* Npc bad for plain maesure 1, because npcs change over time. Less problematic for ipc combinations
* Find out and argue if using sparse format vectors is beneficial for my size or not <https://pytorch.org/docs/stable/sparse.html>
* Different sliding window approaches. Mine was 90/60by1/7 but bwl said 1 year by 1 month is more realistic
* Instead of having a community detection in sliding windows of nodes and edges, we do it only on the edges. All nodes are always present (if computation takes to long or files to big: rethink)
* Mein Topic netzwerk nenne ich jetzt topic (co-)occurance netzwerk um verwirrung zu vermeiden. In ihm möchte ich später predicten wann edges stärker werden und wann somit recombination/diffusion stattfindet. Meine edge weights der projektion sind so gestalltet, dass sie unbalancierte cooccurance von patenten panelized. Zum beispiel: Topic 1, Topic 2, Patent A. wenn 1-A = 0.1 und 2-A = 0.9 dann ist das overall weight 0.9\*0.1 = 0.09. Wenn es aber 1-A: 0.4 und 2-A 0.5 ist, dann 0.4 \* 0.5 = \* 0.2 (Wenn ich die summe statt dem produkt nehmer würde hätte ich 1 und 0,9) Mein argument ist, dass topic recombinationen relevanter sind, oder man eher von einer rekombionation spricht, wenn das verhältnis aubalancierter ist -> das produkt wäre dafür das bessere maß. 0.4 und 0.5 sollte sehr viel mehr gewicht bekommen als 0.1 & 0.9, wobei letzteres auch gewicht haben sollte. Mir scheint als wäre das product wirklich! das beste maß dafür. Feedback der btw dafür bekommen
* Gleiches argument für das topic similarity netzwerk. Patente die über ein topic mit 0.4 und 0.5 verbunden sind, sollen alls more similar representiert werden, als topics die mit 0.1 und 0.9 verbunden sind. Vielleicht ist es hier sogar noch krasser! Man könnte das maß anpassen in dem man beispielsweise den unterschied noch mehr penalized. Beispiel: größeres der beiden \*1.1 und kleineres der beiden \*0.9, bei gleichstand, keine penality. Oder sogar das ganze dynamisch machen. beide teile des products vorher quadrieren!
* Emphazise that this thesis is focusing on the development of this framework and not on it’s optimization!
* For the plain measure and probably community detection to: build the threshold for diffusion (and recombination?) not with all nodes in the window but with all new nodes in the window
* For community detection weighted label propagation we can use all edges instead of the 3 biggest
* Talk about Girvan & Newman not being feasible because of the complexity
* What is GEMSEC? <https://paperswithcode.com/task/community-detection>
* Jonathan: Kann ich soetwas schreiben wie “das ist zwar mit kannonen auf patzen geschossen, aber ich wollte es lernen“ als motivation
* Emphazise that your work is the foundation for a whole research project. The focus was on developing new ideas and implementing them for first insights. The optimization of these ideas and measures is expected to be done in the context of the research project itself.
* The unconnected ness of the graphs reduce the number of community detection algorithms drastically.
* Display communities as graph. Nodes are the communities and links are how communities merge into other communities. Edge weight is the number of communities that merged into one
* When talking about community lifetimes, keep in mind that you favored the oldest community, if there were multiple communities identified by one topk in a window

# unsuitable algorithms - crip #  
'''  
lp2\_commu = {} # Label Propagation (cdlib) | no weight consideration  
leiden\_commu = {} # | need connected graph   
walktrap\_commu = {} # | need connected graph   
eigenvector\_commu = {} # | need connected graph   
spinglass\_commu = {} # | need connected graph   
  
gdmp2\_commu = {} # | does not procude communities  
paris\_commu = {} # | does not seem to work porperly #todo check why?  
  
sbm\_dl\_commu = {} # | need GraphTool  
sbm\_dl\_nested\_commu = {} # | need GraphTool  
infomap\_commu = {} # | needs Linux (I assume)  
aslpaw\_commu = {} # | needs Linux (I assume)  
'''

# try lfm as well if desperate  
  
# unsuitable algorithms - overlapping #  
'''  
wCommunity\_commu = {} # | need connected graph OR nodes with at least one degree  
'''

# filter every community of size 1 out (or even of size 1 and 2, but that will require more effort)  
# construct similar list containing recombinations  
  
# problem: some graphs are not connected: 'networkx.exception.AmbiguousSolution: Disconnected graph: Ambiguous solution for bipartite sets.'  
# solution 1: take not top 3 edges for bipartite graphs but all  
# if not working, take biggest component? Probably not so cool..  
# Take different cd algorithms that can handle disconnectedness?  
# keep in mind for overlapping community detection as well

#--- Identify TopK degree nodes of communities ---#  
  
 # label prop #  
 lp\_topK = {}  
  
 for i in range(len(lp\_clean)):  
 lp\_window = lp\_clean['window\_{0}'.format(i\*30)]  
  
 surviver\_window = []  
  
 for community in lp\_window:  
  
 suriviver = []  
  
 for patent in community:  
  
 suriviver.append((patent, topicSim['window\_{0}'.format(i \* 30)].degree(patent)))  
 # Here we take the overall degree, not the degree restricted to nodes in the community. This is due to the assumption that most  
 # high degree labeled to be in a community also have the most edges to nodes in this community. This assumption can be falsified later.  
 # Later on not only the degree, but rather the sum of edges weighes might be used to find this core node of the community.  
 # This approach might be extended to consider not only the top degree node as core, but the top k degree nodes.  
  
 suriviver.sort(key=operator.itemgetter(1), reverse=True)  
 suriviver\_topK = suriviver[0:1]  
 surviver\_window.append(suriviver\_topK)

# Assumption: highly connected nodes in (/within) a community are somewhat stable parts of communities. We assume that  
# these high degree nodes are not randomly changing community affiliation in the clustering algorithms, and even are  
# propably the least likely and thereby the least nodes that leave the community

Pseudo code for community tracing array :

#for each window (array x-axis)  
# if not first row:  
# for each column (community id)  
# if topk of above column is in a current community  
# insert topk as community id  
# elif highest surving of above topk community in a current community  
# look for highest suring that is in a solo community  
# insert topk as community id  
# else (if no nodes appears anymore) # This means: if a community is dying and on the last tick, at least one patent switches the community, than this counts as merging (which is ok, i guess. At least arguable)  
# insert 0  
# for each community in dic window  
# if topk not in array window  
# open new community  
#