Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Elèctrica

Títol: Self-healing per a xarxes elèctriques

Document: 1. Memòria

Alumne: Marc Padilla Morales

Tutor: Sergio Herraiz Jaramillo

Departament: Enginyeria elèctrica, electrònica i automàtica

Àrea: Enginyeria elèctrica

Convocatòria (mes/any): juny/2022

INDEX

1. INTRODUCCIÓ

1.1. Antecedents

Degut a causes meteorològiques o a fenòmens aleatoris és inevitable que la xarxa pugui danyar-se provocant l'aparició de faltes o pertorbacions en el sistema elèctric. Amb el desenvolupament de les xarxes intel·ligents es pretén dotar a la xarxa de la capacitat de poder aïllar les faltes o pertorbacions de forma autònoma de manera que es pugi continuar amb el subministrament d'energia i reduir l'impacte de la falta. Per aquest motiu s'anomena "self-healing", ja que es realitza una reconfiguració de la xarxa.

1.2. Objecte

La reconfiguració es pot planificar en base a diferents criteris com minimitzar els consumidors afectats o el número d'intervenció d'elements de commutació. Aquests criteris comporten la modificació de diferents consignes en els elements de maniobra de la xarxa. L'objectiu del treball es realitzar un estudi sobre les possibles solucions "self-healing" a adoptar en una xarxa de prova envers a l'aparició de faltes. Mitjançant l'ús d'algoritmes de cerca de solucions es calcularan les actuacions dels elements de maniobra amb diferents criteris, tot simulant l'operació de la xarxa (amb les noves consignes i la falta aïllada) garantint que la xarxa funciona en règim segur i no sobrepassa cap límit establert.

1.3. Abast

Es pretén dissenyar una estratègia per la planificació del "self-healing" i buscar solucions òptimes per a les consignes dels elements utilitzant una xarxa de test. Seguidament s'analitzarà l'estat de la xarxa mitjançant la simulació i es descartaran possibles criteris si sobrepassen límits de seguretat. Finalment es discutirà quines solucions a adoptar són vàlides i quins beneficis i conseqüències comporten.

2. SELF-HEALING

Els sistemes elèctrics son cada vegada més complexes i inclouen més elements, ja sigui en les càrregues, amb la incorporació dels vehicles elèctrics i cada vegada més incorporació de càrregues no lineals. En la generació amb les fonts de generació distribuïda o els punts d'emmagatzematge d'energia. Per aquests motius les xarxes amb self-healing son més necessàries que mai per a millorar la fiabilitat de la xarxa.

El terme self-healing en aquest treball es focalitza en l'aïllament de faltes de forma automàtica. Es dissenya un algorisme per a en cas de falta poder detectar les zones critiques i actuar sobre aquestes i aïllar la falta. D'aquesta manera la falta queda aïllada evitant la propagació d'aquesta. Una vegada aïllada la falta, poder aproximar una configuració que minimitzi paràmetres com les pèrdues de potencia, l'afectació als usuaris, entre d'altres segons convingui. Per al canvi de configuració s'utilitza la obertura i tancament de diferents interruptors i per a aproximar la solució s'implementa la cerca per mitja d'un algorisme evolutiu que cerca solucions a partir d'unes dades inicial i combinacions d'aquestes que minimitzen una funció objectiu.

3. PANDAPOWER

En aquest estudi s'ha utilitzat l'eina pandapower que esta construïda sobre l'eina d'anàlisis de dades pandas i sobre la toolbox de PYPOWER ambdues sobre l'entorn de Python. La llibreria pandapower permet la creació de xarxes elèctriques i la simulació d'aquestes. En aquest s'ha partit des d'uns fitxers .xlsx on hi ha les dades de les xarxes a estudiar i aquestes s'han importat amb llibreries de Python per posteriorment crear la xarxa amb l'eina pandapower.

Entre les seves funcionalitats en destaquen l'analitzador de flux de potències que permet estudiar els diferents fluxos de potència en la xarxa elèctrica. En el cas del treball s'utilitza per a determinar si les solucions trobades per l'algorisme de cerca de solucions es troben dins dels rangs de seguretat en termes de flux de potència.

Una altra funcionalitat a destacar es la possibilitat de realitzar cerques en grafs a través de la llibreria NetworkX. En aquest treball s'utilitzen els grafs i les cerques topològiques per a extreure'n les zones de falta que cal aïllar.

En darrer lloc també s'utilitzen en aquest estudi eines gràfiques per a mostrar gràficament diferents paràmetres de la xarxa, com la localització i els flux de potència. Hi ha la possibilitat de mostrar un esquema de la xarxa sobre un mapa i de mostrar amb diferents colors si paràmetres de seguretat de la xarxa sobrepassen certs límits.

3.1. Dades inicials

Per a utilitzar la llibreria del pandapower cal disposar del model de la xarxa en format pandapower. En aquesta memòria es mencionen dues xarxes, aquestes xarxes s'han introduït al pandapower a través d'un fitxer .xls que ja tenia el format requerit. Els arxius de les xarxes es poden trobar en la carpeta del projecte.

El model de la xarxa inclou paràmetres com les característiques de les línies, transformadors, busos, entre d'altres. És a dir variables com la tensió nominal, càrrega màxima, consums, impedàncies, etc.

4. PREPARACIÓ DE LES DADES

En el següent apartat s'exposen les diferents funcions que s'utilitzen a l'hora de preparar les dades per a l'execució del posterior algorisme.

4.1. Identificar zones

Per tal de poder identificar els interruptors amb que es podran generar solucions en l'algoritme de solucions, el primer pas es identificar les diferents zones que queden aïllades quan els interruptors queden oberts (figura 1). D'aquesta manera si es presenta un curtcircuit en una zona determinada, els interruptors connectats a aquella zona romandran oberts per tal d'aïllar la falta i aquests no podran exercir com a variables de possibles solucions.

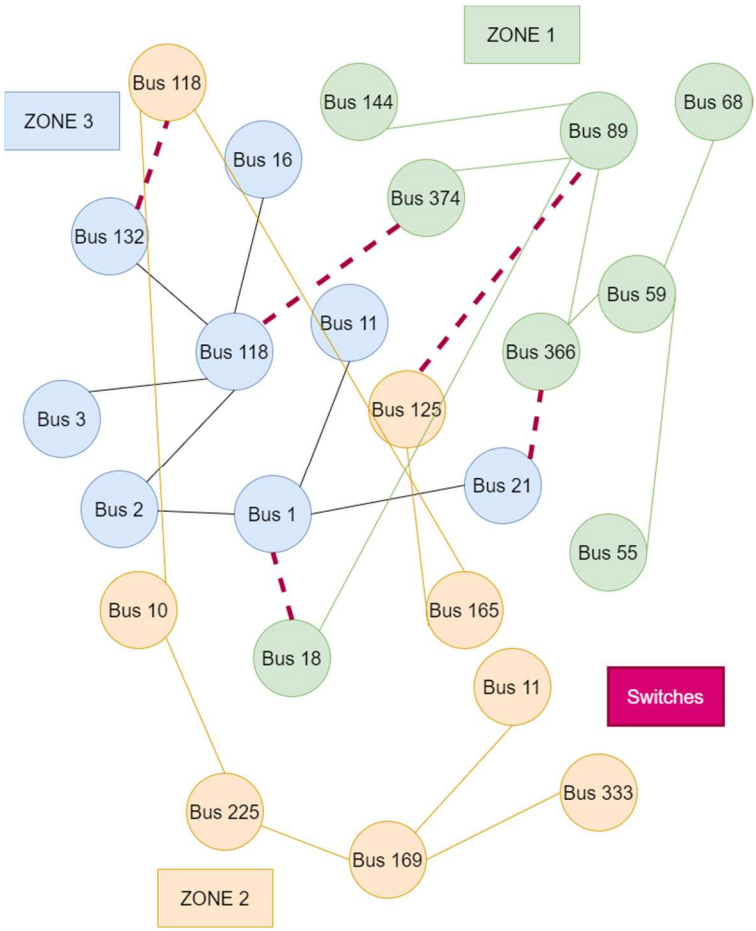

Figura 1: Separació entre zones mitjançant la obertura dels interruptors

Figura 2: Diagrama funció identifyZones()

Amb el codi que es troba a continuació, s'identifiquen i anomenen diferents zones. En el diagrama de la figura 2, es poden apreciar els passos seguits a l'hora d'implementar la funció, els blocs amb forma de fletxa representen entrades i sortides. D'altra banda els blocs amb forma quadrada representen accions. La primera acció consisteix a crear una variable auxiliar que contindrà la xarxa, d'aquesta manera si es fan canvis es canviaran a la variable auxiliar i no a la xarxa. Seguidament cal obrir tots els interruptors, d'aquesta manera es crearan diverses zones i en cas que es presenti un curtcircuit caldrà que la zona afectada s'aïlli. En tercer lloc, es crea un graf ja que així la llibreria pandapower permet fer una cerca topològica de quins busos estan interconnectats entre ells i així poder identificar zones. La funció acaba retornant un diccionari que identifica els diferents busos de la xarxa amb una zona determinada. Cal remarca la importància d'escollir un diccionari, ja que aquests no permeten

elements duplicats i redueix la probabilitat d'errors. En el cas d'escollir per exemple una llista per a desar les dades es podria duplicar una zona. A la part superior s'ha inserit una imatge representativa del procés per a ajudar a entendre la separació que es fa entre zones.

```
def identifyZones(net): #Creating a dictionary with all areas
represented with
    auxNet = net #Creting a new net for open all switches
    auxNet.switch.closed.loc[auxNet.switch.index] = False
    netGraph = pp.topology.create_nxgraph(auxNet)
    ar_dict = {}
    i = 0
    for area in pp.topology.connected_components(netGraph):
#Dictionary because not allows duplicate members
        ar_dict[i] = area
        i = i+1
    return ar_dict
```

La imatge de la figura 3 reflexa un exemple de la creació d'un diccionari de busos i el format que adopta el diccionari.

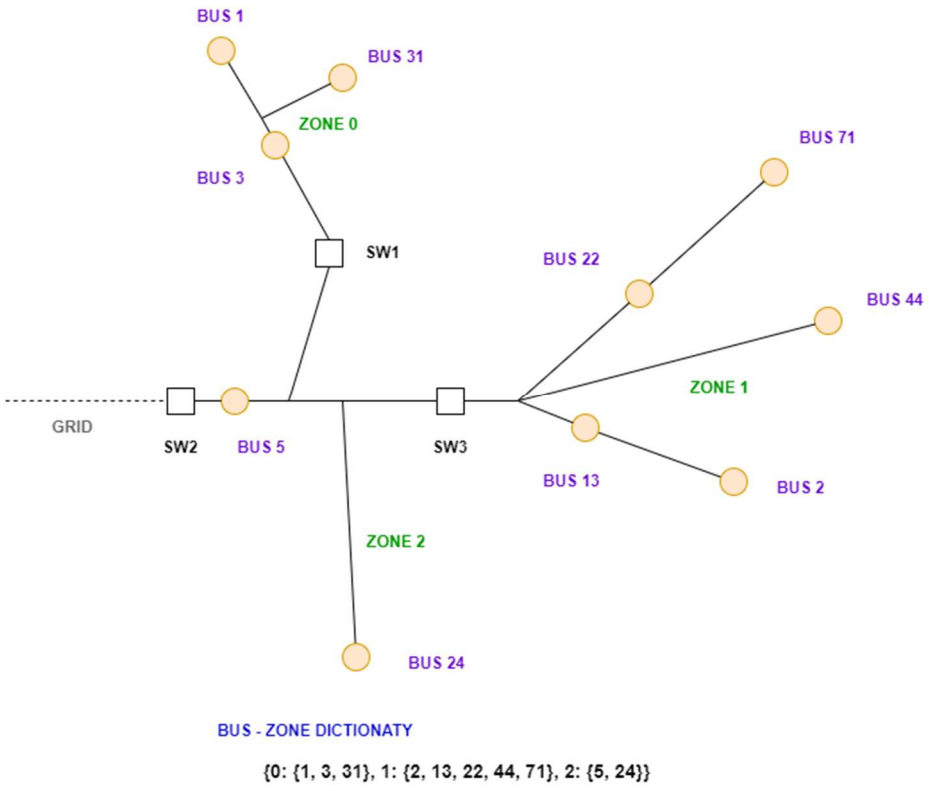

Figura 3: Creació d'un diccionari de busos zones

4.2. Identificar busos amb zones

A continuació es necessària la creació d'una funció per a poder identificar un bus amb una zona. Dit d'una altra manera quan es disposa d'un bus, per exemple el bus d'un curtcircuit i es vol identificar la zona del bus en el cas exemple per aïllar-la s'utilitza aquesta funció.

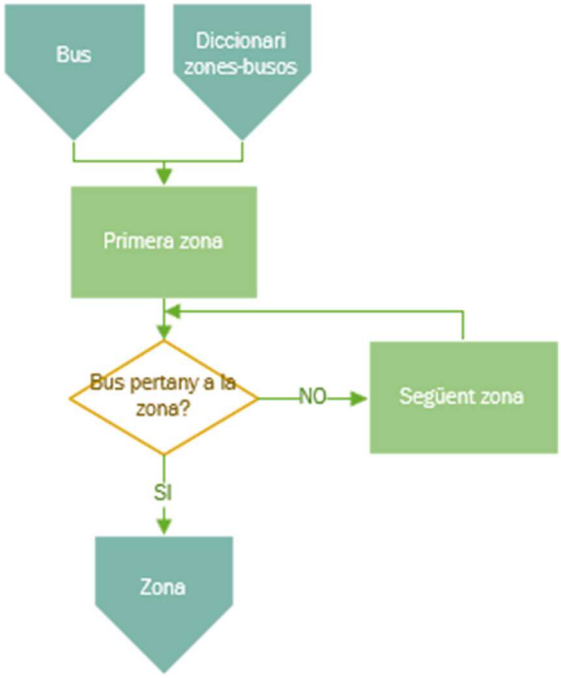

Figura 4: Diagrama de la funcio busZone()

El diagrama anterior mostra el procés que realitza la funció. S'entren dues variables, el diccionari busos-zones anteriorment creat i el bus a identificar, per exemple el bus de falta. La funció comença inspeccionant la primera zona en cas de que el bus es trobi a la primera zona retorna la zona, en cas contrari inspecciona la següent zona i la funció itera fins a trobar la zona buscada. A continuació es deixa el codi de la funció amb els comentaris pertinents.

```
def busZone(zone_dict, bus_ev): #Identifing the zone that has a
specific bus
    for key, value in zone_dict.items():
        if bus_ev in value:
            return key
```

4.3. Diccionari de interruptors

Amb motiu de poder aïllar una determinada zona cal saber quins interruptors es troben connectats a una zona determinada per a posteriorment poder obrir-los. Amb la funció creada a continuació es crea un diccionari de interruptors on s'identifica cada interruptor amb els dos busos amb els que esta connectats i a partir dels busos s'identifiquen les zones amb la funció anteriorment creada. Cal destacar-ne l'ús del diccionari ja que no permet elements duplicats, per tant no hi ha possibilitat de que es dupliquin els interruptors.

```
def idSwZone(net, zone_dict): #Making a dictionary of switches and
its zones
    swZ_dict = {}
    #Switch between 2 buses
    for swId, swElements in net.switch.iterrows():
        swZ_dict[swId] =
[busZone(zone_dict,swElements.bus),busZone(zone_dict,swElements.elem
ent)]
    return swZ_dict
```

El codi anterior reflecteix el procediment i el diagrama posterior dona una idea sobre el procés realitzat. En primer lloc s'entren dues variables, la xarxa i el diccionari de zones-busos, s'inspecciona el primer interruptor i s'identifiquen els busos entre els quals està connectat, seguidamentdementa funció busZone() s'identifiquen les zones a les quals pertanyen els busos i s'afegeix el paràmetre al diccionari dl'interruptor amb les dues zones que interconnecta. La funció itera i executa el procediment anterior amb els diferents busos que es troben a la xarxa. La funció .iterrows() es l'encarregada de realitzar el procés iteratiu sense necessitat de crear una estructura lògica.

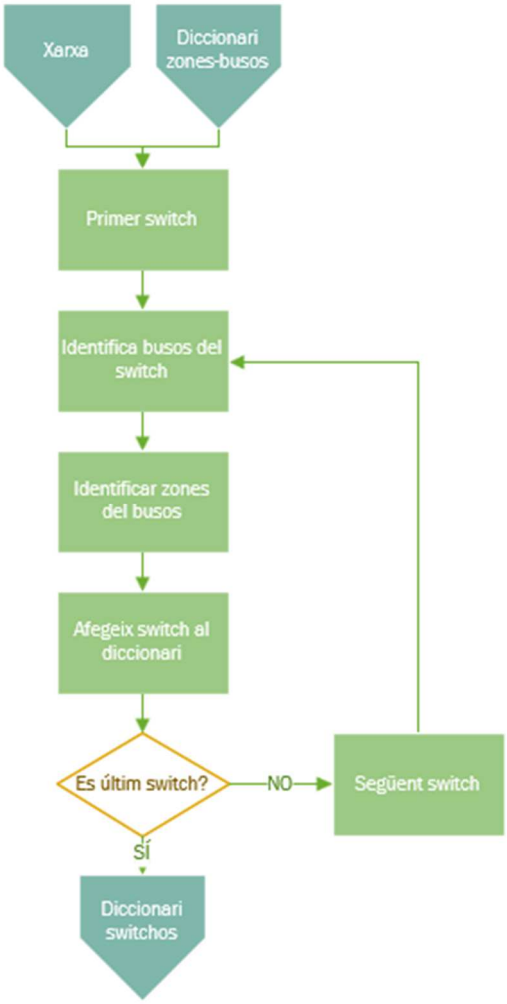

Figura 5: Diagrama funcio idSwZone()

4.4. Interruptors a obrir

Per a poder aïllar una falta, cal saber quina zona aïllar i per tant quins interruptors es necessari obrir obligatòriament, en altres paraules els que estan connectats a la zona. Per aquest motiu s'ha desenvolupat la funció openSwitches(). El codi es pot visualitzar a continuació.

```
def openSwitches(sz_dict, afected_zone): #Function that gives you
all switches that has to be open because are in contact with a zone
that has a failure
    sw_open = []
    for key , values in sz_dict.items():
        if afected_zone in values:
            sw_open.append(key)
    return sw_open
```

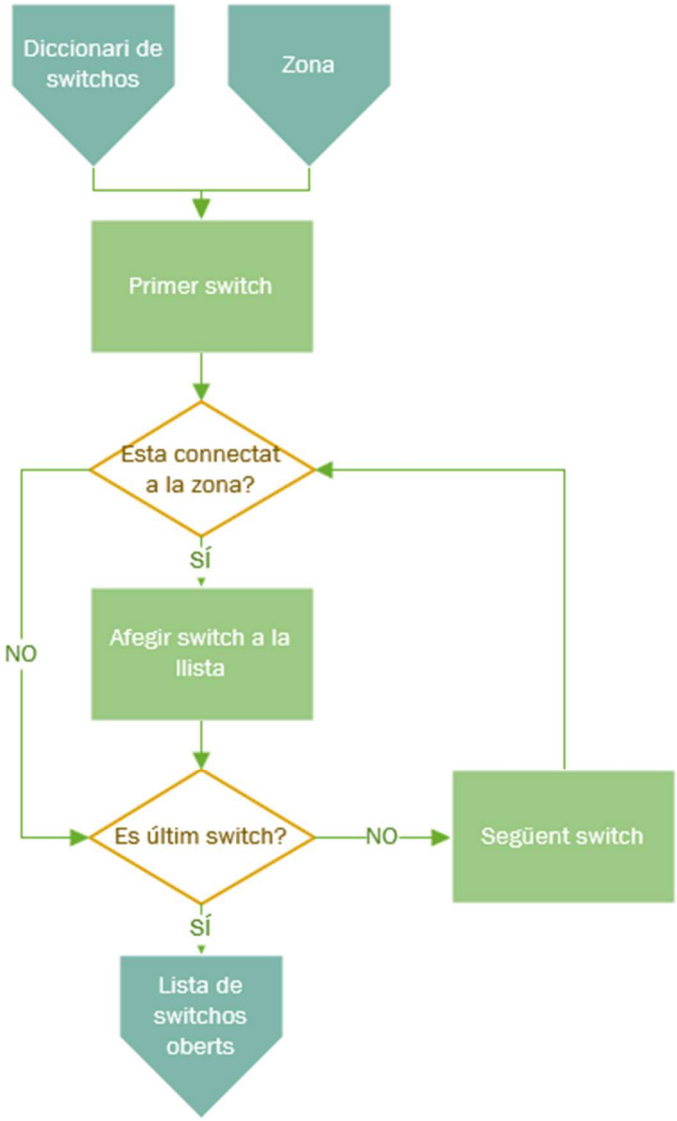

Figura 6: Diagrama de la funció openSwitches()

Amb l'ajuda del diagrama anterior es pot seguir el procediment seguit per a implementar la funció. Com a paràmetres d'entrada hi ha el diccionari de interruptors i la zona. La funció comprova els interruptors un per un per a verificar si la zona que s'ha passat com a paràmetre d'entrada es troba en connexió amb l'interruptor, en cas que això es compleixi s'afegeix a la llista de interruptors que la funció proporciona com a sortida. La funció dona la llista de interruptors que hauran de romandre oberts en cas de falta. Ja que si hi ha una falta en una zona concreta tots els interruptors que proporcioni la funció, és a dir tots aquells que interconnectin la zona avaluada hauran d'esdevenir oberts per a poder aïllar la falta.

4.5. Interruptors configurables

Abans de la implementació de l'algorisme de cerca de solucions cal saber amb quins interruptors es poden realitzar maniobres, en altres paraules quins interruptors es podran canviar d'estat per a la cerca de possibles solucions. Amb aquesta premissa, s'ha implementat la funció conSwitches(), que proporciona una llista dels interruptors als quals s'hi podran configurar diferents estats.

```
def conSwitch(sz_dict, open_switches): #Switches that are able to be
considered
    conf_switches = []
    for key, values in sz_dict.items():
        if key not in open_switches:
            #Append into list
            conf_switches.append(key)
    return conf_switches
```

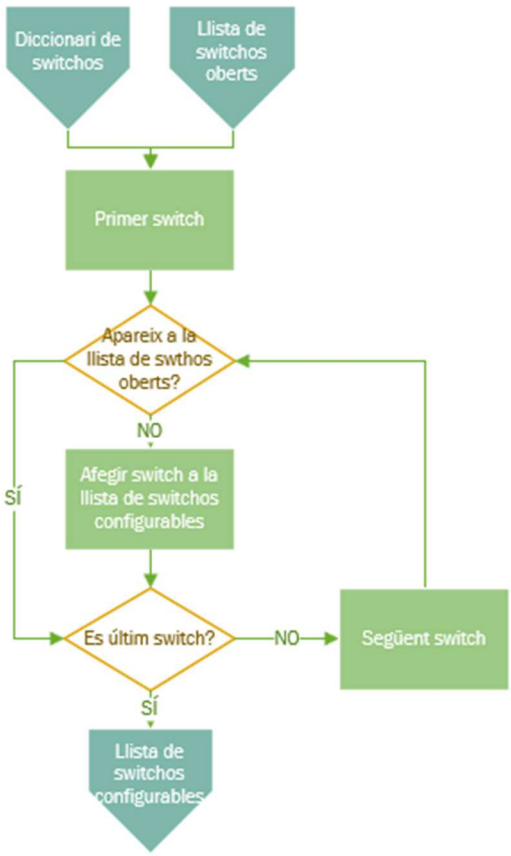

Figura 7: Diagrama de la funció conSwitches()

Si es segueix el diagrama anterior, les variables d'entrada són el diccionari de interruptors-zones i la llista de interruptors oberts. La funció itera sobre els diferents interruptors del diccionari i comprova que no apareguin a la llista del interruptors oberts, en cas que hi apareguin els descarta i en cas contrari afegeix l'interruptor corresponent a la llista de interruptors configurables.

4.6. Obir els interruptors per aïllar falta

Al haver-se detectat els interruptors que cal obrir degut a la falta. Cal implementar una funció que obri els interruptors per tal de poder aïllar la falta, s'utilitza la funció openForFault(), que conté com a entrades la llista de interruptors a obrir i la xarxa i mitjançant una iteració de la llista va obrint els interruptors corresponents.

```
def openForFault(net, swOpen):
    for i in swOpen:
        net.switch.closed.loc[i]=False
    retorn
```

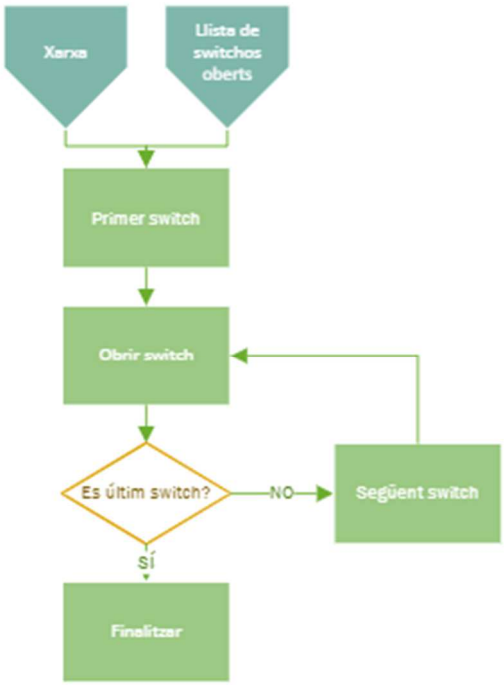

Figura 8: Diagrama funció openForFault()

5. RECONFIGURACIÓ

Per a començar la cerca de solucions, cal partir d'una configuració possible. Amb la ajuda del conjunt de funcions descrites en aquest apartat, s'arriba a una configuració valida que pot ser o no optima. A partir de la configuració vàlida, es podrà córrer l'algorisme de cerca de solucions. En la reconfiguració es parteix de la falta aïllada, es a dir es poden commutar els interruptors que no aïllen la falta, és a dir els interruptors que no interconnecten la zona de la falta amb una altra zona.

5.1. Tancar el interruptors

Per a una reconfiguració de la xarxa una vegada la falta ha sigut aïllada, el primer que es fa és tancar tots els interruptors. Seguidament s'aniran obrint interruptors comprovant que cap part de la xarxa quedi sense alimentació.

Per a tancar els interruptors es realitza amb la funció closeAllSwitches(). Les entrades de la funció son la xarxa a la qual s'han de tancar els interruptors i els interruptors a tancar, en aquest cas seran tots els interruptors que no aïllin la falta. En la funció s'itera sobre la llista de interruptors, es cerquen a la xarxa els interruptors a tancar i es tanquen. El codi i el diagrama de flux reflecteixen el procés iteratiu. Com es pot observar la funció no retorna cap valor.

```
def closeAllSwitches(net, switchToClose): #Function that closes all
the switches
    for i, switch in net.switch.iterrows():
        if i in switchToClose:

            net.switch.closed.loc[i] = True
    retorn
```

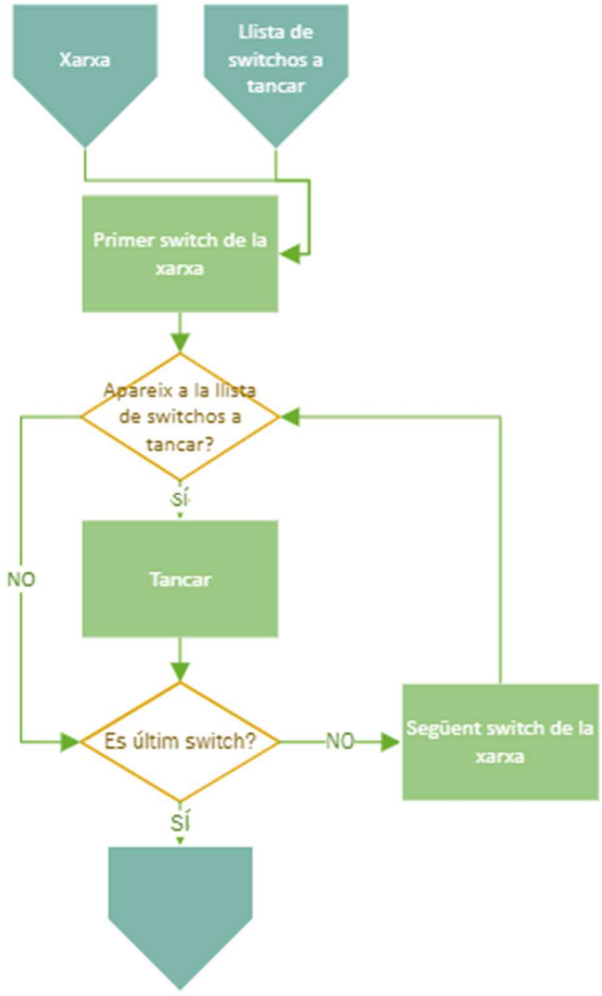

Figura 9: Diagrama de la funció closeAllSwitches()

5.2. Fer una llista de zones inicial connectades a feeders

Anomenarem feeder a un conjunt d'external grid i transformador, aquest permet el subministrament de potència a la xarxa. Una external grid, és un paràmetre que la llibreria pandapower utilitza per subministrar potència en la simulació, aquesta té un valor nominal de tensió i unes potències de curtcircuit definides segons sigui necessari. Es realitza una llista amb valors booleans, aquesta llista indica si la zona corresponent està connectada a un feeder. La llista s'utilitzarà posteriorment per a comprovar que les zones interconnectades contenen un feeder. En el model de pandapower s'interpreta un feeder com una external grid. La figura següent ajuda a entendre el concepte, en el cas de la figura estarien alimentades per un feeder les zones 0 i 3, entre d'altres. Les zones mencionades corresponen a les zones que s'obtenien en l'apartat de la preparació de les dades amb tots els interruptors oberts.

S'identifica cada booleà amb la zona corresponent a través de l'ordre, és a dir la posició 0 de la llista correspon a la zona 0 i així successivament.

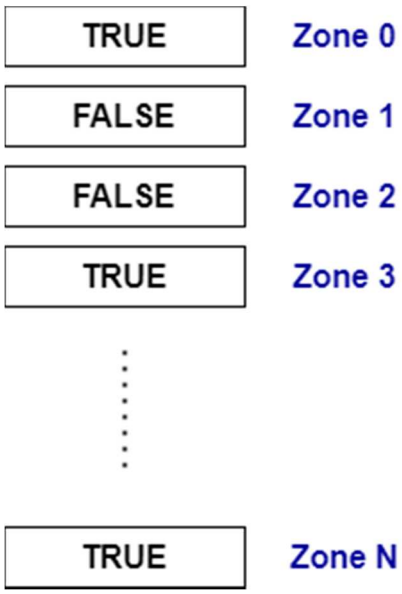

Figura 10: Llista de zones connectades a un feeder

La funció que governa la identificació de la zona amb si està alimentada per un feeder s'anomena initialZonesToExtGrid() ja que associa les zones inicials amb un booleà verdader o fals segons estiguin o no connectades a una xarxa externa del model pandapower. Les entrades de la funció son el diccionari de busos-zones i la xarxa, la funció itera sobre el diccionari de busos-zones i comprova si algun bus de la zona hi ha connectada una xarxa externa (feeder), si es així s'assigna el valor TRUE a la zona corresponent, en cas contrari s'assigna el valor de FALSE. El codi i diagrama reflecteixen el procediment lògic i d'execució de la funció.

```
def initialZonesToExtGrid(net, ZoneBusDic):
    ZoneExtGridList = []
    for i, buses in ZoneBusDic.items():
        ExtGridPresence = False
        for index, val in net.ext_grid.iterrows():
            if val.bus in buses:
                ExtGridPresence = True
        ZoneExtGridList.append(ExtGridPresence)
    return ZoneExtGridList
```

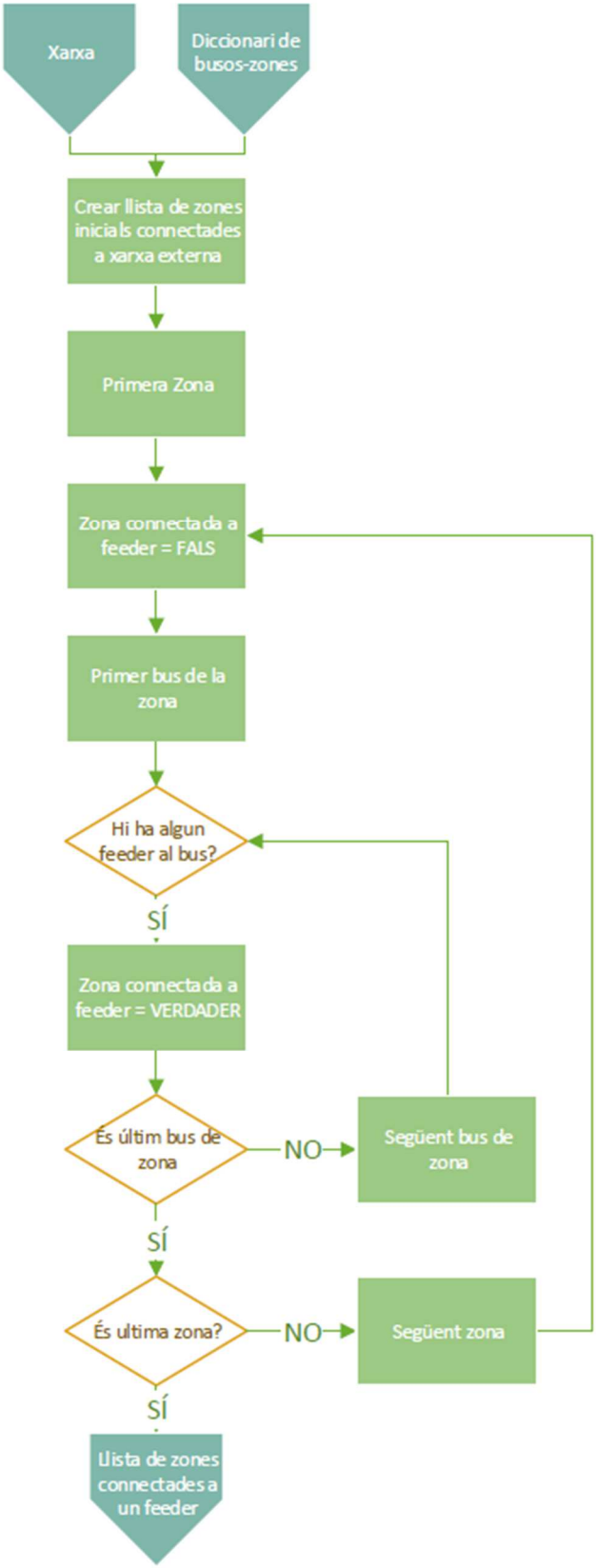

Figura 11: Diagrama de la funció initialZonesToExtGrid()

5.3. Fer una llista de zones interconnectades

A continuació es necessita implementar una llista de zones interconnectades. Partint de la premissa que s'han tancat tots els interruptors possibles (els que no aïllen la falta). Cal comprovar quines de les zones inicials queden interconnectades a través dels interruptors tancats, la imatge posterior descriu gràficament el propòsit de la funció que agrupa en una llista de conjunts les diferents zones que queden interconnectades. S'utilitza una llista de conjunts ja que els conjunts es una estructura de dades que no permet la repetició, d'aquesta manera no hi poden haver zones repetides en conjunts d'interconnexió. Les N zones queden repartides en M conjunts. Les zones que pertanyen a un mateix conjunt estan interconnectades mitjançant els interruptors tancats. Cal remarcar que poden existir conjunts amb un sol element, i que el conjunt que contingui la zona de la falta serà d'un sol element ja que la falta s'aïlla. A la figura 12 es mostra un exemple del que proporciona la funció.

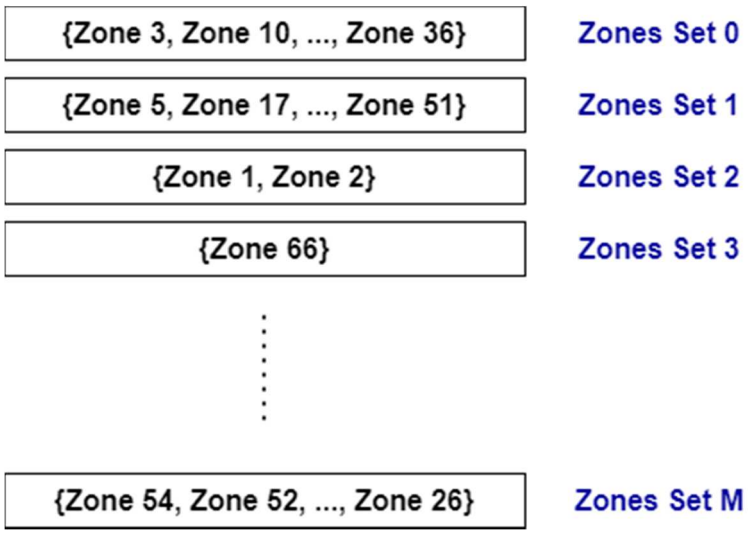

Figura 12: Lista de conjunts de zones

A la funció que implementa la llista de conjunts de zones se li passen dos paràmetres, la xarxa i el diccionari de zones-busos. En primer lloc, es crea un graf que representa la topologia de la xarxa i s'inicialitza la llista de zones. Seguidament es realitza una cerca topològica i s'itera sobre els diferents conjunts de zones no connectats (àrees). S'inicialitza un conjunt, per cada bus de l'àrea és busca la zona a la qual pertany i s'afegeix al conjunt de zones, com que els conjunts de zones no permeten repetició les zones seran afegides només una vegada en cas que dos busos pertanyin a la mateixa zona.

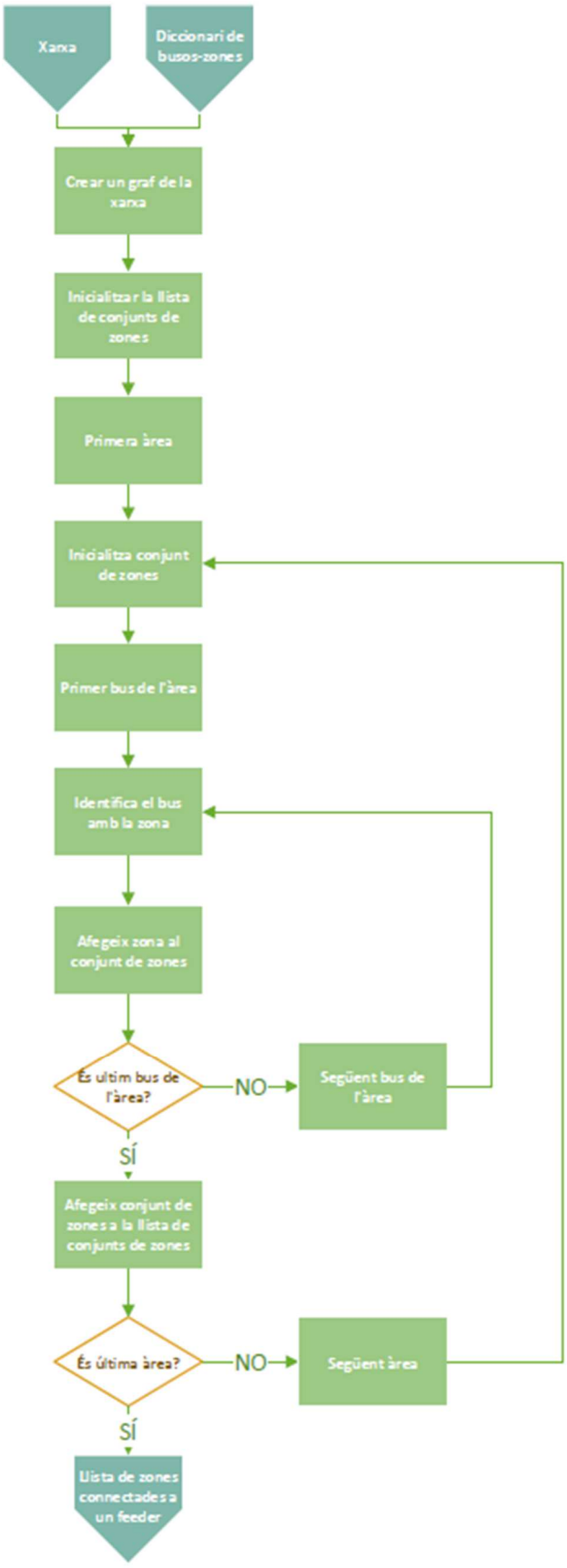

Figura 13: Diagrama de flux de la funció makeZoneList()

```
def makeZoneList(net, ZoneBusDict):
    netGraph = pp.topology.create_nxgraph(net)
    ZonesList = []
    for area in pp.topology.connected_components(netGraph):
        zones = set()
        for bus in area:
            zones.add(busZone(ZoneBusDict, bus))
        ZonesList.append(zones)
    return ZonesList
```

5.4. Comprovació de àrees connectades a un feeder

Per a cada conjunt de zones que anomenarem àrea, es necessita saber si hi ha un feeder que alimenta aquella àrea, ja que les àrees estan aïllades entre elles. Per a comprovar si la l'àrea té connectat un feeder nomes cal comprovar que alguna de les zones de l'àrea conté un feeder connectat. La identificació es fa a través de la llista de zones inicials connectades a feeders.

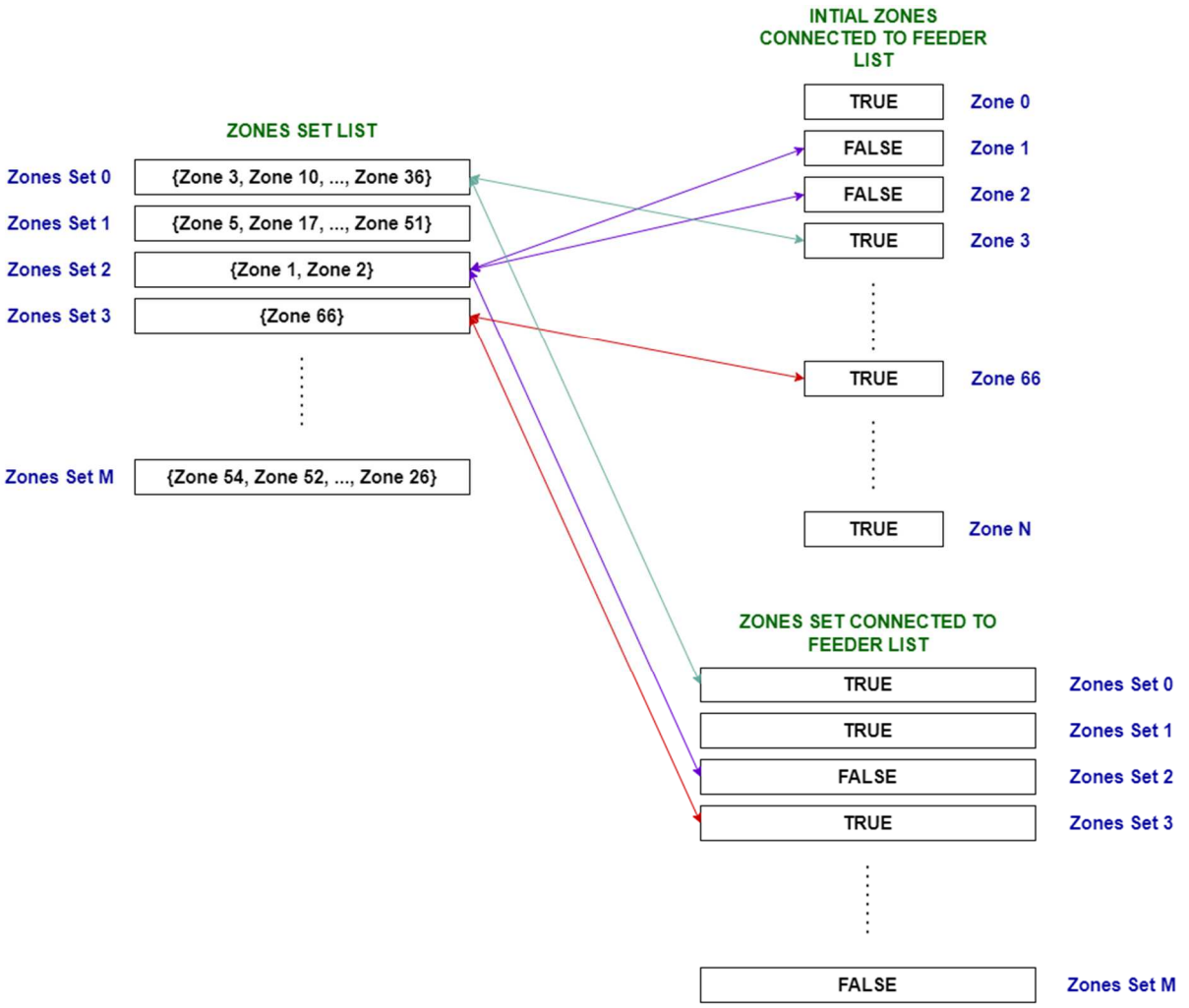

Figura 14: Procés de comprovació de conjunt de zones connectades a un feeder

```
def extGridComprovation(ZoneList,EGZoneList):
    extGridList = []
    for zoneset in ZoneList:
        zoneset_to_extgrid = False
        for setelement in zoneset:
            if EGZoneList[setelement]:
                zoneset_to_extgrid = True
        extGridList.append(zoneset_to_extgrid)
    return extGridList
```

La funció extGridComprovation() s'encarrega de comprovar quins conjunts de zones es troben connectats a feeders. Per a fer-ho utilitza dues entrades, la llista de zones inicials connectades a feeders i la llista de conjunts de zones, altrament anomenats àrees. La funció realitza un procés d'iteració sobre les diferents àrees per a cercar si alguna zona de l'àrea inspeccionada està connectada a un feeder, si es el cas, l'àrea estarà connectada a un feeder i pren la posició de la llista associada aquell conjunt de zones pren el valor True, en cas contrari pren un valor False.

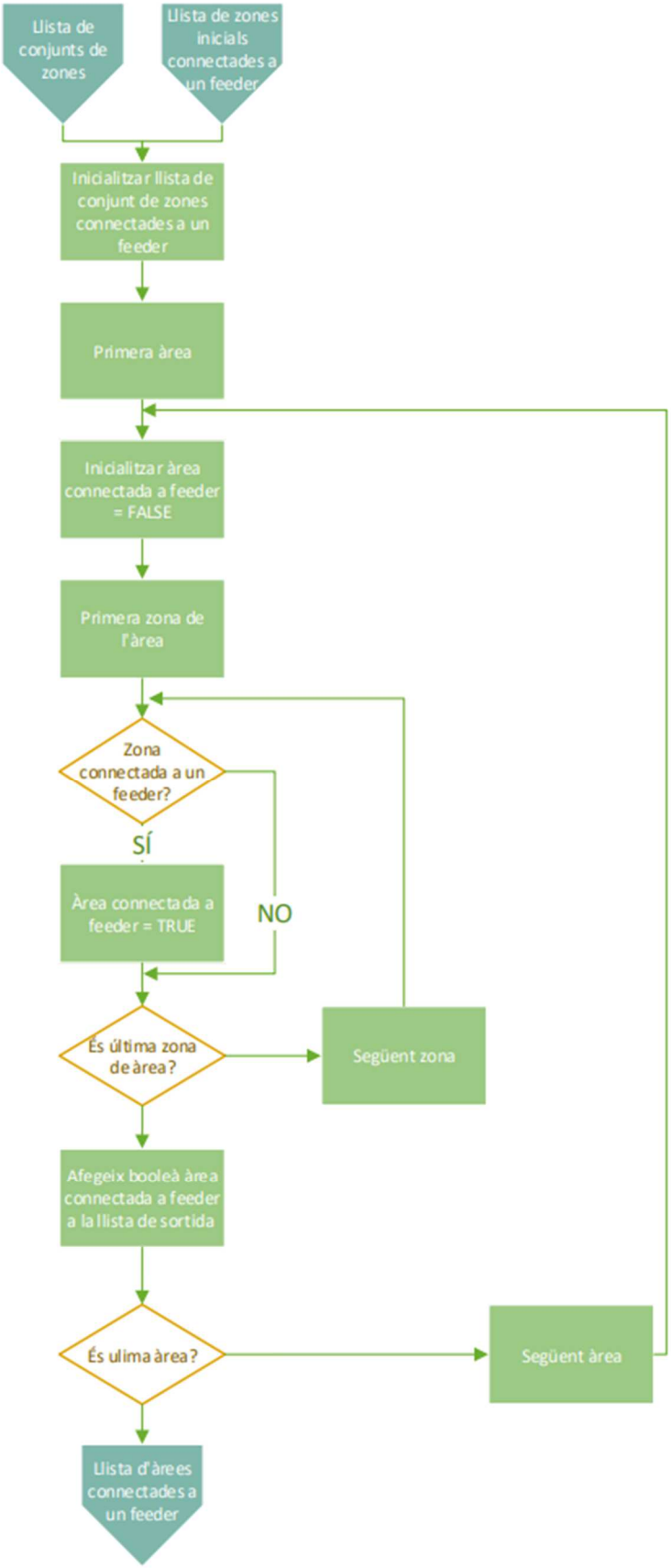

Figura 15: Diagrama de la funció extGridComprovation()

5.5. Comptador de àrees no connectades a feeder

Després d'aïllar la falta i tancar tots els interruptors que no aïllen la falta, cal comptar el nombre de falsos que hi ha a la llista de conjunts de zones connectades a feeders.

Per a argumentar l'obriment de interruptors, s'utilitza un exemple demostratiu en el que s'analitzen diferents casuístiques. En la figura 16 hi ha un exemple de xarxa. L'objectiu de la reconfiguració és obrir el màxim de interruptors possibles sense que cap zona quedi sense alimentació. Si per exemple, s'obrís l'interruptor 1 de la imatge, cap zona quedaria sense alimentació, en canvi si s'obre l'interruptor 4 les zones 4 i 5 queden sense subministrament. En ambdós casos es mostra la llista de conjunts de zones interconnectades i la llista de conjunts de zones connectades a un feeder. Com s'aprecia, el cas 1 (obrir SW1) conté el mateix nombre de falsos que la configuració inicial que és zero, en canvi el cas 2 (obrir SW4) és té un fals i no coincideix amb el nombre de la configuració inicial, per tant, és descarta l'acció d'obrir SW4. En altres paraules, que el nombre de falsos sigui diferent significa que un conjunt nou de zones ha quedat sense alimentació per tant cal descartar l'acció que ha provocat aquest fet.

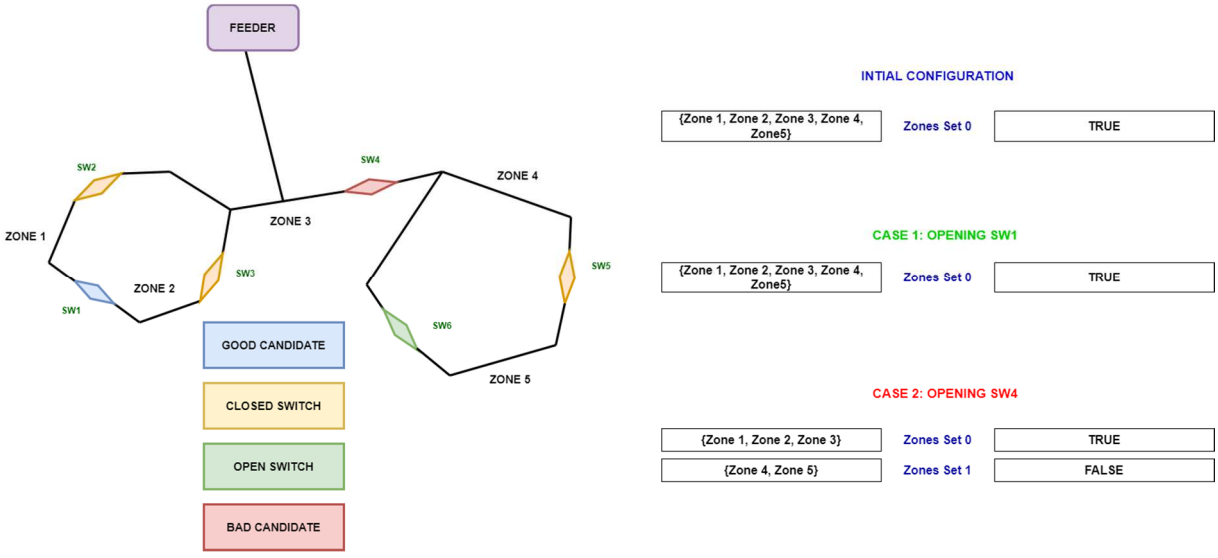

Figura 16: Anàlisi de casos, obrir un interruptor

La funció countFalsesinList() s'encarrega de comptar el número de falsos en una llista.

Per a comptar el nombre de falsos s'itera sobre la llista i amb l'ajuda d'un comptador s'acumula el nombre d'elements amb valors fals de la llista. El codi i el diagrama ajuden a entendre el procés de comptatge.

```
def countFalsesinList(analysedlist):
    i = 0
    for element in analysedlist:
        if element is False:
            i = i+1
    return i
```

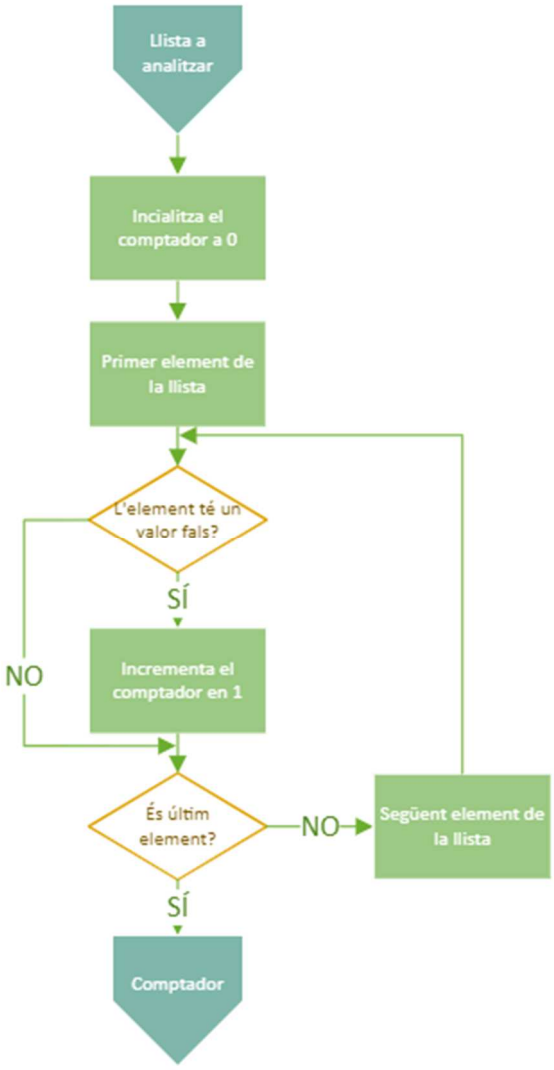

Figura 17: Diagrama funció CountFalsesinList()

5.6. Obriment d'interruptors per a la reconfiguració

A través del nombre de falsos que hi ha en la llista de conjunt de zones connectades a un feeder, es poden anar obrint diferents interruptors fins a aconseguir una reconfiguració de la xarxa amb el major nombre de interruptors obert evitant així l'aparició de llaços redundants.

La figura 18 s'inclou per a resumir els apartats anteriors i explicar quines condicions cl complir per a poder obrir un interruptor. Com es veu en la imatge, hi ha 3 tipus de zones, les zones connectades a un feeder, la zona aïllada que seria la zona on apareix una falta i zones no connectades a un feeder. Que les zones no estiguin connectades a un feeder es degut a que els interruptors de color taronja romandran oberts ja que son aquells que aïllen la falta i no hi podrà existir una connexió física entre el feeder i les zones.

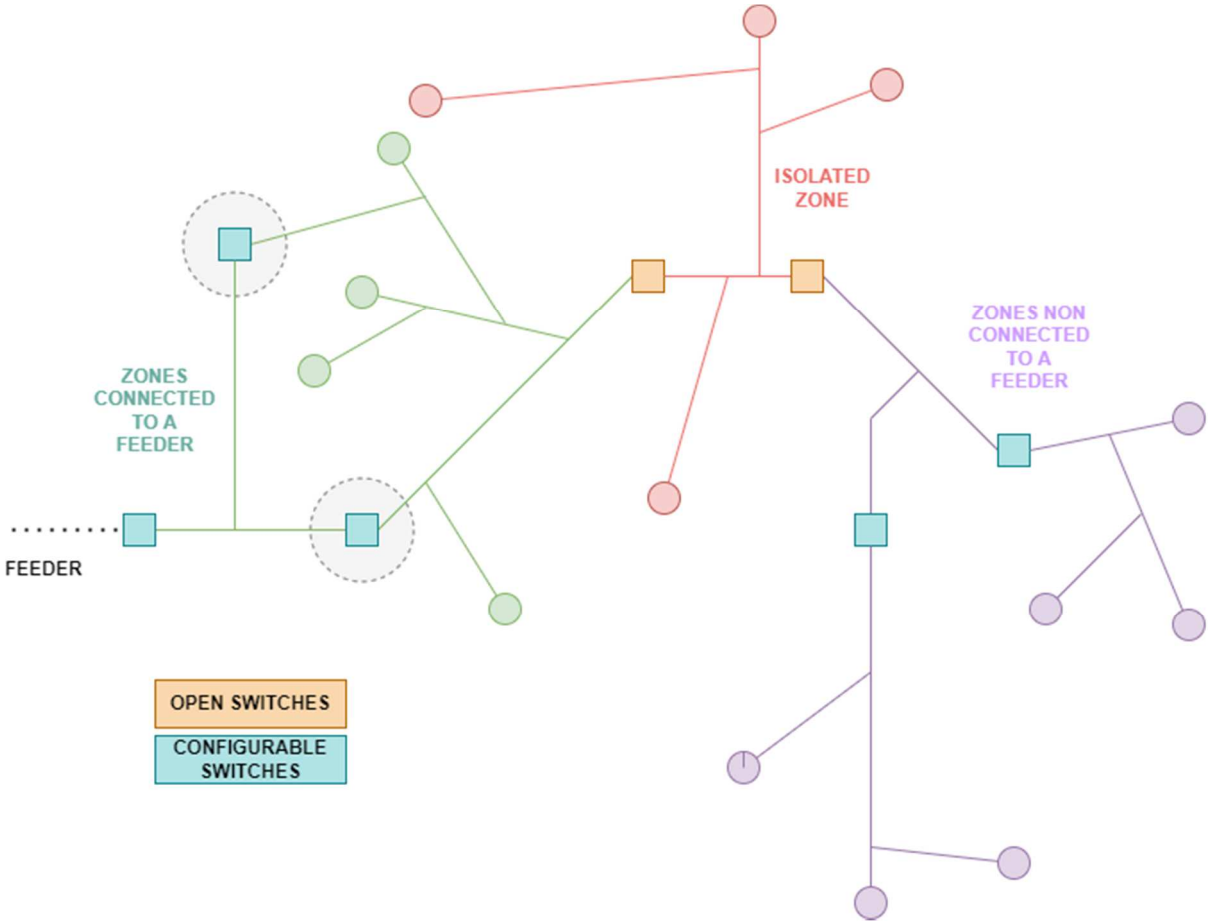

Figura 18: Justificació de la obertura d'interruptors

Es parteix de tots els interruptors de color cian tancats, per cada un d'aquests el procés per a obrir-lo consistirà en obrir l'interruptor, comprovar que cap conjunt de zones de les quals

estaven alimentades, perdi alimentació, és a dir, que el nombre de falsos en la llista de conjunts de zones sigui igual que inicialment, i si es dona el cas, mantenir l'obertura de l'interruptor, en cas contrari tancar-lo.

En la imatge de la figura 18 inicialment hi hauria 3 conjunts de zones, zones de color verd, zones de color vermell i zones de color lila, ja que els interruptors cian es troben tancats i els interruptors taronges es troben oberts. D'altra banda, el nombre de falsos seria 2, corresponent a les àrees vermella i lila. En aquest cas només es podria obrir un dels dos interruptors que estan encerclats per una àrea rodona grisa o els dos a la vegada. Si només s'obrís un dels dos interruptors encerclats la llista de conjunts de zones no incrementaria en mida i el nombre de falsos no incrementaria. En el cas que s'obrissin els dos la llista de conjunts de zones incrementaria la mida fins a 4 conjunts de zones, però hi hauria 2 valors verdaders i un 2 de falsos a la llista d'àrees connectades a feeders, per tant, el nombre de falsos no incrementaria. En el cas de l'algorisme implementat, com que s'itera sobre cada interruptor s'obririen els 2.

El codi i diagrama que es deixen a continuació permeten seguir el procés seguit en la programació.

```
def openingOfswitches(numFalses_initial, net, feasibleSwitch,
ZoneBusDict, EGZoneList):
    for element in feasibleSwitch:
        net.switch.closed.loc[element] = True
        if not (numFalses_initial ==
countFalsesinList(extGridComprovation(makeZoneList(net,
ZoneBusDict),EGZoneList))):
            net.switch.closed.loc[element]=False
    retorn
```

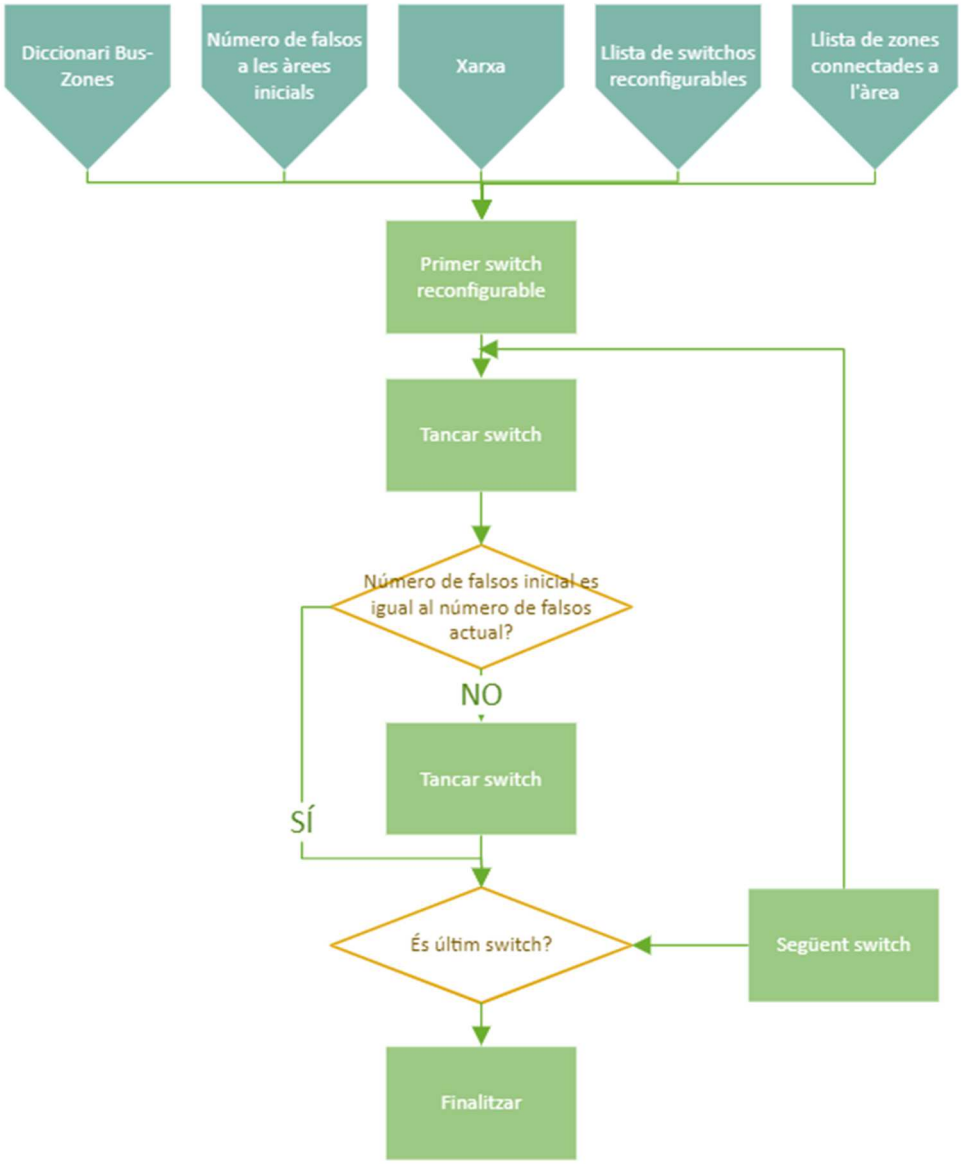

Figura 19: Diagrama funció openingOfswitches()

5.7. Fer una llista de interruptors

Una vegada realitzada la reconfiguració s'implementa una funció que crea una llista dels diferents estats dels interruptors. Aquesta funció permet visualitzar en quins estats han quedat els interruptors i quina configuració ha adoptat la xarxa.

```
def makeListOfSwitches(net):
    swList = []
    for element in net.switch.closed:
        swList.append(element)
    return swList
```

La funció itera sobre els diferents interruptors i afegeix a la llista l'estat d'aquests, cada interruptor s'identifica segons l'ordre de la llista. L'ordre de la llista coincideix amb l'ordre existent en l'arxiu .xls de la xarxa.

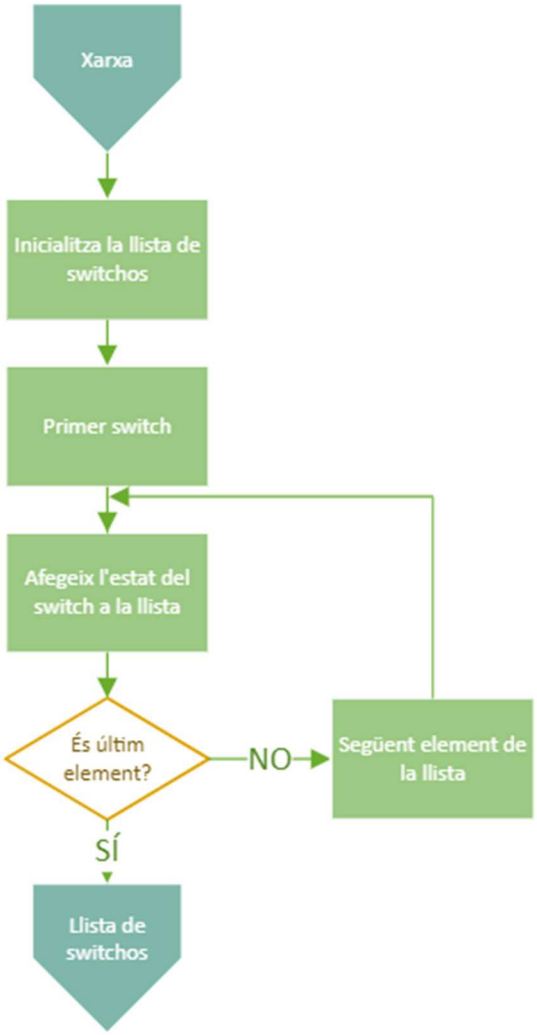

Figura 20: Diagrama de la funció makeListOfSwitches()

6. ALGORISME GENÈTIC

En aquest apartat s'explica el disseny de l'algorisme implementat per a la cerca de solucions. S'inclouen diverses imatges per a ajudar a entendre la funcionalitat i el motiu d'implementació.

6.1. Motiu d'implementació

El motiu d'utilització d'un algorisme genètic és degut a l'alt cost computacional que suposa l'anàlisi de totes les possibles combinacions de interruptors. Per a cada interruptor hi ha dos estats, obert i tancat, per tant, si es volgués trobar la solució òptima global, caldria analitzar les 2^n combinacions d'estats dels interruptors, on n es el nombre de interruptors. Per fortuna, existeix la possibilitat d'implementar un algorisme genètic que sense un cost computacional desmesurat permet la cerca de combinacions òptimes locals, i amb suficients iteracions és molt probable que la solució trobada sigui òptima global.

6.2. Concepte d'algorisme genètic

Els algorismes genètics s'utilitzen per a la cerca de solucions o optimització. S'anomenen genètics perquè s'inspiren en el procés genètic dels organismes vius. La teoria de l'evolució dicta que les especies evolucionen segons un principi de millors condicions genètiques, és a dir, es van generant canvis aleatoris en la genètica de les espècies i les espècies amb millors condicions per adaptar-se sobreviuen i procreen, propagant així la genètica. En la resolució de problemes d'optimització s'adopta una idea similar en la qual es parteix d'un conjunt de solucions que no tenen perquè ser òptimes i es combinen les que proporcionen millors resultats per a generar una nova solució. Si les noves solucions milloren els resultats, es donen com a vàlides i es recombinen per a seguir buscant solucions, en cas que proporcionin pitjors resultats, es descarten.

Per a explicar millor la optimització mitjançant un algoritme genètic, es proposa un exemple. Suposem que volem minimitzar la funció f(x) de la figura 21, en primer lloc es generen punts inicials aleatoris, en el cas de la imatge x_1, x_2 i x_3. Seguidament, es combinen els punts inicials amb millors característiques, en aquest cas com que son pocs punts es combinen tots i es genera la primera iteració de solucions $x_{2\text{-}1}$ i $x_{2\text{-}3}$, els subíndex fan referència als punts combinats. En els cas de la figura la combinació podria estar donada per una formula semblant a:

$$x_{A-B} = \alpha \cdot x_A + (1-\alpha) \cdot x_B \qquad \alpha \in (0,1) \qquad \text{(Eq. 1)}$$

Seguidament, com que $x_{2\text{-}1}$ té un valor de f(x) major que els seus progenitors x_2 i x_1 (pitjors condicions), es descartaria. En el cas de $x_{2\text{-}3}$ com que millora les condicions dels seus progenitors, $f(x_3)$ és menor, es seguiria combinant amb altres punts.

El criteri de recombinació, descart i combinació pot ser diferent segons el tipus de problema, però la idea que recull l'algorisme segueix un patró similar.

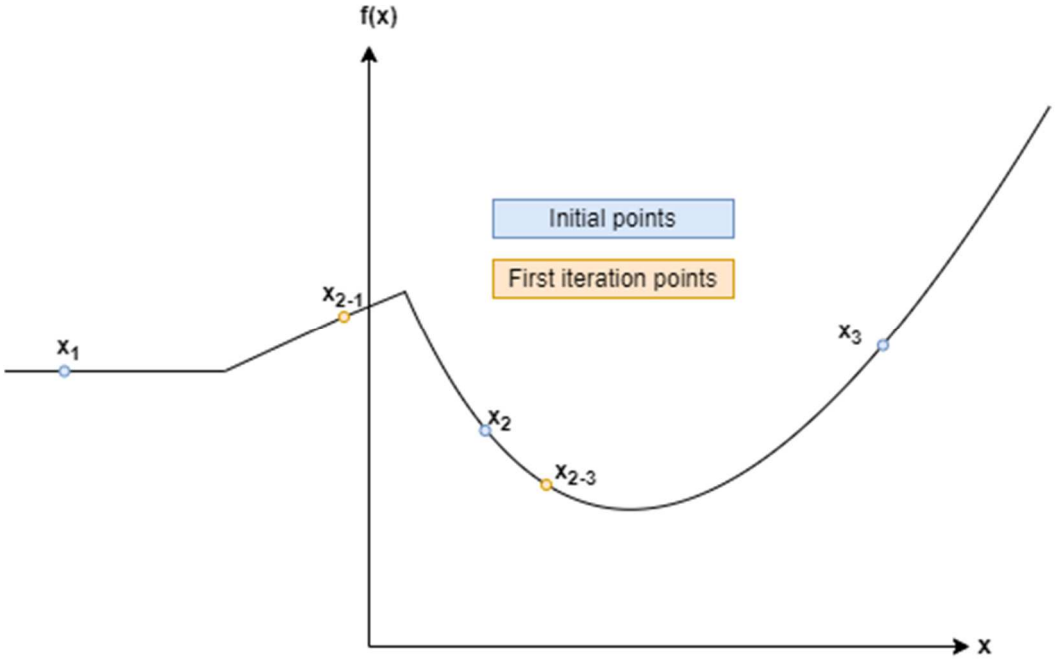

Figura 21: Exemple de minimització.

6.3. Funció de l'algorisme genètic

Per a implementar l'algorisme genètic, s'han utilitzat les idees exposades a (Mesut, Baran / Felix, Wu). Que plantegen una manera de buscar solucions a partir d'una configuració inicial de la xarxa. En primera lloc cal obtenir una configuració vàlida, amb els les funcions de preparació de les dades, es pot aïllar la falta i amb les funcions de reconfiguració s'obté una configuració inicial per a l'algorisme genètic.

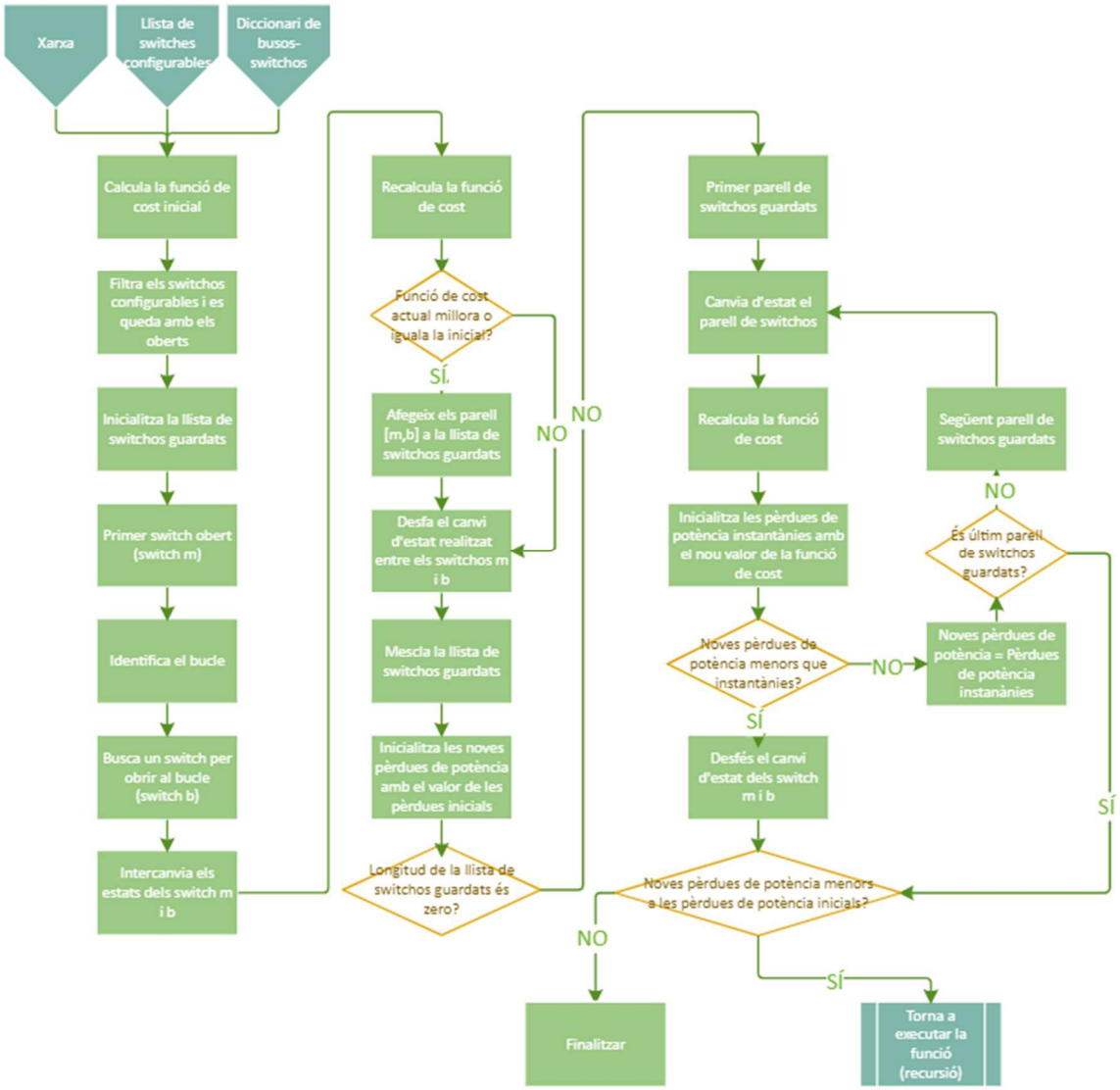

Figura 22: Diagrama de l'algorisme genètic

El següent pas, consisteix en la generació de noves combinacions dels interruptors. Es parteix de una llista d'interruptors configurables, és a dir aquells que no aïllen la falta, l'obtenció d'aquesta llista queda explicada en el punt 4.5. Per cada interruptor d'aquesta llista obert en la configuració inicial, que anomenarem interruptor m, cal tancar-lo, una vegada tancat, s'identificarà el bucle en el que es troba i dins d'aquest bucle es cercarà algun interruptor diferent al interruptor m, que anomenarem interruptor b. S'entén per bucle a un sistema de busos, transformadors, línies i interruptors connectats en anell. L'interruptor b intercanviarà l'estat amb l'interruptor m, en altres paraules, l'interruptor m es tancarà mentre que l'interruptor b s'obrirà. Cal remarcar que un interruptor apareix en el bucle només si es troba tancat, per

aquest motiu, es diu que s'intercanvia l'estat i també per aquesta raó cal tancar l'interruptor m per a identificar el bucle.

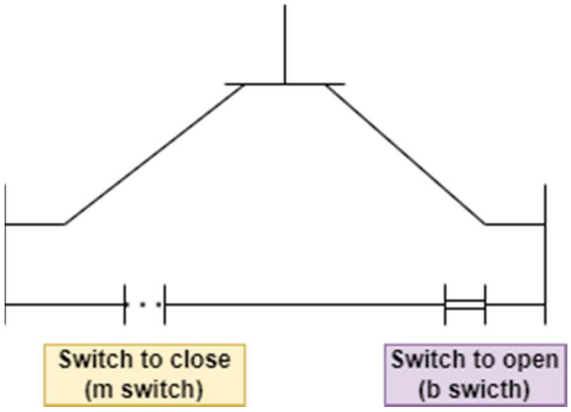

Figura 23: Interruptors m i b

Una vegada intercanviats els estats, es comprova novament la funció de cost i si aquesta disminueix respecte la configuració inicial, es guardarà el parell d'interruptors [m,b] en una llista en cas que la funció de cost, és a dir, les pèrdues de potència, no millorin, es descartarà el parell d'interruptors. Es desfarà el canvi d'estat independentment de si milloren o no la funció de cost. El procés descrit d'intercanvi d'estats es reprodueix per a tots els interruptors oberts de la llista de configurables.

El fet que els interruptors que s'intercanvien estiguin en el mateix bucle és d'important rellevància, Ja que aquest intercanvi manté el criteri de màxim nombre de clients amb subministrament perquè intercanviant l'estat d'interruptors al mateix bucle amb el procediment descrit no produeix l'aïllament de cap zona.

Una vegada avaluats tots els interruptors oberts, s'obté una llista de parells d'interruptors. Aquesta llista es mesclarà aleatòriament i es recorrerà la llista executant els canvi d'estat entre els m i b fins que deixin de millorar la funció de cost. La imatge de la figura 24 descriu el procés, de mescla i canvi d'estat. En l'exemple de la imatge, es mantindran els canvis de [m_{27}, b_{27}] i [m_4, b_4] mentre que es descartaran tots els altre canvis. Podria passar que amb el canvi algun [m_a, b_a] posterior a [m_{33}, b_{33}] millorés la funció de cost però com que a partir de [m_{33}, b_{33}] la funció de cost no millora l'anterior es descarten els canvis de [m_{33}, b_{33}] i posteriors.

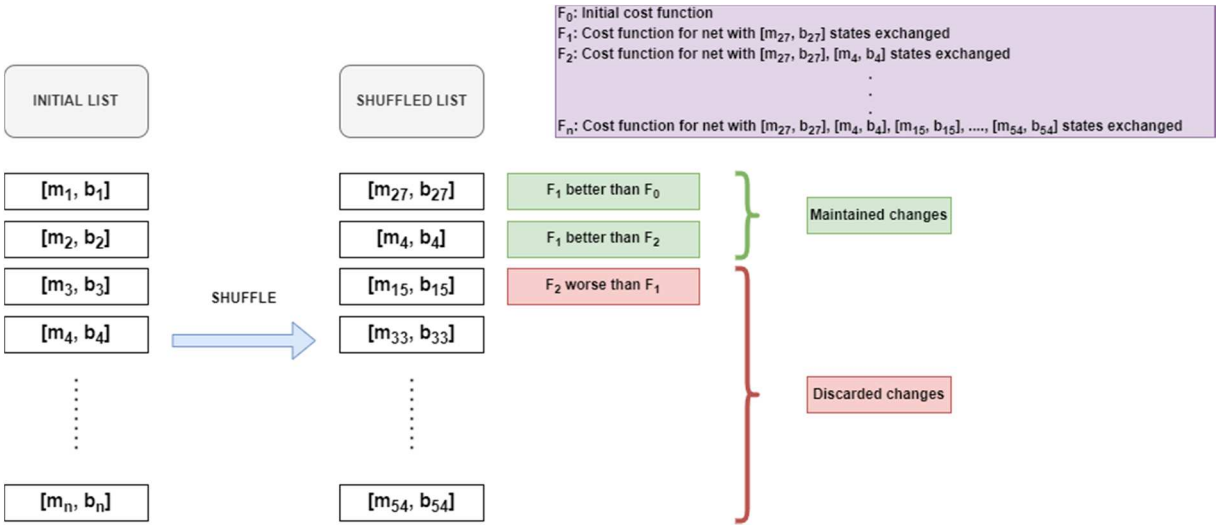

Figura 24: Procés de mescla i execució de canvis d'estat

6.4. Funció de cost

L'algorisme genètic busca noves solucions que millorin la funció de cost, en aquest cas la el paràmetre a millorar és la pèrdua de potència. L'algorisme buscarà solucions que disminueixin les pèrdues de potència.

```
def powerLosses(net): #Functions for power losses
    pp.runpp(net)
    powerLoss = 0
    for key,loss in net.res_line.pl_mw.items():
        powerLoss = powerLoss + loss
    for key, loss in net.res_trafo.pl_mw.items():
        powerLoss = powerLoss + loss
    return powerLoss
```

La funció calcula els fluxos de potència, seguidament afegeix totes les pèrdues de les línies i els transformadors. En el codi i el diagrama es pot comprovar el procés de funcionament de la funció de pèrdues de potència.

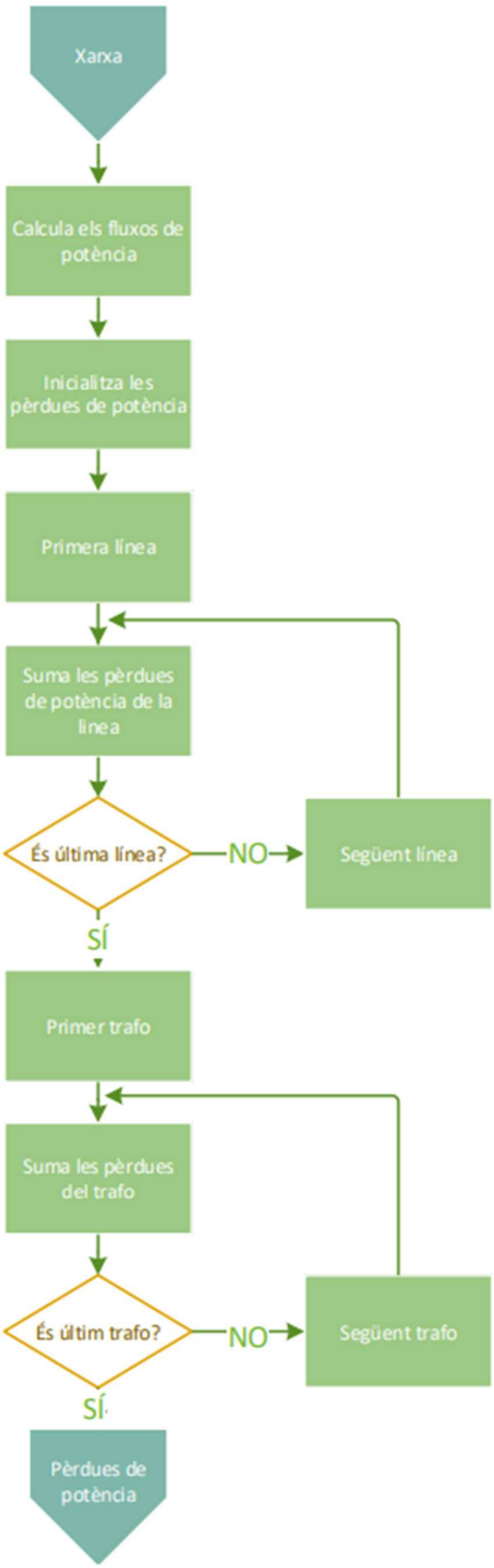

Figura 25: Diagrama de la funció powerLosses()

6.5. Identificar el bucle

Com s'ha mencionat anteriorment per a fer un canvi entre un interruptor a tancar i un interruptor a obrir, cal que ambdós es trobin en el mateix bucle. Per aquest motiu es construeix una funció per a identificar bucles, aquesta funció fa ús d'una llibreria anomenada networkX que permet analitzar grafs. La xarxa es tractada com un graf on els busos son nodes i els interruptors tancats i les línies són arestes.

```
def identifyLoop(net, close_candidate_sw_bus): #Has to be modified
for encountering swID
    net.switch.closed.loc[close_candidate_sw_bus] = True
    netGraph = pp.topology.create_nxgraph(net)
    cycle = list(nx.find_cycle(netGraph,
source=close_candidate_sw_bus, orientation='ignore'))
    net.switch.closed.loc[close_candidate_sw_bus] = False
    return cycle
```

Figura 26: Diagrama de la funció identifyLoop()

Es passen el paràmetres de la xarxa i el bus de l'interruptor a tancar, per a poder identificar el bucle cal tancar l'interruptor per a que quan la xarxa esdevingui una graf, l'interruptor actuï com a aresta. Una vegada tancat l'interruptor i creat el graf es busca el cicle amb la funció find_cycle de la llibreria networkX que troba si el bus que es passa com a paràmetre pertany a un cicle.

En un graf, es defineix un cicle si existeix un camí d'arestes des de un vèrtex fins al mateix vèrtex sense passar dues vegades per una mateixa aresta.

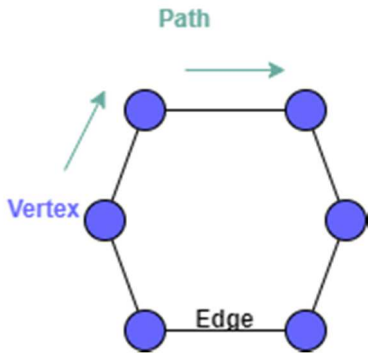

Figura 27: Cicle en un graf

El cicle trobat per la funció find_cycle s'adjunta en una llista, aquesta conté informació de les diferents arestes del cicle, cada element de la llista té la forma:

[bus1, bus2, ('line o 'switch' o 'trafo' ,identificador de l'element), 'forward' o 'backward']

Finalment es torna a obrir l'interruptor a tancar ja que abans de tancar-lo definitivament cal comprovar si millora la funció de cost.

6.6. Trobar interruptor al bucle

Tal i com s'ha mencionat anteriorment, s'obté el bucle mitjançant una llista on els paràmetres són de la forma:

[bus1, bus2, ('line o 'switch' o 'trafo' ,identificador de l'element), 'forward' o 'backward']

La funció findSwitchInLoop() rastreja els elements de la llista i busca un interruptor que sigui diferent del que s'ha de tancar (interruptor m) al bucle. En el cas de la figura 28, si s'hagués de tancar l'interruptor 2, la funció buscaria un interruptor diferent de 2 i retornaria el primer que trobés, en aquest cas amb l'interruptor 1. En taronja es marca l'interruptor que es descarta i en blau es mostra l'interruptor que retorna. L'interruptor que retorna la funció (interruptor b), serà el candidat per a tancar-se.

```
[(79, 93, ('line', 65), 'forward'), (93, 85, ('line', 72), 'forward'), (85, 120, ('line', 76),
'forward'), (120, 90, ('switch', 2), 'forward'), (90, 87, ('trafo', 14), 'forward'), (87, 117,
('line', 99), 'forward'), (117, 116, ('line', 98), 'forward'), (116, 115, ('line', 97), 'forward'),
 (115, 107, ('trafo', 15), 'forward'), (107, 118, ('switch', 1), 'forward'), (118, 98, ('line',
82), 'forward'), (98, 99, ('line', 80), 'forward'), (99, 105, ('line', 86), 'forward'), (105, 96,
('line', 78), 'forward'), (96, 79, ('switch', 0), 'forward')]
```

Figura 28: Exemple de funcionament de la funció findSwitchInLoop()

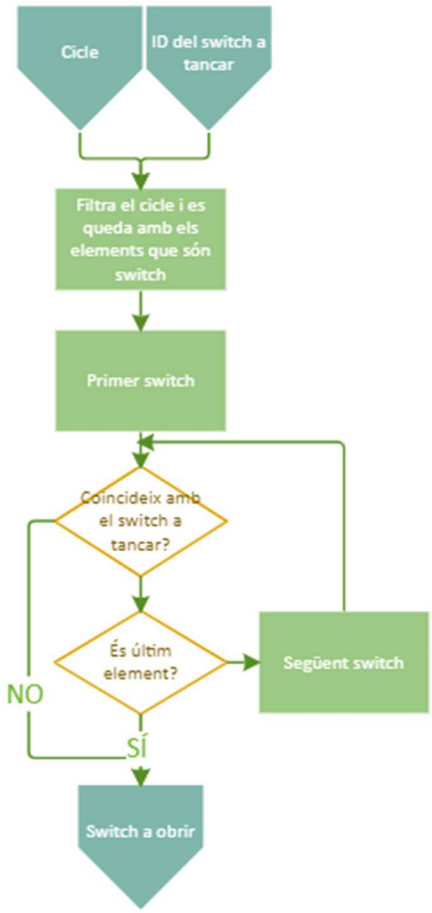

Figura 29: Diagrama de la funció findSwitchInLoop()

```
def findSwitchInLoop(cycle, close_candidate_sw_id):
    switchesOnLoop = list(filter(filterCycleFunction,cycle))
    print("Filtered switches:")
    for element in switchesOnLoop:
        if not(element[2][1] == close_candidate_sw_id):
           return element[2][1]
```

6.7. Filtre del bucle

Per a filtrar els elements del bucle és necessària la incorporació d'una funció que retorni el valor True quan es tracti d'un interruptor i False en qualsevol altre cas, el següent codi fa la funció desitjada.

```
def filterCycleFunction(tuple):
    return True if 'switch' in tuple[2] else False
```

6.8. Filtre d'interruptor obert

Per a filtrar els interruptors oberts, cal una funció com la que es troba a continació, que retorni el valor True si l'interruptor es troba obert i False si es troba tancat.

```
def filterOpenSwitches(net, switchID):
    return not(net.switch.closed.loc[switchID])
```

7. XARXA DE L'ESQUIROL

Per a la realització del treball, s'ha utilitzat una xarxa situada a l'Esquirol. La xarxa té una topologia radial, com es veu en la figura 30, la xarxa conté 44 busos i està connectada a 2 feeders. Per a la reconfiguració en cas de falta es pot aïllar sobre 3 interruptors. A més a més, també conté 2 transformadors de mitja tensió a baixa tensió. Hi ha connectat un sistema de bateries que permet el funcionament de la xarxa en mode illa.

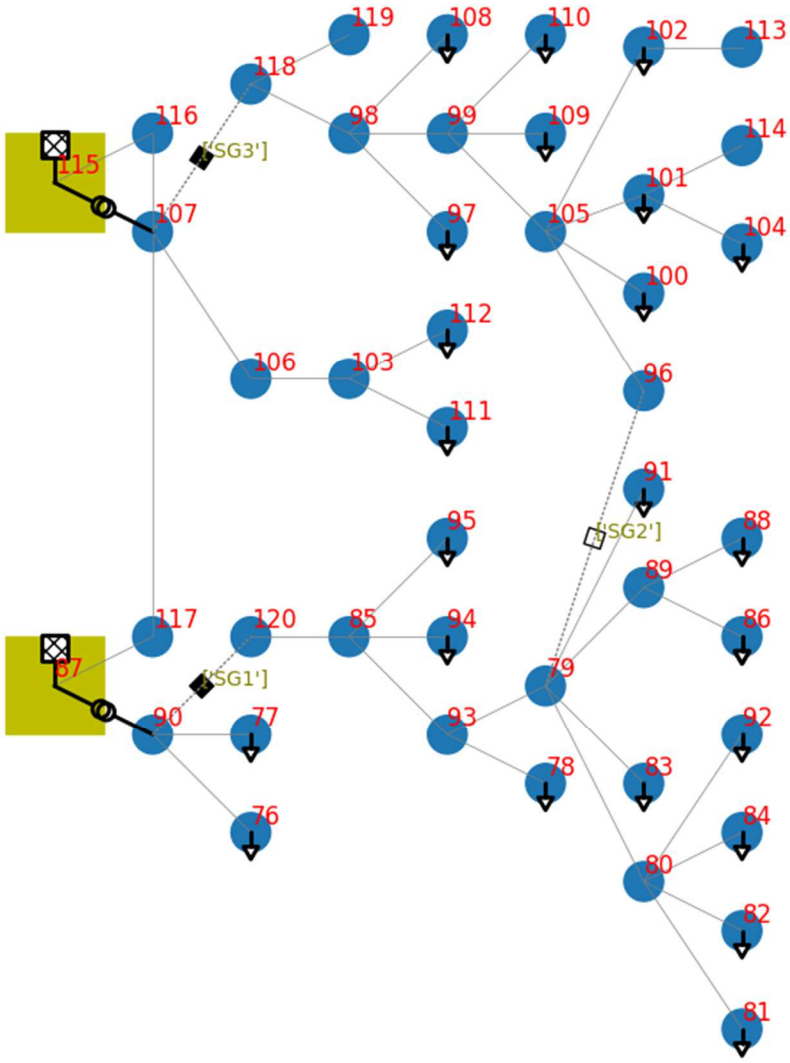

Figura 30: Topologia de la xarxa de l'Esquirol

En la xarxa s'hi distingeixen 3 zones separades pels corresponents interruptors que es resumeixen en taula 1.

Zona 0	Zona 1	Zona 2
103, 106, 107, 76, 77, 111, 112, 115, 116, 117, 87, 90	88, 78, 79, 80, 81, 82, 83, 84, 85, 86, 120, 89, 91, 93, 94, 95	96, 97, 98, 99, 100, 101, 102, 104, 105, 108, 110, 113, 114, 118, 119, 120

Taula 1: Busos corresponents a cada zona (Xarxa L'esquirol)

La zona 0 correspon a la zona on s'hi connecten els feeders, les faltes en aquesta zona no s'estudiaran ja que se suposa que a la xarxa sempre hi ha alimentació. Se suposa que en el cas que hi hagi un problema en zona dels feeders o aigües amunt la xarxa es reconfigura per a poder donar subministrament.

En l'apartat actual es mostren resultats gràfics, però les sortides que proporciona el codi es troben en l'annex B.

7.1. Fallada en la zona 1

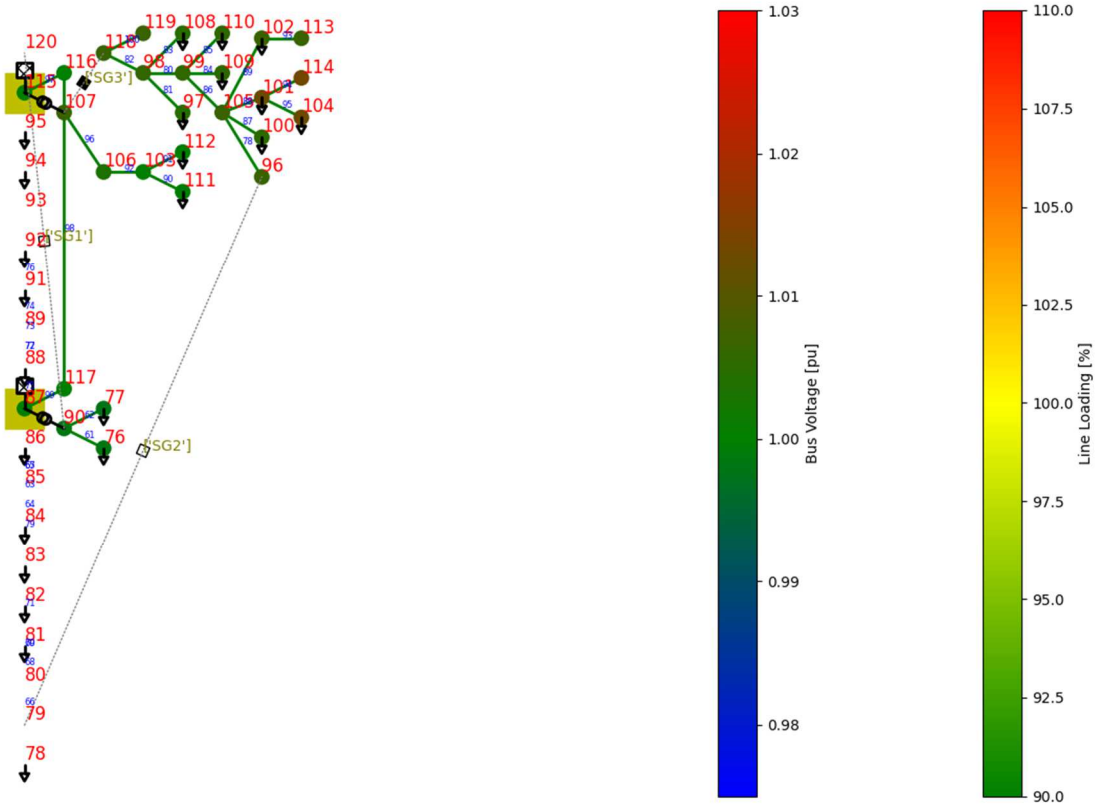

Figura 31: Fluxos de potència quan hi ha fallada en la zona 1

La figura 31 mostra gràficament els fluxos de potència i les tensions, com s'aprecia, els elements que no estan alimentats queden sense color i els alimentats es tenyeixen amb diferents colors segons el percentatge de càrrega de les línies i els nivells de tensió als busos.

Es pot veure que la zona 1 queda desalimentada. A més a més cap bus o línia apareix de color vermell, per aquest motiu es pot concloure que la falta queda correctament aïllada i que el sistema funciona sense contingències. Per tant, la reconfiguració donada és vàlida.

7.2. Fallada en la zona 2

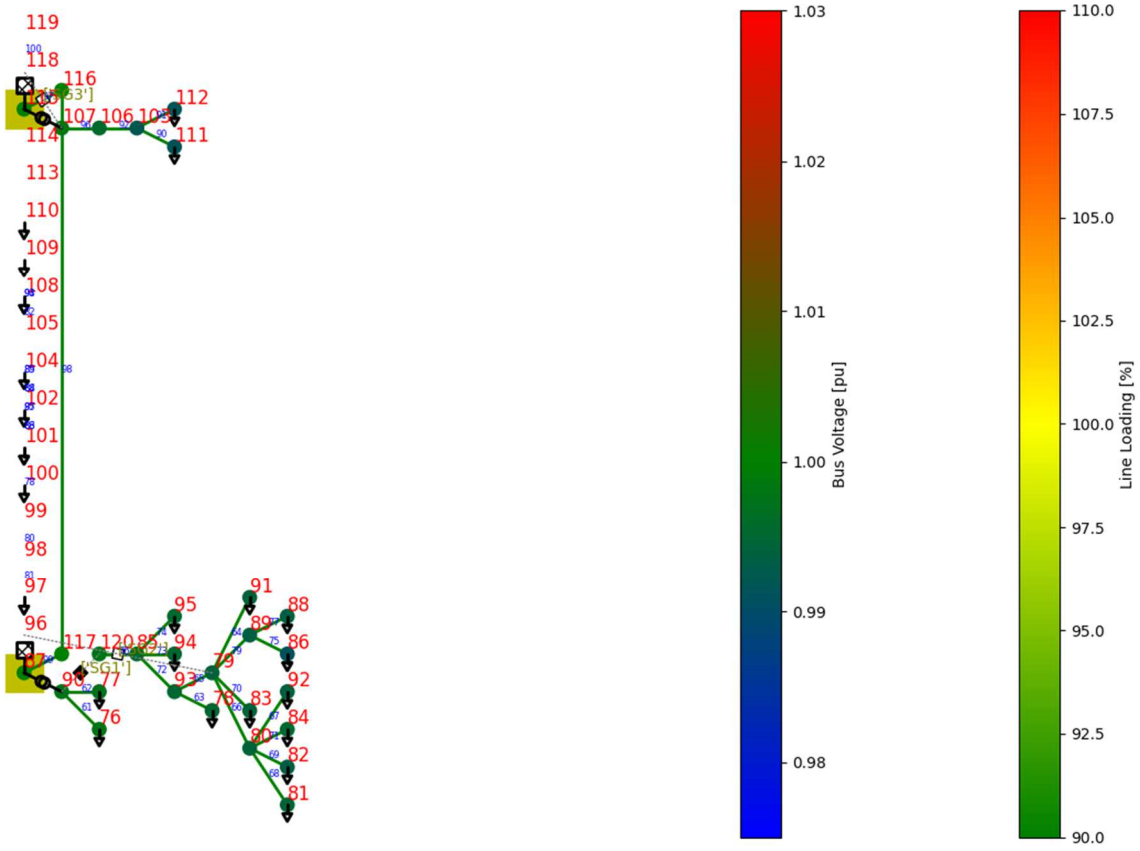

Figura 32: Fluxos de potència quan hi ha una fallada en la zona 2

Anàlogament a l'apartat anterior, la zona 2 queda aïllada i en l'anàlisi de fluxos de potència i ni vell de tensions no apareixen elements tenyits de vermell, per tant es pot assumir que la solució proporcionada és acceptable.

8. XARXA IEEE 33

La xarxa anterior (Esquirol) queda no demostra el funcionament de l'algorisme. Per aquest motiu s'ha creat una nova xarxa inspirada en la xarxa IEEE 33 (figura 33), aquesta xarxa es un model teòric utilitzada per a diferents investigadors per a contrastar hipòtesis.

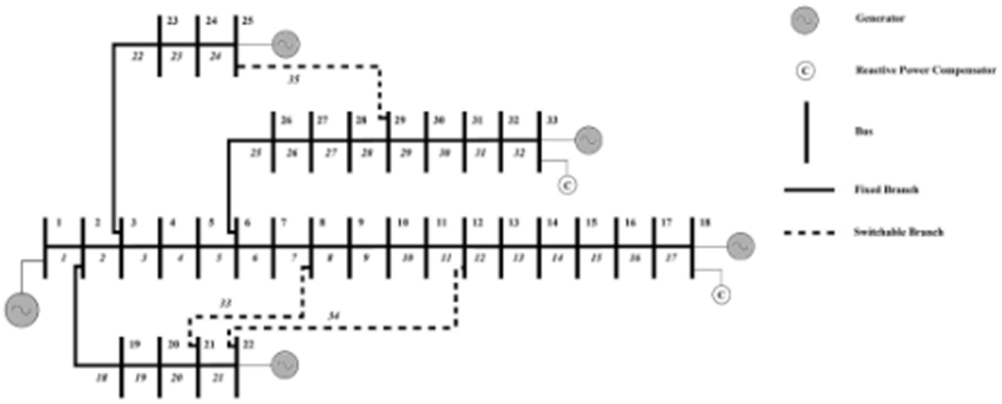

Figura 33: Xarxa IEEE 33

Cal comentar que els busos i línies són els mateixos que en la figura 33 però s'han afegit interruptors per a que l'algorisme pugui realitzar més combinacions i s'han tret els generadors per a afegir conjunts de feeders i transformadors.

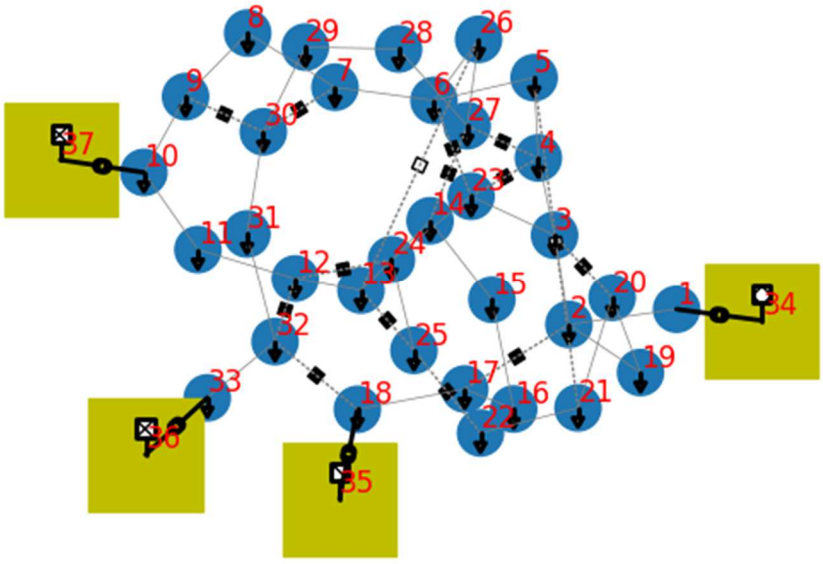

Figura 34: Configuració inicial de la xarxa IEEE 33

La xarxa disposa de 26 interruptors per tant el nombre de combinacions que es poden donar (2^{26}) és molt alt per a comprovar-les totes, per aquest motiu l'algorisme genètic pren valor, ja que es més assequible en termes computacionals.

En aquest cas s'ha simulat una falta en la zona 2 per a comprovar que l'algorisme provoca una millor reconfiguració. La zona 2 és la formada per els busos 24, 25, 3 i 23 (figura 35).

```
{0: {1, 34}, 1: {2}, 2: {24, 25, 3, 23}, 3: {4, 5, 6}, 4: {8, 9, 7}, 5: {10, 11, 12, 37}, 6: {13, 14, 15, 16, 17}, 7:
 {18, 35}, 8: {19, 20, 21, 22}, 9: {26}, 10: {27, 28}, 11: {32, 33, 36, 29, 30, 31}}
```

Figura 35: Zones de la xarxa IEEE 33

En una primera fase, l'algorisme aïlla la falta i reconfigura la xarxa per a que el màxim de consumidors tinguin subministrament, en la figura 36 es pot observar la zona de falta aïllada i la configuració post-falta.

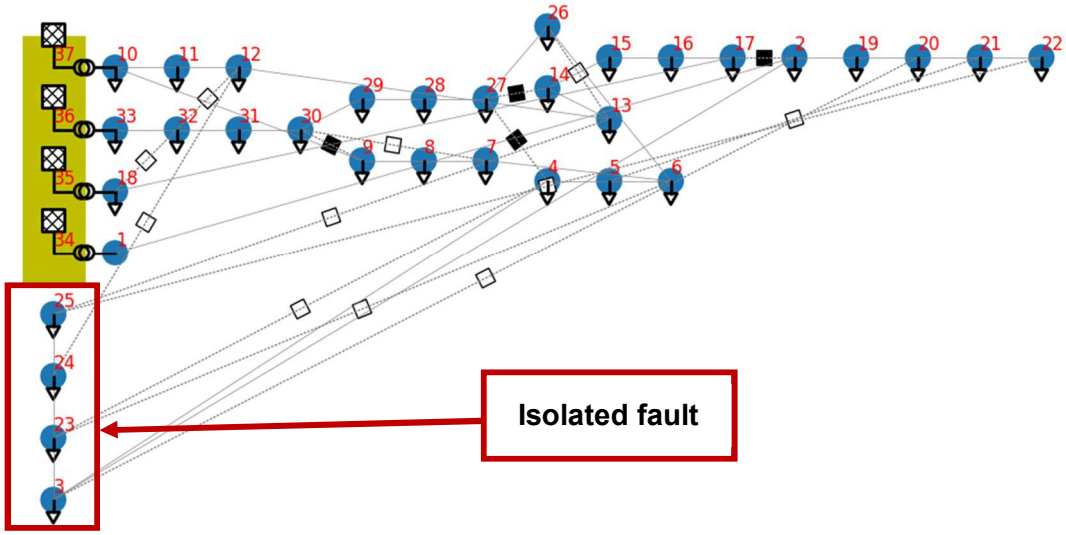

Figura 36: Reconfiguració incial de la xarxa IEEE 33

Es calculen les pèrdues inicials per a comprovar si l'algorisme millora la solució inicial, aquestes són de 25,4565 MW (figura 37).

```
Initial power losses, value of cost function for active power losses:
25.456567906249465
```

Figura 37: Pèrdues de potència inicials, xarxa IEEE 33

Seguidament entra en funcionament l'algorisme genètic, aquest proposa una solució que millora la funció de cost, és a dir redueix les pèrdues de potència. Tanmateix, la nova solució manté la falta aïllada (figura 38). Així com, s'observa que es manté el criteri de màxim subministrament als consumidors, ja que els únics busos aïllats són els de la zona de falta.

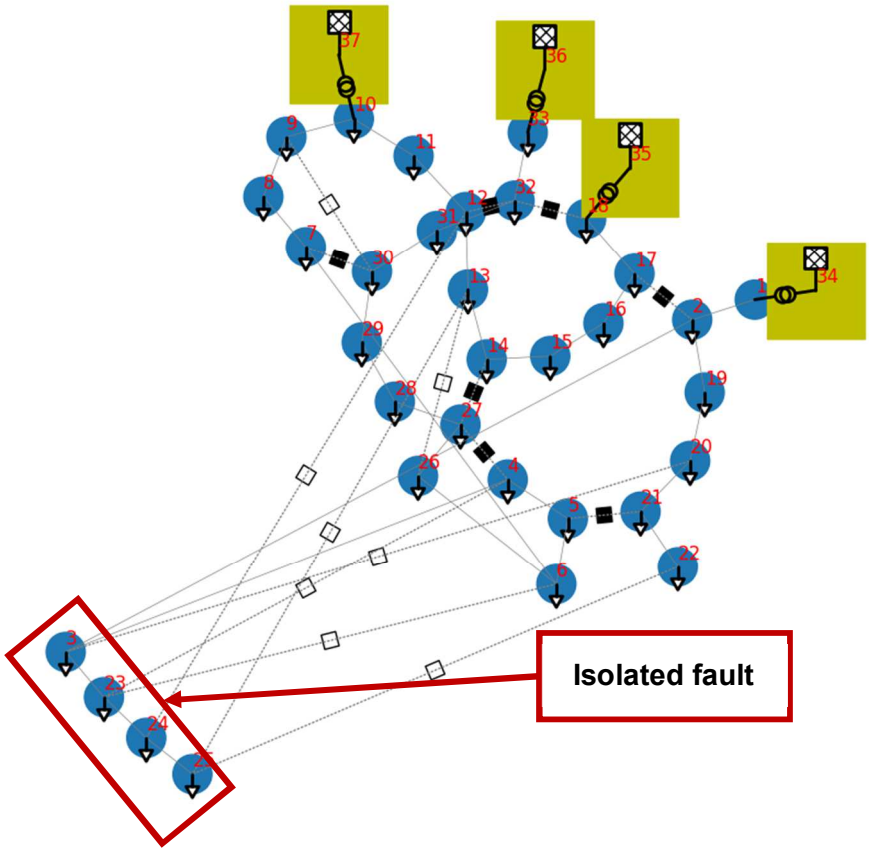

Figura 38: Proposta de solució final

Cal remarcar que l'algorisme no té perquè generar sempre la mateixa solució ja que hi ha un pas en el que una llista d'interruptors guardats per a fer canvis es mescla aleatòriament (apartat 6.3).

Finalment, per a demostrar que funciona és mostren les pèrdues de potència finals (figura 39) per a una execució del codi i com es veu, aquestes s'han reduït.

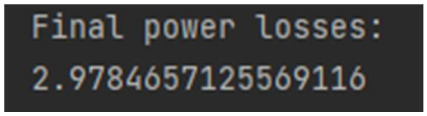

Figura 39: Pèrdues de potència final

Degut a la complexitat de la xarxa l'algorisme utilitzat per a calcular la funció de cost no convergia, ja que es basa en el mètode iteratiu Newton-Raphson, amb altres mètodes iteratius disponibles a la llibreria pandapower tampoc convergia. Per aquest motiu s'ha utilitzat la simulació de la xarxa amb un mètode lineal, que s'utilitza sovint en l'anàlisi de contingències.

Aquest mètode suposa que els voltatges als busos són unitaris i elimina les resistències de les línies, d'aquesta manera és té una solució que s'aproxima al model real però que es pot calcular mitjançant un sistema lineal d'equacions. Amb el mètode lineal no intervé cap procés de càlcul iteratiu, per tant no s'hi troben problemes de convergència en el càlcul.

Per a calcular les pèrdues s'ha utilitzat la intensitat que passen per les diferents línies i les resistències de les línies.

L'objectiu d'aquest apartat és demostrar que l'algorisme millora la solució inicial, per tant si la simulació de la xarxa es aproximada i no exacte no comporta cap problemàtica. Es pretén mostrar mitjançant un indicador que l'algorisme funciona correctament, és indiferent si aquest indicador adopta un valor real o aproximat, la importància es troba en que aquest millori com en el cas estudiat.

9. BIBLIOGRAFÍA

ARMANIOUS, MARIO. The Self-Healing grid (http://environment.umn.edu/education/susteducation/pathways-to-renewable-energy/the-self-healing-grid/, 20 de febrer de 2022)

HACOPIAN, DOLATABADI. GHOTBANIAN, MAEDEH. SIANO, PIERLUIGI. HATZIARGYRIOU, NIKOS. An Enhanced IEEE 33 Bus Benchmark Test System for Distribution System Studies. (https://ieeexplore.ieee.org/abstract/document/9258930, 21 de febrer de 2022)

MATPLOTLIB. Documentació de la llibreria Matplotlib. (https://matplotlib.org/, 5 de març de 2022)

MESUT, BARAN. FELIX, WU. Network reconfiguration in distribution systems for loss reduction and load balancing. (https://ieeexplore.ieee.org/document/25627, 10 de febrer de 2022)

NETWORKX. Documentació de la llibreria NetworkX. (https://networkx.org/, 16 de març de 2022)

PANDAPOWER. Documentació de la llibreria pandapower. (http://www.pandapower.org/, 10 de febrer de 2022)

STACKOVERFLOW. Fòrum d'ajuda a la programació. (https://stackoverflow.com/, 3 de març de 2022)

10. CONCLUSIONS

El projecte pretenia dissenyar una estratègia de planificació del self-healing. Mitjançant la creació d'un algorisme genètic, es crea una estratègia de configuració òptima en termes de menor afectació als consumidors, menor número d'intervenció d'elements de commutació tancats i menors pèrdues de potència.

En primer lloc s'aïlla la falta amb l'obriment d'interruptors propers a aquesta, seguidament es proposa una reconfiguració que minimitza l'afectació al nombre de consumidors i en la qual s'eviten les connexions en anell, per tant es tanquen el menor nombre d'interruptors. Es duu a terme a través del tancament inicial de tots els interruptors que no aïllen la falta i seguidament es van obrint tots els interruptors possibles de manera que no quedin zones sense alimentació, així s'aconsegueix el màxim d'interruptors oberts és a dir el mínim de tancats i la menor afectació de desconnexió als consumidors.

A continuació, partint d'una reconfiguració inicial s'utilitza la part genètica per a buscar solucions que millorin les pèrdues de potència inicial. Dit d'una altra manera, és substitueixen interruptors per d'altres per a crear una nova configuració on hi hagi menys pèrdues en les línies, interruptors i transformadors. Els interruptors intercanviats pertanyen al mateix anell, de manera que és manté el criteri de menor afectació als consumidors i menor nombre d'intervenció d'interruptors tancats.

Finalment, per a comprovar el funcionament de l'algorisme, s'executa i s'estudien els fluxos de potència d'una xarxa model situada al poble de l'Esquirol, també es fa una menció a que la solució adoptada per l'algorisme en termes de flux de potència. D'altra banda amb una xarxa amb estructura més complexa (IEEE 33) s'estudia el comportament de l'algorisme i es reflecteix que aquest proposa una solució millor a la inicial.

MARC PADILLA MORALES - DNI 41588630F

Firmado digitalmente por MARC PADILLA MORALES - DNI 41588630F
Fecha: 2022.06.10 20:50:26 +02'00'

Marc Padilla Morales
Graduat en enginyeria elèctrica
Girona, 6 de juny del 2022

A. PROGRAMA ALGORISME GENÈTIC

```
import networkx
import pandapower as pp
import pandas as pd
import pandapower.topology
import pandapower.plotting as plot
from pandapower.plotting import get_collection_sizes
import networkx as nx
import matplotlib.pyplot as plt
import copy
import random
import math
from pandapower import networks
try:
    import seaborn
    colors = seaborn.color_palette()
except:
    colors = ["b", "g", "r", "c", "y"]

evaluedNet = pp.create_empty_network()

path = 'Files/Xarxa_IEE.xlsx'
bus = pd.read_excel(path, sheet_name=  "bus", decimal = ',')
#busgeo = pd.read_excel(path, sheet_name=  "bus_geodata", decimal =
',')
# Importing the net

for _, busd in bus.iterrows():
    pp.create_bus(evaluedNet, vn_kv = busd.vn_kv, name =
busd['name'],type = busd.type, in_service = busd.in_service,index =
busd["Unnamed: 0"])

lines = pd.read_excel(path, sheet_name=  "line", decimal = ',')

pp.add_zero_impedance_parameters(evaluedNet)

for _, lined in lines.iterrows():
    from_bus = lined.from_bus
    to_bus = lined.to_bus
    pp.create_line_from_parameters(evaluedNet, from_bus, to_bus,
length_km=lined.length_km, r_ohm_per_km = lined.r_ohm_per_km,
x_ohm_per_km = lined.x_ohm_per_km, name = lined["name"],
in_service=lined.in_service, index = lined["Unnamed: 0"],
c_nf_per_km = lined.c_nf_per_km, max_i_ka = lined.max_i_ka)


loads = pd.read_excel(path, sheet_name=  "load", decimal = ',')

for _, loadd in loads.iterrows():
    pp.create_load(evaluedNet, loadd.bus, name=loadd["name"],p_mw =
loadd.p_mw, q_mvar = loadd.q_mvar, in_service=loadd.in_service,
index = loadd["Unnamed: 0"])
```

```
pp.create_transformer_from_parameters(evaluedNet, 34, 1, sn_mva=3,
vn_hv_kv=20, vn_lv_kv=0.4, vkr_percent=0.06,
                                      vk_percent=8, pfe_kw=0,
i0_percent=0, tp_pos=0, shift_degree=0, name='T1')
pp.create_transformer_from_parameters(evaluedNet, 35, 18, sn_mva=3,
vn_hv_kv=20, vn_lv_kv=0.4, vkr_percent=0.06,
                                      vk_percent=8, pfe_kw=0,
i0_percent=0, tp_pos=0, shift_degree=0, name='T2')
pp.create_transformer_from_parameters(evaluedNet, 36, 33, sn_mva=3,
vn_hv_kv=20, vn_lv_kv=0.4, vkr_percent=0.06,
                                      vk_percent=8, pfe_kw=0,
i0_percent=0, tp_pos=0, shift_degree=0, name='T3')
pp.create_transformer_from_parameters(evaluedNet, 37, 10, sn_mva=3,
vn_hv_kv=20, vn_lv_kv=0.4, vkr_percent=0.06,
                                      vk_percent=8, pfe_kw=0,
i0_percent=0, tp_pos=0, shift_degree=0, name='T4')


'''
trafos = pd.read_excel(path, sheet_name=  "trafo", decimal = ',')

for _, trafod in trafos.iterrows():
    pp.create_transformer(evaluedNet, hv_bus=trafod.hv_bus,
lv_bus=trafod.lv_bus, std_type=trafod.std_type, name=trafod["name"],
tap_pos=trafod.tap_pos, in_service=trafod.in_service, df=trafod.df,
index = trafod["Unnamed: 0"])
'''
switch = pd.read_excel(path, sheet_name=  "switch", decimal = ',')

for _, switchd in switch.iterrows():
    pp.create_switch(evaluedNet, bus=switchd.bus,
element=switchd.element, et=switchd.et, closed=switchd.closed,
name=switchd["name"], z_ohm=switchd.z_ohm, index = switchd["Unnamed:
0"])

ext_grids = pd.read_excel(path, sheet_name=  "ext_grid", decimal =
',')

for _, extgd in ext_grids.iterrows():
    pp.create_ext_grid(evaluedNet, name=extgd["name"],
bus=extgd.bus, vm_pu=extgd.vm_pu, va_degree=extgd.va_degree,
in_service=extgd.in_service, s_sc_max_mva=extgd.s_sc_max_mva,
rx_max=extgd.rx_max, x0x_max=extgd.x0x_max, r0x0_max=extgd.r0x0_max,
index = extgd["Unnamed: 0"])
'''
sgens = pd.read_excel(path, sheet_name=  "sgen", decimal = ',')

for _, sgend in sgens.iterrows():
    pp.create_sgen(evaluedNet, name=sgend["name"], bus=sgend.bus,
p_mw=sgend.p_mw, q_mvar=sgend.q_mvar, scaling=sgend.scaling,
in_service=sgend.in_service, type=sgend.type,
current_source=sgend.current_source, index = sgend["Unnamed: 0"])
'''


# Fault zone isolation functions
```

```
def identifyZones(net): #Creating a dictionary with all areas
represented with
    auxNet = copy.deepcopy(net) #Creting a new net for open all
switches
    auxNet.switch.closed.loc[auxNet.switch.index] = False
    netGraph = pp.topology.create_nxgraph(auxNet)
    ar_dict = {}
    i = 0
    for area in pp.topology.connected_components(netGraph):
#Dictionary because not allows duplicate members
        ar_dict[i] = area
        i = i+1
    return ar_dict

def busZone(zone_dict, bus_ev): #Identifing the zone that has a
specific bus
    for key, value in zone_dict.items():
        if bus_ev in value:
            return key

def idSwZone(net, zone_dict): #Making a dictionary of switches and
its zones
    swZ_dict = {}
    #Switch between 2 buses
    for swId, swElements in net.switch.iterrows():
        swZ_dict[swId] =
[busZone(zone_dict,swElements.bus),busZone(zone_dict,swElements.elem
ent)]
    return swZ_dict

def openSwitches(sz_dict, afected_zone): #Function that gives you
all switches that has to be open because are in contact with a zone
that has a failure
    sw_open = []
    for key , values in sz_dict.items():
        if afected_zone in values:
            sw_open.append(key)
    return sw_open

def conSwitch(sz_dict, open_switches): #Switches able to be
considered
    conf_switches = []
    for key, values in sz_dict.items():
        if key not in open_switches:
            #Append into list
            conf_switches.append(key)
    return conf_switches

def openForFault(net, swOpen):
    for i in swOpen:
        net.switch.closed.loc[i]=False
    return
```

```
# Reconfiguration functions

def closeAllSwitches(net, switchToClose): #Function that closes all
the switches
    for i, switch in net.switch.iterrows():
        if i in switchToClose:

            net.switch.closed.loc[i] = True
    return

def initialZonesToExtGrid(net, ZoneBusDic):
    ZoneExtGridList = []
    for i, buses in ZoneBusDic.items():
        ExtGridPresence = False
        for index, val in net.ext_grid.iterrows():
            if val.bus in buses:
                ExtGridPresence = True
        ZoneExtGridList.append(ExtGridPresence)
    return ZoneExtGridList

def makeZoneList(net, ZoneBusDict):
    netGraph = pp.topology.create_nxgraph(net)
    ZonesList = []
    for area in pp.topology.connected_components(netGraph):
        zones = set()
        for bus in area:
            zones.add(busZone(ZoneBusDict, bus))
        ZonesList.append(zones)
    return ZonesList

def extGridComprovation(ZoneList,EGZoneList):
    extGridList = []
    for zoneset in ZoneList:
        zoneset_to_extgrid = False
        for setelement in zoneset:
            if EGZoneList[setelement]:
                zoneset_to_extgrid = True
        extGridList.append(zoneset_to_extgrid)
    return extGridList

def countFalsesinList(analysedlist):
    i = 0
    for element in analysedlist:
        if element is False:
            i = i+1
    return i

def openingOfswitches(numFalses_initial, net, feasibleSwitch,
ZoneBusDict, EGZoneList):
    for element in feasibleSwitch:
        net.switch.closed.loc[element] = False
        if not (numFalses_initial ==
countFalsesinList(extGridComprovation(makeZoneList(net,
ZoneBusDict),EGZoneList))):
            net.switch.closed.loc[element]=True
    return
```

```
def makeListOfSwitches(net):
    swList = []
    for element in net.switch.closed:
        swList.append(element)
    return swList


# Genetic algorithm functions
def powerLosses(net): #Functions for power losses
    pp.rundcpp(net)
    powerLoss = 0
    for key,loss in net.res_line.i_to_ka.items():
        if(math.isnan(loss)):
            loss = 0
        powerLoss = powerLoss +
pow(loss,2)*net.line.loc[key].r_ohm_per_km*net.line.loc[key].length_
km
    '''
    for key, loss in net.res_trafo.pw_hv_mw.items():
        powerLoss = powerLoss + loss
    '''
    return powerLoss


def identifyLoop(net, close_candidate_sw_id): #Has to be modified
for encountering swID
    net.switch.closed.loc[close_candidate_sw_id] = True
    netGraph = pp.topology.create_nxgraph(net)
    cycle = list(nx.find_cycle(netGraph,
source=net.switch.bus.loc[close_candidate_sw_id],
orientation='ignore'))
    net.switch.closed.loc[close_candidate_sw_id] = False
    return cycle


def filterCycleFunction(tuple):
    return True if 'switch' in tuple[2] else False


def findSwitchInLoop(cycle, close_candidate_sw_id):
    switchesOnLoop = list(filter(filterCycleFunction,cycle))
    print("Filtered switches:")
    for element in switchesOnLoop:
        if not(element[2][1] == close_candidate_sw_id):
            return element[2][1]


def filterOpenSwitches(net, switchID):
    return not(net.switch.closed.loc[switchID])


def geneticFunction(net, configurable_switches, switches_bus_dict):
    initial_powerLosses = powerLosses(net)
    print("Initial power losses, value of cost function for active
```

```
power losses:")
    print(initial_powerLosses)

    openedSwitches = list(filter(lambda switchID:
filterOpenSwitches(net, switchID), configurable_switches))
    print("Switches that are opened in configurable switches:")
    print(openedSwitches)

    keeped_switches = []
    for m in openedSwitches:
        print("Open switch for replacement: " + str(m))
        print("Loop identified:")

        try:
            loop = identifyLoop(net, m)
            print(loop)

            replaced_switch = findSwitchInLoop(loop, m)
            print("Switch that replace candidate: " +
str(replaced_switch))

            #Executing switch change
            evaluedNet.switch.closed.loc[m] = True
            evaluedNet.switch.closed.loc[replaced_switch] = False

            #Recalculating cost function
            current_powerLosses = powerLosses(net)
            if (initial_powerLosses >= current_powerLosses):
                keeped_switches.append([m, replaced_switch])
            evaluedNet.switch.closed.loc[m] = False
            evaluedNet.switch.closed.loc[replaced_switch] = True

        except:
            print("Loop non identified")

    # Mix different options
    random.shuffle(keeped_switches)
    new_powerLosses = initial_powerLosses
    if not(len(keeped_switches) == 0):
        for n in keeped_switches:
            # Switches improve cost function
            evaluedNet.switch.closed.loc[n[0]] = True
            switch_state = False
            if (evaluedNet.switch.closed.loc[n[1]]):
                switch_state = True
            evaluedNet.switch.closed.loc[n[1]] = False

            #Non decremental power losses --> undo the change and
break the loop
            instant_powerlosses = powerLosses(net)
            if (new_powerLosses < instant_powerlosses):
                evaluedNet.switch.closed.loc[n[0]] = False


                if (switch_state):
                    evaluedNet.switch.closed.loc[n[1]] = True
```

```
                else:
                    evaluedNet.switch.closed.loc[n[1]] = False
                break

            else:
                new_powerLosses = instant_powerlosses

    if new_powerLosses < initial_powerLosses:
        geneticFunction(net, configurable_switches,
switches_bus_dict)

    return


# Print functions
def printNodes(net):
    net.bus_geodata.drop(net.bus_geodata.index, inplace=True)
    net.line_geodata.drop(net.line_geodata.index, inplace=True)
    plot.create_generic_coordinates(net, respect_switches=True)
    #plot.fuse_geodata(net)
    #bc = plot.create_bus_collection(net, net.bus.index, size=.2,
color=colors[0], zorder=10)
    #tlc, tpc = plot.create_trafo_collection(net, net.trafo.index,
color="g")
    #lcd = plot.create_line_collection(net, net.line.index,
color="grey", linewidths=0.5, use_bus_geodata=True)
    #sc = plot.create_bus_collection(net, net.ext_grid.bus.values,
patch_type="rect", size=.5, color="y", zorder=11)



    sizes = get_collection_sizes(net)

    # create collections for elements
    collections = []
    #collections.append(plt.create_bus_collection(net,
size=sizes["bus"]))
    #collections.append(plt.create_line_collection(net,
use_bus_geodata=True))
    collections.append(plot.create_bus_collection(net,
net.bus.index, size=.2, color=colors[0]))
    collections.append(plot.create_bus_collection(net,
net.ext_grid.bus.values, patch_type="rect", size=.5, color="y"))
    collections.append(plot.create_line_collection(net,
net.line.index, color="grey", linewidths=0.5, use_bus_geodata=True))
    collections.append(plot.create_trafo_collection(net,
size=sizes["trafo"]))
    collections.append(plot.create_ext_grid_collection(net,
size=sizes["ext_grid"], orientation=0))
    collections.append(plot.create_bus_bus_switch_collection(net,
size=sizes["switch"]))
    collections.append(plot.create_load_collection(net,
size=sizes["load"]))

    ax = plt.gca()
    for idx in net.bus_geodata.index:
```

```
        x = net.bus_geodata.loc[idx, "x"]
        y = net.bus_geodata.loc[idx, "y"] + sizes["bus"] * 1.
        ax.text(x, y, str(idx), fontsize=12, color="r")
    '''
    for idx in net.switch.index:
        idbus = net.switch.loc[net.switch.index == idx].bus
        idbus2 = net.switch.loc[net.switch.index == idx].element
        x1 = net.bus_geodata.loc[idbus, "x"]
        y1 = net.bus_geodata.loc[idbus, "y"]
        x2 = net.bus_geodata.loc[idbus2, "x"]
        y2 = net.bus_geodata.loc[idbus2, "y"]
        x = (x1.iloc[0] + x2.iloc[0]) / 2.0
        y = (y1.iloc[0] + y2.iloc[0]) / 2.0
        ax.text(x, y, str(net.switch.loc[net.switch.index == idx,
"name"].values), fontsize=10, color="olive")
    '''
    plot.draw_collections(collections, ax=ax)

    plt.show()
    return

def printPowerFlow(net):
    net.bus_geodata.drop(net.bus_geodata.index, inplace=True)
    net.line_geodata.drop(net.line_geodata.index, inplace=True)
    plot.create_generic_coordinates(net, respect_switches=True)


    sizes = get_collection_sizes(net)


    cmap_list=[(90, "green"), (100, "yellow"), (110, "red")]
    cmap, norm = plot.cmap_continuous(cmap_list)
    lc = plot.create_line_collection(net, net.line.index, zorder=1,
cmap=cmap, norm=norm, linewidths=2, use_bus_geodata=True)
    cmap_list=[(0.975, "blue"), (1.0, "green"), (1.03, "red")]
    cmap, norm = plot.cmap_continuous(cmap_list)
    bc = plot.create_bus_collection(net, net.bus.index.values,
size=0.2, zorder=2, cmap=cmap, norm=norm)
    egrect = plot.create_bus_collection(net,
net.ext_grid.bus.values, patch_type="rect", size=.5, color="y")
    tf = plot.create_trafo_collection(net, size=sizes["trafo"])
    egsymbol = plot.create_ext_grid_collection(net,
size=sizes["ext_grid"], orientation=0)
    swsymbol = plot.create_bus_bus_switch_collection(net,
size=sizes["switch"])
    loadsymbol = plot.create_load_collection(net,
size=sizes["load"])

    ax = plt.gca()
    for idx in net.bus_geodata.index:
        x = net.bus_geodata.loc[idx, "x"]
        y = net.bus_geodata.loc[idx, "y"] + sizes["bus"] * 1.
        ax.text(x, y, str(idx), fontsize=12, color="r")

    for idx in net.line.index:
        idbus = net.line.loc[net.line.index == idx].from_bus
```

```
        idbus2 = net.line.loc[net.line.index == idx].to_bus
        x1 = net.bus_geodata.loc[idbus, "x"]
        y1 = net.bus_geodata.loc[idbus, "y"]
        x2 = net.bus_geodata.loc[idbus2, "x"]
        y2 = net.bus_geodata.loc[idbus2, "y"]
        x = (x1.iloc[0]+x2.iloc[0])/2.0
        y = (y1.iloc[0]+y2.iloc[0])/2.0
        ax.text(x, y, str(idx), fontsize=6, color="b")
    '''
    for idx in net.switch.index:
        idbus = net.switch.loc[net.switch.index == idx].bus
        idbus2 = net.switch.loc[net.switch.index == idx].element
        x1 = net.bus_geodata.loc[idbus, "x"]
        y1 = net.bus_geodata.loc[idbus, "y"]
        x2 = net.bus_geodata.loc[idbus2, "x"]
        y2 = net.bus_geodata.loc[idbus2, "y"]
        x = (x1.iloc[0] + x2.iloc[0]) / 2.0
        y = (y1.iloc[0] + y2.iloc[0]) / 2.0
        ax.text(x, y, str(net.switch.loc[net.switch.index == idx,
"name"].values), fontsize=10, color="olive")
    '''
    plot.draw_collections([egrect,lc,
bc,tf,egsymbol,swsymbol,loadsymbol], figsize=(8,6), ax = ax)
    plt.show()


# Executing functions for results
# Printing net
print("Initial grid and switches state (Graph):")
printNodes(evaluedNet)


# Isolating the fault
print("Creating a dictionary buses-zones:")
dicti = identifyZones(evaluedNet)
print(dicti)

print("Creating a dictionary switches-zones:")
swZonedict = idSwZone(evaluedNet,dicti)
print(swZonedict)


print("Switches that have to be open for fault isolation:")
openSwi = openSwitches(swZonedict,2)
print(openSwi)


print("Switches that are able to consider for algorithm:")
playing_swit = conSwitch(swZonedict, openSwi)
print(playing_swit)


print("Switch state, before isolating the fault:")
print(evaluedNet.switch)
```

```
openForFault(evaluedNet,openSwi)
print("Switch state, after isolating the fault:")
print(evaluedNet.switch)


# Reconfiguration
print("Closing all the switches (non fault isolation switches):")
closeAllSwitches(evaluedNet,playing_swit)
print(evaluedNet.switch.closed)


print("List of zones connected to an external grid:")
ExternalGridZonesList_Initial =
initialZonesToExtGrid(evaluedNet,dicti)
print(ExternalGridZonesList_Initial)


print("Creating a zone set list immediately after fault isolation:")
newZonesList_afterFault = makeZoneList(evaluedNet, dicti)
print(newZonesList_afterFault)


print("Zone sets connected to an external grid, evaluated
immediately after fault isolation:")
extGridList_afterFault =
extGridComprovation(newZonesList_afterFault,
ExternalGridZonesList_Initial)
print(extGridList_afterFault)


print("False number of zone sets connected to an external grid,
immediately evaluated after fault:")
falseNumber_AF = countFalsesinList(extGridList_afterFault)
print(falseNumber_AF)


print("Intermediate switch list, for visualizing switches state
after reconfiguration:")
openingOfswitches(falseNumber_AF,evaluedNet,playing_swit, dicti,
ExternalGridZonesList_Initial)
intermediate_list_SW = makeListOfSwitches(evaluedNet)
print(intermediate_list_SW)


print("Grid and switches state after reconfiguration (Graph):")
printNodes(evaluedNet)
#printPowerFlow(evaluedNet)

print("Running genetic algorithm:")
geneticFunction(evaluedNet, playing_swit, swZonedict)


print("Final results:")
print("Final switch list:")
finalSwitchList = makeListOfSwitches(evaluedNet)
print(finalSwitchList)
```

```
print("Grid and switches state final configuration (Graph):")
printNodes(evaluedNet)
#printPowerFlow(evaluedNet)
print("Final power losses:")
print(powerLosses(evaluedNet))
```

B. RESULTATS DEL CODI

En aquest apartat és mostren els diferents resultats corresponents a les diferents simulacions.

B.1. Resultats de la simulació de falta en zona 1 (Xarxa Esquirol)

Creating a dictionary buses-zones:

```
{0: {103, 106, 107, 76, 77, 111, 112, 115, 116, 117, 87, 90}, 1:
{88, 78, 79, 80, 81, 82, 83, 84, 85, 86, 120, 89, 91, 92, 93, 94,
95}, 2: {96, 97, 98, 99, 100, 101, 102, 104, 105, 108, 109, 110,
113, 114, 118, 119}}

Creating a dictionary switches-zones:

{0: [1, 2], 1: [0, 2], 2: [0, 1]}

Switches that have to be open for fault isolation:

[0, 2]

Switches that are able to consider for algorithm:

[1]

Switch state, before isolating the fault:

   bus  element et  type  closed name  z_ohm

0   79       96  b  None   False  SG2    0.0

1  107      118  b  None    True  SG3    0.0

2   90      120  b  None    True  SG1    0.0

Switch state, after isolating the fault:

   bus  element et  type  closed name  z_ohm

0   79       96  b  None   False  SG2    0.0

1  107      118  b  None    True  SG3    0.0

2   90      120  b  None   False  SG1    0.0

Closing all the switches (non fault isolation switches):

0    False

1     True
```

```
2    False

Name: closed, dtype: bool

List of zones connected to an external grid:

[True, False, False]

Creating a zone set list immediately after fault isolation:

[{0, 2}, {1}]

Zone sets connected to an external grid, evaluated immediately after
fault isolation:

[True, False]

False number of zone sets connected to an external grid, immediately
evaluated after fault:

1

Intermediate switch list, for visualizing switches state after
reconfiguration:

[False, True, False]

Grid and switches state after reconfiguration (Graph):

Running genetic algorithm:

Initial power losses, value of cost function for active power
losses:

0.004262563843786619

Switches that are opened in configurable switches:

[]

Final results:

Final switch list:

[False, True, False]

Grid and switches state final configuration (Graph):
```

B.2. Resultats de la simulació de falta en zona 1 (Xarxa Esquirol)

```
Creating a dictionary buses-zones:

{0: {103, 106, 107, 76, 77, 111, 112, 115, 116, 117, 87, 90}, 1:
{88, 78, 79, 80, 81, 82, 83, 84, 85, 86, 120, 89, 91, 92, 93, 94,
```

```
95}, 2: {96, 97, 98, 99, 100, 101, 102, 104, 105, 108, 109, 110,
113, 114, 118, 119}}

Creating a dictionary switches-zones:

{0: [1, 2], 1: [0, 2], 2: [0, 1]}

Switches that have to be open for fault isolation:

[0, 1]

Switches that are able to consider for algorithm:

[2]

Switch state, before isolating the fault:

   bus  element et  type  closed name  z_ohm

0   79       96  b  None   False  SG2    0.0

1  107      118  b  None    True  SG3    0.0

2   90      120  b  None    True  SG1    0.0

Switch state, after isolating the fault:

   bus  element et  type  closed name  z_ohm

0   79       96  b  None   False  SG2    0.0

1  107      118  b  None   False  SG3    0.0

2   90      120  b  None    True  SG1    0.0

Closing all the switches (non fault isolation switches):

0    False

1    False

2     True

Name: closed, dtype: bool

List of zones connected to an external grid:

[True, False, False]

Creating a zone set list immediately after fault isolation:

[{0, 1}, {2}]

Zone sets connected to an external grid, evaluated immediately after
fault isolation:

[True, False]
```

```
False number of zone sets connected to an external grid, immediately
evaluated after fault:

1

Intermediate switch list, for visualizing switches state after
reconfiguration:

[False, False, True]

Grid and switches state after reconfiguration (Graph):

Running genetic algorithm:

Initial power losses, value of cost function for active power
losses:

0.004376704078010812

Switches that are opened in configurable switches:

[]

Final results:

Final switch list:

[False, False, True]
```

B.3. Resultats de la simulació de falta en zona 2 (Xarxa IEEE 33)

```
C:\Users\Marc\PycharmProjects\esquirolEST\venv\Scripts\python.exe
C:/Users/Marc/PycharmProjects/esquirolEST/IEEE33.py

Initial grid and switches state (Graph):

The number of given colors (1) is smaller than the number of nodes
(37) to draw! The colors will be repeated to fit.

Creating a dictionary buses-zones:

{0: {1, 34}, 1: {2}, 2: {24, 25, 3, 23}, 3: {4, 5, 6}, 4: {8, 9, 7},
5: {10, 11, 12, 37}, 6: {13, 14, 15, 16, 17}, 7: {18, 35}, 8: {19,
20, 21, 22}, 9: {26}, 10: {27, 28}, 11: {32, 33, 36, 29, 30, 31}}

Creating a dictionary switches-zones:

{1: [7, 6], 2: [6, 5], 3: [6, 9], 4: [10, 9], 5: [5, 4], 6: [1, 0],
7: [3, 2], 8: [8, 3], 9: [4, 3], 10: [2, 1], 11: [3, 2], 12: [11,
10], 13: [1, 7], 14: [11, 4], 15: [7, 11], 16: [1, 6], 17: [2, 8],
18: [3, 2], 19: [8, 2], 20: [6, 10], 21: [11, 5], 22: [3, 2], 23:
[10, 3], 24: [6, 2], 25: [4, 11], 26: [2, 5]}

Switches that have to be open for fault isolation:
```

```
[7, 10, 11, 17, 18, 19, 22, 24, 26]
Switches that are able to consider for algorithm:
[1, 2, 3, 4, 5, 6, 8, 9, 12, 13, 14, 15, 16, 20, 21, 23, 25]
Switch state, before isolating the fault:
    bus  element et  type  closed  name  z_ohm
1    18       17  l  None    True   SW1    0.0
2    13       12  l  None    True   SW2    0.0
3    13       26  b  None   False   SW3    0.0
4    27       26  l  None    True   SW4    0.0
5    10        9  l  None    True   SW5    0.0
6     2        1  l  None    True   SW6    0.0
7     4        3  l  None    True   SW7    0.0
8    21        5  b  None   False   SW8    0.0
9     7        6  l  None    True   SW9    0.0
10    3        2  l  None    True  SW10    0.0
11    6       25  l  None    True  SW11    0.0
12   29       28  l  None    True  SW12    0.0
13    2       18  l  None    True  SW13    0.0
14   30        7  b  None    True  SW14    0.0
15   18       32  b  None    True  SW15    0.0
16    2       17  b  None    True  SW16    0.0
17    3       20  b  None    True  SW17    0.0
18    4       23  b  None    True  SW18    0.0
19   22       25  b  None    True  SW19    0.0
20   14       27  b  None    True  SW20    0.0
21   32       12  b  None    True  SW21    0.0
22    6       23  b  None    True  SW22    0.0
23   27        4  b  None    True  SW23    0.0
24   13       25  b  None    True  SW24    0.0
```

```
25    9       30  b  None    True  SW25    0.0
26   24       12  b  None    True  SW26    0.0
Switch state, after isolating the fault:
    bus  element et  type  closed  name  z_ohm
1    18       17  l  None    True   SW1    0.0
2    13       12  l  None    True   SW2    0.0
3    13       26  b  None   False   SW3    0.0
4    27       26  l  None    True   SW4    0.0
5    10        9  l  None    True   SW5    0.0
6     2        1  l  None    True   SW6    0.0
7     4        3  l  None   False   SW7    0.0
8    21        5  b  None   False   SW8    0.0
9     7        6  l  None    True   SW9    0.0
10    3        2  l  None   False  SW10    0.0
11    6       25  l  None   False  SW11    0.0
12   29       28  l  None    True  SW12    0.0
13    2       18  l  None    True  SW13    0.0
14   30        7  b  None    True  SW14    0.0
15   18       32  b  None    True  SW15    0.0
16    2       17  b  None    True  SW16    0.0
17    3       20  b  None   False  SW17    0.0
18    4       23  b  None   False  SW18    0.0
19   22       25  b  None   False  SW19    0.0
20   14       27  b  None    True  SW20    0.0
21   32       12  b  None    True  SW21    0.0
22    6       23  b  None   False  SW22    0.0
23   27        4  b  None    True  SW23    0.0
24   13       25  b  None   False  SW24    0.0
25    9       30  b  None    True  SW25    0.0
```

```
26   24       12  b  None   False  SW26    0.0
Closing all the switches (non fault isolation switches):
1      True
2      True
3      True
4      True
5      True
6      True
7     False
8      True
9      True
10    False
11    False
12     True
13     True
14     True
15     True
16     True
17    False
18    False
19    False
20     True
21     True
22    False
23     True
24    False
25     True
26    False
Name: closed, dtype: bool
```

```
List of zones connected to an external grid:

[True, False, False, False, False, True, False, True, False, False,
False, True]

Creating a zone set list immediately after fault isolation:

[{0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11}, {2}]

Zone sets connected to an external grid, evaluated immediately after
fault isolation:

[True, False]

False number of zone sets connected to an external grid, immediately
evaluated after fault:

1

Intermediate switch list, for visualizing switches state after
reconfiguration:

[False, False, False, True, False, False, False, False, False,
False, False, True, True, False, False, True, False, False, False,
True, False, False, True, False, True, False]

Grid and switches state after reconfiguration (Graph):

The number of given colors (1) is smaller than the number of nodes
(37) to draw! The colors will be repeated to fit.

Running genetic algorithm:

Initial power losses, value of cost function for active power
losses:

25.456567906249465

Switches that are opened in configurable switches:

[1, 2, 3, 5, 6, 8, 9, 14, 15, 21]

Open switch for replacement: 1

Loop identified:

Loop non identified

Open switch for replacement: 2

Loop identified:

Loop non identified

Open switch for replacement: 3

Loop identified:
```

```
[(13, 14, ('line', 13), 'forward'), (14, 27, ('switch', 20),
'forward'), (27, 26, ('line', 26), 'forward'), (26, 13, ('switch',
3), 'forward')]

Filtered switches:

Switch that replace candidate: 20

Open switch for replacement: 5

Loop identified:

[(10, 9, ('line', 9), 'forward'), (9, 30, ('switch', 25),
'forward'), (30, 29, ('line', 29), 'forward'), (29, 28, ('line',
28), 'forward'), (28, 27, ('line', 27), 'forward'), (27, 14,
('switch', 20), 'forward'), (14, 13, ('line', 13), 'forward'), (13,
12, ('line', 12), 'forward'), (12, 11, ('line', 11), 'forward'),
(11, 10, ('line', 10), 'forward')]

Filtered switches:

Switch that replace candidate: 25

Open switch for replacement: 6

Loop identified:

Loop non identified

Open switch for replacement: 8

Loop identified:

[(21, 20, ('line', 20), 'forward'), (20, 19, ('line', 19),
'forward'), (19, 2, ('line', 18), 'forward'), (2, 17, ('switch',
16), 'forward'), (17, 16, ('line', 16), 'forward'), (16, 15,
('line', 15), 'forward'), (15, 14, ('line', 14), 'forward'), (14,
27, ('switch', 20), 'forward'), (27, 4, ('switch', 23), 'forward'),
(4, 5, ('line', 4), 'forward'), (5, 21, ('switch', 8), 'forward')]

Filtered switches:

Switch that replace candidate: 16

Open switch for replacement: 9

Loop identified:

[(7, 6, ('line', 6), 'forward'), (6, 5, ('line', 5), 'forward'), (5,
4, ('line', 4), 'forward'), (4, 27, ('switch', 23), 'forward'), (27,
28, ('line', 27), 'forward'), (28, 29, ('line', 28), 'forward'),
(29, 30, ('line', 29), 'forward'), (30, 9, ('switch', 25),
'forward'), (9, 8, ('line', 8), 'forward'), (8, 7, ('line', 7),
'forward')]

Filtered switches:
```

```
Switch that replace candidate: 23

Open switch for replacement: 14

Loop identified:

[(30, 7, ('switch', 14), 'forward'), (7, 8, ('line', 7), 'forward'),
(8, 9, ('line', 8), 'forward'), (9, 30, ('switch', 25), 'forward')]

Filtered switches:

Switch that replace candidate: 25

Open switch for replacement: 15

Loop identified:

[(18, 17, ('line', 17), 'forward'), (17, 16, ('line', 16),
'forward'), (16, 15, ('line', 15), 'forward'), (15, 14, ('line',
14), 'forward'), (14, 27, ('switch', 20), 'forward'), (27, 28,
('line', 27), 'forward'), (28, 29, ('line', 28), 'forward'), (29,
30, ('line', 29), 'forward'), (30, 31, ('line', 30), 'forward'),
(31, 32, ('line', 31), 'forward'), (32, 18, ('switch', 15),
'forward')]

Filtered switches:

Switch that replace candidate: 20

Open switch for replacement: 21

Loop identified:

[(32, 31, ('line', 31), 'forward'), (31, 30, ('line', 30),
'forward'), (30, 29, ('line', 29), 'forward'), (29, 28, ('line',
28), 'forward'), (28, 27, ('line', 27), 'forward'), (27, 14,
('switch', 20), 'forward'), (14, 13, ('line', 13), 'forward'), (13,
12, ('line', 12), 'forward'), (12, 32, ('switch', 21), 'forward')]

Filtered switches:

Switch that replace candidate: 20

Initial power losses, value of cost function for active power
losses:

4.887269906683056

Switches that are opened in configurable switches:

[3, 5, 9, 14, 15, 16, 21]

Open switch for replacement: 3

Loop identified:
```

```
[(13, 14, ('line', 13), 'forward'), (14, 27, ('switch', 20),
'forward'), (27, 26, ('line', 26), 'forward'), (26, 13, ('switch',
3), 'forward')]

Filtered switches:

Switch that replace candidate: 20

Open switch for replacement: 5

Loop identified:

[(10, 9, ('line', 9), 'forward'), (9, 30, ('switch', 25),
'forward'), (30, 29, ('line', 29), 'forward'), (29, 28, ('line',
28), 'forward'), (28, 27, ('line', 27), 'forward'), (27, 14,
('switch', 20), 'forward'), (14, 13, ('line', 13), 'forward'), (13,
12, ('line', 12), 'forward'), (12, 11, ('line', 11), 'forward'),
(11, 10, ('line', 10), 'forward')]

Filtered switches:

Switch that replace candidate: 25

Open switch for replacement: 9

Loop identified:

[(7, 6, ('line', 6), 'forward'), (6, 5, ('line', 5), 'forward'), (5,
4, ('line', 4), 'forward'), (4, 27, ('switch', 23), 'forward'), (27,
28, ('line', 27), 'forward'), (28, 29, ('line', 28), 'forward'),
(29, 30, ('line', 29), 'forward'), (30, 9, ('switch', 25),
'forward'), (9, 8, ('line', 8), 'forward'), (8, 7, ('line', 7),
'forward')]

Filtered switches:

Switch that replace candidate: 23

Open switch for replacement: 14

Loop identified:

[(30, 7, ('switch', 14), 'forward'), (7, 8, ('line', 7), 'forward'),
(8, 9, ('line', 8), 'forward'), (9, 30, ('switch', 25), 'forward')]

Filtered switches:

Switch that replace candidate: 25

Open switch for replacement: 15

Loop identified:

[(18, 17, ('line', 17), 'forward'), (17, 16, ('line', 16),
'forward'), (16, 15, ('line', 15), 'forward'), (15, 14, ('line',
14), 'forward'), (14, 27, ('switch', 20), 'forward'), (27, 28,
```

```
('line', 27), 'forward'), (28, 29, ('line', 28), 'forward'), (29,
30, ('line', 29), 'forward'), (30, 31, ('line', 30), 'forward'),
(31, 32, ('line', 31), 'forward'), (32, 18, ('switch', 15),
'forward')]

Filtered switches:

Switch that replace candidate: 20

Open switch for replacement: 16

Loop identified:

[(2, 19, ('line', 18), 'forward'), (19, 20, ('line', 19),
'forward'), (20, 21, ('line', 20), 'forward'), (21, 5, ('switch',
8), 'forward'), (5, 4, ('line', 4), 'forward'), (4, 27, ('switch',
23), 'forward'), (27, 14, ('switch', 20), 'forward'), (14, 15,
('line', 14), 'forward'), (15, 16, ('line', 15), 'forward'), (16,
17, ('line', 16), 'forward'), (17, 2, ('switch', 16), 'forward')]

Filtered switches:

Switch that replace candidate: 8

Open switch for replacement: 21

Loop identified:

[(32, 31, ('line', 31), 'forward'), (31, 30, ('line', 30),
'forward'), (30, 29, ('line', 29), 'forward'), (29, 28, ('line',
28), 'forward'), (28, 27, ('line', 27), 'forward'), (27, 14,
('switch', 20), 'forward'), (14, 13, ('line', 13), 'forward'), (13,
12, ('line', 12), 'forward'), (12, 32, ('switch', 21), 'forward')]

Filtered switches:

Switch that replace candidate: 20

Initial power losses, value of cost function for active power
losses:

3.2981496999070794

Switches that are opened in configurable switches:

[3, 16, 20, 21, 23, 25]

Open switch for replacement: 3

Loop identified:

[(13, 12, ('line', 12), 'forward'), (12, 11, ('line', 11),
'forward'), (11, 10, ('line', 10), 'forward'), (10, 9, ('line', 9),
'forward'), (9, 8, ('line', 8), 'forward'), (8, 7, ('line', 7),
'forward'), (7, 30, ('switch', 14), 'forward'), (30, 29, ('line',
29), 'forward'), (29, 28, ('line', 28), 'forward'), (28, 27,
```

```
('line', 27), 'forward'), (27, 26, ('line', 26), 'forward'), (26,
13, ('switch', 3), 'forward')]

Filtered switches:

Switch that replace candidate: 14

Open switch for replacement: 16

Loop identified:

[(7, 8, ('line', 7), 'forward'), (8, 9, ('line', 8), 'forward'), (9,
10, ('line', 9), 'forward'), (10, 11, ('line', 10), 'forward'), (11,
12, ('line', 11), 'forward'), (12, 13, ('line', 12), 'forward'),
(13, 14, ('line', 13), 'forward'), (14, 15, ('line', 14),
'forward'), (15, 16, ('line', 15), 'forward'), (16, 17, ('line',
16), 'forward'), (17, 18, ('line', 17), 'forward'), (18, 32,
('switch', 15), 'forward'), (32, 31, ('line', 31), 'forward'), (31,
30, ('line', 30), 'forward'), (30, 7, ('switch', 14), 'forward')]

Filtered switches:

Switch that replace candidate: 15

Open switch for replacement: 20

Loop identified:

[(14, 13, ('line', 13), 'forward'), (13, 12, ('line', 12),
'forward'), (12, 11, ('line', 11), 'forward'), (11, 10, ('line',
10), 'forward'), (10, 9, ('line', 9), 'forward'), (9, 8, ('line',
8), 'forward'), (8, 7, ('line', 7), 'forward'), (7, 30, ('switch',
14), 'forward'), (30, 29, ('line', 29), 'forward'), (29, 28,
('line', 28), 'forward'), (28, 27, ('line', 27), 'forward'), (27,
14, ('switch', 20), 'forward')]

Filtered switches:

Switch that replace candidate: 14

Open switch for replacement: 21

Loop identified:

[(32, 31, ('line', 31), 'forward'), (31, 30, ('line', 30),
'forward'), (30, 7, ('switch', 14), 'forward'), (7, 8, ('line', 7),
'forward'), (8, 9, ('line', 8), 'forward'), (9, 10, ('line', 9),
'forward'), (10, 11, ('line', 10), 'forward'), (11, 12, ('line',
11), 'forward'), (12, 13, ('line', 12), 'forward'), (13, 14,
('line', 13), 'forward'), (14, 15, ('line', 14), 'forward'), (15,
16, ('line', 15), 'forward'), (16, 17, ('line', 16), 'forward'),
(17, 18, ('line', 17), 'forward'), (18, 32, ('switch', 15),
'forward')]

Filtered switches:
```

```
Switch that replace candidate: 14

Open switch for replacement: 23

Loop identified:

[(27, 28, ('line', 27), 'forward'), (28, 29, ('line', 28),
'forward'), (29, 30, ('line', 29), 'forward'), (30, 31, ('line',
30), 'forward'), (31, 32, ('line', 31), 'forward'), (32, 18,
('switch', 15), 'forward'), (18, 17, ('line', 17), 'forward'), (17,
16, ('line', 16), 'forward'), (16, 15, ('line', 15), 'forward'),
(15, 14, ('line', 14), 'forward'), (14, 13, ('line', 13),
'forward'), (13, 12, ('line', 12), 'forward'), (12, 11, ('line',
11), 'forward'), (11, 10, ('line', 10), 'forward'), (10, 9, ('line',
9), 'forward'), (9, 8, ('line', 8), 'forward'), (8, 7, ('line', 7),
'forward'), (7, 6, ('line', 6), 'forward'), (6, 5, ('line', 5),
'forward'), (5, 4, ('line', 4), 'forward'), (4, 27, ('switch', 23),
'forward')]

Filtered switches:

Switch that replace candidate: 15

Open switch for replacement: 25

Loop identified:

[(9, 8, ('line', 8), 'forward'), (8, 7, ('line', 7), 'forward'), (7,
30, ('switch', 14), 'forward'), (30, 31, ('line', 30), 'forward'),
(31, 32, ('line', 31), 'forward'), (32, 18, ('switch', 15),
'forward'), (18, 17, ('line', 17), 'forward'), (17, 16, ('line',
16), 'forward'), (16, 15, ('line', 15), 'forward'), (15, 14,
('line', 14), 'forward'), (14, 13, ('line', 13), 'forward'), (13,
12, ('line', 12), 'forward'), (12, 11, ('line', 11), 'forward'),
(11, 10, ('line', 10), 'forward'), (10, 9, ('line', 9), 'forward')]

Filtered switches:

Switch that replace candidate: 14

Initial power losses, value of cost function for active power
losses:

2.9784657125569116

Switches that are opened in configurable switches:

[14, 16, 20, 23, 25]

Open switch for replacement: 14

Loop identified:

[(30, 29, ('line', 29), 'forward'), (29, 28, ('line', 28),
'forward'), (28, 27, ('line', 27), 'forward'), (27, 26, ('line',
26), 'forward'), (26, 13, ('switch', 3), 'forward'), (13, 12,
```

```
('line', 12), 'forward'), (12, 11, ('line', 11), 'forward'), (11,
10, ('line', 10), 'forward'), (10, 9, ('line', 9), 'forward'), (9,
8, ('line', 8), 'forward'), (8, 7, ('line', 7), 'forward'), (7, 30,
('switch', 14), 'forward')]

Filtered switches:

Switch that replace candidate: 3

Open switch for replacement: 16

Loop identified:

[(13, 14, ('line', 13), 'forward'), (14, 15, ('line', 14),
'forward'), (15, 16, ('line', 15), 'forward'), (16, 17, ('line',
16), 'forward'), (17, 18, ('line', 17), 'forward'), (18, 32,
('switch', 15), 'forward'), (32, 31, ('line', 31), 'forward'), (31,
30, ('line', 30), 'forward'), (30, 29, ('line', 29), 'forward'),
(29, 28, ('line', 28), 'forward'), (28, 27, ('line', 27),
'forward'), (27, 26, ('line', 26), 'forward'), (26, 13, ('switch',
3), 'forward')]

Filtered switches:

Switch that replace candidate: 15

Open switch for replacement: 20

Loop identified:

[(13, 12, ('line', 12), 'forward'), (12, 32, ('switch', 21),
'forward'), (32, 31, ('line', 31), 'forward'), (31, 30, ('line',
30), 'forward'), (30, 29, ('line', 29), 'forward'), (29, 28,
('line', 28), 'forward'), (28, 27, ('line', 27), 'forward'), (27,
26, ('line', 26), 'forward'), (26, 13, ('switch', 3), 'forward')]

Filtered switches:

Switch that replace candidate: 21

Open switch for replacement: 23

Loop identified:

[(27, 26, ('line', 26), 'forward'), (26, 13, ('switch', 3),
'forward'), (13, 12, ('line', 12), 'forward'), (12, 11, ('line',
11), 'forward'), (11, 10, ('line', 10), 'forward'), (10, 9, ('line',
9), 'forward'), (9, 8, ('line', 8), 'forward'), (8, 7, ('line', 7),
'forward'), (7, 6, ('line', 6), 'forward'), (6, 5, ('line', 5),
'forward'), (5, 4, ('line', 4), 'forward'), (4, 27, ('switch', 23),
'forward')]

Filtered switches:

Switch that replace candidate: 3
```

```
Open switch for replacement: 25

Loop identified:

[(13, 14, ('line', 13), 'forward'), (14, 15, ('line', 14),
'forward'), (15, 16, ('line', 15), 'forward'), (16, 17, ('line',
16), 'forward'), (17, 18, ('line', 17), 'forward'), (18, 32,
('switch', 15), 'forward'), (32, 31, ('line', 31), 'forward'), (31,
30, ('line', 30), 'forward'), (30, 29, ('line', 29), 'forward'),
(29, 28, ('line', 28), 'forward'), (28, 27, ('line', 27),
'forward'), (27, 26, ('line', 26), 'forward'), (26, 13, ('switch',
3), 'forward')]

Filtered switches:

Switch that replace candidate: 15

Final results:

Final switch list:

[True, True, True, True, True, True, False, True, True, False,
False, True, True, False, True, False, False, False, False, False,
True, False, False, False, False, False]

Grid and switches state final configuration (Graph):

The number of given colors (1) is smaller than the number of nodes
(37) to draw! The colors will be repeated to fit.

Final power losses:

2.9784657125569116


Process finished with exit code 0
```