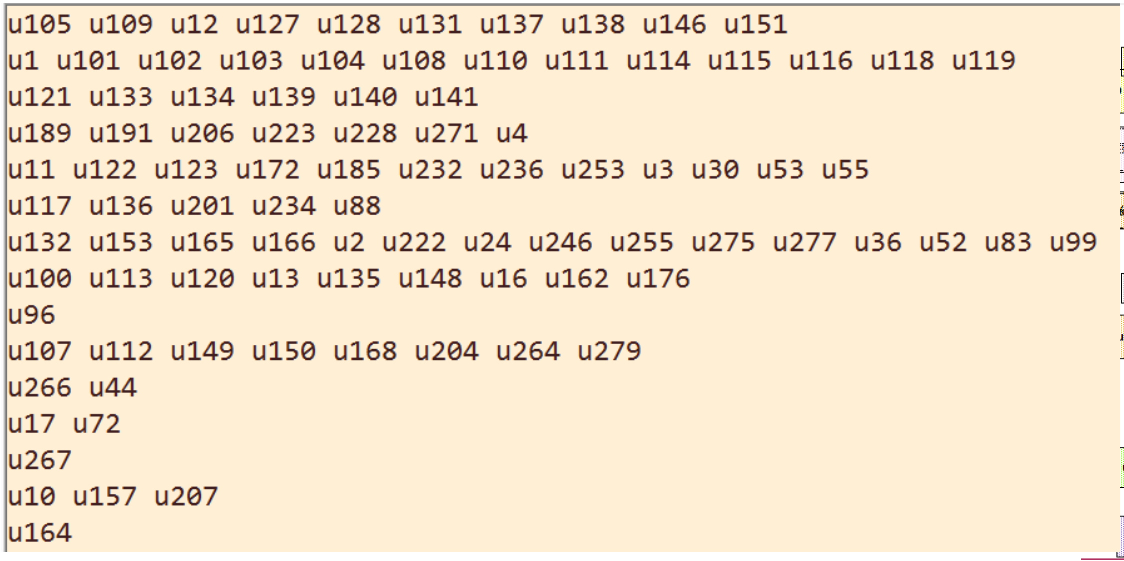


```
u1 504 187 552 226
u2 479 407 532 445
u3 144 146 202 186
u4 587 166 645 197
u5 460 12 518 55
u6 552 106 607 147
u7 418 173 463 212
u8 449 503 500 536
u9 19 71 66 118
u10 285 530 326 570
u11 477 69 523 114
```

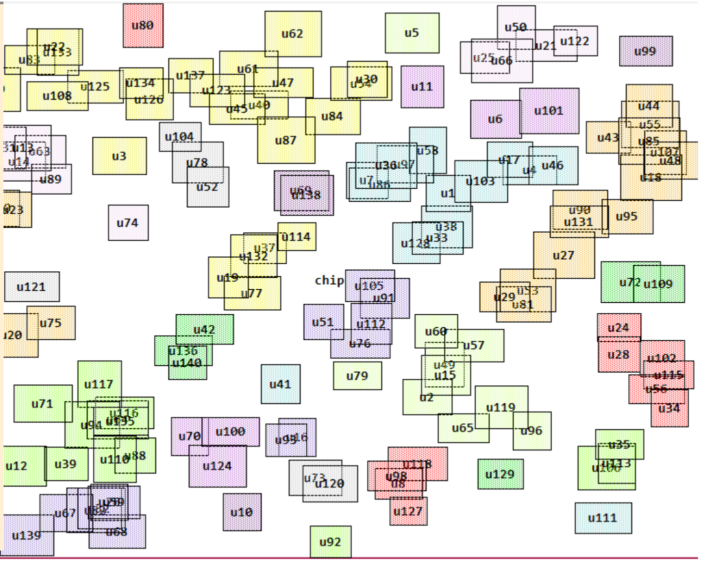


PROBLEM 2 (continued)

- The clusters output file should be named after the units file:
  units file is: `cells17.loc`
  clusters file is: `cells17.clusters`
- Clusters file should look as follows:
  - Each cluster is printed in one line (unit names only!)
  - Example: in the below `cells117.clusters` consists of 15 clusters (15 lines). The first cluster consists of 10 units, second cluster of 13 units, ..

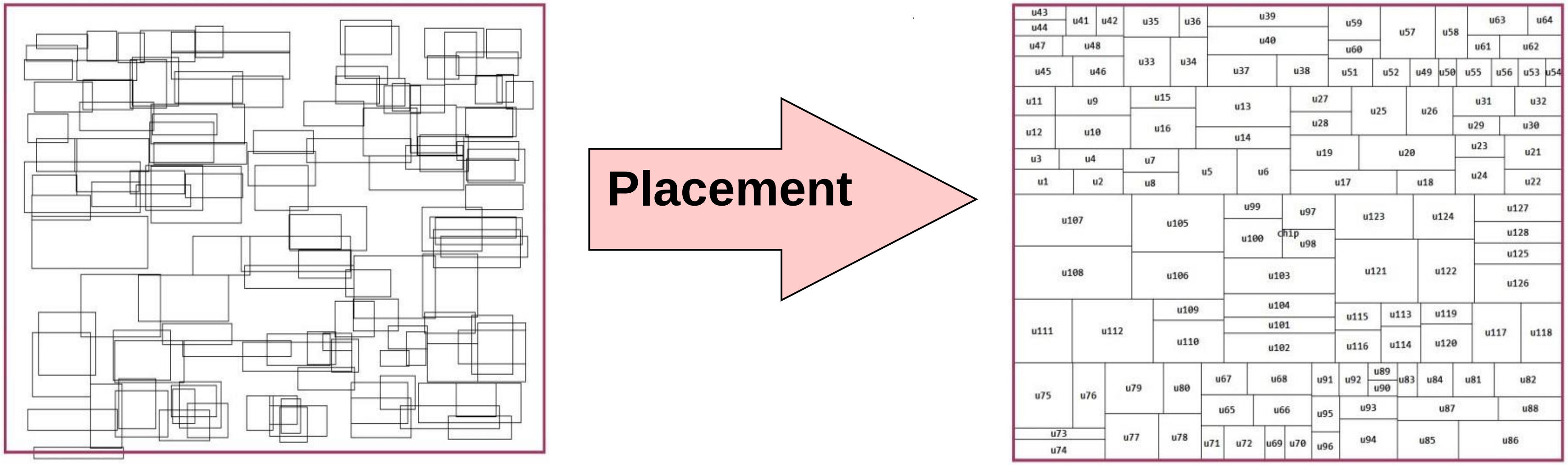


```
u105 u109 u12 u127 u128 u131 u137 u138 u146 u151
u1 u101 u102 u103 u104 u108 u110 u111 u114 u115 u116 u118 u119
u121 u133 u134 u139 u140 u141
u189 u191 u206 u223 u228 u271 u4
u11 u122 u123 u172 u185 u232 u236 u253 u3 u30 u53 u55
u117 u136 u201 u234 u88
u132 u153 u165 u166 u2 u222 u24 u246 u255 u275 u277 u36 u52 u83 u99
u100 u113 u120 u13 u135 u148 u16 u162 u176
u96
u107 u112 u149 u150 u168 u204 u264 u279
u266 u44
u17 u72
u267
u10 u157 u207
u164
```



PROBLEM 3

- **Placement Problem**
  Download the test location files to your project directory from:
  https://samyzaf.com/braude/DSAL/projects/final/locfiles.zip
- Take into account that your algorithm will be checked on more test cases that are not included in these files, so you must optimize it to all possible cases
- In each location file test case, you have to move the units within the chip boundary so that no two units intersect and as many as possible units fit into the chip. You may not rotate or flip units!



PROBLEM 3 (continued)

- **Placement Program (placer)**
  You have to write a function placer(units, chip) which accepts a list of unit objects (as defined in the file unit.py) and a chip which is also a unit object

```
def placer(units, chip):
    # Place units in chip area. Make sure no two units
    # intersect and as many as possible units are placed
    # return the list of units that could not be placed
    # The success criteria will be calculated by the formula:
    #         placed_area / total_chip_area
```

- **Chip size**
  In all 6 cases, the chip size is the same: **width=800**, **height=600**.
  So your programs will use a lot the following two lines:

```
chip = Unit("chip", 0, 0, 800, 600)
chip.draw(outline='maroon', width=2)
```

PROBLEM 3 (continued)

- **Output files**
  After applying your placer algorithm on the location files:

  cells1.loc
  cells2.loc
  cells3.loc
  cells4.loc
  cells5.loc
  cells6.loc …

  you should save the results of your placement algorithm in the following location files:

  cells1_placed.loc
  cells2_placed.loc
  cells3_placed.loc
  cells4_placed.loc
  cells5_placed.loc
  cells6_placed.loc ……

- It is recommended to have a function for drawing a location file so you can easily verify that your results were saved correctly (we will use such function to check the correctness of your results)
- You may not rotae or filp uinits.
- Total run time: **<u>should not exceed two hours</u>** per location file (for any of the test cases that you were given)

END OF PROJECT

Useful Links

- The following links should help you find information related to drawing on the Tkinter canvas and some EDA algorithms stuff:

- https://samyzaf.com/braude/EDA
- http://effbot.org/tkinterbook/canvas.htm
- http://effbot.org/tkinterbook/tkinter-index.htm
- https://wiki.python.org/moin/TkInter
- http://effbot.org/tkinterbook/tkinter-index.htm
- http://www.engr.uconn.edu/~tehrani/teaching/cad/14_floorplanning.pdf
- http://vlsicad.ucsd.edu/Publications/Journals/j46.pdf
- http://www.or.uni-bonn.de/~vygen/files/analyt.pdf

SUBMISSION

- Due date: January 12, 2014
- You can work in pairs if you want (no triples!)
- Please read the coding guideline in the Python course site and make sure you follow the coding style
- You will have to present your work to the course instructors (Samy and Ofer) at the last week of the semester (with possibly extra date in the first week of the semester break) – you will be reviewed and will have to explain your work and demonstrate it working
- More details about grading and other questions will be added to this project soon (do not print it as it is going to change several times)
- Pleas report to samyz@braude.ac.il on any errors that you find, things that are not clear, etc.
- Have fun solving the problems ☺