Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili

Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili

Estructura de dades

Pràctica 2:

Estructures de dades avançades

Estudiants: Leandro Favio Gomez Racca

Miriam Gertrudix Pedrola

21 de juny del 2020

GEI Universitat Rovira i Virgili

Índex de la práctica:

1. Decisions de disseny:

1.1. Node:

Classe Nodo:

En aquesta classe conte tota la informació d'un node així com els mètodes per operar amb un node en concret.

Estructura:

Id: *enter*, identificador del node

Info: *genèric (T1)*, informació extra que se'ns dona amb el node (ex: coordenades, nom del node...)

listaDeEnlaces: *llista de enlace*, en aquesta llista es on guardarem tots els enllaços que conte el node.

Constructor:

Constructor:	descripció
`Nodo(int id, T info)`	Constructor per crear un node a partir de l'id i la informació si en té (si no en té li passem un null).
`Nodo(Nodo<T> n)`	Fer una copia del node actual al node que li passem per paràmetre

Mètodes:

tipus	mètode	descripció
void	`anadirEnlace(Enlace enlace)`	Mètode per afegir un enllaç
Int	`compareTo (Nodo<T> n,boolean str)`	Compara el grau dels nodes si li passem cert i l'strength si li passem fals.
Void	`eliminarEnlace(int idNodo)`	Mètode que elimina l'enllaç amb un node
Void	`eliminarEnlaces()`	Mètode que elimina tots els enllaços
Boolean	`existeEnlace(int idNodoB)`	Mètode per consultar si existeix un enllaç
Int	`getGrado()`	Getter del grau del node
Int	`getId()`	Getter de l'id del node
T	`getInfo()`	Getter de la informació del node
LinkedList	`getListaDeEnlaces()`	Getter de la llista d'enllaços
Double	`getStrength()`	Getter del pes total dels enllaços

Int	`gradoNodo()`	Mètode que retorna el grau del node
Int	`idNodoDestino(int idx)`	Mètode per obtenir l'id del node en la posició idx
Double	`pesoTotalEnlacesNodo()`	Mètode que retorna el pes total de tots els enllaços del node
String	`toString()`	Mètode to string

1.2. Enllaç:

Classe Enlace:

En aquesta classe conte tota la informació d'un enllaç així com els mètodes per operar amb un enllaç en concret.

Estructura:

nodoB: *enter*, identificador del node destí.

peso: *Double*, pes del enllaç.

Constructor:

Constructor:	descripció
`Enlace(int nodoB, java.lang.Double peso)`	Constructor per crear un enllaç a partir de l'id del node destí i el pes

Mètodes:

tipus	mètode	descripció
Int	`getNodoB()`	Getter del node destí
Double	`getPeso()`	Getter del pes
String	`toString()`	Mètode to string

1.3. Graf:

Per a la implementació del graf hem decidit crear tres classes diferents: *Grafo, Nodo* i *Enlace.*

Classe Grafo:

En aquesta classe hem implementat tota la estructura necessària per a implementar un graf així com els mètodes que utilitzem per operar amb aquest.

Com que la informació que guardarem als nodes del graf no sabem en quin format estarà hem implementat un genèric (T1);

Estructura:

listaNodos: *llista de nodes*, en aquesta llista es on guardarem tots els nodes que tinguem en el nostre graf.

nNodos: *enters*, atribut que ens indica la quantitat de nodes que conte el nostre graf.

Constructor:

Constructor:	descripció
`Grafo(int nNodos)`	Constructor per crear un node a partir del número de nodes
`Grafo(java.lang.String filePath)`	Constructor per crear un node a partir d'un fitxer .net
`Grafo(Grafo<T> other)`	Fer una copia del graf actual al graf que li passem per paràmetre

Mètodes:

tipus	mètode	descripció
void	`addEnlace(int nodoA, int nodoB, java.lang.Double peso)`	Mètode per afegir un enllaç entre dos nodes
Void	`addNodo(int id)`	Mètode per afegir un node sense informació al graf
Void	`addNodo (int id, T info)`	Mètode per afegir un node al graf
Boolean	`enlaceExiste (int idNodoA, int idNodoB)`	Mètode per consultar si existeix un enllaç
ArrayList	`getListaNodos()`	Getter de la llista de nodes
Int	`getnNodos()`	Getter del numero de nodes que hi ha a la llista
String	`toString()`	Mètode to string

1.4. Heap:

Classe maxHeap:

En aquesta classe hem implementat tota la estructura necessària per a implementar l'estructura d'ordenació maxHeap.

Hem implementat un genèric (T1) perquè no sabem quin tipus de dades guardarem als nodes;

Estructura:

heap: *llista de nodes ordenats*, en aquesta llista es on guardarem tots els nodes que tinguem la nostra xarxa.

tipo: *boolea*, aquest atribut el farem servir per saber si evaluem grau (fals) o strength (cert)

Constructor:

Constructor:	descripció
`MaxHeap()`	Constructor basic que utilitzara per defecte en el cas de grau
`MaxHeap(boolean str)`	Constructor per escollir la key

Mètodes:

tipus	mètode	descripció
void	`add(Nodo<T> n)`	Mètode per afegir un node al heap
Void	`descender(Nodo<T> n)`	Mètode per fer descendir el node en cas de que sigui necessari
Void	`eliminar()`	Mètode per eliminar el node amb el valor més alt
Boolean	`isVacio()`	Mètode per consultar el heap es buit
Nodo<T>	`ver()`	Mètode per consultar el node més gran
Nodo<T>	`verYeliminar()`	Mètode per consultar el node més gran i eliminar-lo
String	`toString()`	Mètode to string

1.5. Percolació:

Classe percolacion:

En aquesta classe hem implementat totes les operacions i estructures necessàries per a fer els càlculs i les avaluacions de la percolació

Estructura:

listaNodosActivos: *llista d'index de nodes actius*, en aquesta llista es on guardarem tots els nodes que no han estat eliminats.

heap: *maxHeap*, heap que utilitzarem per a evaluar.

nNodosActivos: *enter*, aquest atribut ens indica quants nodes actius tenim.

NodosEliminar: *enter*, aquest atribut ens indica quants nodes hem d'eliminar.

op: double, aquest atribut ens indica el valor del op del nostre graf.

ncc: double, aquest atribut ens indica el valor del ncc del nostre graf.

gcc: double, aquest atribut ens indica el valor del gcc del nostre graf.

slcc: double, aquest atribut ens indica el valor del slcc del nostre graf.

Constructor:

Constructor:	descripció
`Percolacion(Grafo<T> g)`	Constructor que s'utilitza per evaluar a partir d'un graf

Mètodes:

tipus	mètode	descripció
void	`evaluacionPercolacion (int opt)`	Mètode que crea un fitxer .csv amb totes les evaluacions fins a destruir completament el graf
int	`eliminarNodoAleatorio()`	Mètode per eliminar un node de forma aleatòria, ens retorna l'id del node que volem eliminar
int	`eliminarNodoGrado()`	Mètode per eliminar el node amb el grau més alt, ens retorna l'id del node que volem eliminar
int	`eliminarNodoStr()`	Mètode per eliminar el node amb el pes total més alt, ens retorna l'id del node que volem eliminar

int	`eliminarNodoHeapGrado()`	Mètode per eliminar el node amb el grau més alt utilitzant maxheap, ens retorna l'id del node que volem eliminar
int	`eliminarNodoHeapStr()`	Mètode per eliminar el node amb el pes total més alt utilitzant heap, ens retorna l'id del node que volem eliminar
void	`actualizarGccSlcc(int i)`	Mètode per actualitzar i evaluar el gcc i el slcc
void	`evaluar()`	Mètode que calcula tots els valors de la percolació
double	`getOp()`	Getter de l'op
double	`getNcc()`	Getter del ncc
double	`getGcc()`	Getter del gcc
double	`getSlcc()`	Getter del slcc

2. Instruccions d'execució:

2.1. Execució percolació:

Per a una fàcil execució hem fet un programa principal amb un menú. En aquest menú se'ns demana triar un dels fitxer ".net" que sens faciliten així com un que hem afegit per fer tests.

```
1.networks\basico.net
2.networks\airports_UW.net
3.networks\email_URV-edges_betw.net
4.networks\powergrid_USA-edges_betw.net
5.networks\wtw2000-sym.net
Escoja una red
```

També sens demana quin mètode d'extirpació volem utilitzar:

```
1.Metodo aleatorio
2.Metodo grado
3.Metodo "strength"
4.Metodo grado con heap
5.Metodo "strength" con heap
Escoja un metodo de extirpacion
```

Descripció:

Opció	Mètode	descripció
1	aleatorio	Va eliminant mètodes de forma aleatòria
2	Grado	Va eliminant el node que te mes grau recalculant el grau de tots els nodes
3	Strength	Va eliminant el node que te més pes recalculant el pes de tots els nodes
4	Grado con heap	Va eliminant el node que te més grau utilitzant el maxHeap
5	Strength con heap	Va eliminant el node que te el pes més gran utilitzant el maxHeap

*Grado i Strength son implementacions extra

Després de especificar el test i el mètode d'extirpació s'executarà la percolació i ens generarà un document ".csv" i s'executarà un script que hem creat amb python que ens crearà una imatge ".png". El document csv es el resultat del op, ncc, gcc i slcc de cada iteració del programa. I el png és la gràfica amb aquests paràmetres.

També ens imprimirà per pantalla el temps que ha tardat en executar-se.

En el cas de l'aleatori ens fa el procés 100 vegades i ens fa una mitja dels resultats del op, el ncc, el gcc i el slcc així com el temps d'execució per a obtenir resultats més fiables.

3. Resultats de la percolació:

Anàlisi dels paràmetres:

L'op representa la quantitat de nodes actius del graf. Com que eliminem una quantitat constant de nodes a cada iteració això provoca que la gràfica també decreixi constantment. En quan al Ncc, quantitat de components connexes, amb aquesta component podem veure quant robusta es la nostra xarxa i podem observar com a mesura que s'apropa a 1 es va fragmentant.

Del Gcc ens dona la component connexa amb mes nodes. Aquest paràmetre també decent de la robustesa de la xarxa. En una xarxa on la connectivitat dels nodes es fluixa el gcc cau en picat en les primeres iteracions, ja que a mesura de que eliminem nodes la xarxa es va fragmentant i per tant la quantitat de nodes connectats també disminueix. Amb el Slcc ens mostra un petit pic. Quan es dona aquest pic significa que hi ha dos components connexes molt grans a la nostra xarxa. Per això aquest pic sol coincidir amb la corba del gcc. Es podria dir que es un moment en que la xarxa es parteix en dos.

Anàlisis de les gràfiques:

(Les gràfiques estan adjuntades en l'annex i referenciades a continuació)

En el cas de l'extirpació aleatòria al ser un atac aleatori com el seu nom indica podem veure la connectivitat de la nostra xarxa, es a dir, si el gcc cau molt ràpid es perquè la xarxa es poc connexa. Un clar exemple es la comparació de la xarxa email_URV-edges_betw (il·lustració 4) i la xarxa del powergrid (il·lustració 7). En la email que es una xarxa molt connexa veiem que el gcc tarda moltes més iteracions en caure que en la xarxa del powergrid que no ho es tant.

Després podem analitzar el wtw2000-sym (il·lustracions 10, 11 i 12), en aquest cas ens és indiferent quin tipus d'extirpació utilitzem perquè al ser una xarxa casi complerta, es a dir, com que gairebé tots els nodes estan connectats entre ells, tindran un comportament similar ultilitzant qualsevol dels tres mètodes implementats.

EN el cas del powergrid (il·lustracions 7, 8 i 9) i airports (il·lustracions 1, 2 i 3) son dos xarxes que tenen pocs enllaços en comparació amb la gran

quantitat de nodes que tenen, això fa que quan extirpem en mode grau o strength podem fragmentar la xarxa molt més ràpid.

Quantitat d'enllaços airports: 14.141 Quantitat d'enllaços per ser un graf complert: 6.543.153 Percentatge connexa: 0,2%	Quantitat d'enllaços web: 10.136 Quantitat d'enllaços per ser un graf complert: 16.653 Percentatge connexa: 60%

Per ultim si ens fixem en la xarxa emails (il·lustracions 4, 5 i 6) es més robusta que la powergrid i l'airport en relació quantitat de nodes respecte enllaços i això fa que els atacs dirigits siguin més potents que l'aleatori tot i que els aguanti prou bé.

4. Temps de càlcul:

Temps en segons:

	aleatori	Grau heap	strength heap
airports	7,06803	3,742211	3,845519
email	0,72463	0,4576817	0,4668542
powergrid	8,80329	8,101652	7,8326319
web	0,0432	0,148421	0,1361667

Especificacions de l'ordinador:

Processador:	cpu ryzen 5 3600 gpu nvidia geforce rtx 2060 super
Placa base:	msi tomahawk b450
Memòria:	ram: 16(2x8)gb ddr4 3200mhz ssd crucial MX500
Sistema Operatiu:	Windows 10 student (més recent)
Versió de java:	Java 14.0.1 (més recent)

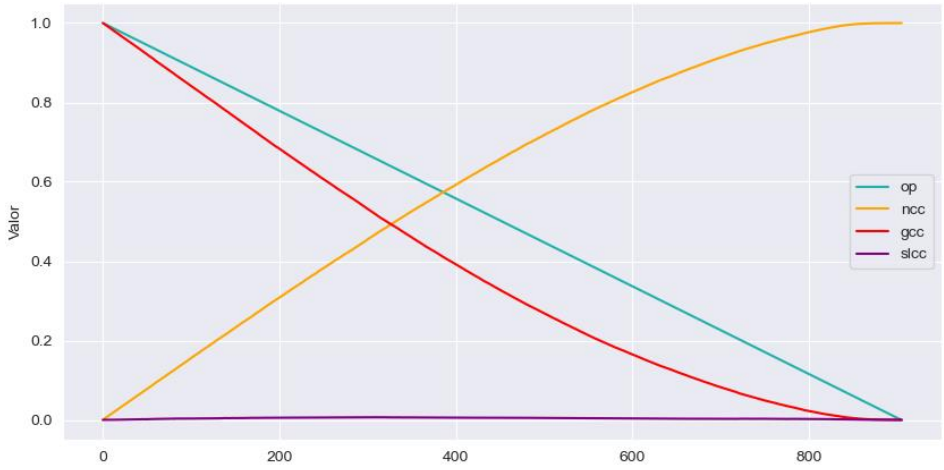
5. Interpretació dels resultats:

Un cop hem realitzat totes les proves i hem executat tots els tests hem tret les nostres pròpies conclusions. Si la xarxa te una bona conductivitat (que els nodes estiguin el màxim enllaçats entre ells possible), ens serà indiferent com l'ataquem ja que tindrà un comportament similar en qualsevol tipus de extirpació de les que hem vist. Però en el cas de que tinguem una xarxa que no pugui garantir aquesta bona connectivitat llavors un atac dirigit (de grau o de strength) es molt més efectiu i farà molt més mal.

6. Annex:

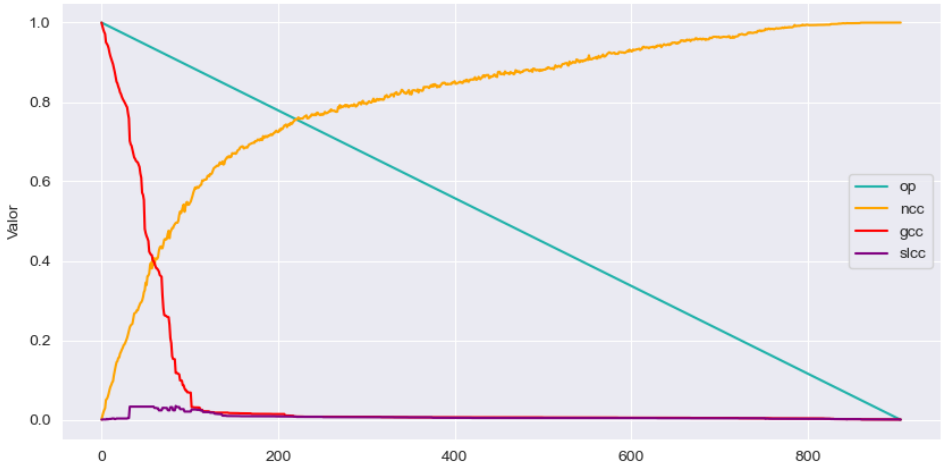
6.1. Xarxa airports_UW :

6.1.1. Mètode d'extirpació aleatori



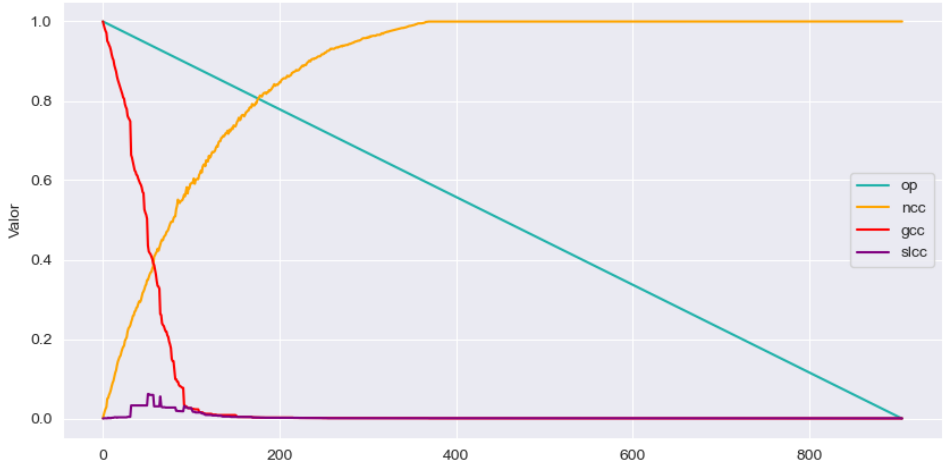
Il·lustració 1

6.1.2. Mètode d'extirpació per grau



Il·lustració 2

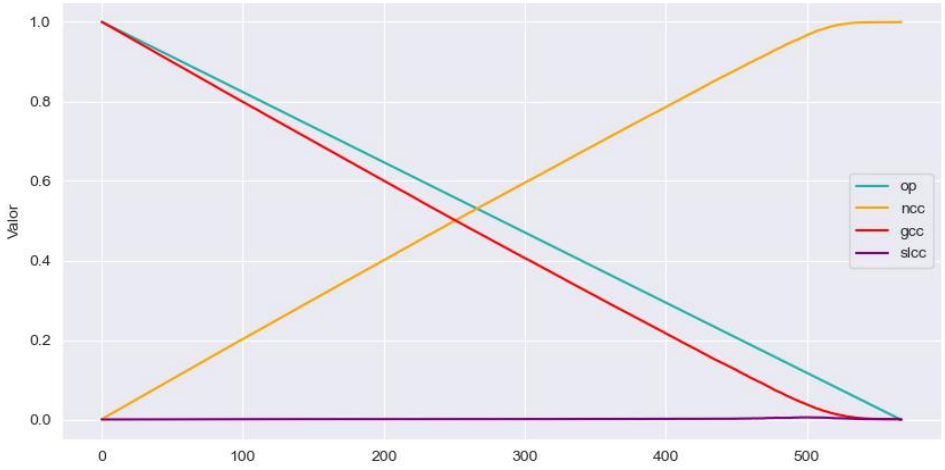
6.1.3. Mètode d'extirpació per strength



Il·lustració 3

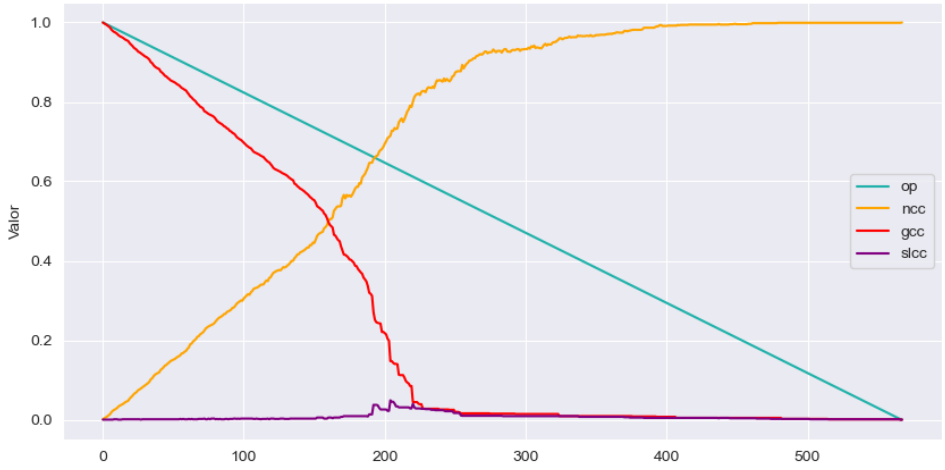
6.2. Xarxa email_URV-edges_betw :

6.2.1. Mètode d'extirpació aleatori



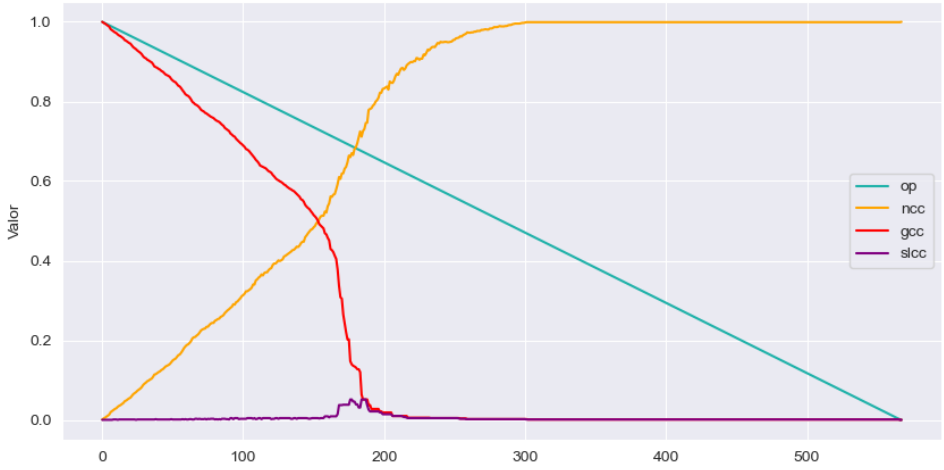
Il·lustració 4

6.2.2. Mètode d'extirpació per grau



Il·lustració 5

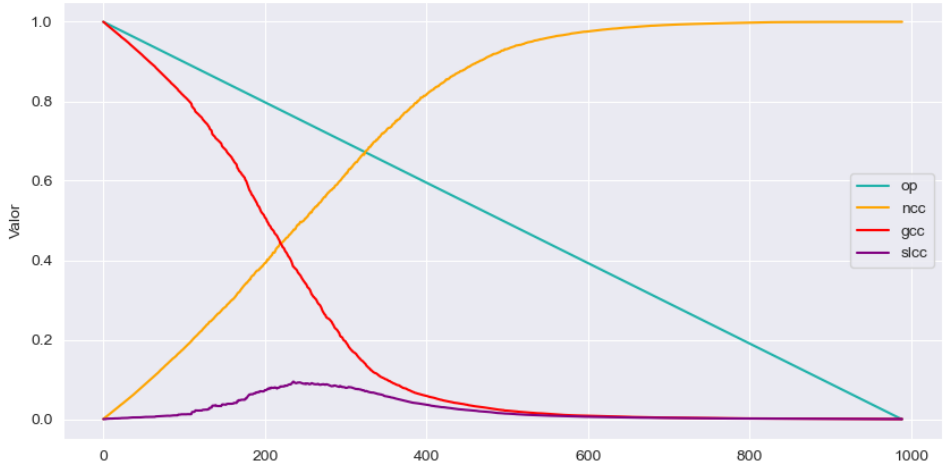
6.2.3. Mètode d'extirpació per strength



Il·lustració 6

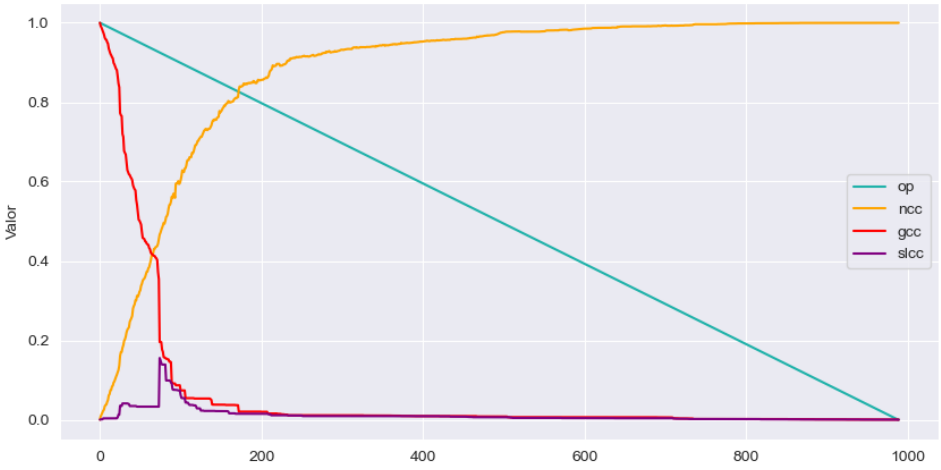
6.3. Xarxa powergrid_USA-edges_betw :

6.3.1. Mètode d'extirpació aleatori



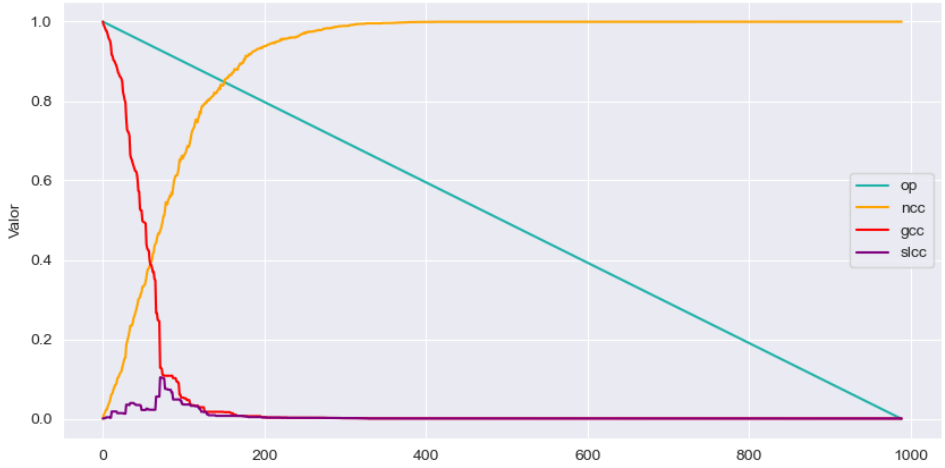
Il·lustració 7

6.3.2. Mètode d'extirpació per grau



Il·lustració 8

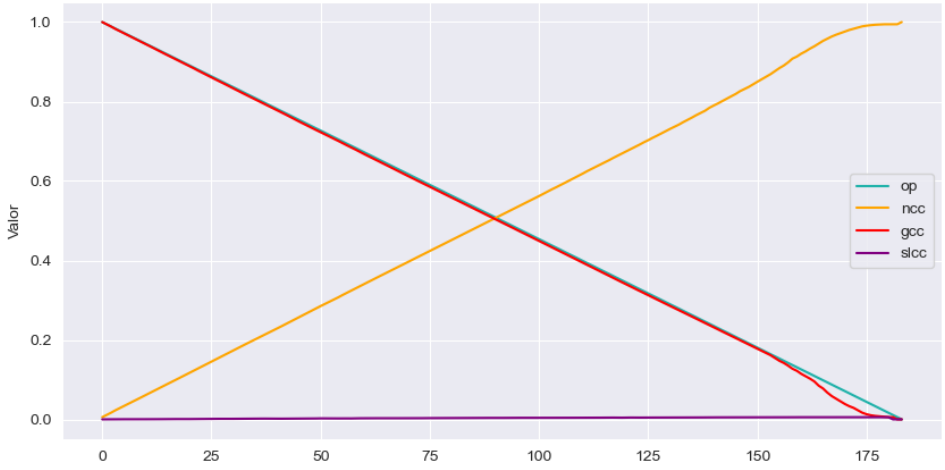
6.3.3. Mètode d'extirpació per strength



Il·lustració 9

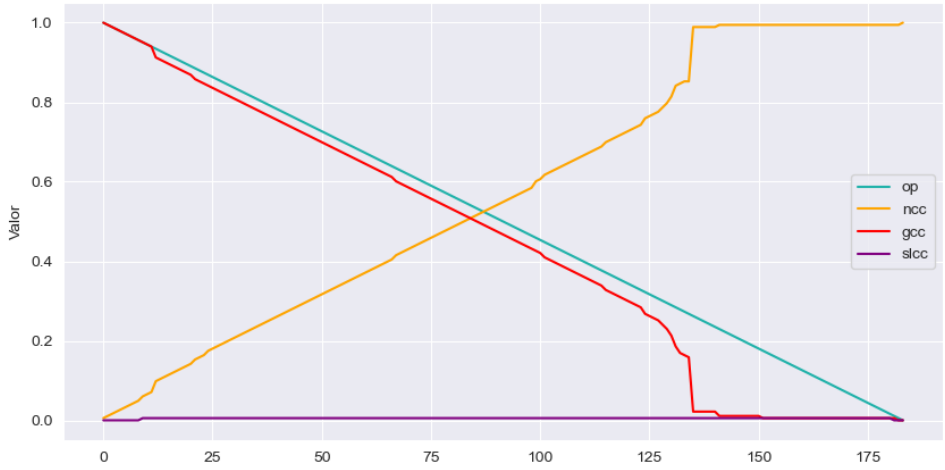
6.4. Xarxa wtw2000-sym :

6.4.1. Mètode d'extirpació aleatori



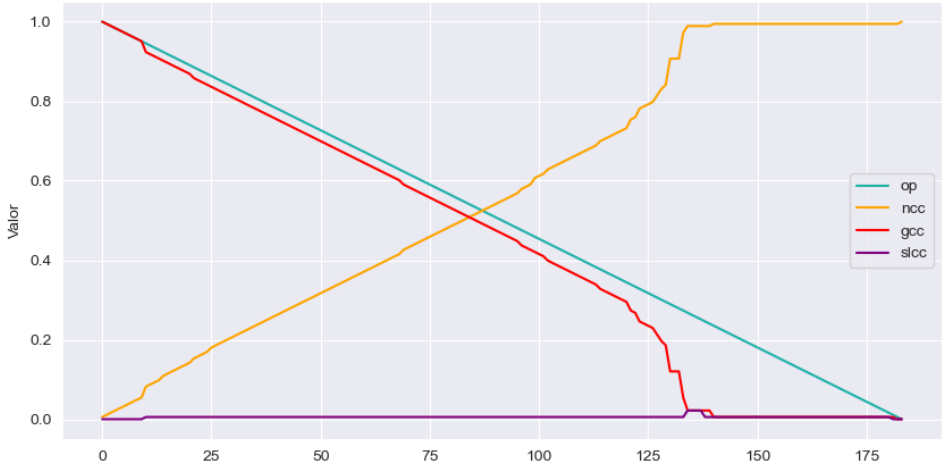
Il·lustració 10

6.4.2. Mètode d'extirpació per grau



Il·lustració 11

6.4.3. Mètode d'extirpació per strength



Il·lustració 12