Estructures de Dades
Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili

Pràctica 2: Arbres i Grafs

Josep Bello Curto
Lautaro Russo Bertolez

18-06-2020

Índex

1. Estructures de dades

Implementeu un graf i un max heap genèric en Java.

1.1 Graf

1.1.1 Disseny

A l'hora de decidir com fer el graf vam veure dues opcions:

- Amb una llista on cada element de la llista sigui un node, amb el seu valor, i una llista amb tots els nodes amb que te connexió i el valor de l'aresta.
- Amb una taula de hashing on la clau es el contingut del node i el valor es una llista amb totes les connexions del node i valor de l'aresta.

Ens hem decidit per la segona opció ja que així tenim un cost en búsqueda constant en tot moment tot i sacrificar el cost en memòria que es superior.

1.1.2 Implementació

Per implementar-ho amb una taula de hashing primer vam pensar en utilitzar la classe HashMap de java, però ja que en alguns casos ens interessa accedir als nodes inserits a la xarxa amb un l'índex d'inserció per inserir una aresta ho hem fet amb LinkedHashMap ja que assegura que es mantindrà en tot moment l'ordre amb el que hem inserit les dades, mentre que amb una HashMap no.

Per guardar els enllaços d'un node hem decidit utilitzar una LinkedList i crear una classe EdgeT, de la cual farem la LinkedList i on es guardara el node al que enllaça la clau i el valor de l'enllaç, tot en tipus genèrics.

A l'hora d'afegir una aresta entre dos nodes o eliminar un node hem implementat dues formes de fer-ho:

- Utilitzant el contingut del node, que sera una clau de la taula de hasing, per seleccionar els nodes.
- Utilitzant l'ordre d'inserció dels nodes al graf, aquestes funcions s'han implementat per facilitar el afegir arestes amb el format Pajek, on primer afegim tots els nodes i després arestes entre aquests.

La resta de detalls d'implementació, com que retorna cada mètode, paràmetres, etc estan detallats a la Javadoc del projecte.

1.1.3 Joc de proves

De totes les proves que hem fet, ens agradaria destacar les següents, ja que considerem que son les mes importants.

- Afegir una aresta utilitzant el contingut dels nodes per seleccionar-los, entre dos nodes que no existeixen crea els nos nodes amb el contingut que s'ha ficat al afegir l'aresta.
- Afegir una aresta utilitzant l'index d'inserció dels nodes per seleccionar-los, entre dos nodes que no existeixen retorna fals.
- Eliminar un node elimina també totes les aparicions a les llistes dels nodes adjacents.
- Eliminar un node passant com a paràmetre un node que no existeix, la funció retorna fals.

1.2 Max heap

1.2.1 Disseny

En el disseny del MaxHeap hem decidit utilitzar una taula dinàmica ja que es la forma més senzilla i eficient d'implementar-ho en aquest cas. En quan als algorismes d'inserció d'un nou element i eliminació de l'arrel hem decidit utilitzar els vistos a classe.

Inserció: Ficar l'element al final de la llista i re-equilibrar l'arbre.

Extracció de l'arrel: Intercambiar l'últim element de l'arbre per l'arrel, eliminar l'últim i re-equilibrar.

1.2.2 Implementació

Per implementar-ho en Java hem decidit fer-ho amb un ArrayList ja que tracta de forma interna la gestió de memòria i ja te un contador intern d'elements, per tant no ens cal implementar-ho a nosaltres. Ja que la classe MaxHeap ha de ser genèrica però necessitem comparar els elements de l'arbre per ordenar-lo hi ha dues formes d'implementar-ho en Java:

1. Comparable: Fent que la classe de la cual es fa el MaxHeap implementi l'interficie Comparable i utilitzant el mètode compareTo d'aquesta.
2. Comparator: Fent que l'usuari ens passi una expressió al instanciar la classe que ens dira com hem de comparar-ho.

Ens hem decantat per la primera opció ja que creiem que es molt més simple per l'usuari ja que ha de guardar elements al heap que puguin ser comparables, com ara números, i en el nostre cas hem de guardar un identificador del node, com el seu contingut i un valor en coma flotant que es el que s'utilitzarà per ordenar l'arbre.

Aixo ho hem implementat amb una classe anomenada NodeT que implementa Compable amb ella mateixa i on guardem el contingut del node, i el valor a comparar com un Float. Llavors la funció compareTo es simplement la que ja està implementada a la classe Float.

Destacar també que hem decidit implementar el Heap a partir de l'índex 0 de la taula i no de l'1.

La resta de detalls d'implementació, com que retorna cada mètode, paràmetres, etc estan detallats a la Javadoc del projecte.

1.2.3 Joc de proves

Per fer les proves del Heap ho hem provat amb Integers ja que es molt més simple de veure que funciona correctament, i de les proves que hem fet ens agradaria destacar les següents:

- Extraure l'arrel d'un arbre buit retorna null.

2. Precoloració

2.1 Enunciat

L'anàlisi de percoloració d'una xarxa ens permet mesurar com de robusta es aquesta quan algun dels nodes s'elimina, esta inactiu. Bàsicament consisteix en repetir els següents passos fins que la xarxa sigui buida:

1. Eliminar un node de la xarxa.
2. Mesurar la funcionalitat de la xarxa a partir dels paràmetres OP, NCC, GCC i SLCC.

Els paràmetres anteriors ens permeten veure com evoluciona la xarxa enfront d'un atac, i signifiquen

1. OP: Fracció de nodes que queden a la xarxa.
2. NCC: Nombre de components connexes.
3. GCC: Mida de la component connexa més gran.
4. SLCC: Mida de la segona component connexa més gran.

2.1.1 Formes d'eliminar un node

Per eliminar un node de la xarxa hi ha bàsicament tres formes de fer-ho. Una que simula un atac aleatori i dues que simulen atacs dirigits.

Aleatoria
Eliminar un node que encara esta actriu de forma aleatoria.

Grau del node
Eliminar el node actiu amb un grau més gran.
Es defineix el grau del node com a la quantitat d'arestes que te aquest.

Strength del node
Eliminar el node actiu amb més strength.
Es defineix strength com la suma dels pesos dels seus enllaços.

2.1.2 Càlculs

Per mesurar la funcionalitat de la xarxa hem decidit utilitzar un mètode d'exploració per amplitud, BFS, en lloc de per amplada, DFS, ja que tots dos tenen el mateix cost i el primer ens ha semblat més fàcil d'implementar.
A més a més DFS es sol implementar de forma recursiva la qual cosa podria fer que la pila d'execució es desbordes en una xarxa gran.

2.1.3 Extracció de dades

Per extraure totes les dades dels paràmetres descrits anteriorment hem decidit guardar les dades en csv on tenum un arxiu per cada atac aleatori i un per cada atac dirigit, strength i grau del node.
Per calcular les mitjanes dels atacs aleatoris hem fet un script en la llibreria Pandas de Python que genera un csv amb la mitjana ja calculada i posteriorment hem generat les gràfiques amb Matplotlib.

2.2 Temps d'execució

A continuació es mostra una taula amb els temps, en segons que tardem en executar cada tipus d'atac, en el cas de l'atac aleatori es una mitjana entre les 100 execucions.

Xarxa	Atac aleatori	Atac per grau	Atac per strength
World Trade Web	0,37s	0,44s	0,38s
Correu electrònic de la URV	6,30s	11,50s	13,02s
Transport aeri	149,65s	238,53s	222,54s
Distribució elèctrica USA	85,92s	125,42s	132,35s

Com podem veure tot i que la xarxa de distribució elèctrica te més de 1000 nodes més que la de transport aeri es tarda més en realitzar tot el procès de percoloració a la xarxa de transport aeri ja que te moltes més connexions.

2.2.1 Entorn d'execució

Les proves s'han executat en un MSI P63 amb:

- Processador: Intel i7 8565U de 4 nuclis, 8 fils
- RAM: 16GB DDR4
- S.O: Windows 10 Pro
- IDE: Eclipse 2019
- Versió del JDK de Java: 12.0.2
- Versió del JRE de Java: 8 update 211

3. Gràfiques del NCC, SLCC i GCC

3.1 Xarxa World Trade Web

3.1.1 Atac aleatori

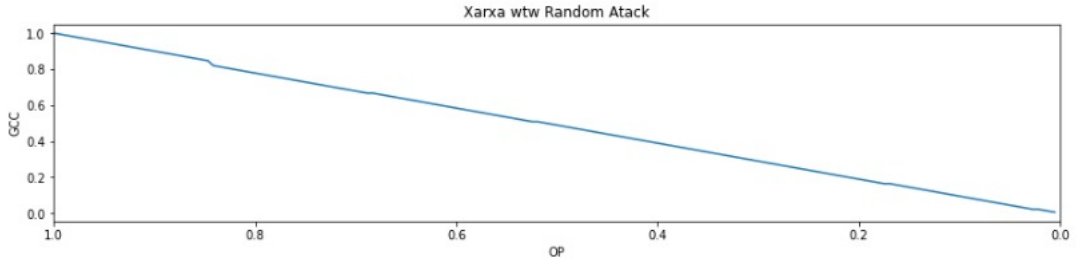

Figura 3.1: Random GCC wtw

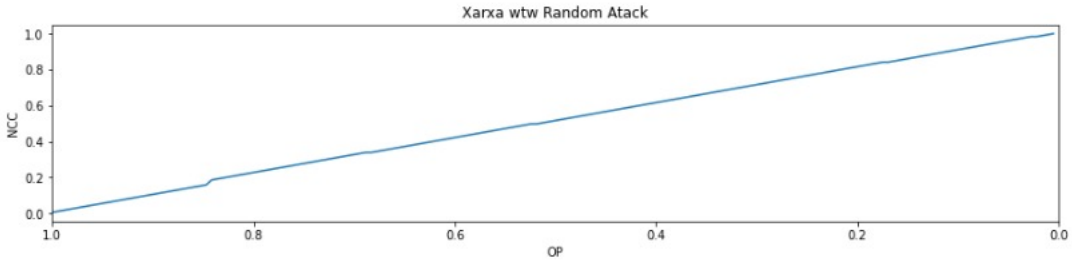

Figura 3.2: Random NCC wtw

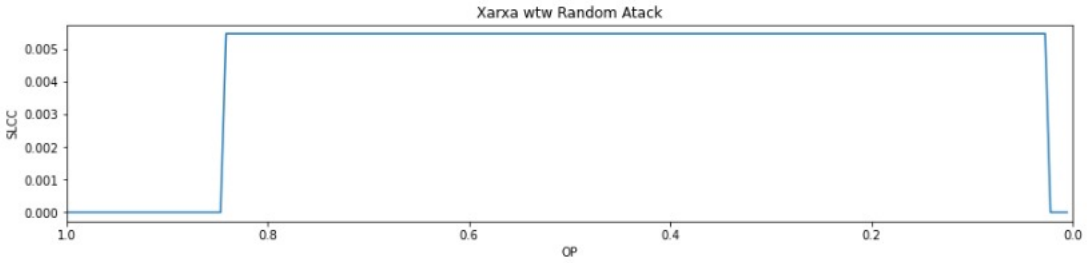

Figura 3.3: Random SLCC wtw

3.1.2 Atac per grau

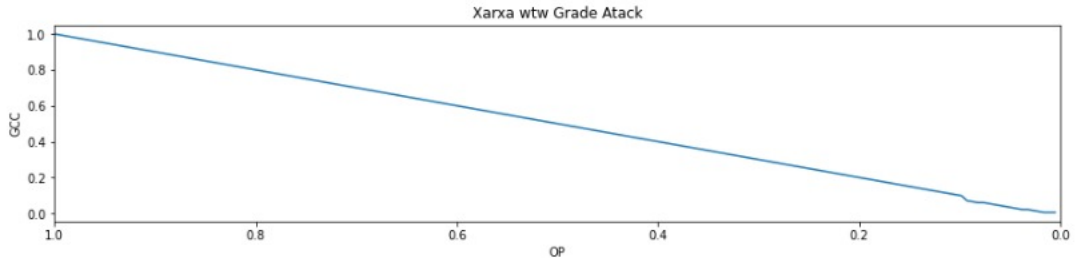

Figura 3.4: Grau GCC wtw

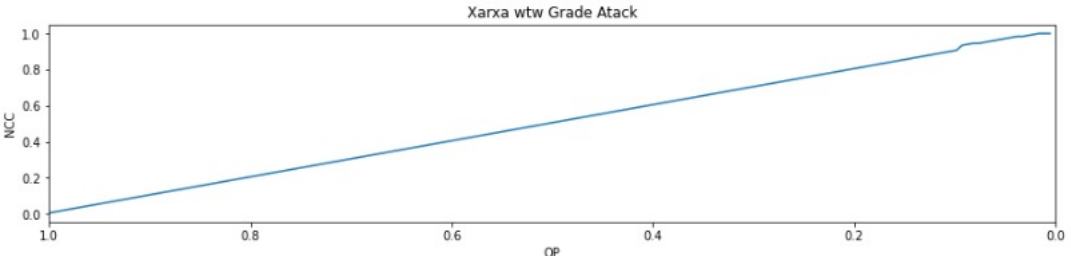

Figura 3.5: Grau NCC wtw

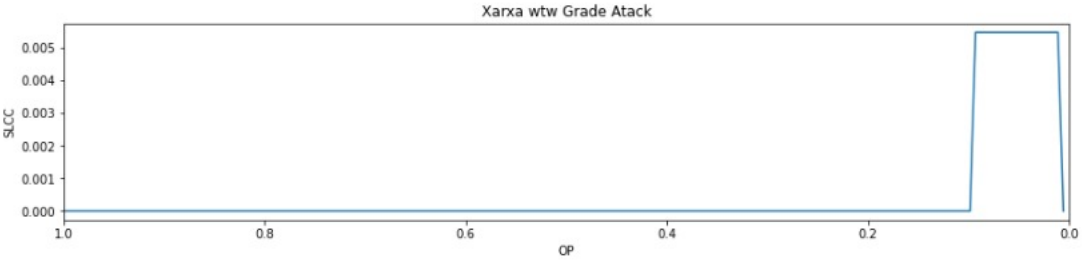

Figura 3.6: Grau SLCC wtw

3.1.3 Atac per strength

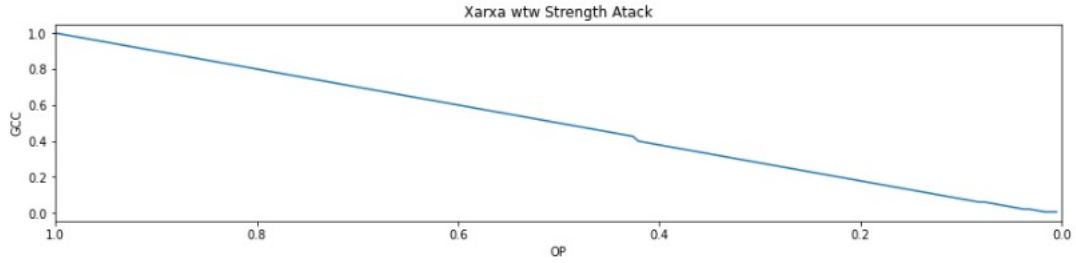

Figura 3.7: Strength GCC wtw

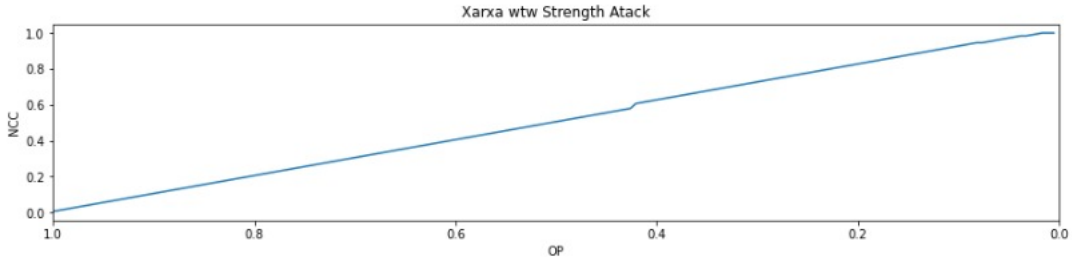

Figura 3.8: Strength NCC wtw

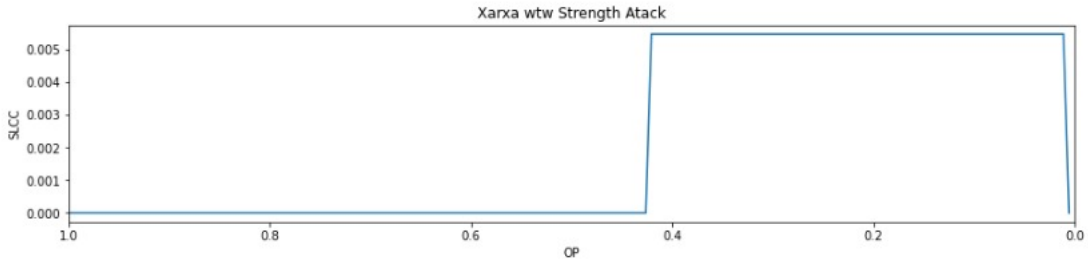

Figura 3.9: Strength SLCC wtw

3.2 Xarxa del correu electrònic de la URV

3.2.1 Atac aleatori

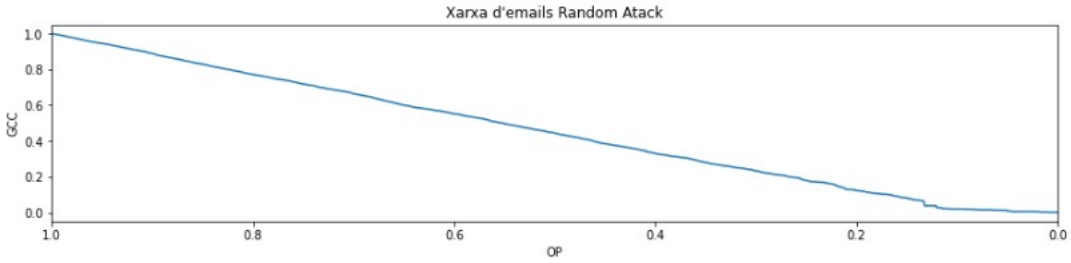

Figura 3.10: Random GCC email

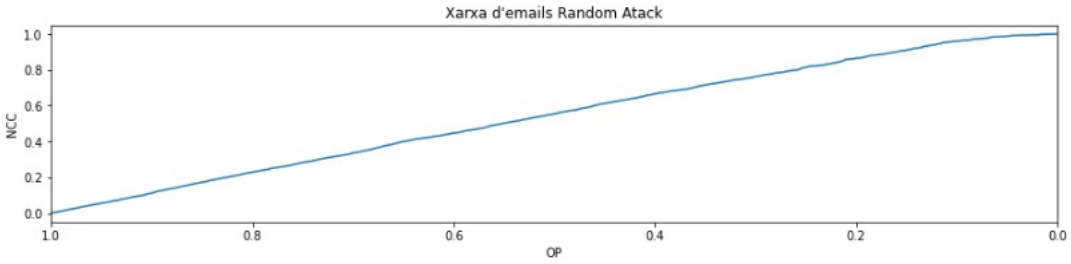

Figura 3.11: Random NCC email

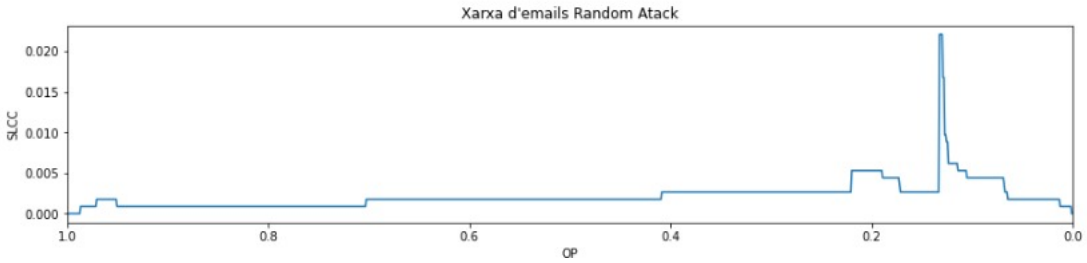

Figura 3.12: Random SLCC email

3.2.2 Atac per grau

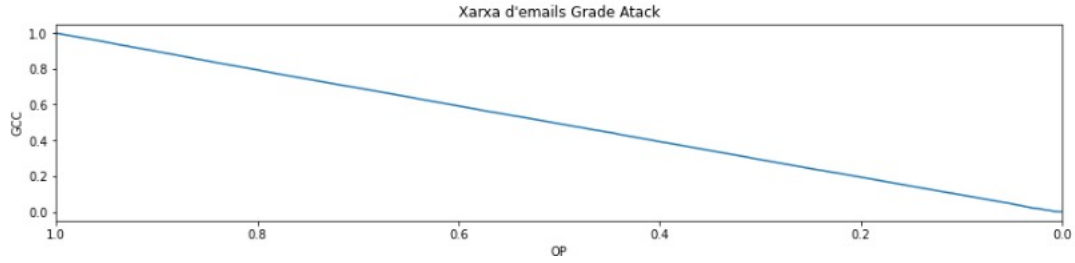

Figura 3.13: Grau GCC email

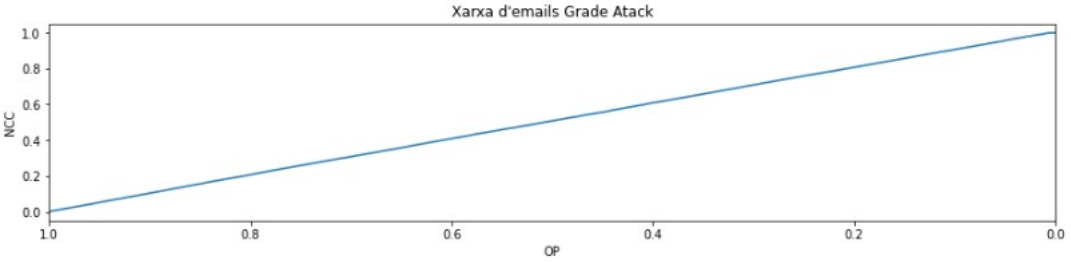

Figura 3.14: Grau NCC email

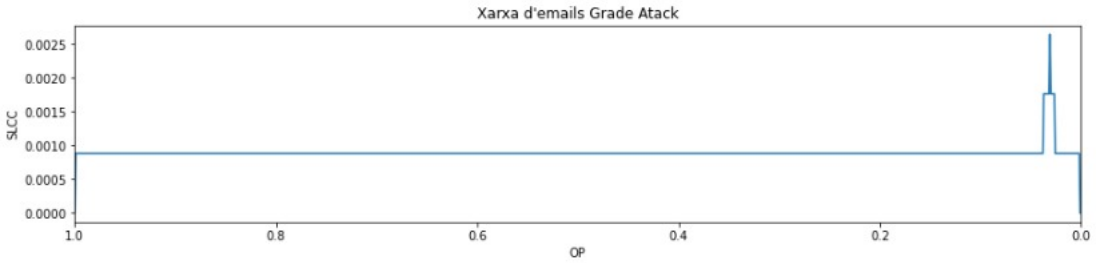

Figura 3.15: Grau SLCC email

3.2.3 Atac per strength

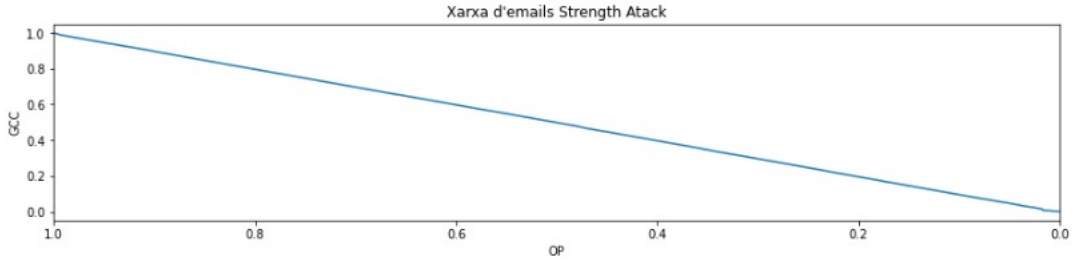

Figura 3.16: Strength GCC email

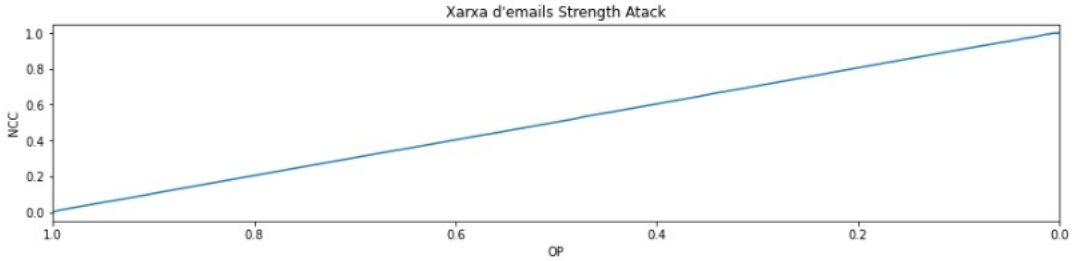

Figura 3.17: Strength NCC email

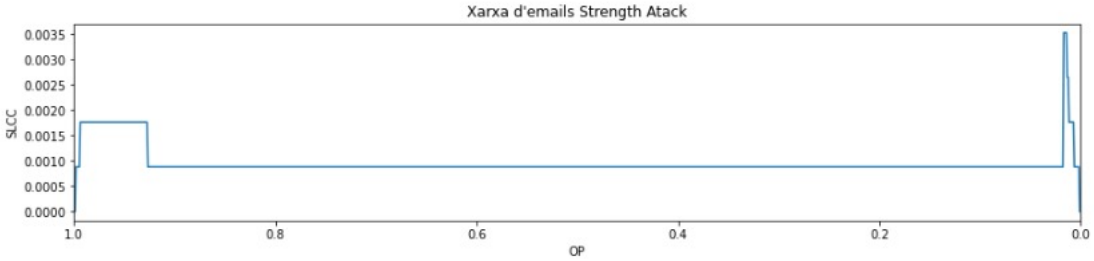

Figura 3.18: Strength SLCC email

3.3 Xarxa de transport aeri

3.3.1 Atac aleatori

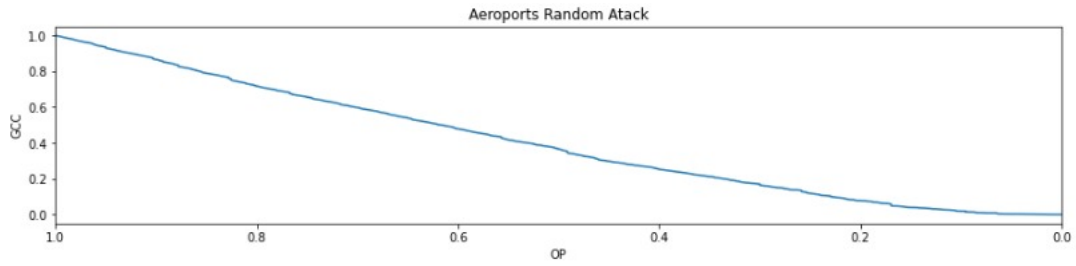

Figura 3.19: Random GCC transport aeri

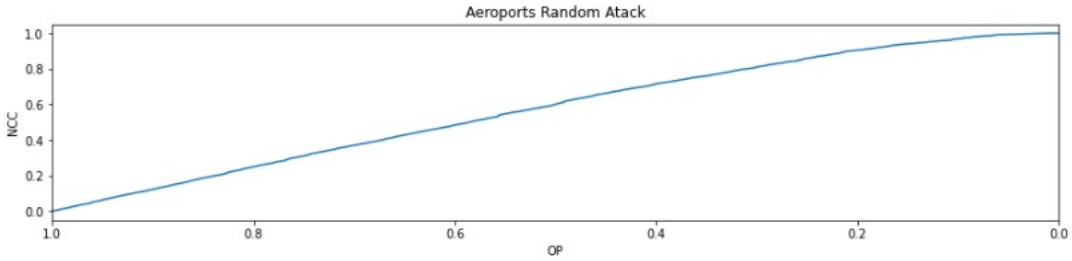

Figura 3.20: Random NCC transport aeri

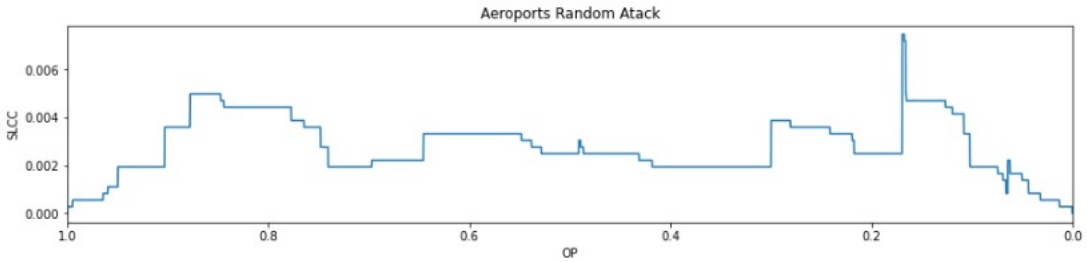

Figura 3.21: Random SLCC transport aeri

3.3.2 Atac per grau

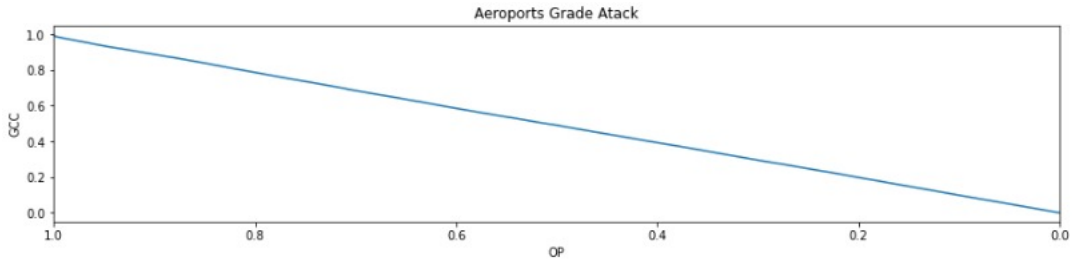

Figura 3.22: Grau GCC transport aeri

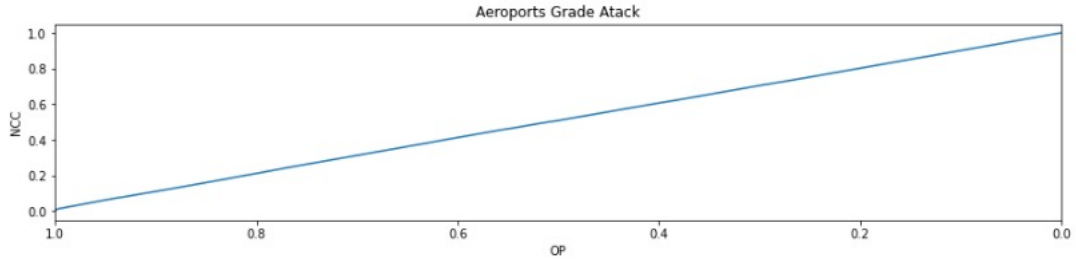

Figura 3.23: Grau NCC transport aeri

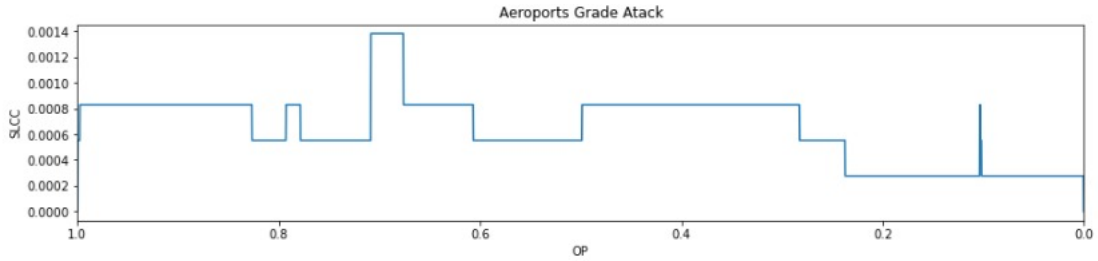

Figura 3.24: Grau SLCC transport aeri

3.3.3 Atac per transport aeri

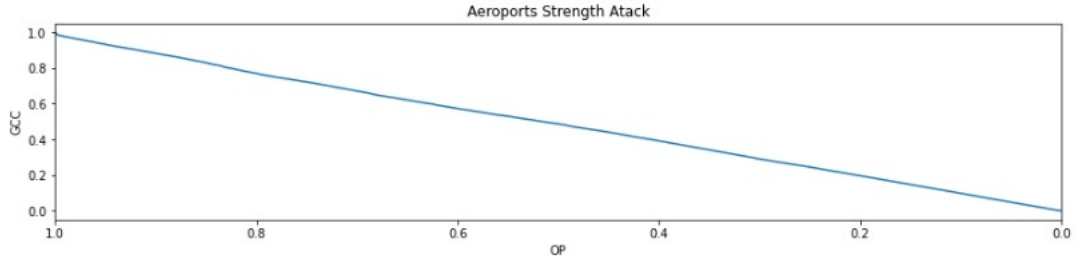

Figura 3.25: Strength GCC transport aeri

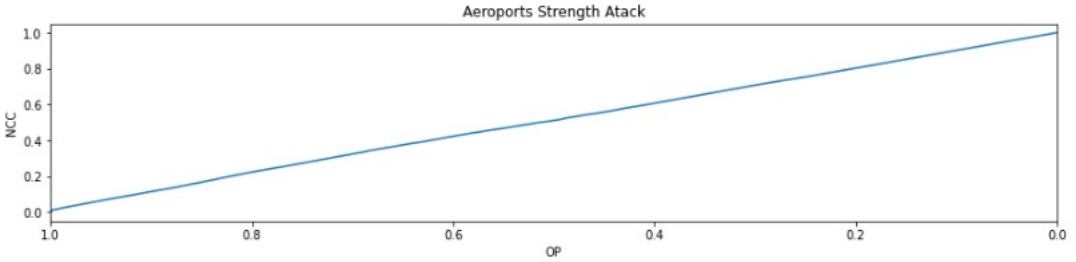

Figura 3.26: Strength NCC transport aeri

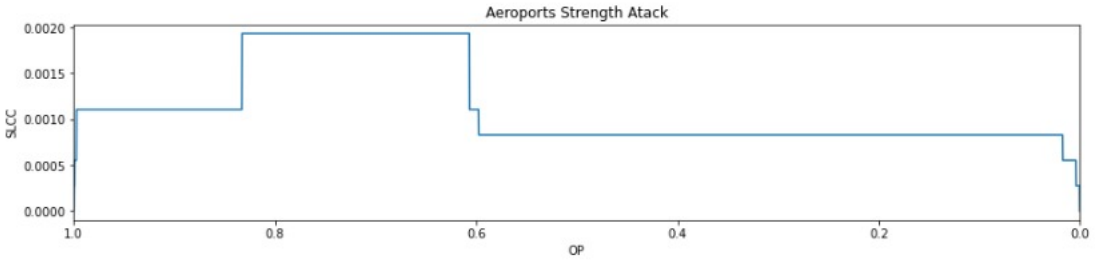

Figura 3.27: Strength SLCC transport aeri

3.4 Xarxa de distribució elèctrica USA

3.4.1 Atac aleatori

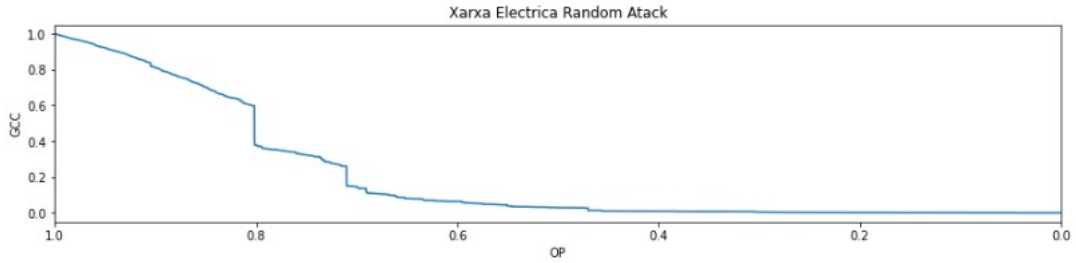

Figura 3.28: Random GCC xarxa de distribució elèctrica

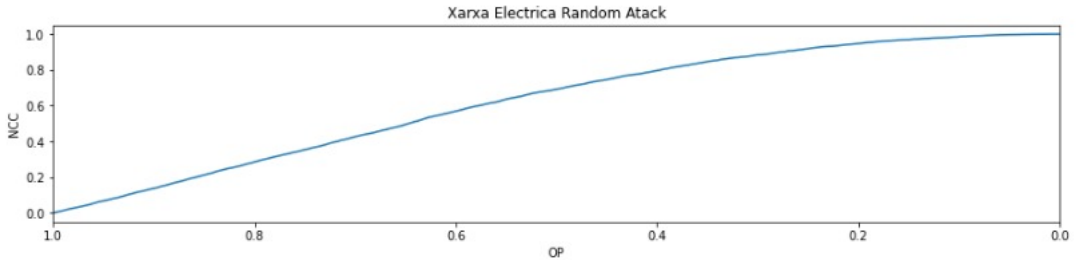

Figura 3.29: Random NCC xarxa de distribució elèctrica

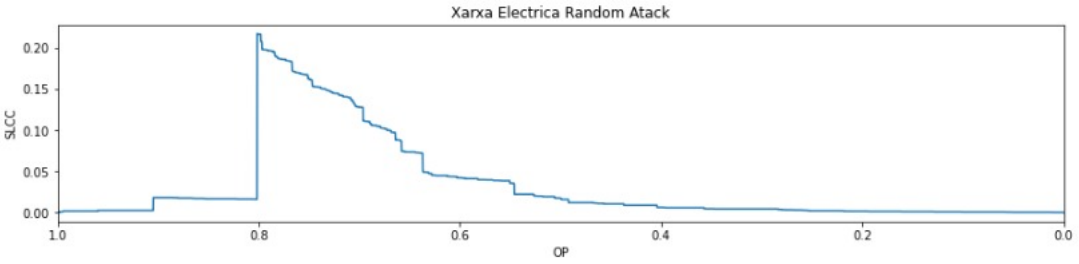

Figura 3.30: Random SLCC xarxa de distribució elèctrica

3.4.2 Atac per grau

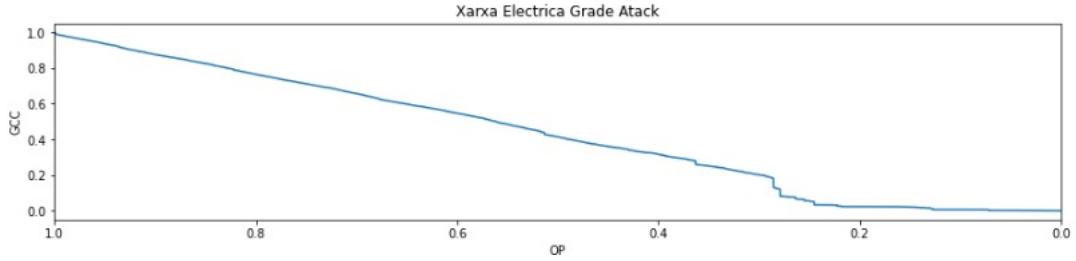

Figura 3.31: Grau GCC xarxa de distribució elèctrica

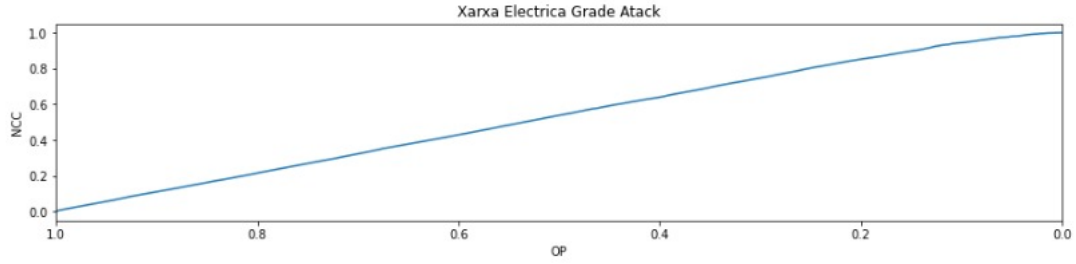

Figura 3.32: Grau NCC xarxa de distribució elèctrica

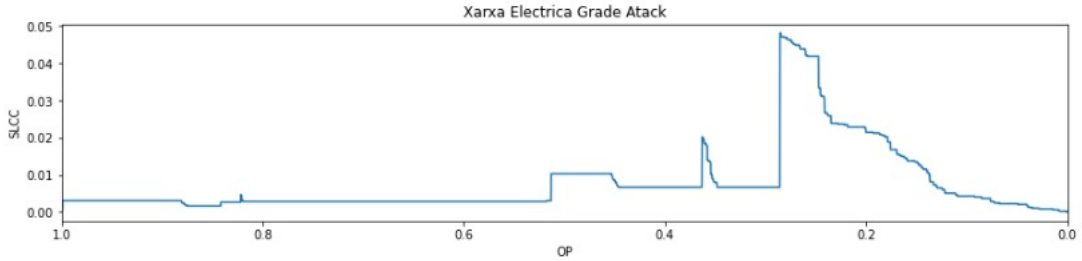

Figura 3.33: Grau SLCC xarxa de distribució elèctrica

3.4.3 Atac per xarxa de distribució elèctrica

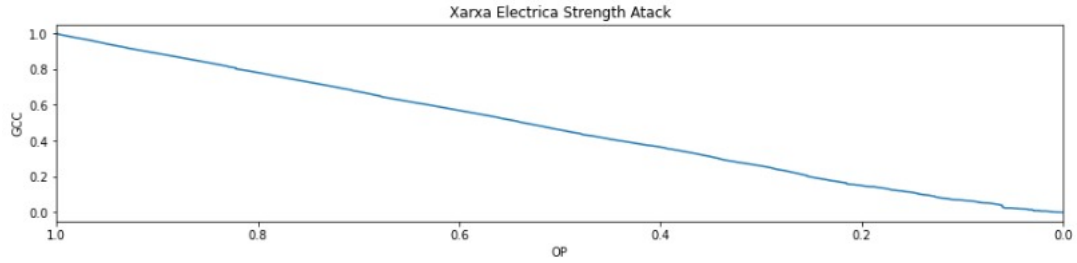

Figura 3.34: Strength GCC xarxa de distribució elèctrica

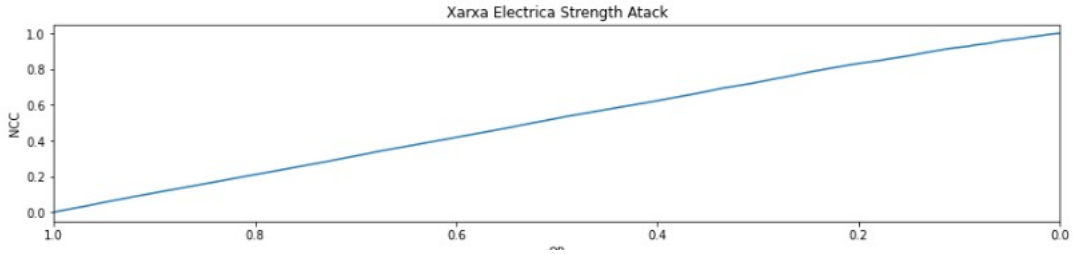

Figura 3.35: Strength NCC xarxa de distribució elèctrica

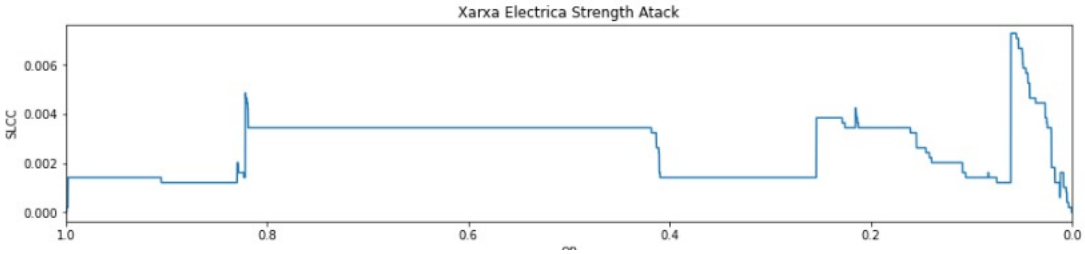

Figura 3.36: Strength SLCC xarxa de distribució elèctrica

4. Anàlisi de les gràfiques

Podem observar una tendència clara en les gràfiques de tots els atacs, on la GCC cau de forma lineal, o quasi lineal, la NCC creix de forma lineal quasi lineal creuant-se amb la GCC quan OP es 0,5 i la gràfica del SLCC, com es lògic, té pics ascendents quan la gràfica de la GCC té pics descendents.

Cabria esperar que, ja que els atacs per grau i strength són atacs dirigits les xarxes es desestabilitzin molt ràpidament, però com podem observar això no es així ja que aquestes xarxes són bastant robustes.

Índex de figures