Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Millora del software de càlcul per al disseny de sistemes d'electrificació autònoms per a comunitats rurals

MEMÒRIA

Autor: Sergi de la Fuente Vallespí

Directors: Daniela Tost Pardell, Bruno Domenech Léga

Convocatòria: Juliol 2016

Escola Tècnica Superior

d'Enginyeria Industrial de Barcelona

Resum

El TFG s'emmarca en el projecte d'Energies renovables a Amèrica Llatina, del Centre de Cooperació per al Desenvolupament (CCD), el qual té per objectiu la instal·lació de microxarxes en zones rurals. Les microxarxes són un sistema d'electrificació basat en la generació elèctrica amb energies renovables (solar i eòlica) i la interconnexió dels sistemes mitjançant cables, i controlat amb reguladors, bateries i inversors.

Aquest TFG contribueix a aquest objectiu proposant la millora del software d'optimització de microxarxes que es feia servir fins ara. Es parteix d'una metodologia que calcula l'arbre de possibles microxarxes solució dividint el càlcul en tres etapes, en les quals es tenen en compte unes restriccions, com ara el nombre d'usuaris per microxarxa o el percentatge d'energia solar que volem.

El principal problema d'aquesta metodologia és que, per poder separar el càlcul, tant la introducció de restriccions com la visualització dels resultats s'ha de fer manualment, obrint i editant els fitxers corresponents. Si es vol un càlcul automàtic de principi a fi, s'obtenen moltes solucions que no són d'interès per l'usuari.

El primer objectiu del projecte és que la resolució es pugui fer per nivells, fent participar l'usuari en cada etapa del refinament de les solucions, la qual cosa evita els càlculs inútils i redueix en gran mesura el temps de càlcul necessari.

El segon objectiu és millorar la presentació dels resultats, ja que inicialment no es disposa d'una visualització que permeti comparar fàcilment les diverses microxarxes calculades.

Finalment, el TFG contribueix a fer més fàcil l'ús del programa en aportar un executable instal·lable sobre ordinadors amb el sistema operatiu Windows.

En la memòria es descriu l'anàlisi i disseny de les solucions aportades i es demostra l'abast de les millores a través de diverses proves.

Sumari

Glossari

API: *conjunt de definicions de rutines, eines i protocols per crear programes i aplicacions.*

Metodologia: *procediment de disseny de microxarxes sobre el qual es basa el projecte i s'hi fan les millores.*

Programa: *el software creat durant aquest TFG per millorar la metodologia i afegir-li funcionalitats.*

ILOG*:* *és el software d'optimització utilitzat en aquest projecte, desenvolupat per l'empresa IBM.*

JExcelAPI: *API que permet llegir i escriure dades en fitxers de text des de Java.*

Fitxer multicriteri: *fulla de càlcul emprada perquè el programa interactuï amb l'usuari, on s'escriuen els resultats i s'introdueixen les restriccions.*

1. Introducció

1.1. Origen del projecte

El TFG forma part del projecte d'Energies renovables a Amèrica Llatina, del Centre de Cooperació per al Desenvolupament (CCD). En el marc d'un contracte de pràctiques, es vol millorar l'estat d'un programa que es fa servir en aquest projecte.

1.2. Motivació

L'energia elèctrica s'ha convertit en un pilar fonamental durant l'últim segle, i en els països en vies de desenvolupament serveix per promoure tot tipus d'activitats molt necessàries: potenciar la indústria, poder fer ús d'eines més potents en l'agricultura, millorar les condicions sanitàries o fins i tot ajudar en l'educació mitjançant ordinadors i internet.

Malgrat això, hi ha zones que, ja sigui per la seva geografia o la seva localització, queden aïllades de la xarxa elèctrica i es veuen limitades en recursos i capacitat de prosperar per la manca l'electricitat.

La solució a aquest problema és el disseny de microxarxes, un sistema d'electrificació per a comunitats rurals concretes basat en la generació elèctrica amb energies renovables (solar i eòlica) i la interconnexió dels edificis amb un cablejat degudament regulat per millorar el subministrament, l'autonomia i la seguretat dels sistemes.

Actualment, al CCD disposen d'una metodologia de disseny de microxarxes, desenvolupada per Bruno Domenech Léga en el marc de la seva tesi doctoral dirigida per Laia Ferrer Martí i Rafael Pastor Moreno.

Aquesta metodologia presenta algunes deficiències: calcula totes les solucions sense que es pugui intervenir en la presa de decisions, els resultats no es poden comparar fàcilment perquè s'escriuen en fitxers de text individuals, té un cost computacional elevat, i cal tenir coneixements de programació per introduir les restriccions.

La motivació del projecte és contribuir a la millora d'aquesta metodologia creant un programa que la farà més eficient i pràctica d'utilitzar. Amb això es reduiran els costos del disseny i la instal·lació de les microxarxes, un dels factors decisius en zones de pocs recursos, la qual cosa propiciarà la seva difusió i permetrà millorar la qualitat de vida de la població.

1.3. Objectius del projecte

L'objectiu principal és facilitar la feina als usuaris del programa perquè puguin prendre millors decisions respecte al disseny de microxarxes, gràcies a un disseny del programa més flexible i automàtic que permetrà obtenir solucions més òptimes en menys temps.

Les millores a implementar s'agrupen al voltant de tres objectius específics:

- Reduir el temps de càlcul del procés d'optimització limitant les solucions a explorar, per mitjà de permetre a l'usuari fer una tria de les solucions de més interès i imposar restriccions a mesura que obté els resultats.

- Desar els resultats en un fitxer de dades de manera estructurada, perquè siguin més entenedors per a l'usuari del programa i perdi menys temps fent-ne la comparació.

- Crear un sistema que permeti introduir les dades de partida necessàries de manera fàcil i intuïtiva.

El primer objectiu és la reducció del temps de processament per mitjà de la interacció amb l'usuari, al qual es vol oferir la possibilitat de seleccionar quines opcions vol estudiar abans de començar a fer càlculs, i a partir de quines solucions vol seguir explorant un cop se li presenta l'arbre de solucions.

Després, cal que els resultats de cada solució es mostrin per pantalla amb la major claredat possible, fent servir les fulles de càlcul del programa Excel i valorant la possibilitat d'implementar una interfície gràfica dedicada.

El treball s'enfoca a les persones responsables del disseny d'aquest sistema. Es pretén agilitzar el procés de tria de solucions que acompanya la optimització de les variables més rellevants, aconseguint que es puguin contemplar totes les opcions de manera personalitzada.

1.4. Abast del projecte

- El software s'ha desenvolupat exclusivament per PC amb el sistema operatiu Windows.

- Les interaccions amb les fulles de càlcul estan limitades per les funcions de la API JExcelAPI, que va ser escollida al principi del projecte per fer poder editar, amb el codi de Java, les fulles de càlcul.

- Cal disposar del programa IBM ILOG Cplex Optimization Studio, que és qui s'encarrega d'executar els algorismes d'optimització.

- L'ordinador ha de tenir el JDK de Java instal·lat per poder executar el programa.

- Les variables sobre les quals es permet establir restriccions són: energia consumida pels punts, potència necessària, autonomia del sistema, nombre de microxarxes, nombre màxim d'usuaris per microxarxa, nombre d'usuaris independents, percentatge d'energia solar sobre la total generada, nombre d'equips de generació i percentatge de l'energia que anirà a usuaris independents.

2. Descripció del problema

A continuació es definirà el concepte de microxarxa, i s'estudiarà amb detall la metodologia emprada pel disseny de microxarxes. En el següent apartat es desglossaran les funcions principals de la metodologia, les quals llegeixen i escriuen dades en fitxers. Finalment, un cop vista la seva estructura, s'analitzaran els punts de la implementació que cal millorar.

2.1. Anàlisi de la metodologia

Les microxarxes consisteixen en un cablejat que uneix diversos edificis, juntament amb elements generadors i reguladors de l'energia, perquè aquesta es pugui repartir segons les necessitats. Juntament amb la col·locació de reguladors solars, bateries, inversors i medidors, permeten una certa autonomia energètica a aquestes comunitats, per evitar que hi hagi una manca d'electricitat en moments de més consum o quan no es donen les condicions favorables per generar-ne.

La figura 1 mostra l'esquema de la metodologia proposada, la qual comença quan s'escull una comunitat per electrificar. Primerament, es realitza una avaluació inicial per recollir la informació necessària per al disseny: els punts de consum (cases, centres sanitaris, etc), les seves necessitats energètiques reals I alguns detalls socials que podrien influir en la gestió del Sistema (com ara conflictes entre veïns); el potencial dels recursos eòlics i solars a cada punt de consum; i les característiques tècniques i econòmiques dels equips disponibles a la regió.

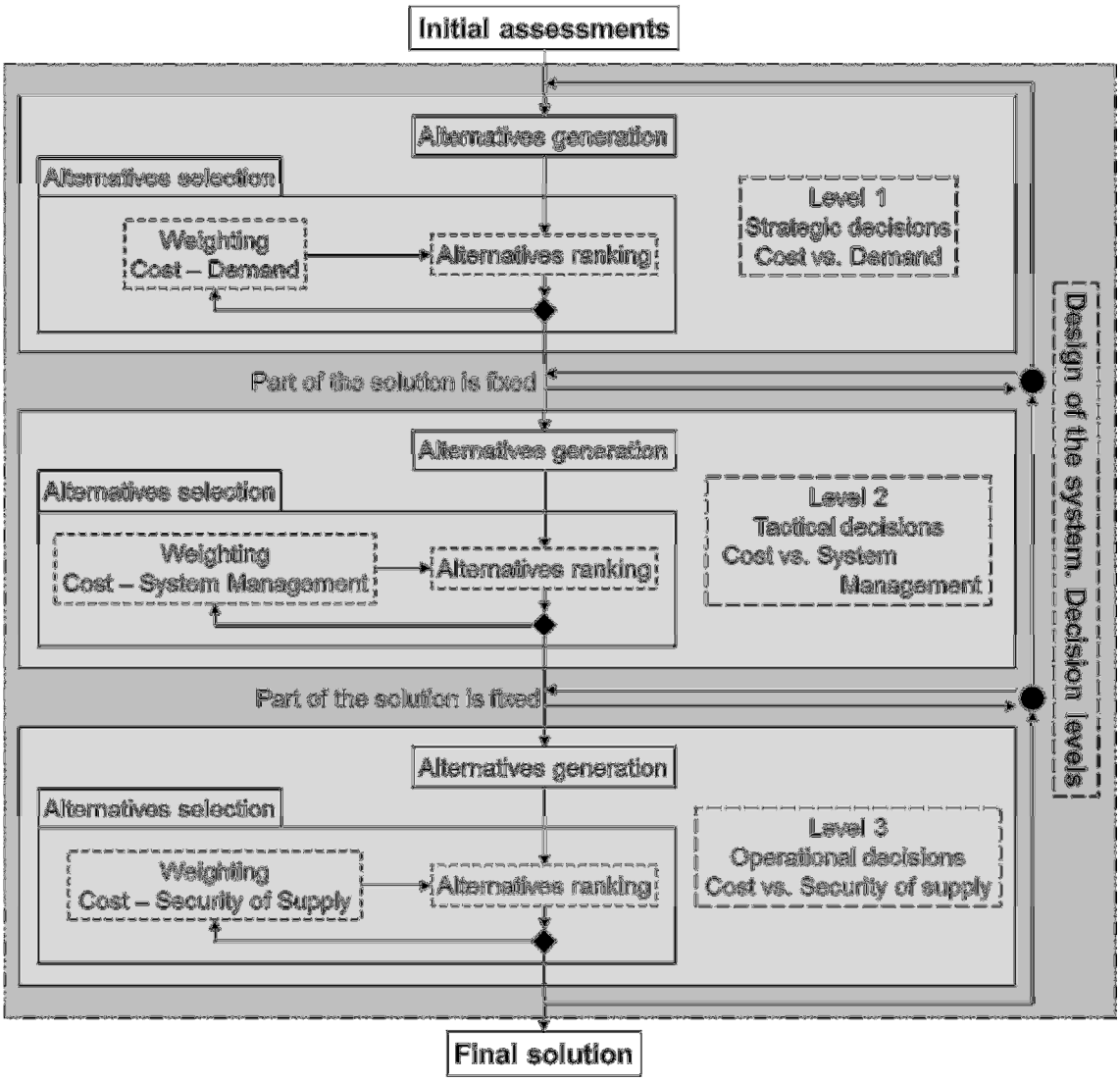

*Fig. 1 Metodologia per al disseny de projectes d'electrificació eòlics i fotovoltaics híbrids autosuficients. Recuperat de [1] "*Methodology for the design of hybrid wind-PV stand-alone electrification projects*" per Bruno Domenech, Laia Ferrer-Martí i Rafael Pastor. Reimprès amb permís.*

El procés de disseny d'una microxarxa s'estructura en tres nivells de decisió. Aquests nivells es classifiquen segons la importància de les decisions preses, de manera que l'usuari del programa pot analitzar i decidir unes característiques abans de considerar-ne d'altres de menor rellevància. En tots els nivells es busca minimitzar el cost de la inversió inicial com a criteri principal, ja que és un limitant important en les zones rurals objectiu del projecte. Els criteris secundaris són: en el primer nivell, la demanda, per garantir que es cobreixen les necessitats energètiques de la població; en el segon, la configuració de la distribució elèctrica, per simplificar la gestió del sistema; i en el tercer nivell, el tipus d'equips de generació que es faran servir, per millorar la seguretat del subministrament.

A la figura 2 es mostra l'esquema de la optimització per nivells. Cal notar que després de cada nivell trobem un valor per al cost que no és res més que la cota mínima que podem assolir amb aquella configuració, que s'incrementarà en nivells successius a mesura que anem especificant altres paràmetres.

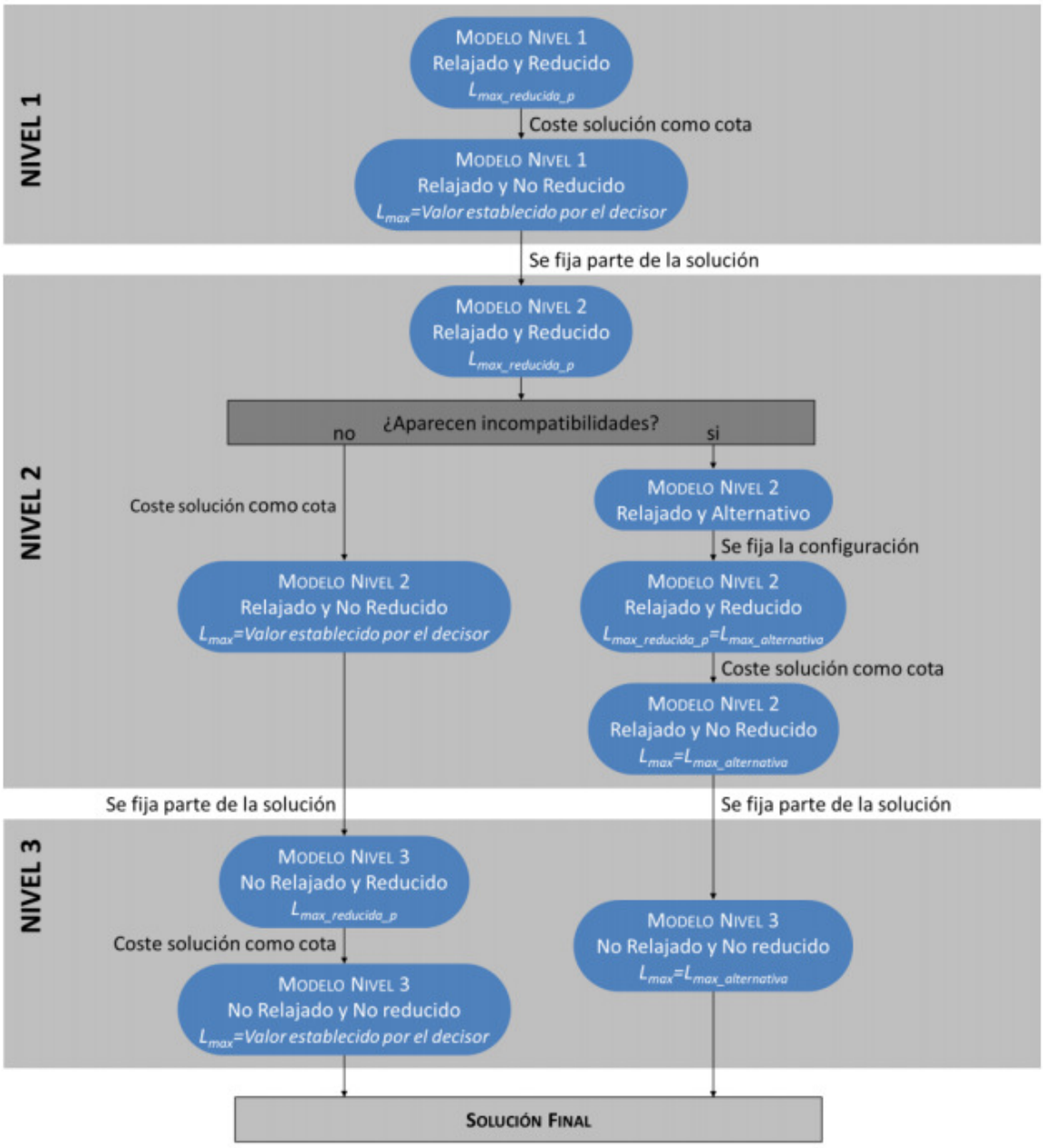

Fig. 2 Diagrama del procés d'optimització. Recuperat de "Metodología para el diseño de sistemas de electrificación autónomos para comunidades rurales" per Bruno Domenech. Reimprès amb permís.

Dins de cada nivell, el programa empra dos algorismes per minimitzar el cost i considerar detalladament la generació, l'acumulació i la distribució elèctrica: el primer permet arribar a una solució acceptable en poc temps, i el segon porta a una solució òptima en la majoria de casos, malgrat que és més lent.

El software permet gestionar els dos models, fent els càlculs amb el primer algorisme per estar segurs de tenir una primera solució, i després utilitzant el model per intentar arribar a una solució millor dins del temps establert per fer càlculs.

Un cop s'obtenen les solucions, se selecciona l'alternativa més adient basada en criteris econòmics, tècnics i socials. Aquesta solució serà el punt de partida per calcular el següent nivell.

2.2. Flux de dades

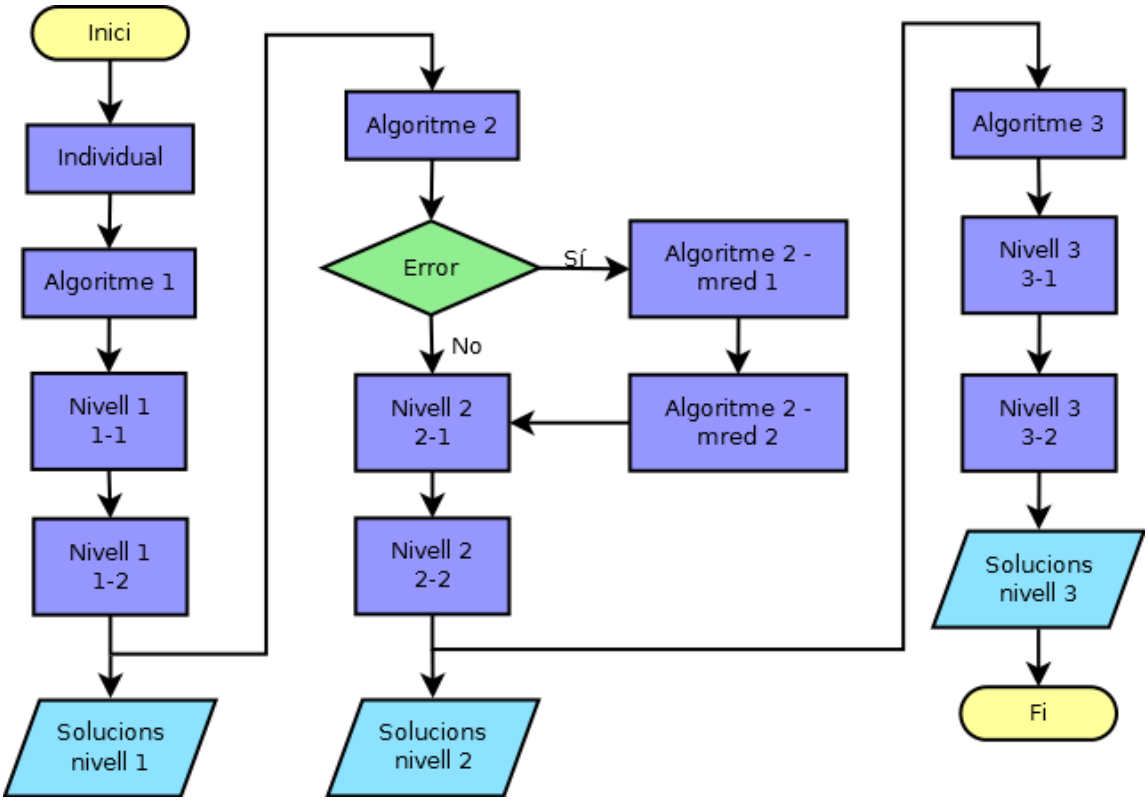

Fig. 3 Diagrama del programa abans de la implementació de les millores.

El treball comença amb una metodologia creada específicament pel disseny de les microxarxes. Implementada en un petit programa, llegeix dades introduïdes en un fitxer de dades (.xlsx) i en altres fitxers (.dat), i se serveix de diverses funcions per introduir aquestes dades al programa IBM ILOG CPLEX. El programa ILOG és l'encarregat d'executar

l'algoritme i el model matemàtic per trobar solucions, que escriu en fitxers de text (.txt). A la figura 3 podem veure un diagrama amb el flux de dades del programa inicial.

El bloc "Individual" és el càlcul de la solució on cap edifici està connectat amb els altres i cadascun té la seva pròpia font de generació. Els blocs "Algoritme" representen l'execució de l'algoritme ràpid, mentre que els blocs "Nivell" són l'execució del model més precís. Els blocs blau cel són els arxius de text amb les solucions que retorna el programa. Finalment, el bloc verd simbolitza una cruïlla en l'execució, que porta cap un camí o l'altre.

També, com es pot observar, l'usuari no té cap paper un cop s'ha iniciat l'execució, ni es pot tornar enrere en el fil del programa automàticament, és a dir, sense haver de fer-ho manualment.

Les solucions dels nivells es creen amb els càlculs de l'ILOG, que és el software d'optimització pròpiament dit, desenvolupat per IBM. És el nucli sobre el qual es basa tot el programa, la funció del qual és preparar els arxius que l'ILOG necessita per funcionar i gestionar la creació dels arxius de text. Tots els arxius que interaccionen amb l'ILOG es mostren a la figura 4, i s'agrupen en una funció anomenada "Ejecutar".

Els resultats estan compostos per quatre fitxers de text. La terminació "_X" representa que depenen del nivell que s'està calculant.

- Arxiu "Solución_X": conté el número de nodes, el cost de la solució, la matriu de connexions dels punts de la xarxa i el temps que s'ha trigat a calcular aquella solució concreta.

- Arxiu "Gráfico_X": s'escriu en un format que pot reconèixer un programa de creació de grafs. Serveix per crear una imatge que representa la posició i la connexió dels punts de consum segons la microxarxa solució.

- Arxiu "Resolución_X": guarda els valors de variables del procés d'optimització. Són internes del programa i no tenen una correspondència amb elements físics del sistema.

- Arxiu "Multicriterio_X": Conté el cost de la solució i el valor que prenen les variables del nivell calculat. Són els valors que després es mostren a l'usuari amb el fitxer multicriteri.

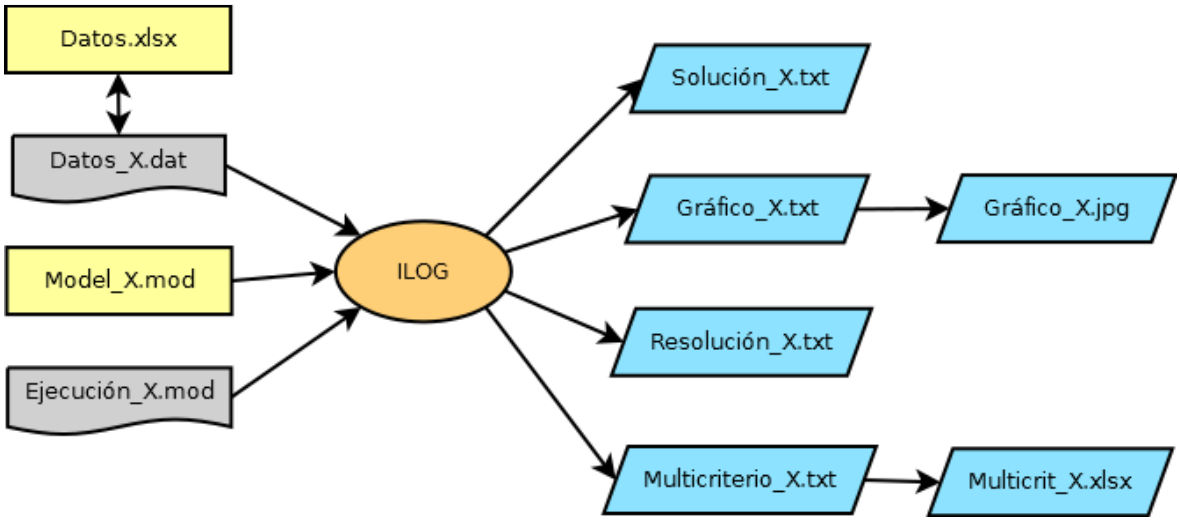

Fig. 4 Diagrama de la funció "Ejecutar", amb els arxius d'entrada i sortida del programa ILOG.

Els arxius que necessita l'ILOG per funcionar són els següents:

- L'arxiu de dades "Datos_X.dat" conté la informació sobre les restriccions de cada nivell, de tots els elements dels qual es composa la microxarxa, i de les característiques de tots els punts de consum. Bona part d'aquesta informació s'agafa d'un arxiu de dades anomenat Datos.xlsx.

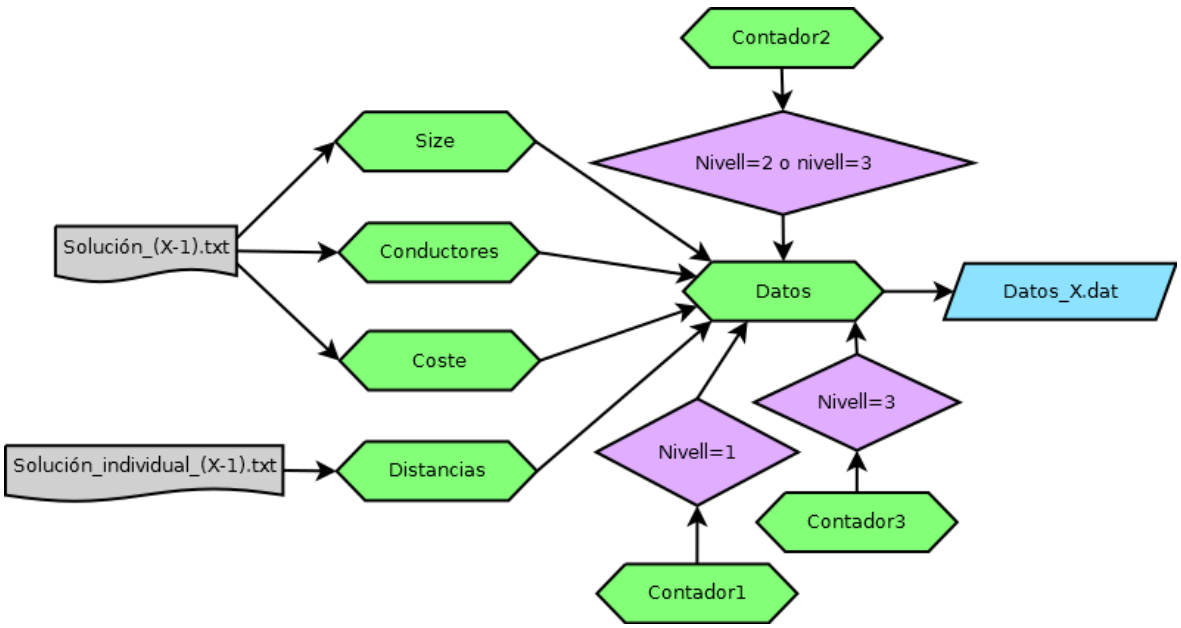

Fig. 5 Diagrama del flux de dades de la funció "Datos" per crear l'arxiu de Dades necessari en tots els nivells.

A la figura 5 podem veure el flux de dades de la funció Datos, encarregada de crear el fitxer. En verd hi ha les funcions del programa, que extreuen dades dels fitxers de text i les envien a la funció central (Size, Conductores, Coste i Distancias), o que fan els càlculs de les restriccions necessàries segons el nivell (Contador1, 2 i 3). En gris hi ha els fitxers de text calculats en el nivell anterior on el la execució individual (en el nivell 1), i en lila hi ha els condicionals. Els arxius acabats en "_X" són dependents del nivell actual, i els "_(X-1)" són els que s'agafen del nivell anterior.

- El "Model_X.mod" és l'algorisme matemàtic que es fa servir en aquell processament concret. Varia segons si és el ràpid o l'altre, i també canvia segons el nivell que s'estigui executant.

- L'arxiu "Ejecución_X.mod" es crea a partir d'una plantilla predeterminada que el programa ja incorpora, i que permet generar els arxius mitjançant les funcions existents. Aquest arxiu es crea amb la funció "Ejecución", que tal i com es mostra a la figura 6, necessita una plantilla a partir de la qual crear els arxius. N'hi ha tres possibles: la de l'execució individual, la de l'algorisme ràpid i la de l'algorisme lent. També fa servir l'arxiu amb el model matemàtic a executar segons el punt del programa en el qual ens trobem; l'arxiu Datos_X del qual s'anirà a buscar les dades un cop el ILOG comenci l'execució; i les dades relatives al nivell en el qual ens trobem i el temps de càlcul que li donem perquè realitzi les operacions.

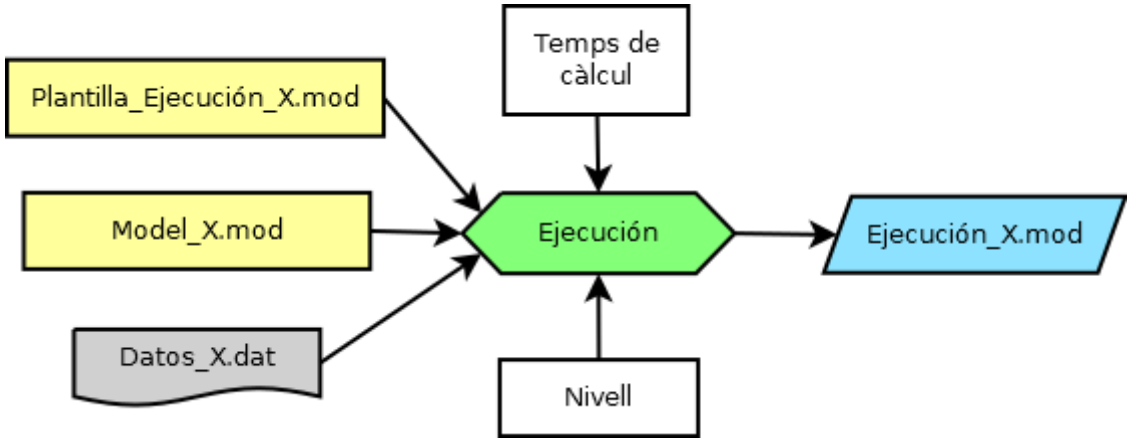

Fig. 6 Flux de dades de la funció "Ejecución".

2.3. Punts a millorar

Els fitxers de dades tenen una estructura per camps, que permet que el ILOG llegeixi les dades corresponents. Aquesta estructura és molt rígida, i cal crear-los manualment. Per tant, l'usuari ha de disposar dels coneixements suficients per escriure el fitxer que s'adapti a les seves necessitats, modificant els camps com li sigui convenient mentre respecti la nomenclatura que fa servir el programa.

A partir d'aquí, el programa s'encarrega de crear els nous fitxers de dades que necessita en els següents càlculs, i l'usuari no ha de tornar a editar cap més fitxer.

A nivell del codi, els tres nivells en els quals s'estructura el programa formen un sol bloc que s'executa conjuntament. Cada nivell s'encarrega de la creació dels fitxers de text amb els resultats mitjançant la execució dels models matemàtics amb el programa ILOG. Alguns d'aquests resultats serveixen per calcular el següent nivell, i la resta per mostrar a l'usuari i guiar-lo en la tria de solucions. Aquesta estructura fa que les solucions es generin en forma d'arbre, malgastant temps en calcular el següent nivell de solucions que potser en nivells superiors ja s'haurien descartat per la seva naturalesa.

Els resultats dels nivells de cada solució es generen en un fitxer de text que es desa a la carpeta corresponent. L'estructura de les carpetes és la següent: una per cada solució del nivell 1, i cadascuna d'aquestes conté una carpeta per cada solució del següent nivell, i així successivament. Cal crear algun mètode per poder veure els resultats sense haver d'obrir els arxius de text un per un per comparar les solucions.

3. Anàlisi i disseny

A l'apartat 3.1 es fa l'anàlisi del programa de partida i dels objectius que es van marcar per millorar-lo, tant els que es van plantejar inicialment com els que han anat sorgint a mesura que ha anat avançant el treball.

A l'apartat 3.2 hi ha els canvis finalment aplicats, amb una explicació detallada de com s'han implementat.

Finalment, a l'apartat 3.3 es descriuen les regles que s'han seguit per implementar les funcions, i a l'apartat 3.4 s'expliquen les millores finals per facilitar l'execució del programa.

3.1. Model de requeriments

3.1.1. Generals

- Es demana que es pugui utilitzar amb Windows, que és el mateix suport del programa de partida.
- Es recomana fer servir el llenguatge de programació Java, per facilitar el manteniment el codi per la persona que va dissenyar la versió inicial.
- Les dades d'entrada s'han de poder introduir amb el format d'una fulla de càlcul, per mantenir la manera de treballar dels usuaris fins ara.

3.1.2. Funcionals

- **Nivells independents**

Com ja s'ha comentat anteriorment, els tres nivells es criden mútuament de manera seqüencial per generar totes les solucions possibles, partint d'unes dades inicials que es coneixen tan bon punt s'inicialitza el programa.

Es vol separar els tres nivells de manera que es puguin cridar independentment, per poder així fer els càlculs de les solucions que interessen a l'usuari sense haver de calcular-les totes.

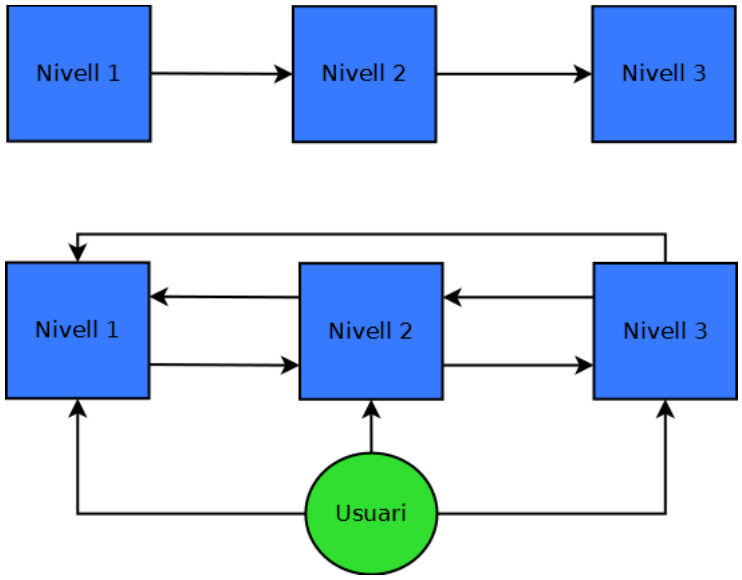

Fig. 7 Flux del programa abans i després de la millora. Les fletxes entre els nivells representen els camins que es poden seguir de manera automàtica.

Gràcies a la millora es vol aconseguir un sistema més versàtil on l'usuari pugui influir en el procés de decisió, tal com s'indica a la figura 7.

- **Selecció de la millor solució per part de l'usuari**

Un cop es tenen els nivells separats es vol aconseguir que, en lloc de generar totes les alternatives del següent nivell per cada una de les solucions, l'usuari s'encarregui d'escollir la solució que li sembli més adient i que només es generin alternatives d'aquesta, la qual cosa redueix enormement el temps de càlcul necessari.

- **Possibilitat de tornar a un nivell anterior**

En cas de trobar que les solucions que s'han trobat en un nivell concret a partir de les restriccions inicials no compleixen les expectatives de l'usuari, es vol fer que es pugui tornar automàticament a nivells anteriors per poder modificar les restriccions entrades. En comparació a aturar l'execució i començar-la de nou, s'aconsegueix estalviar el temps de càlcul de la solució individual així com de les solucions del nivell 1 i, si es vol refer el nivell 3, també del nivell 2.

Llavors, el flux de dades del programa es vol modificar tal com es pot veure a la figura 8, utilitzant la valoració de l'usuari sobre les solucions obtingudes per determinar si es continua amb el següent nivell o és millor tornar enrere per explorar noves opcions.

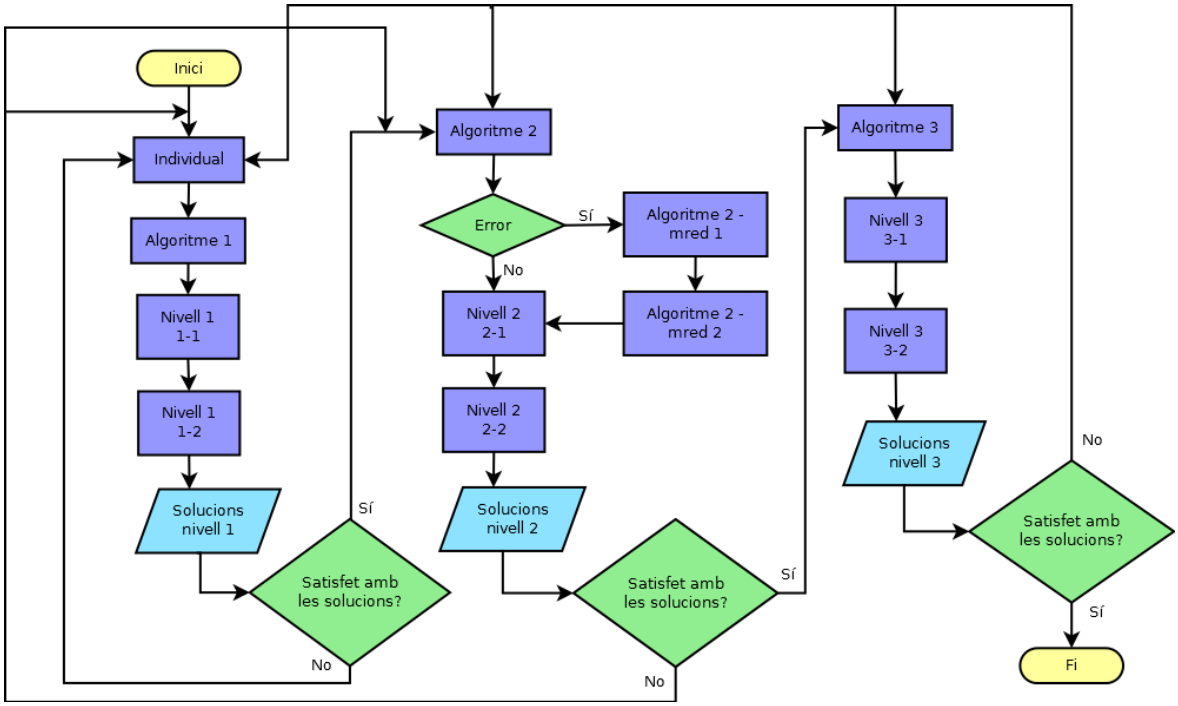

Fig. 8 Diagrama del programa després de la implementació de les millores.

- **Creació d'un fitxer d'estructuració de les dades**

En un principi es volia fer una interfície gràfica basada en una fulla de càlcul ja que els usuaris que feien funcionar el programa inicial ja treballaven en aquest entorn. Així que, per compatibilitat, s'ha decidit mostrar tots els resultats del programa en una fulla de càlcul, que d'ara en endavant anomenarem multicriteri, la qual permet una implementació directa i sense grans canvis en la manera de treballar d'aquestes persones.

Les solucions es mostraran en columnes, en cadascuna de les quals hi haurà 4 blocs de dades amb els resultats de la microxarxa solució:

- El nom de la solució amb les restriccions imposades per l'usuari (figura 9).

Input data	AP-D0	AP-D1
Energy [Wh/day]	280	280
Power [W]	200	200
Autonomy [days]	1	2

Fig. 9 Taula amb les restriccions de les dues primeres solucions del primer nivell.

- Els valors que han pres les variables en aquell cas concret, tractades perquè siguin fàcilment interpretables (figura 10).

Solution data		
Cost [$]	28718	29998
Energy [Wh/day]	282	299
Power [W]	200	200
Autonomy [days]	1	4

Fig. 10 Taula amb els resultats de les dues primeres solucions del primer nivell.

- Els valors que han pres les variables tal i com les dóna el programa, ocultes a l'usuari però ocupant quatre files. S'utilitzen més tard per fer càlculs.
- Tres files de fórmules basades en uns factors de ponderació que l'usuari pot editar, i que permeten que en una quarta fila aparegui una classificació que ordena les solucions de millor a pitjor (figura 11).

Internal calculus		
L1	0,352	0,153
Linf	0,352	0,153
Average	0,352	0,153
Ranking	2	1

Fig. 11 Taula amb els resultats de les fórmules de ponderació i classificació de solucions.

Aquests factors de ponderació permeten donar més pes a unes variables o les altres, a criteri de l'usuari del programa. Per poder-los introduir, s'ha afegit una taula que clarifica on ha d'entrar les dades l'usuari. També hi ha dues columnes que agafen el millor i el pitjor valor de cada paràmetre, d'entre totes les solucions calculades. La taula la podem trobar a la figura 12.

Weights				Ideal	Antiideal
Cost	0,48	Cost	1	28718	32745
Demand	0,52	Energy	0,39	1	1
		Power	0,32	1	1
		Autonomy	0,28	2	1

Fig. 12 Taula per introduir els factors de ponderació, amb la millor i la pitjor solució segons els resultats

En els fitxers de dades dels nivells 2 i 3 també hi ha d'aparèixer els valors que han agafat les variables en les solucions escollides en nivells anteriors.

A més a més, dintre d'aquest fitxer es vol incorporar l'arxiu Datos.xlsx perquè l'usuari només hagi de treballar amb un sol arxiu. S'ha de modificar la ruta que segueix l'ILOG per obtenir

les dades.

- **Creació d'alternatives a partir de restriccions en els nivells 2 i 3**

En el programa inicial, les restriccions a cada nivell eren limitades i s'introduïen directament al codi abans d'executar el programa, limitant en gran mesura la capacitat d'analitzar les opcions que ofereixen diferents configuracions de les variables.

Per això, l'usuari ha de poder introduir a cada nivell les restriccions que es tindran en compte durant el càlcul. Per fer-ho, caldrà poder llegir les dades que s'han entrat en el fitxer multicriteri amb els resultats, amb la qual interacciona l'usuari.

Utilitzant aquestes dades, a cada nivell s'han de generar totes les combinacions de restriccions a partir de diferents valors de cada variable proposats per l'usuari. Abans dels canvis, la metodologia fa la combinatòria de valors amb bucles encadenats, tal com es pot veure a l'annex 1.

- **Representació dels resultats en un graf**

El programa del qual es parteix genera uns fitxers especialment preparats per mostrar un graf amb els resultats de la solució mitjançant un programa anomenat Graphviz. El que es demana és que el programa generi el graf per cada solució calculada i el mostri a l'usuari en el fitxer multicriteri.

- **Creació d'alternatives a partir de restriccions en el nivell 1**

En aquest cas hi ha una diferència amb els nivells 2 i 3: en comptes de proposar els valors que l'usuari vol comprovar en el procés d'optimització (nombres generals aplicables a elements del disseny de la microxarxa), aquest cop s'introdueix per una banda un valor per cada punt de consum de la microxarxa, i per l'altra banda s'introdueixen increments percentuals comuns per tots els punts.

Per exemple: els valors de la primera variable pels tres primers punts és 1, 2 i 3; els de la segona, 6, 7 i 8; i un únic valor de 5 per la tercera (igual a tots els punts, sempre és així). Llavors, l'usuari defineix increments del 20% i del 40% per la primera variable, no escriu res per la segona, i afegeix un increment del 10% en el valor de la tercera. Així doncs, es fa la combinatòria d'aquests percentatges i queden finalment 6 solucions: [0,0,0], [20,0,0], [40,0,0], [0,0,10], [20,0,10], [40,0,10], expressat amb increments percentuals. Per fer els càlculs s'agafarà els valors de cada punt i se'ls aplicarà l'increment corresponent.

- **Generació dels fitxers .dat del nivell 1**

Un dels impediments de cara a la utilització del programa per part d'una persona no experta

en el seu funcionament era la creació dels fitxers de dades necessaris en el primer nivell per poder fer funcionar el ILOG. La principal causa d'això era que, inicialment, les alternatives del primer nivell es generaven partint d'uns arxius de dades que l'usuari s'havia de preocupar de crear manualment i introduir a la carpeta del programa. Cadascun d'aquests arxius corresponia a una solució del primer nivell.

Seria ideal tenir una funció que s'encarregués de crear aquests arxius, a partir d'una plantilla igual per totes les execucions. L'usuari ha de poder introduir les dades dels punts de consum en una fulla de càlcul i que els arxius es creïn agafant les dades directament d'allà.

- **Facilitar l'execució del codi**

Atès que l'usuari no necessàriament té un entorn de desenvolupament que permeti executar el codi durant el procés de creació, seria desitjable que es pogués cridar el programa des d'un altre lloc del sistema, i preferiblement des de la mateixa fulla de càlcul on es mostren els resultats.

Cal crear un botó o un enllaç que cridi un executable del programa per iniciar l'execució des del fitxer multicriteri quan l'usuari el premi.

- **Millorar la compatibilitat entre idiomes en l'entrada de nombres**

Un problema que ha aparegut durant el desenvolupament és que en la versió anglesa de l'Excel la separació decimal es representa amb punts i la dels milers amb comes (1,423.38), mentre que en castellà la separació es fa a l'inrevés (1.423,38).

Cal unificar els dos mètodes d'escriptura per tal que les dades es mostrin correctament segons l'idioma del sistema de l'usuari. Això permetrà que el programa es pugui utilitzar des d'ordinadors els quals no tenen l'espanyol com el seu idioma predeterminat.

- **Creació d'una interfície gràfica**

La fulla de càlcul és una opció pràctica, però no la més adequada de cara a fer públic el projecte perquè d'altres el puguin aprofitar. Per això s'ha proposat crear una interfície dedicada, tant per aconseguir un acabat més polit en la mostra dels resultats com per facilitar la entrada de dades per part de l'usuari, que no haurà de conèixer tant a fons els camps del full de càlcul i tindrà més ajuda sobre les dades que es demanen a cada casella.

3.2. Disseny

- **Nivells independents**

A nivell de codi, la solució parteix de separar els tres bucles que constitueixen cada nivell. Es creen funcions independents per cada un d'ells i es reestructura la crida de cadascuna d'elles per mantenir la cadena de nivells.

En convertir-se en funcions separades, cal passar com a paràmetres les dades que cada nivell necessita:

- En el cas del nivell 1, es necessita tenir el vector amb els noms de cada solució, el qual es calcula en una funció prèvia anomenada "contador1", i el fitxer multicriteri en el qual s'escriuran les solucions del nivell.

- En el nivell 2, calen els noms de les solucions dels nivells 1 i 2, el fitxer multicriteri on s'han escrit les solucions del nivell 1 i el número de la solució escollida en el nivell previ.

- En el nivell 3 cal el mateix que a l'anterior canviant el fitxer multicriteri pel que es calcula durant el nivell 2, i s'afegeix el vector amb els noms de les solucions del nivell 3 i la solució escollida durant el nivell 2.

- **Selecció de la millor solució per part de l'usuari**

Primerament, es crea una finestra amb el títol desitjat i unes mides acceptables perquè sigui fàcil d'utilitzar per l'usuari. En aquesta finestra s'hi introdueix un JPanel (panell on afegir-hi altres objectes) i se li afegeix un Layout vertical (això és, que els objectes afegits ocupin tot l'espai horitzontal i s'afegeixin per ordre de dalt a baix). Llavors, es creen els botons numerats generats automàticament en un vector i s'afegeixen un per un al JPanel, perquè l'usuari pugui clicar el que correspon a la solució que desitja. Podem trobar un exemple de panell amb botons a la figura 13.

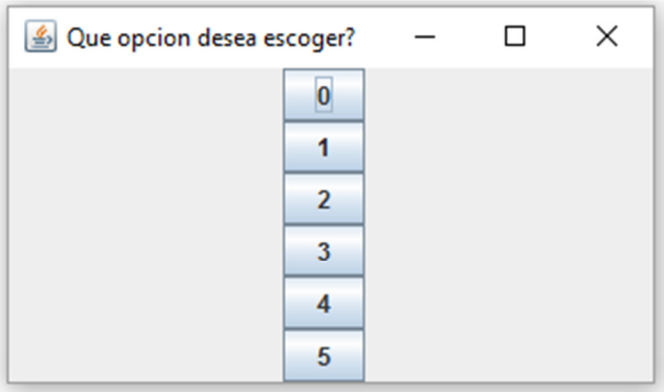

Fig. 13 Exemple de quadre de selecció per un nivell amb 6 solucions.

Un cop fet això, cal establir la connexió entre aquests botons i la crida al següent nivell amb les dades de la solució per seguir amb el procés d'optimització. Per saber quin botó ha clicat l'usuari, es recorre el vector de botons i es busca l'índex del botó en el qual s'ha generat l'acció de clicar.

Addicionalment, per facilitar la feina a l'usuari, es fa que s'obri automàticament el nou fitxer multicriteri amb els resultats que s'han calculat durant el nivell, de tal manera que no calgui buscar-lo dintre la carpeta, i amb el benefici afegit que es fa més visual el moment en què el programa acaba de fer els càlculs.

- **Possibilitat de tornar a un nivell anterior**

En el nivell 2 s'ha creat un botó extra en el panell de l'apartat superior, que en clicar-lo obre el fitxer multicriteri on s'introdueixen les restriccions per al nivell 2, i que crida la funció de generar les alternatives del nivell 2 un cop l'usuari ha introduït les dades.

En el nivell 3 s'han creat dos botons: un botó igual que el del cas anterior i un altre botó que obre la fulla de càlcul del nivell 3 per introduir les restriccions d'aquest nivell, i que crida la funció de generar les alternatives corresponents també després que l'usuari hagi introduït les dades.

Aquests botons es creen un cop s'han generat els botons per seleccionar cadascuna de les solucions. Segons el nivell i la posició del botó dins del panell, el programa detecta si s'ha premut un botó de selecció o un dels botons de tornar a un altre nivell, i s'executen accions diferents per cadascun.

Dins del bucle per recórrer els botons esmentat anteriorment, si el botó clicat és l'últim o el penúltim (en els nivells 2 i 3, respectivament) s'executa el codi per tornar a obrir l'arxiu de dades per introduir les restriccions del nivell 2, i si el botó clicat és l'últim (en el nivell 3) s'executa el codi per poder introduir les restriccions del nivell 3.

- **Creació d'un fitxer d'estructuració de les dades (fitxer multicriteri)**

Per poder treballar amb fulles de càlcul des de Java s'ha fet servir una API anomenada JExcelApi, que permet llegir i escriure dades en fitxers .xls (el format antic dels programes tipus Excel).

Es parteix d'un model dissenyat per un dels futurs usuaris sobre com s'haurien de disposar els elements que hi ha d'aparèixer, i a partir d'aquí s'escriuen diverses funcions per reproduir-lo. Aquestes funcions s'encarreguen de:

- Escriure, per cada solució, el cost de la instal·lació i el valor de les tres variables característiques del nivell que resulten del càlcul, emmagatzemats al fitxer de text "Multicrit".

- Escriure, per cada solució, el valor teòric de les tres variables que correspon a la combinació de les restriccions introduïdes per l'usuari. En el cas de que una restricció estigui no activa, s'escriurà el valor "Libre".

- Llegir les restriccions introduïdes per l'usuari de cada variable, en forma de columna.

- Escriure el valor que prenen les variables de la solució escollida en els nivells anteriors.

S'ha afegit l'arxiu "Datos.xlsx" com la quarta fulla d'aquest arxiu de dades, i s'ha modificat la ruta que fa servir l'ILOG per anar a llegir les dades.

Un problema a l'hora d'escriure els resultats al fitxer multicriteri és la puntuació que es fa servir pel sistema decimal: depenent de l'idioma de l'ordinador de l'usuari, les comes s'interpreten com el separador decimal o com la marca dels milers, així com passa amb els punts.

La plantilla conté també funcions d'Excel que serveixen per acabar d'adequar els resultats del programa a valors fàcilment interpretables per l'usuari. A més a més, disposa d'un sistema de classificació de les solucions de millor a pitjor basat en uns factors de ponderació que l'usuari pot modificar des de la fulla de càlcul directament.

- **Creació d'alternatives a partir de restriccions en els nivells 2 i 3**

La base de la generació de solucions és fer la combinatòria dels valors que introdueix l'usuari abans dels nivells 2 i 3. S'ha fet una exploració de les eines que hi havia disponibles per fer combinatòries i totes requereixen de coneixements avançats de programació en Java o costa saber com aplicar-les pel cas concret amb el qual ens trobem (3 o 4 vectors d'elements, dels quals volem totes les possibles combinacions agafant només un element

de cada vector, mantenint l'ordre). Finalment, s'ha pres la decisió d'escriure funcions pròpies per fer la combinatòria, personalitzades per cada nivell.

En aquests dos nivells, la manera que té l'ILOG de rebre les restriccions és per mitjà d'un vector de 4 variables en el nivell 2 i de 3 variables en el nivell 3, acompanyat d'un vector d'uns i zeros per seleccionar quina restricció està activa (1) i quina no ho està (0) a cada solució (vector d'activació).

L'usuari disposa d'una columna per cada variable per introduir-hi les restriccions, sempre començant per la fila superior i escrivint-les consecutivament, ja que el programa les llegeix a partir d'una cel·la i anant baixant fins que troba una cel·la buida. En començar a fer la combinatòria, aquests valors es guarden en un vector de Strings.

Cal comptar quants d'aquests vectors estan buits per no haver de recorre'ls, la qual cosa podria donar problemes si es fes. Separem els possibles casos en 8: tots els vectors buits; 1 d'ells amb alguna dada i la resta buit (3 casos segons quin sigui); 2 d'ells amb alguna dada i l'altre buit (3 casos segons quin sigui); o tots ells amb alguna dada. També es fa el càlcul de quantes solucions hi haurà finalment per poder crear el vector que s'anirà omplint amb les combinacions de variables.

A partir d'aquí se separa el càlcul en dos blocs, segons si ens trobem en un dels primers 4 casos descrits abans o en un dels 4 últims, per decidir si cal fer la combinatòria amb bucles un dins de l'altre o és factible fer-la amb un de sol.

Aquestes funcions combinatòries estan dissenyades per no generar mai dues solucions equivalents. Per aconseguir-ho, no n'hi ha prou amb fer totes les combinacions possibles del vector d'activació per cada combinació de restriccions. Exemple: Amb dues restriccions per la primera variable, com ara 2 i 3, i cap restricció en les altres dues (valor per defecte 0), aquestes dues solucions serien equivalents (figura 14).

	Solució 1			**Solució 2**		
Valor de les restriccions	**2**	**0**	**0**	**3**	**0**	**0**
Activació	**No**	**Sí**	**Sí**	**No**	**Sí**	**Sí**

Fig.14 Exemple de dues combinacions que són interpretades com a iguals pel programa.

Per tant, la manera d'abordar aquest problema ha estat fer la combinatòria del vector d'activació agafant la primera restricció de cada variable, i anar iterant tots els valors de les restriccions, eliminant les combinacions que resultarien en una solució repetida.

Començarem explicant el nivell 3, que té tres vectors de restriccions. El nivell 2 es calcula exactament igual que el 3, excepte que els bucles han de recórrer un vector més (compost d'un element, sempre el mateix valor) i, per tant, tots ells fan el doble d'iteracions, en tots els casos.

Així doncs, el càlcul s'inicia agafant el primer valor de tots els vectors de restriccions que estiguin plens, i posant un 0 a la resta. Segons els diferents casos, el procediment és diferent:

- Cas 0 (cap restricció en cap variable)

És el cas trivial, on l'usuari no imposa cap restricció. Només hi ha una solució, i és la que no restringeix cap variable.

En el nivell 2, donat que l'últim vector sempre conté un element, hi haurà dues solucions al final: la solució on no s'imposa cap restricció, i la solució en què s'imposa la restricció d'aquest quart vector.

- Casos 1, 2 i 3 (alguna restricció en una sola variable)

Són casos simples, en el sentit que només cal iterar sobre una variable. S'agafa el valor de la primera restricció i es crea la solució lliure, una solució amb la variable restringida per cada valor que l'usuari ha introduït.

En el nivell 2 hi ha dues solucions per cada restricció imposada per l'usuari, amb i sense restringir el 4t vector.

- Casos 4, 5 i 6 (alguna restricció en dues variables)

En aquests casos ja hem de començar a fer combinacions amb els valors introduïts per generar totes les solucions. S'agafa el primer valor de les dues restriccions i un valor prefixat 0 per la variable lliure, i es calculen les 4 solucions que surten de fer les combinacions d'activació o desactivació de les restriccions. Després, si hi ha més d'un valor en alguna restricció, es van combinant les variables afegint 2 noves solucions per cada nova combinació de restriccions (les altres 2 solucions que falten són solucions repetides).

En el cas d'estar calculant nivell 2 es crearien 8 solucions inicialment i per cada nova combinació se n'afegirien 4 més.

- Cas 7 (alguna restricció en totes tres variables)

S'agafa el primer valor de les tres restriccions i es calculen les 8 solucions que surten de fer les combinacions d'activació o desactivació de les restriccions. Després, si hi ha més d'un

valor en alguna restricció, es van combinant les variables afegint 4 noves solucions per cada nova combinació de restriccions (les altres 4 solucions que falten són solucions repetides).

En el cas d'estar calculant nivell 2 es traurien 16 solucions només començar i se n'afegirien 8 per cada nova combinació.

- **Creació d'alternatives a partir de restriccions en el nivell 1**

En disposar ja de la funció que calcula la combinatòria de tres vectors, s'ha copiat la funció del nivell 3 i s'ha modificat per guardar els resultats en vectors dedicats únicament al nivell 1 i per eliminar tota la part de restriccions actives o no actives que no era necessària aquí.

- **Generació dels fitxers .dat del nivell 1**

Primerament, ha fet falta una plantilla basada en els fitxers .dat que es feien servir fins llavors. Aquests fitxers consisteixen en diverses línies que llegeix el ILOG que li indiquen la variable que ha de llegir i en quin camp del fitxer Excel les ha d'anar a buscar. L'única diferència entre un fitxer i un altre es troba en tres línies que representen els requeriments d'energia, de potència i d'autonomia de cada punt, encara que l'autonomia és per tots els punts la mateixa i, per tant, només cal escriure un valor.

Tenint en compte això, es crea una funció dedicada a l'escriptura d'aquests fitxers que funciona de la següent manera: es copien les línies prèvies a aquestes tres línies, s'escriuen els dos vectors i el valor d'autonomia, i es copien la resta de línies de la plantilla. Per cada punt s'ha d'escriure el seu valor d'energia, que es llegeix del fitxer Excel, i al qual se li aplica l'augment/reducció que correspongui segons la combinatòria de l'apartat anterior, i posteriorment s'escriu al fitxer .dat d'aquella alternativa. Es fa el mateix pel vector de potència, i per l'autonomia només cal llegir l'únic valor i aplicar-li la modificació corresponent.

- **Representació dels resultats en un graf**

S'ha creat una funció que crida el programa Graphviz, passant com a paràmetres el nom del fitxer que conté les dades per crear el graf, el camí relatiu fins el fitxer i la iteració en la qual ens trobem. Aquesta funció retorna una imatge .jpg amb el graf com la de la figura 15, que queda desada a una carpeta anomenada "graficos" junt amb totes les altres imatges.

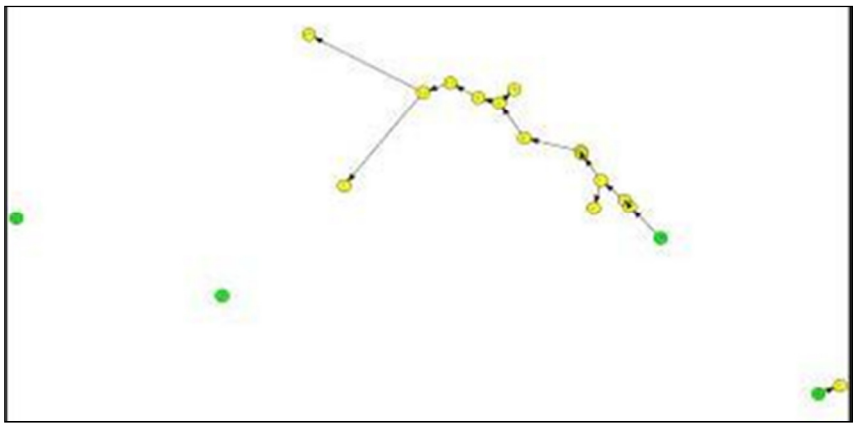

Fig. 15 Detall del graf per una solució concreta.

Els punts representen els edificis que cal abastir d'energia, i les fletxes són el cablejat que conforma la microxarxa.

Cada cop que una imatge es guarda a la carpeta, s'agafa i es col·loca centrada en una cel·la del fitxer multicriteri, just a sota de la solució corresponent, per facilitar així la visualització dels resultats. També es modifica la seva mida perquè encaixi exactament amb la mida de la cel·la, i les cel·les s'han dissenyat prou amples perquè les imatges siguin entenedores mentre conserven una mida que permet veure diverses solucions alhora sense haver de desplaçar la pantalla lateralment.

En el procés de col·locació de les imatges, també s'ha creat una funció que els hi posa un marge al voltant de color negre, la qual cosa permet distingir on acaba una i comença la següent, donat que estan tocant-se unes amb les altres per aprofitar l'espai. Això s'aconsegueix creant un rectangle de color negre centrat en la imatge i de mida una mica superior, el qual s'enganxa a la imatge per darrera, de manera que fa l'efecte de que la imatge tingui un marc.

- **Millorar la compatibilitat entre idiomes en l'entrada de nombres**

S'ha emprat la funció getLocale() de WorkbookSettings del JExcelAPI, que permet obtenir el local de la fulla en la qual estem treballant, la qual permet obtenir l'idioma del sistema de l'usuari i per extensió que els nombres que s'escriuen a les cel·les quedin formatats segons aquest criteri. Llavors, en la creació del primer Workbook, se li passa com a paràmetre aquest WorkbookSettings, i així els Workbooks que es creen a partir d'aquest tindran tots aquest local.

3.3. Implementació

En aquest apartat es descriuen les regles generals que s'han tingut en compte a l'hora d'elaborar el codi.

- **Divisió del codi en subfuncions**

Atenent a l'objectiu de clarificar el codi escrit, les funcions més llargues s'haurien de dividir en funcions més curtes que permetin aprofitar codi i alhora separar funcionalitats. Això permet detectar errors més ràpidament i estructura el programa de manera més clara.

L'exemple més clar d'aquesta divisió és la separació dels tres nivells en execucions independents, la qual cosa ha permès tirar endavant i endarrere en l'execució del programa.

També s'ha aplicat aquesta regla en el càlcul de les combinatòries, ja que s'ha diferenciat la part de preparació per al càlcul de saber en quin cas ens trobem, els 4 primers casos que comparteixen una estructura similar i els 4 últims, que també s'implementen de manera semblant. Cada part disposa de la seva pròpia funció.

- **Definir com a privades les funcions que ho puguin ser**

Totes les funcions s'han definit com a públiques durant la implementació de noves funcions per no tenir problemes d'accés entre classes, però idealment les funcions que només es criden dins d'una mateixa classe haurien de ser privades per evitar interaccions no desitjades.

- **Definir una nomenclatura única per les variables, classes, funcions i mètodes**

Per garantir la comprensió del codi el més adient és nombrar-ho tot seguint una mateixa norma, perquè no hi hagi dubtes de a què s'està referint quan apareix un nom en alguna part del codi. Tant és així que caldrà especificar una nomenclatura particular per les variables, una altra per les classes i una diferent per les funcions, com per exemple que totes les variables s'escriguin en minúscules, o que els noms de les classes estiguin separats pel caràcter '_'. Ex: variableprimera, objecte_principal.

S'ha decidit anomenar les variables seguint l'estil de nomenclatura de Java recomanat per Google [3].

Segons aquesta guia, els noms de les classes s'han d'escriure amb el que s'anomena UpperCamelCase, que és l'escriptura amb la primera lletra de cada paraula amb majúscula. Els noms dels mètodes, funcions i variables s'escriuen tots en lowerCamelCase, que és igual que el UpperCamelCase però amb la primera lletra minúscula.

- **Documentar les funcions**

Escriure quin és el propòsit de cada funció facilita la comprensió del software en general, i ajuda a no perdre el fil del que s'intenta aconseguir en particular. L'objectiu final és tenir totes les funcions explicades a les respectives capçaleres.

S'ha fet servir la convenció de documentació recomanada per Oracle [4]. S'explica el propòsit de la funció, què és cada paràmetre d'entrada i el que retorna. Si es fa seguint el sistema que s'il·lustra a la figura 16, es crea la documentació de Java, això és, les dades introduïdes es poden veure quan es posa el ratolí sobre el nom de la funció allà on es crida.

```
/**
 * Creates the .dat files needed in the first level, reading information of
 * the Excel "Datos".
 *
 * @param archivo  the name of the template to use when creating the .dat files
 * @param iteration  the number of the solution
 */
private static void writeDat(String archivo, int iteration) {
```

Fig. 16 Exemple de funció documentada.

Totes les funcions han estat documentades, amb els paràmetres d'entrada referenciats juntament amb els valors de retorn.

3.4. Millores d'implementació

- **Camins relatius**

Per poder trobar els arxius sense saber el directori on l'usuari guardarà la carpeta del projecte, es fa servir la funció System.getProperty("user.dir"), que permet obtenir el nom absolut del directori des del qual s'executa el programa (en aquest cas, la carpeta "MicrogridOptimizer"). Aquest nom es guarda en una variable global anomenada "path", i s'utilitza cada cop que es crida un arxiu dins del programa, ja que és el punt de partida a partir del qual s'estructura l'arbre de fitxers.

- **Execució del codi directament des del fitxer amb els resultats**

Donat que es volia facilitar l'ús del programa fora de l'entorn de desenvolupament, s'ha compilat el projecte en un fitxer .jar que es pot cridar des de la consola de Windows i mostra per pantalla el procés d'execució. Mitjançant una opció de l'entorn de desenvolupament, aquest arxiu es crea automàticament incloent totes les llibreries de les quals depèn el projecte, en aquest cas la API JExcel.

Un cop s'ha obtingut el fitxer .jar en la compilació del projecte, es copia i es desa a la seva carpeta principal, perquè d'aquesta manera, quan s'executa el programa des de la línia de comandes, la carpeta principal del projecte esdevé la que conté el .jar, i tots els fitxers es criden relativament respecte aquesta.

Després, emprant un programa anomenat JSmooth, la funció del qual és transformar un fitxer .jar en un fitxer executable de Windows .exe, es crea aquest arxiu, que també es copia a la carpeta del projecte. Amb això, qualsevol usuari que executi l'arxiu podrà fer córrer el programa des de la mateixa carpeta.

Per anar un pas més enllà, es crea en el fitxer multicriteri una forma rectangular amb el text “Start”, que serà el botó des del qual volem executar el programa en fer-hi clic. Finalment, creem un hipervincle des del botó, que l’enllaça amb l’arxiu executable que hem guardat prèviament dins la carpeta del projecte.

- **Creació d’un fitxer ReadMe per explicar el funcionament del programa**

S’han escrit els passos a seguir per executar el software en un fitxer de text, per facilitar als nous usuaris l’aprenentatge del programa. En aquest fitxer es descriuen les accions a realitzar en cada moment per la seva correcta execució, des del moment de l’arrencada fins a l’obtenció dels resultats.

4. Validació dels resultats

És vital que, un cop s'han implementat totes les millores, hi hagi un procés de validació dels objectius establerts en un principi. En aquest apartat es descriuen els mètodes emprats i els resultats obtinguts.

4.1. Taula resum de les millores programades

A la següent pàgina hi ha una taula amb una breu explicació dels objectius funcionals vistos anteriorment i de les millores que s'han implementat per complir-los (figura 17).

4.2. Reducció del temps de processament

Degut a que no s'ha actuat directament sobre el procés purament d'optimització, no té sentit mesurar el temps d'execució del programa, malgrat que sí que podem fer un càlcul de l'abast teòric de la millora en un exemple concret:

Suposant només una restricció per cada variable en cadascun dels nivells:

$$Solucions\ nivell\ 1: \quad 2 * 2 * 2 = 8$$

$$Solucions\ nivell\ 2: \quad 2 * 2 * 2 * 2 = 16$$

$$Solucions\ nivell\ 3: \quad 2 * 2 * 2 = 8$$

$$Nombre\ total\ de\ solucions\ calculades\ en\ l'execució\ automàtica\ inicial:$$

$$8 * 16 * 8 = 1024$$

$$Nombre\ total\ de\ solucions\ calculades\ després\ d'implementar\ la\ tria\ de\ solucions:$$

$$8 + 16 + 8 = 32$$

En els tres nivells tenim per cadascuna de les tres variables dos casos: la restricció és activa o no ho és. El segon nivell té una variable extra que, tal com s'ha comentat anteriorment, sempre té dos valors: activa o inactiva.

Tal com es pot veure, i agafant de mitjana un segon per solució calculada, s'ha passat d'una execució de 20 minuts a una de 30 segons. La diferència augmenta exponencialment com més restriccions imposi l'usuari.

Objectius funcionals	Estat de la seva implementació en finalitzar el projecte
Nivells independents	Els nivells estan separats i es poden cridar independentment.
Selecció de la millor solució	L'usuari pot escollir la solució que li sembla més adequada a través d'un panell amb botons, i el següent nivell s'executa partint d'aquesta.
Possibilitat de tornar a un nivell anterior	Si a l'usuari no li agraden les solucions calculades en un nivell, pot tornar un o dos nivells enrere automàticament clicant un botó que apareix al mateix panell de triar la solució.
Fitxer d'estructuració de les dades	A mesura que es calculen els nivells es va generant un fitxer multicriteri que conté els resultats del programa, i en el qual s'introdueixen les restriccions a tenir en compte en el següent nivell.
Creació d'alternatives a partir de restriccions en els nivells 2 i 3	Amb les restriccions introduïdes per l'usuari en el fitxer multicriteri es generen totes les solucions possibles fent la combinatòria de les restriccions, sense que n'hi hagi cap de repetida.
Representació dels resultats en un graf	Per cada solució es genera una imatge amb un graf que representa els punts de consum i les unions entre ells, i s'afegeix al fitxer multicriteri.
Creació d'alternatives a partir de restriccions en el nivell 1	A partir de les restriccions escrites per l'usuari en el fitxer de dades inicial es generen totes les solucions possibles fent la combinatòria de les restriccions, sense que n'hi hagi cap de repetida.

Generació dels fitxers .dat del nivell 1	Ja no cal escriure els fitxers de dades del nivell 1 manualment, sinó que es generen automàticament a partir d'una plantilla.
Facilitar l'execució del codi	El programa es pot cridar des d'un entorn de desenvolupament, des de la consola de Windows, fent doble clic sobre l'executable que hi ha dins la carpeta o prement el botó que hi ha dintre de l'arxiu de dades.
Millorar la compatibilitat entre idiomes en l'entrada de nombres	El programa detecta automàticament l'idioma de l'ordinador de l'usuari i adapta els separadors decimals segons això perquè no hi hagi problemes d'escriptura.
Creació d'una interfície gràfica	S'ha dissenyat la interfície a un nivell bàsic tant visualment com funcionalment.

Fig. 17 Taula comparativa de les funcions a implementar en un principi i del seu estat al final del projecte.

5. Continuació del treball

En aquest apartat es descriuen les funcionalitats que es podrien afegir al programa en el cas que es vulgui seguir amb el projecte de millora del software dissenyat. Algunes s'han comentat a l'apartat d'implementació i d'altres han sorgit durant la realització del treball però no s'han pogut dur a terme per falta de temps.

5.1. Implementació d'una interfície gràfica dedicada

Durant el treball s'ha plantejat la creació d'una interfície que permeti a l'usuari introduir les dades del programa de manera més pràctica i senzilla, on tota la execució sigui dinàmica i compacta, i que sigui intuïtiva d'utilitzar. Per manca de temps no s'ha pogut realitzar aquesta tasca, que requereix d'una gran planificació i estructura, però sí que s'ha dissenyat una proposta d'interfície que permetria realitzar totes les funcions que incorpora ara per ara el programa i alhora afegir noves característiques.

Aquesta interfícies està dividida en 7 pestanyes: 3 pestanyes d'introducció de fitxers de dades i restriccions dels nivells, 3 de visualització dels resultats i una pestanya final on es mostrin les solucions escollides.

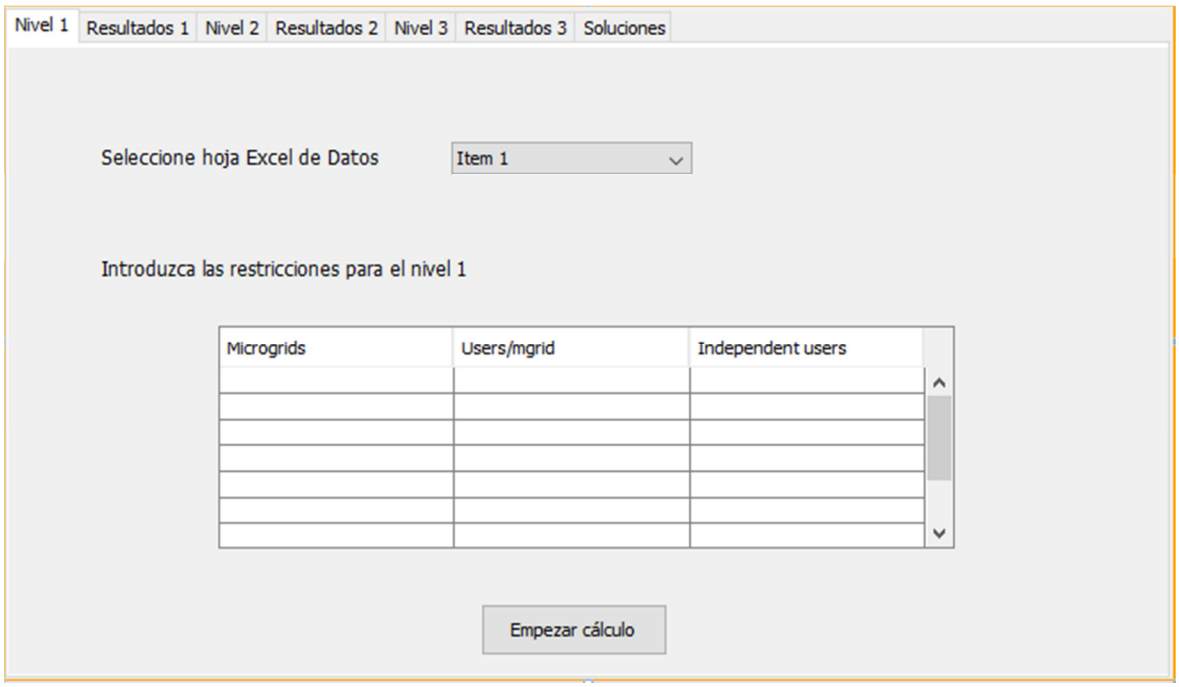

Fig. 18 Primera pestanya del model de la interfície gràfica.

A la part superior hi ha un menú on es pot anar de pestanya en pestanya per veure els resultats dels diferents nivells, introduir les restriccions i veure les solucions seleccionades. A la primera pestanya es podria seleccionar també el fitxer de dades inicial, tal com es pot veure a la figura 18.

A les pestanyes de resultats (figura 19) es mostren les microxarxes solució calculades per l'ILOG. Tenen un format semblant al de les pestanyes del fitxer multicriteri actual, però aquí es fa èmfasi en la claredat i en la possible implementació de noves característiques, com ara la selecció de més d'una solució, la ordenació de les solucions seguint un criteri de manera automàtica, o la possibilitat de que es mostri el graf de la microxarxa en una finestra a part quan es clica sobre la solució, per resoldre el problema de la mida limitada de les imatges.

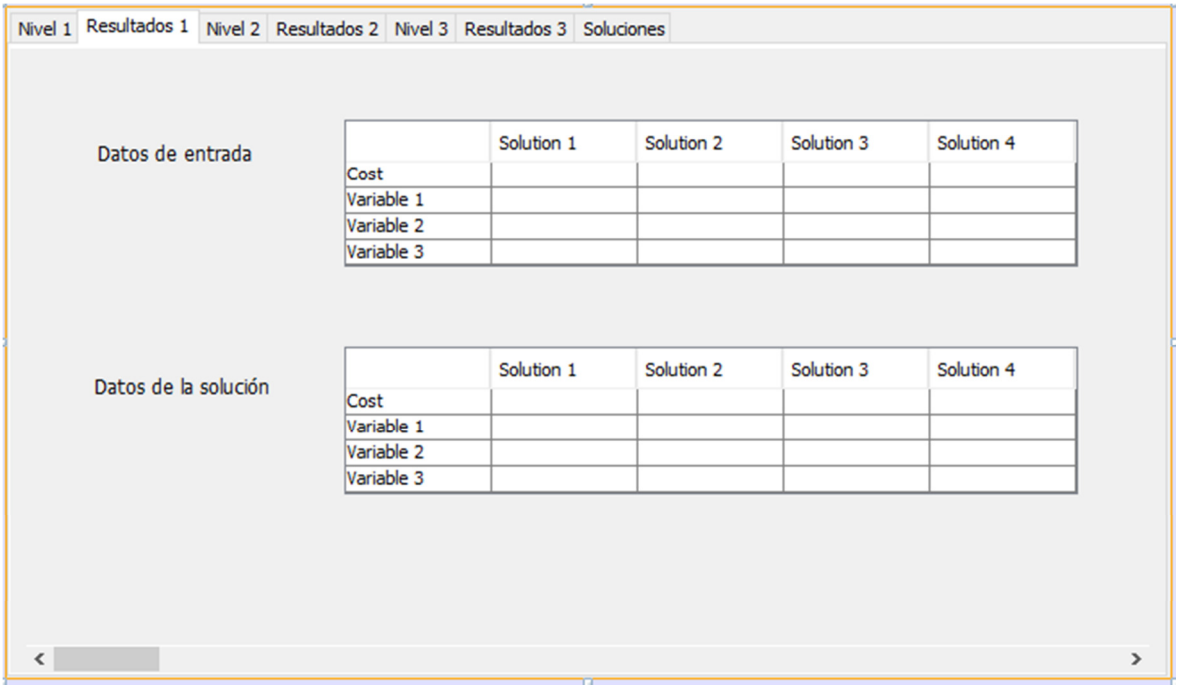

Fig. 19 Pestanya de resultats genèrica.

5.2. Selecció de múltiples solucions

Una millora que facilitaria la feina dels dissenyadors de microxarxes que facin servir el programa és que poguessin seleccionar més d'una solució a cada nivell, perquè de vegades no hi ha una microxarxa perfecta, sinó que es volen explorar diverses opcions per trobar la que més convingui en cada cas.

No s'ha pogut implementar perquè requereix d'una gran reestructuració en la manera de cridar les funcions i d'escriure a l'arxiu multicriteri, i es va plantejar quan el projecte estava

molt avançat. Caldria crear classes per cada nivell i fer una funció que s'encarregués d'instanciar els nivells i gestionar la seva crida, a més de totes les funcions internes per escriure les dades allà on toqués i emmagatzemar les solucions escollides per l'usuari.

Conclusions

Les funcions implementades finalment permeten a l'usuari: escollir des d'una fulla de càlcul quines restriccions vol imposar per cada nivell; fer la combinatòria de les restriccions; executar els nivells de manera independent a partir de la solució que seleccioni l'usuari; veure els resultats de cada solució calculada de manera clara i ordenada; tornar un o dos nivells enrere per triar unes altres restriccions; i veure gràficament la distribució de les microxarxes solució amb grafs superposats a la fulla de càlcul.

La resta de millores han anat destinades a l'estructuració del codi per facilitar-ne la comprensió i la creació d'un executable per fer funcionar el programa en ordinadors amb el sistema operatiu Windows.

Agraïments

Primer de tot, voldria agrair a Bruno Domenech la cooperació i el suport constant al llarg de tot el treball, ja que amb la seva ajuda i les seves recomanacions he pogut enfocar la manera de realitzar una tasca o sortir de punts on m'havia quedat encallat. Malgrat tota la feina que tenia sempre ha trobat un moment per ajudar-me.

També voldria dedicar aquest treball a la meva tutora, Daniela Tost, pel seu enfocament metòdic i la seva capacitat de fer-me visualitzar el projecte des d'una altra perspectiva, tan útil quan em trobava submergit profundament dins de la programació.

.

Bibliografia

Referències bibliogràfiques

[1] Bruno Domenech. Metodología para el diseño de sistemas de electrificación autónomos para comunidades rurales (2013)

[2] Bruno Domenech, Laia Ferrer-Martí, Rafael Pastor. Methodology for the design of hybrid wind-PV stand-alone electrification projects (2014)

[3] Google (02/06/16) Style Guide for C++ Code

https://google.github.io/styleguide/cppguide.html#General_Naming_Rules

[4] Oracle Corporation (02/06/16) How to write Doc Comments for the Javadoc Tool

http://www.oracle.com/technetwork/articles/java/index-137868.html

Bibliografia complementària

- Oracle Corporation (01/05/16) Document information. What is a Locale?

  https://docs.oracle.com/cd/E23824_01/html/E26033/glmbx.html

- Tutorials Point (I) Pvt. Ltd. (29/10/15) Java.io Package Tutorial

  http://www.tutorialspoint.com/java/io/index.htm

- Alexander, Alvin (17/02/16) Java/Scala JOptionPane Examples

  http://alvinalexander.com/source-code/java/javascala-joptionpane-examples

- NetBeans, Oracle Corporation (26/05/16) Native packaging in Netbeans IDE. Docs & Support

  https://netbeans.org/kb/docs/java/native_pkg.html

- JExcelAPI (23/10/15) Java Documentation

  http://jexcelapi.sourceforge.net/resources/javadocs/current/docs/

- Van Rossum, Guido (03/06/16) Style Guide for Python Code

  https://www.python.org/dev/peps/pep-0008/#indentation

- Microsoft (05/05/16) Función CONTAR.SI. Ayuda de Office

  https://support.office.com/es-es/article/CONTAR-SI-funci%C3%B3n-CONTAR-SI-e0de10c6-f885-4e71-abb4-1f464816df34

- Oracle (04/06/16) Adding classes to the JAR file's classpath

  https://docs.oracle.com/javase/tutorial/deployment/jar/downman.html

Annex

```
int contador2 = NMRED.length * NUXMRED.length * NUIND.length * MED.length;
String nivel2[] = new String[contador2];
int[] Gestion[] = new int[contador2][4];
int j0 = 0;
for (int j1=0; j1<NMRED.length; j1++) {
        for (int j2=0; j2<NUXMRED.length; j2++) {
                for (int j3=0; j3<NUIND.length; j3++) {
                        for (int j4=0; j4<MED.length; j4++) {
                                nivel2[j0] = (j0+1)+"_"+NMRED[j1]+"_"+NUXMRED[j2]+"_"+NUIND[j3]+"_"+MED[j4];
                                Gestion[j0][0] = j1; Gestion[j0][1] = j2; Gestion[j0][2] = j3; Gestion[j0][3] = j4;
                                j0 = j0+1;
                        }
                }
        }
}

int contador3 = SOL.length * GEN.length * MDEM.length;
String nivel3[] = new String[contador3];
int[] Seguridad[] = new int[contador3][3];
int k0 = 0;
for (int k1=0; k1<SOL.length; k1++) {
        for (int k2=0; k2<GEN.length; k2++) {
                for (int k3=0; k3<MDEM.length; k3++) {
                        nivel3[k0] = (k0+1)+"_"+SOL[k1]+"_"+GEN[k2]+"_"+MDEM[k3];
                        Seguridad[k0][0] = k1; Seguridad[k0][1] = k2; Seguridad[k0][2] = k3;
                        k0 = k0+1;
                }
        }
}
```

Annex 1 Combinatòria de valors dels nivells 2 i 3 a la metodologia, abans d'aplicar els canvis.