UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

PRÀCTICA DE CERCA LOCAL

Algorismes bàsics per la intel·ligència artificial, GIA, UPC

Autors:
Moya Morera, Adrià
Puerta del Valle, Javier
Spiridonov Poch, Daniel
Tutor:
Álvarez Napagao, Sergio

27/10/2022

INDEX:

1. Objectiu Principal de la Pràctica

La primera pràctica proposada a l'assignatura d'ABIA consisteix a resoldre un problema proposat de tal forma que aconseguim el millor resultat possible per una empresa amb la implementació de diferents estats inicials, operadors i una funció heurística que ens permet maximitzar els beneficis generats per aquesta empresa.

El problema consisteix en el fet que una empresa de centrals vol arribar a generar el màxim benefici tenint en compte certs criteris:

- Els 3 tipus de central. D'aquí havíem de tenir en compte el cost de producció i el de parada de cadascun.
- Els 3 tipus de clients. D'aquí havíem de tenir en compte si es tractava d'un client garantit (sempre se'ls ha de subministrar energia) o no garantit, el consum en Mw de cada tipus i la indemnització en cas de no subministrar a un client No Garantit (els garantits no es poden deixar de subministrar en cap cas).
- La pèrdua d'energia segons la distància entre la central i el client (aquesta energia que es perd no la paga el client sinó que és a compte de la central, la qual produeix de més).

L'objectiu al qual hem d'arribar és assignar els clients a les centrals de tal forma que obtinguem el màxim benefici seguint les restriccions anteriors.

A partir de l'arxiu donat pel professorat, anomenat abia_energia.py, és d'on hem tret les classes Centrals i Clients que ens donen una llista d'aquests amb els atributs necessaris per tenir coneixement de la seva capacitat de producció, energia requerida, etc. I VEnergia que ens dona els costos de la central, tarifes dels clients segons el tipus i la pèrdua d'energia segons la distància.

1.1 Cerca Local

Primer de tot hem d'entendre que una cerca local es tracta d'un procés que, donada una solució inicial vàlida, trobem una solució final a partir de fer una cerca per un espai de solucions.

En el cas de la nostra pràctica, es tracta d'una cerca local per diversos motius:

- Primer de tot es tracta d'una cerca local ja que en el nostre cas comencem amb una solució inicial vàlida des de la qual busquem una de les millors solucions d'entre totes les possibles per arribar a un màxim o mínim local. No comencem amb una "solució inicial buida" a partir de la qual es construeix la solució inicial, ja que això no seria considerat com una cerca local ja que la natura del problema ens diu que no és solució vàlida.
- A més a més, els operadors que utilitzem per obtenir una solució bona, no tenen cap cost que ajudi a la funció heurística a trobar una solució millor.
- Estem en un espai de solucions i no en un espai d'estats.
- Finalment, *Hill Climbing i Simulated annealing* són els algorismes adequats ja que no busquem un màxim o mínim absolut, sinó un local.

2. Descripció de la implementació de l'estat

Per tal de decidir quina implementació pels estats era la òptima vàrem exposar diverses opcions de les quals finalment ens vàrem quedar només amb tres.

Per decidir quina d'aquestes tres possibles opcions era la millor vàrem discutir el cost espacial que necessitaríem per accedir a cada client de cada central i vàrem valorar com hauria de ser la còpia per cada cas.

Opció 1: La primera opció consisteix en crear una llista de longitud nombre de clients i a cada posició el índex de la central que li subministra al client.

- Cost de la implementació:
    - El cost espacial d'aquesta representació seria de CL * log_2 C. El log_2C ve donat per que només escrivim el índex de la central.

- Còpia:
    - Es faria una còpia només de la llista de clients. El cost de la copia seria també lineal respecte al nombre de clients.

- Aspectes a favor :
    - Té un cost petit en comparació amb altres implementacions. La còpia per a cada estat té un cost també baix. Permet representar l'estat d'una manera senzilla i visual.

- Aspectes en contra :
    - Per tal de veure quants clients té una central qualsevol, s'hauria de recórrer tota la llista de clients per trobar aquells amb el mateix valor de central. Com té llargada nombre de clients, podem patir d'una quantitat molt elevada d'aquests i per tant fer que la llista es faci molt gran i, per tant, un cost molt elevat.

Opció 2: La segona opció consisteix a fer una llista de llistes o matriu, és a dir que cada posició de la llista principal fossi l'índex d'una central i que, dins d'aquesta, hi hagi una llista binària de longitud clients on no si es posa 1, significa que el client està vinculat a la central, i 0 que no ho està .

- Cost de la implementació:
    - El cost de la implementació, al ser una "matriu" binària tenim que el cost és igual a C*CL*1. Sent 1 el cost de representar un dígit binari.

- Còpia :
    - Per tal de realitzar la còpia de la representació, s'hauria de fer una còpia de cada subllista, a més de la de la llista general. Per tant, el cost seria el mateix, C*CL.
- Aspectes a favor :
    - A l'hora de veure quants clients té una central, tindríem un cost "petit", ja que caldria veure l'índex de la central que volem i mirar quins índexs tenen un 1 de la subllista. Altre aspecte positiu és que degut a que és un valor binari, el cost no augmenta a més, en quant a representació numèrica.
- Aspectes en contra :

    - Té un cost espacial bestial, tant per la representació com per la còpia. No és res eficient per fer cerca d'elements en una central, ja que ha de recórrer tota la matriu fins a trobar els elements a trobar.

Opció 3: La tercera opció consisteix en un diccionari que la seva clau és el índex de la central i el seu valor és un set amb el índex dels clients que hi ha en la central.

- Cost de la implementació:
    - El cost d'aquesta implementació és de C + Cl * log2, ja que la quantitat de valors al set és molt variable i no podem trobar un valor constant.

- Còpia:
    - A l'hora de fer la còpia, necessitaria fer una còpia de cada set de cada clau del diccionari, a més que la còpia del diccionari en general.
- Aspectes a favor :
    - Té un cost espacial bastant acceptable, sent un entremig entre les opcions 1 i 2. A l'hora de fer cerca també té un cost raonable, ja que ha de mirar per a cada clau si existeix l'element al seu valor. Finalment, la representació és molt fàcil d'interpretar per un extern.
- Aspectes en contra :
    - Amb un alt nombre de centrals, poden existir problemes de memòria. En el cas pitjor, tots els clients estan ubicats a l'última central, per tant hauria de recórrer totes les claus i sets.

Finalment, després d'haver estudiat aquestes tres opcions ens hem decantat per **utilitzar la tercera opció** com a implementació de l'estat. Hem decidit quedar-nos amb aquesta opció perquè pensem que és la que més equilibri té en quant a cost i eficiència a l'hora d'aplicar l'algorisme. A més, també tenim diversitat en el tipus d'estructura de dades, fet que pot arribar a optimitzar algunes parts. En addicció, és la que més explicabilitat en té, ja que es trivial la forma de veure aquestes relacions entre clients i centrals.

Per tant, com es pot observar a la figura 8.1.A, la nostra representació de l'estat consta de diversos atributs. Aquests són : les llistes de clients i centrals, el diccionari on guardem les relacions, una llista per guardar els estats de les centrals i altra llista per guardar l'índex d'aquells clients que no tinguin cap central.

Per altre costat, en quant al mètode de còpia, com es pot veure a l'apartat 8.1.B, tenim que, creem de nou la llista dels estats de les centrals, creem de nou el diccionari fent una *copy()* de cada set del seu interior i creem de nou la llista de clients que no tenen cap central. Una vegada s'ha fet això, retornem el nou estat amb les còpies fetes.

3. Operadors

Per decidir quins operadors volem usar vàrem apuntar tots els possibles operadors que podríem necessitar per tal d'arribar a una solució final vàlida.

- Moure el client d'una central a una altra(CL, C1, C2).
  - Factor ramificació: CENTRALS, ja que hem de mirar totes les centrals per assegurar-nos de que existeix alguna que minimitza el subministrament requerit.
  - Condició d'aplicabilitat : Un client es mourà d'una central a una altra si l'energia que necessita és menor en la nova que en la que ja estava. Els efectes que aconseguim són : guanyar més espai per a més clients i minimitzar la pèrdua d'energia que una central ha de generar per subministrar a un client.

- Intercanviar dos clients(CL1, CL2,C1,C2)
  - Factor ramificació: CLIENTS^2, ja que s'han de fer comparacions de tots els clients amb tots.
  - Condició d'aplicabilitat : 2 Clients es poden intercanviar si al fer el canvi, caben en les centrals i a més necessiten un subministrament menor al que ja tenien, a més han d'existir almenys 1 client en 2 o més centrals. Els efectes que aconseguim són tant positius com negatius. La part positiva és que minimitzem les pèrdues de subministrament i, negativament, paguem un cost computacional molt alt.

- Canviar estat de la central(C)
  - Factor ramificació: 2 ^ CENTRALS, ja que per cada central podem tenir 2 estats possibles.
  - Condició d'aplicabilitat : Una central es manté encesa si té algun client amb subministrament garantits. Si no en té, llavors calculem el cost de mantenir-la encesa o apagada i, depenent del resultat, encenem o apaguem, respectivament. Els efectes que tenim són : poden haver situacions en que és millor no subministrar a clients amb tarifa de no subministrar i també situacions on hi hagi 1 client a una central i que aquest sigui garantits i que, a més, estigui en pèrdues la central.

- Introduir client(CL, C)
  - Factor ramificació: CENTRALS, ja que només introduïm un client, llavors s'han de veure totes les centrals disponibles.
  - Condició d'aplicabilitat : Un client s'introduirà a aquella central on el seu percentatge de pèrdua en subministrament sigui la mínima possible. Els efectes són : maximitzar el benefici d'una central, ja que no ha de subministrar energia de més i minimitzar els canvis de centrals entre clients, ja que es s'introdueixen a la central més propera, que a simple vista, pensem que és la més òptima.

Per tant, havent pensat en aquests operadors, considerem que hem de fer experiments per tal de poder decidir amb quina combinació d'aquests ens permet obtenir una solució òptima en quant a temps i beneficis. En un primera vista crítica, pensem que potser el que pitjor resultat ens doni sigui l'operador d'intercanviar clients, degut al seu elevat factor de ramificació. Com a visió general, pensem que hem ideat aquests operadors amb unes condicions d'aplicabilitat bastant restrictives per tal de, reduir el temps de còmput, que ens ajudarà a fer els experiments en un menor temps, i per intentar trobar una solució propera a l'òptima. A continuació seguirem amb les explicacions dels operadors.

InsertClient:
Podem veure el codi de l'operador a l'apèndix 8.2 en la figura A.

Aquest operador s'encarrega de inserir un client que no tenia cap associació amb cap central a una central. Per tal de poder portar a terme aquesta funció primer compara l'energia restant de cada central amb els clients que no tenen cap associació amb cap central. Si l'energia restant de la central és major a l'energia que necessita el client se l'afegeix a un set que guarda totes les possibilitats de moviment en cada moment.

MoveClient:
Podem veure el codi de l'operador a l'apèndix 8.2 en la figura B.

Aquest és l'operador que s'encarrega d'associar un client que rebia subministrament d'una central amb una altra central. Però per poder fer aquesta acció abans s'han de tenir en compte diversos passos i comprovacions.

El primer que fem en aquest operador és sumar tots els beneficis que estem obtenint en aquell estat i per fer això hem de mirar si es tracta d'un client garantit o no per poder veure quina tarifa se li ha d'aplicar.
Llavors a aquest benefici li hem de restar el que gasta el client, és a dir, el cost que suposa per la central segons la distància a la qual estigui el client.
Un cop tenim això calculat restem el benefici de la central menys el cost del client per obtenir el benefici que obtenim de tenir aquest client en aquesta central.

El següent pas que fa és calcular el mateix, però en aquest cas per l'opció en què el client està associat a la nova central fent el mateix càlcul i finalment un cop té els dos beneficis calculats comprova que en la nova central tingui la suficient capacitat per subministrar l'energia que necessita el client i llavors comprova que el benefici obtingut en cas de canviar el client de central sigui major al benefici que obtenia abans de fer aquesta acció.
Finalment, si aquestes condicions es donen llavors es mou el client de central.

SwapState:
Podem veure el codi de l'operador a l'apèndix 8.2 en la figura C.

Aquest operador s'encarrega de canviar l'estat d'una central. Per fer això primer crea una llista set on marcarà per cada central si està encesa o apagada. Llavors entra al diccionari i mira a cada client de cada central per comprovar si són clients garantits (prèviament ja es marca la central com a "False", és a dir que no hi ha garantits a totes les centrals). Si veu que hi ha algun garantits marca aquesta variable com a "True".
Llavors comprovem la llista i afegim a swap_state_comb totes les centrals amb clients garantits.
En cas contrari, comprovem si surt més rendible canviar d'estat la central o deixar-ho tal com estava segons els beneficis que s'obtenen (es calculen els beneficis pels dos casos).
En cas que els beneficis siguin menors deixaran el mateix estat, en canvi, si són majors es farà el canvi.

Exchange clients :
Podem veure el codi de l'operador a l'apèndix 8.2 en la figura D.

Aquest operador s'encarrega d'intercanviar 2 clients. Per tal d'evitar iteracions, aquest operador intercanviarà clients de diferents centrals, no de la mateixa, ja que no té sentit. A més, seran intercanviats quan l'energia necessària d'aquest sigui més petita i a més si poden cabre o no a la nova central corresponent.

4. Funció d'avaluació heurística:

Al principi de tot, vàrem decidir provar diverses funcions heurístiques tot i que al final només podíem utilitzar la que estava relacionada amb maximitzar els beneficis.
En aquest cas el que buscàvem amb la nostra heurística era tenir-ne una que sigues lo suficientment informada per tal de poder trobar la solució amb el màxim benefici possible.

Com podem veure en el codi que es troba en els apèndixs, a la figura 8.3.A, ens basem exclusivament en els beneficis. Aquesta funció primer de tot guarda en variables el tipus de client, l'energia que necessita per ser subministrat i el contracte que té.
A partir d'aquí comprovem si la central està encesa o apagada. Si està apagada automàticament li restem als beneficis la penalització per no subministrar al client i a més al mateix client no garantit se li cobra (s'afegeix als beneficis) la tarifa mensual tot i no subministrar-li l'energia, en canvi, si està encesa comprovem si el client és garantit o no per saber quina tarifa se li ha d'aplicar i llavors se suma als beneficis el que ha consumit.
A més, si la central està apagada s'ha de restar del benefici el cost de estar parada la central (depenent del tipus serà un cost o un altre). En canvi, si està encesa s'ha de restar dels beneficis el cost diari de la central i el cost de produir l'energia pels clients.
Aquest procés anterior s'ha de fer per tots els clients de totes les centrals per obtenir el benefici total.
Finalment, mirem tots els clients (no garantits) que no estan associats a cap central i per cada un restem dels beneficis la penalització que se'ls ha de pagar per no ser subministrats.
D'aquesta forma aconseguim els beneficis nets d'una possible solució.

- ¿Per què aquesta heurística?

  Hem utilitzat aquesta heurística, ja que hem pogut veure que és la que ens proporciona uns resultats acceptables per tal d'apropar-nos a un òptim en un temps raonable, ja que busquem el benefici net després de descomptar totes i cada una de les despeses que comporta aquesta producció.

  Encara que pensem que poden existir altres heurístiques que guiïn al algorisme millor o altres on existeixin penalitzacions cap a aquelles que siguin no-solució, pensem que en el nostre cas, no es necessari penalitzar res ni fer-la més informada. Això és degut a que els operadors que hem ideat, creiem que són lo suficientment *greedy* per poder arribar a solucions bones.

5. Generació de solució inicial

Per tal de decidir quines dues solucions inicials utilitzaríem vàrem provar-ne vàries de les quals ens vàrem adonar que si la solució inicial retornava un primer estat molt favorable això dificultava molt a l'heurística per trobar una millor distribució mentre que un estat inicial aleatori tampoc era el més indicat per aquesta situació.

Finalment, ens vàrem decantar per les següents dues solucions inicials, ja que junt amb els operadors i l'heurística utilitzada ens retornava una solució propera a l'òptima..

- **Gen_initial_state_only_granted:**
    - El primer estat inicial ideat va ser el de gen_initial_state_only_granted. Aquest estat inicial s'encarrega de repartir tots els clients garantits entre les centrals de forma que cada client garantit estigui associat a la central més propera respecte a les seves coordenades. Els clients no garantits els deixem fora, en aquest cas els fiquem en una llista.

    - Per tant, com es pot veure en la figura 8.4.1.A, primer de tot generem amb els paràmetres donats les centrals i els clients. A continuació, fiquem tots els clients garantits en una llista ("clients_granted") i fem el mateix amb els clients no garantits (clients_no_granted). Llavors comprova per totes les centrals quina és el percentatge de pèrdua que ha de fer cada central per subministrar al client. Guardem tots els resultats a una llista i agafem l'índex del primer mínim. Aquesta serà la central del client.

    - Considerem per tant, que amb aquesta generadora donem una solució vàlida bàsica, ja que ens assegurem que tots els garantits tenen central. Això si, és una mica més *greedy* perquè fica els clients a les centrals properes, significant que les centrals no han de subministrar energia de més. A més a més, amb això ens assegurem que les centrals no s'omplen del tot, excepte en el cas que tots els clients es generen al voltant d'una central en concret.

    - Respecte el cost de trobar la solució inicial, tenim que en el cas pitjor és O(Clients garantits * Centrals) ja que hem de mirar en totes les centrals per cada client garantits. A més s'ha de sumar el cost de trobar el primer mínim de cada pèrdua de cada central.

- **Gen_initial_state_ordered:**
    - Aquest estat inicial s'encarrega de repartir primer tots els clients garantits entre les centrals en ordre d'arribada i un cop tots estan classificats es reparteixen tots els clients amb contracte no garantit per ordre d'arribada també fins emplenar les centrals o fins que no en quedin més clients.

    - Per tant, com es pot veure en la figura 8.4.2.A, primer de tot es classifiquen en dues llistes diferents els clients amb un contracte garantit i els clients amb un contracte no garantit igual que amb l'estat inicial anterior. Llavors mentre quedin clients amb contracte garantit, si la central entrant disposa d'energia lliure per subministrar s'associa aquest client a aquesta central, sinó es comprova la següent central fins trobar una central que tingui la capacitat suficient. Fem el mateix amb els clients amb contracte no garantit un cop ja tots els clients garantits estan associats a les diferents centrals. Si s'emplenen totes les centrals i queden clients amb contracte no garantit els afegim a una llista auxiliar i seran tractats com clients al qui s'ha de pagar la indemnització per no ser subministrats..

    - Des del punt de vista de la bondat de la solució, pensem que és una generadora molt més *greedy* que la de *gen_initial_state_only_granted*. Això és degut a que no ens quedem només en ficar els garantits, sinó que fiquem tots els clients disponibles fins omplir les centrals. Creiem que pot ser, al ser tan *greedy*, no deixem a l'algorisme trobar un òptim, però per poder donar com a vàlida aquesta suposició, s'haurà de fer una prova empírica.

    - Des del punt de vista del cost de generar-la trobem més o menys el mateix cost que l'altre generadora, O(Clients * Central), més un cost afegit d'ordenar els clients no subministrats respecte el seu consum, en el cas de trobar-nos en la situació d'omplir totes les centrals.

6. Experiments

6.1. Experiment 1

6.1.1. Característiques de l'experiment

Observació	Poden haver subconjunts d'operadors que obtinguin millors resultats respecte altres operadors en base a la nostra variable objectiu, els beneficis.
Plantejament	De tots els operadors que hem pensat que es poden utilitzar, provarem diverses combinacions d'aquests per veure quins ens proporcionen millors resultats.
Hipòtesis	El benefici és el mateix independentment dels operadors escollits(H0) o hi ha subconjunts d'operadors que provoquen millors beneficis.
Mètode	● Escollirem n subconjunts d'operadors ● Executarem 5 experiments per cada subconjunt d'operadors i anotarem el benefici i el temps en cada experiment ● Elegirem 10 llavors aleatòries per cada experiment, que s'utilitzen per la generació dels clients i les centrals. ● Escollirem la funció generadora que només col·loca els clients garantits en una de les centrals que té més a prop segons la distancia euclidea i tenint en compte la pèrdua d'energia. ● Utilitzarem els següents paràmetres: ○ proporció de centrals[A,B,C] = [5, 10, 25] ○ Nombre de clients = 1000 ○ Proporció de clients[XG,MG,G] = [0.2, 0.3, 0.5] ○ Proporció de garantits = 0.75 ○ ● Utilitzarem l'algorisme de Hill Climbing

6.1.2. Subconjunts d'operadors

Recordem que hem d'escollir d'entre els següents operadors:

- InsertClient: agafem un client que no està assignat a cap central i l'inserim a una de les centrals que més a prop té, considerem que les centrals que té més a prop són aquelles en les que la pèrdua d'energia és 0.
- MoveClient: Movem un client d'una central que ja té assignada a una altra si l'energia que ha de generar la central a la que volem moure el client, és menor a l'energia que ja genera actualment la central que té assignada.
- SwapState: Canviem l'estat d'una central a encesa o apagada, en funció de si conté un o més clients amb subministrament garantits.
- ExchangeClient: Intercanviar dos clients de dues centrals diferents si un dels dos clients es troba més a prop de l'altre central i no fa que l'altre client s'allunyi més del que el primer client s'apropa a la central del segon client.

6.1.3. Resultats de l'experimentació

A l'executar l'algorisme 10 vegades amb 10 llavors diferents, obtenim els resultats que apareixen en les taules de l'apèndix 8.5.1. Cal especificar que el temps mitjà ha sigut calculat en base a 10 iteracions.

Combinacions d'operadors	Mitjana temps(s)	Desviació temps(s)	Mitjana beneficis(€)	Desviació beneficis(€)
Insert + Move + Swap	23,28	2,64	100635	8379,98
Insert + Move	23,41	2,1	97475,6	8570,2
Insert + Swap	0,57	0,07	100600	8357,1
Move + Swap	0,234	0,067	60846,5	7267,76
Exchange	2123,42	53,5330525	57611,6	7603,54

Havent fet un anàlisi previ a l'experimentació, esperàvem que encara que tingués un temps pitjor, la millor solució seria *Insert + Move + Swap*, ja que pensàvem que donaria una solució propera a la òptima. La part que no esperàvem era que *Insert + Swap* ens donaria el mateix benefici i a més en un temps molt molt baix, convertint-se en una de les sorpreses més inesperades realitzant la pràctica. Aquest fet ens a fet aprendre a no donar res per sabut i realitzar totes les proves empíriques per comprovar.

Per un costat, tenim que l'opció *Insert + Move + Swap* obté més o menys els mateixos beneficis finals que l'opció de *Insert + Swap*, la única diferència és el temps, que en la segona opció és molt millor. Per tant, no fa falta fer cap prova estadística per demostrar quina distribució és millor, ja que a simple vista es veu que la segona té un rendiment superior al aconseguir mateixos resultats en menor temps.

Per altre costat, tenim 2 operadors que la seva mitjana i desviació són bastant semblants, és el cas de *Insert + Move + Swap i Move + Insert*. Per tal de veure quin temps és millor, s'ha fet un t-test comparant amb dues distribucions. El resultat ha sigut el següent.

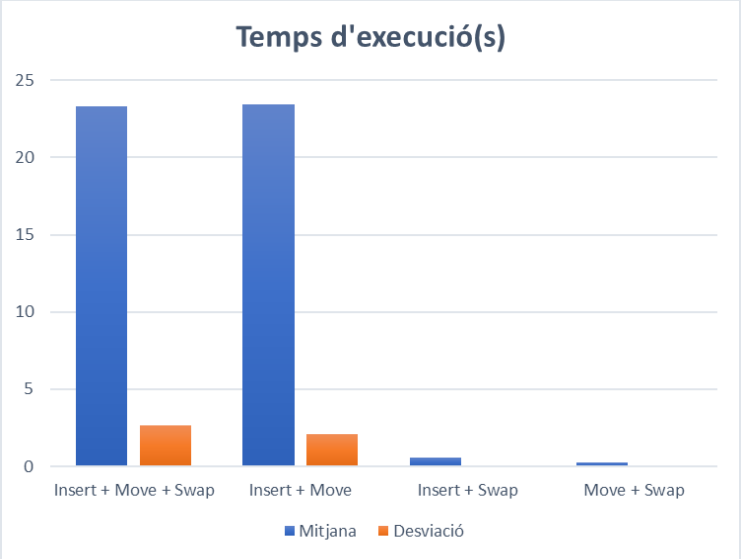
```
        Welch Two Sample t-test

data:  IM and IMS
t = 0.11972, df = 17.141, p-value = 0.9061
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.126263  2.382263
sample estimates:
mean of x mean of y
   23.409    23.281
```

6.1.3.A) Output del T-test.

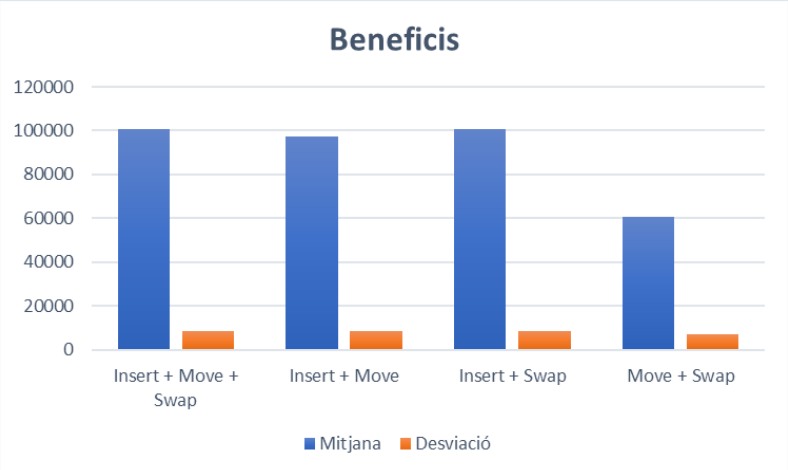
El resultat del t-test dona que existeix una probabilitat del 0.9061 de que les dues distribucions siguin iguals, és a dir, el p-value és superior al nivell de significació(0.05). En conclusió, no tenim evidències per rebutjar la hipòtesi nul·la de que són iguals. Si hauríem d'agafar alguna opció d'aquestes, ens tindrem que fixar als beneficis.
Finalment, tenim l'opció de *Move + Swap*, que, encara que tingui el temps mínim de tots els experiments, no la podem escollir com a la més òptima, ja que és la combinació amb menys beneficis.

En conclusió, una vegada fet aquest anàlisi, **considerem que l'opció que més beneficis aporta en quant a beneficis i temps és la de *Insert* + *Swap***. Tenir aquests operadors ens ajudarà a crear els diferents experiments que queden per fer d'una forma més ràpida i a més, ens aproparem a la solució més òptima.

A continuació podem veure els barplots generats a partir de les mitjanes i desviacions estàndards del temps d'execució i dels beneficis, d'aquesta forma podem apreciar molt més significativament la diferència entre les diferents combinacions d'operadors. En quant al temps no queda dubte que ens quedem entre les combinacions "Insert + Swap" o "Move + Swap", que al fer la intersecció amb els beneficis, on les tres primeres combinacions són les guanyadores, podem confirmar que "Insert + Swap" és la millor combinació d'operadors com hem vist anteriorment.



6.1.3.B) Barplot del temps d'execució en segons per cada combinació d'operadors



6.1.3.C) Barplot dels beneficis finals per cada combinació d'operadors

6.1.4 Explicació del criteri d'elecció dels operadors

Per aquest experiment podíem utilitzar dues funcions generadores de les quals una d'elles és molt greedy ja que assigna els clients only_granted a les centrals més properes i per tant la solució és molt bona, i encara que queden tots els clients no garantits per assignar, l'operador d'insert client, inserta el client en una de les centrals que té més a prop. Per consegüent, hi ha menys recorregut a l'heurística per recórrer les solucions. Per l'altra banda, una funció que assigna els clients garantits primer a les centrals en ordre d'arribada i llavors els no garantits fins que ja no en quedin o no puguem introduir-ne més a les centrals, amb aquest operador, només amb Insert Client i Swap Client, no podem recórrer tot l'espai de solucions.

En el nostre cas vàrem intentar primer utilitzar tres operadors (amb "Exchange") per poder realitzar l'execució i ens donaven uns beneficis raonables però el temps d'execució era molt elevat respecte els beneficis que ens retornava amb les dues generadores. I per aquest motiu, com una de les opcions era molt greedy (only_granted) vàrem provar a l'experiment de prescindir de "Exchange" i de seguida ens donava beneficis molt similars mentre que el temps disminuïa considerablement (de aproximadament 2100s a menys de 1s) el qual clarament suposava un clar benefici de temps per les centrals. Com estàvem utilitzant la funció generadora only granted, tal com hem explicat anteriorment, només amb els operadors d'insert client, swap state i move client, podem recórrer tot l'espai de solucions amb aquesta funció generadora i no vam tenir en compte els altres tipus de generacions inicials que poden haver-hi. Tot i que amb la segona funció generadora (Ordered) el fet de no utilitzar "Exchange" no explorem tot l'espai de solucions el qual tot i no ser vàlid per el problema hem decidit que per aportar un millor resultat utilitzarem "Insert + Swap" junt amb la generació més greedy ja que per temps i beneficis era l'opció més òptima.

Per concloure, encara que en aquest experiment havíem de comparar subconjunts d'operadors que poguessin recórrer tot l'espai de solucions, i d'aquests escollir el subconjunt que maximitzés els beneficis i minimitzés el temps d'execució, no vam tenir en compte que podia haver subconjunts d'operadors que no cobrien tot l'espai de solucions.
És per aquest motiu que fem avaluacions com per exemple l'operador d'intercanviar clients per si sol, per poder veure quant de temps significa introduir aquest operador, que com podem veure en les taules anteriors és un temps completament desorbitat tot i els nostres esforços per optimitzar el codi.
En aquest context, al avaluar els operadors d'aquesta manera, hem incomplert el fet que els operadors han de poder recórrer tot l'espai de solucions. Encara que, en un escenari real, si plantegem el problema amb una certa solució inicial i sabem que sempre s'utilitzarà aquesta funció generadora, podríem utilitzar els operadors descrits anteriorment, des del punt de vista de l'abstracció dels resultats i conclusions d'aprendre a usar un algorisme com Hill Climbing tal com descriu la pràctica, hauríem de no haver descartat l'operador de Exchange Client.
Però a data d'avui ens veiem impossibilitzats de tornar a realitzar tots els experiments amb aquesta premissa i és per això que seguim endavant amb l'elecció dels operadors insert client i swap client, tot i que en el codi podeu veure els altres dos operadors "move_client" i "exchange_clients" comentats, amb possibilitat de ser descomentats i usats.

6.2. Experiment 2

6.2.1. Característiques de l'experiment

Observació	Un cop hem escollit els operadors, pot ser que depèn de la funció generadora obtinguem més o menys operadors.
Plantejament	Hem creat dos funcions generadores, farem proves empíriques amb cada funció i escollirem la que creiem que ens proporciona millors resultats.
Hipòtesis	El benefici és el mateix independentment de la funció generadora(H0) o hi ha una de les dues funcions generadores que ens proporciona millors beneficis.
Mètode	● Escollirem primer la funció generadora Ordered i posteriorment realitzarem l'experiment per Only_Granted ● Executarem 5 experiments per cada llavor i cada funció generadora per tal de calcular el temps mitjà. ● Elegirem 10 llavors aleatòries per cada experiment, que s'utilitzen per la generació dels clients i les centrals. ● Escollirem els operadors elegits en l'anterior experiment. ● Utilitzarem els següents paràmetres: ○ proporció de centrals[A,B,C] = [5, 10, 25] ○ Nombre de clients = 1000 ○ Proporció de clients[XG,MG,G] = [0.2, 0.3, 0.5] ○ Proporció de garantits = 0.75 ○ ● Utilitzarem l'algorisme de Hill Climbing

6.2.2. Funcions generadores

Hem creat dues possible funcions generadores de la solució inicial, són les següents:

- **Ordered:** En primer lloc col·loca els clients amb subministrament garantit per ordre d'arribada i un cop han sigut col·locats, assigna els clients amb el subministrament no garantit, també per ordre d'arribada, els clients que no càpiguen en les centrals són guardats en una llista auxiliar.

- **Only_Granted:** Assigna tots els clients amb subministrament garantit a una de les centrals que té més a prop en base a la pèrdua d'energia del client a cada central. Els clients amb subministrament no garantit es guarden en la llista auxiliar i no s'assignen a cap central.

6.2.3. Conclusions

F. Generadora	Mitjana temps(s)	Desviació temps(s)	Mitjana dif. beneficis(€)	Desviació dif. beneficis(€)	Mitjana beneficis fin.(€)	Desviació beneficis fin.(€)
Ordered	1,42	0,49	16.567	9.812,68	108419,30	4070,375
Only_granted	0,65	0,07	45.068	4.841,52	100915,30	7851,008

Una vegada feta l'experimentació, com es pot observar als resultats finals, es veu com la funció generadora *Only_granted* obté un temps millor que la seva contrincant, però si mirem els beneficis, mirant només mitjanes si que sabem qui guanya però amb la desviació típica no ho podem saber amb claredat.

Per tant, es farà un t-test, per veure si aquestes mitjanes s'assemblen o no. Per tal d'interpretar el resultat, mirarem si la distribució de Only_granted és millor que la de Ordered.

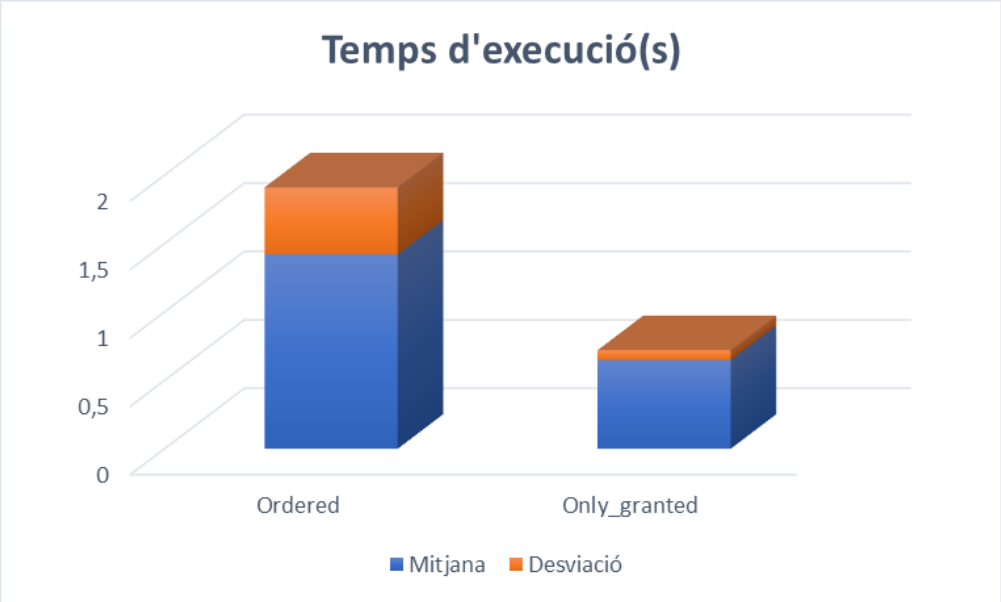
```
        Welch Two Sample t-test

data:  OG and OR
t = 8.2368, df = 13.137, p-value = 7.552e-07
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 22378.11      Inf
sample estimates:
mean of x mean of y
    45068     16567
```

2.4.A) T-test de les dues distribucions de diferencia de beneficis.

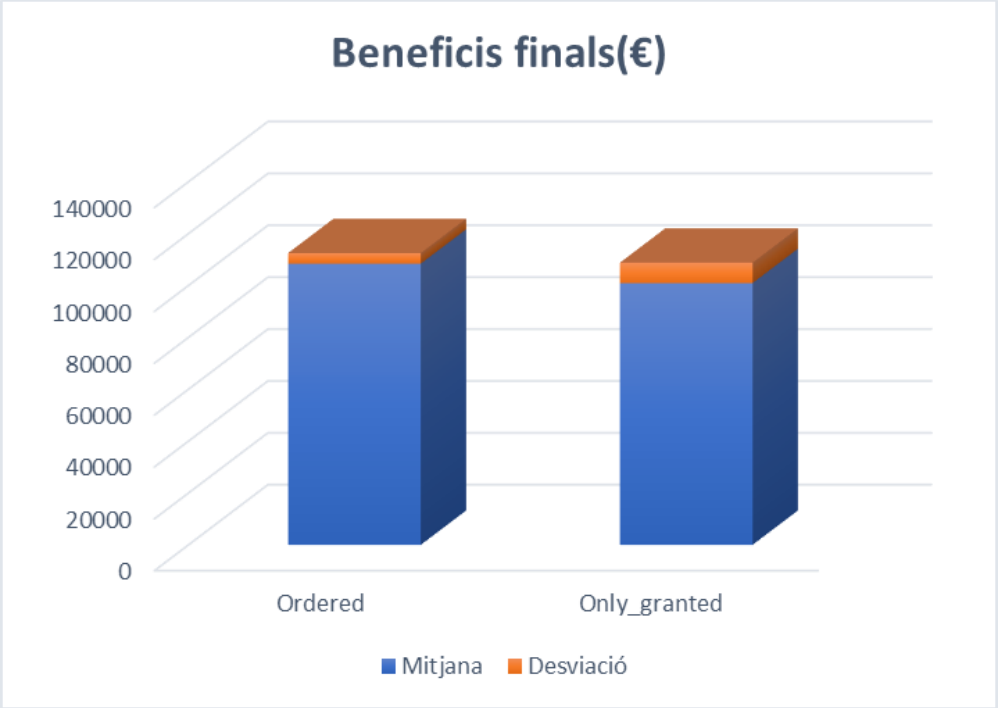
Com es pot observar, el p-value és menor que el nivell de significació(0,05), per tant, podem rebutjar la hipòtesi nul·la de que són igual i acceptar que la de *Only_granted* és millor que la de *Ordered*.

Per tant, havent vist el resultat d'aquest test, ens quedem amb la generadora de *Only_Granted*, ja que obtenim un benefici major i un temps menor en vers a l'altre generadora. A més, la que hem agafat no es tan tan *greedy*, per tant deixa al algorisme suficient espai per buscar les solucions òptimes.

En els gràfics següents podem apreciar de forma més visual que el temps d'execució de "Only Granted" és molt menor que el de "Ordered" i per altra banda si ens fixem en la desviació de "Only Granted" podem veure com la desviació és lleugerament major a la de "Ordered".



2.4.B) Barplot 3D del temps d'execució per cada funció generadora



2.4.B) Barplot 3D dels beneficis finals per cada funció generadora

6.3. Experiment 3

6.3.1. Característiques de l'experiment

Observació	Un cop hem triat la funció heurística, operadors i l'estratègia de generació de l'estat inicial a partir dels experiments anteriors, ara depèn dels paràmetres k (constant proporcional), lambda (exponent) i limit (número d'iteracions) que utilitzem aconseguirem millors o pitjors resultats.
Plantejament	Començarem fent proves amb constants proporcionals i una lambda més alta i un número d'iteracions més baix i llavors anirem reduint les dues primeres per tal de permetre l'algoritme explorar amb més llibertat i farem proves augmentant el número d'iteracions.
Hipòtesis	Amb una constant proporcional i lambda més petites els resultats seran més favorables en termes de beneficis i temps.
Mètode	● Escollirem primer amb quins paràmetres k, lam i limit volem començar l'experiment basat en l'experiment ● Executarem els experiments necessaris per arribar al punt d'inflexió on no retorni error per quedar-se sense clients que intercanviar ● Elegirem 10 llavors aleatòries per cada experiment, que s'utilitzen per la generació dels clients i les centrals. ● Escollirem els operadors, heurística i estat inicial elegits en l'anterior experiment i triarem 10 paràmetres de cada diferents (per cada llavor utilitzarem els mateixos paràmetres). ● Utilitzarem els següents paràmetres: ○ proporció de centrals[A,B,C] = [5, 10, 25] ○ Nombre de clients = 1000 ○ Proporció de clients[XG,MG,G] = [0.2, 0.3, 0.5] ○ Proporció de garantits = 0.75 ● Utilitzarem l'algorisme de Simulated Annealing

6.3.2. Resultats de l'experimentació i Conclusions

Constant Proporcional (k)	Exponent (lam)	Num Iteracions (limit)	Mitjana dif. Beneficis	Desviació dif. beneficis	Mitjana del temps	Desviació del temps
20	0.5	200	21683,56	6370,81	0,98	0,29
70	0.5	10	4851,33	4647,91	0,15	0,06
50	0.1	50	6918,56	4645,58	0,41	0,07
35	0.05	87	9790,78	4900,11	0,57	0,08
30	0.01	128	13479,67	5174,77	0,81	0,13
20	0.01	170	17979,11	5760,48	1,52	0,43
20	0.01	200	21492,44	6754,88	1,69	0,59
10	0.001	220	23979,11	6859,17	0,94	0,22
5	0.0001	240	27906,89	8603,81	0,89	0,18
5	0.0001	200	20536,22	7399,47	0,91	0,24
5	0.001	200	21554,67	6715,70	0,87	0,09

6.3.2.A) Taula de resultats finals

Després de fer tots els experiments hem obtingut els resultats anteriors de mitjana i desviació típica.
Per un costat ens podem fixar en les quantitats de Num Iteracions. Podem veure ràpidament que com més iteracions fem, més diferència de beneficis obtenim respecte els beneficis inicials i més temps triga en executar-se (tot i que es tracten de mil·lisegons).
Tot i això, si el nombre de iteracions és massa elevat junt amb la constant i la lambda, ens trobem amb que no hi ha més clients sense assignar i per tant, com els operadors que utilitzem són insert i swap, salta un error ja que no és capaç de continuar. El punt en que queden 0 clients per assignar i deixem de iterar és el punt amb més beneficis ja que tenim tots els clients associats a centrals (refredament) Aquest esdeveniment es pot visualitzar al pdf de la pràctica on en quan ens apropem a probabilitats de 0 en el cas és quan arribem a la situació en que no queden clients sense assignar i com en el nostre cas els operadors no poden realitzar més operacions un cop que tots estan associat, és quan ens retorna un error.

En el nostre cas hem agafat límits d'iteracions que ens permetin realitzar l'execució amb totes les llavors i per llavors poder realitzar la mitjana de beneficis i temps d'execució i les seves desviacions. D'aquesta forma hem decidit que el límit que utilitzarem és de 240 ja que és compatible per totes les llavors que hem provat. De totes formes, hem de tenir en compte que per cada llavor que utilitzem i els paràmetres k (permet explorar més fills) i lambda (aquests faran que puguem fer més o menys iteracions també) el límit màxim d'iteracions serà diferent.

Per altra banda, la constant proporcional i l'exponent lambda tenen molt poca influència en els resultats, tot i que tal i com vàrem predir al principi de l'experiment el fet de reduir la constant i l'exponent ens permet explorar l'algoritme amb més llibertat i obtenir uns resultats una mica millors i sobretot amb temps molt millors.
Això es veu si agafem les dues files amb el mateix número de iteracions (200) i en fem la relació.

- K = 20 ; LAM = 0,5 ; LIMIT = 200 ; → 21683,56 / 0,98 = 22126,0816 e / segon
- K = 5 ; LAM = 0,001 ; LIMIT = 200 ; → 21554,67 / 0,87 = 24775,4828 e / segon

Com podem veure la relació ens mostra que ens retorna un millor resultat per segon amb variables k i lambda més petites, tot i que si ens fixem en els resultats de diferència de beneficis amb una constant i lambda més alts obtenim més benefici per una diferència de mil·lisegons molt baixa lo qual és un temps insignificant.
Després d'analitzar les dades obtingudes hem pogut veure que és millor mantenir la constant i la lambda més altes com pot ser **K = 20 i LAM = 0,5** i intentar buscar la màxima quantitat de iteracions per arribar al límit màxim per aquella llavor i d'aquesta forma assignar TOTS els clients a alguna central per augmentar el benefici i mantenir un temps d'execució mínim. En el nostre cas hem vist que la millor opció per tal de poder acabar l'execució sense quedar-nos sense clients per inserir és establir el **LIM = 240**. El temps de diferència que obtenim sempre son mil·lisegons, per lo tant, aplicant aquest cas a la vida real val la pena esperar un temps més per tal d'obtenir més beneficis. En el nostre cas com ho hem de provar amb diverses llavors cada vegada, per tal de no buscar el límit per cada estat inicial (llavor) vàrem establir el límit d'iteracions en 240.

Finalment hem pogut verificar que els resultats en termes de rati són millors per una constant i lambda més petites, tot i que hem prioritzat la diferència de beneficis ja que la mitjana de temps era insignificant.

6.4. Experiment 4

6.4.1 Característiques de l'experiment

Observació	Hem d'analitzar l'augment del temps en l'execució quan s'escala el nombre de centrals.
Plantejament	Augmentarem de 40 en 40 el nombre de centrals mantenint les mateixes proporcions que en l'estat plantejat inicialment.
Hipòtesis	El temps no augmentarà(H0) o el temps augmentarà linealment a mesura que augmentem el nombre de centrals
Mètode	● Elegim 10 llavors aleatòries ● Executem l'algorisme amb la funció generadora, heurística i operadors decidits anteriorment ● Un cop executat l'algorisme i guardat el temps, augmentarem el nombre de centrals en la mateixa proporció que en l'estat inicial([5,10,25]) i repetim el procés 10 cops per llavor ● Utilitzarem els següents paràmetres inicials: ○ proporció de centrals[A,B,C] = [5, 10, 25] ○ Nombre de clients = 1000 ○ Proporció de clients[XG,MG,G] = [0.2, 0.3, 0.5] ○ Proporció de garantits = 0.75 ● Augmentarem les centrals de forma que la proporció de centrals serà de [0.13*n, 0.25*n, 0.63*n], sent n el nombre de central, és a dir, 40,80,120, etc.

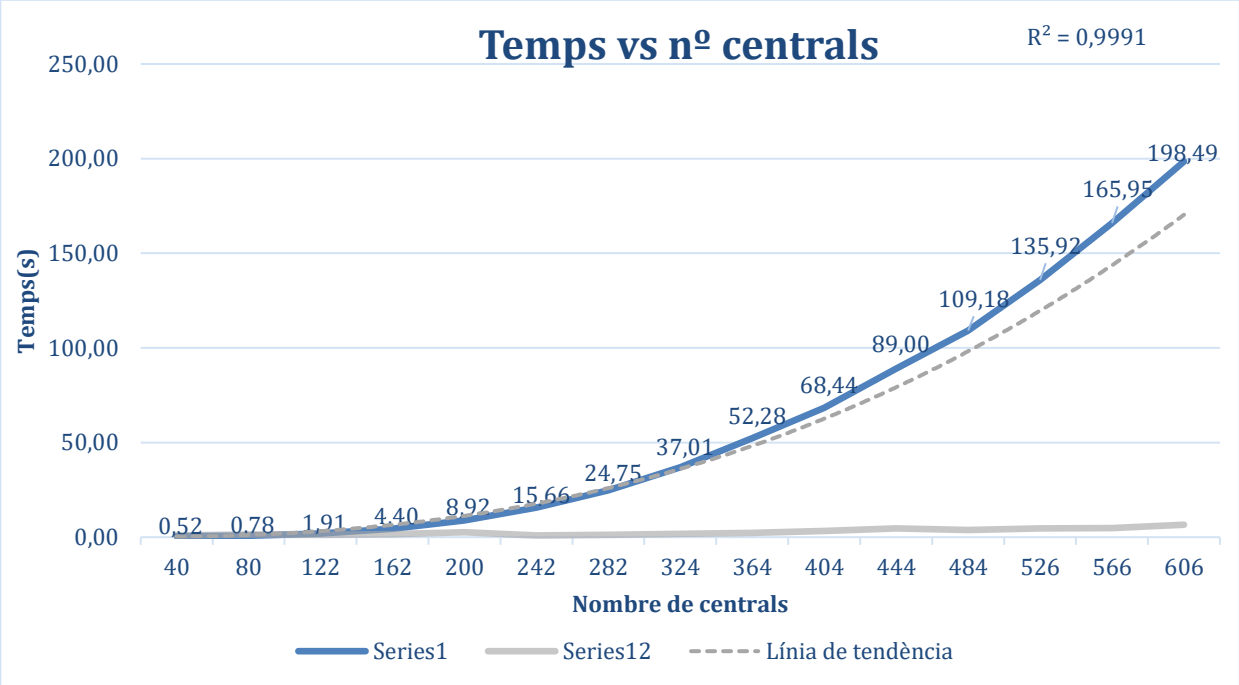
Analitzarem les mitges i desviacions estàndards del temps d'execució de l'algorisme amb els diferents nombres de centrals. Triarem 10 llavors diferents per fer les mitges per tal d'obtenir resultats més fiables. Cada mesura de temps és obtinguda a partir de la mitja de realitzar 10 cops l'experiment. Per tant, s'ha executat 10 cops per cada nombre de centrals de 40 a 606, augmentant n en proporció de 40 en 40 i seguint les proporcions que es descriuen anteriorment. Això ho realitzem amb les 10 llavors i d'aquí trèiem la mitja i la desviació estàndard de les mostres de cada nombre de centrals, amb les que realitzarem l'anàlisi i extraurem les conclusions a continuació.

6.4.2. Resultats de l'experimentació

Per tal de poder veure la tendència del temps al augmentar el nombre de centrals, hem anat augmentant les centrals en la mateixa proporció que hi ha en l'estat inicial amb 40 centrals, és a dir:
Centrals del tipus A: 5/40 = 0.13
Centrals del tipus B: 10/40 = 0.25
Centrals del tipus C: 25/40 = 0.63
De forma que no augmentarem exactament de 40 en 40 el nombre de centrals, però mantindrem les proporcions de centrals. Per tal de veure la tendència, augmentarem 15 cops el nombre de centrals inicials, per tant 15*40 = 600, en l'escenari final tindrem unes 600 centrals, si mantenim la proporció inicial són exactament 606 centrals.
Com podem veure en l'apèndix 8.5.4.A) veiem les execucions de cada llavor per cada nombre de centrals i en les dues columnes finals veiem la mitjana i la desviació per cada nombre de centrals.

És necessari destacar el canvi de criteri que vam tenir i com això va afectar a la tendència del temps, en primer lloc, quan pensàvem que si un client no estava sent subministrat, no havia de pagar la seva quota per MW, i utilitzàvem els operadors de *Insert Client* + *Move Client* + *Swap State*, vam obtenir unes dades diferents amb una tendència lineal.
En canvi, seguint el criteri que vam establir en el laboratori del 20/10/2022, en el que es va especificar que els clients no garantits, encara que no se'ls subministrés energia, havien de pagar la seva quota i se'ls havia de pagar la indemnització, un cop realitzats de nou els experiments 1 i 2 i escollint la nova combinació d'operadors *Insert Client* + *Swap Client*, tenim unes dades diferents que segueixen una tendència exponencial com veiem a la figura 4.3.A.

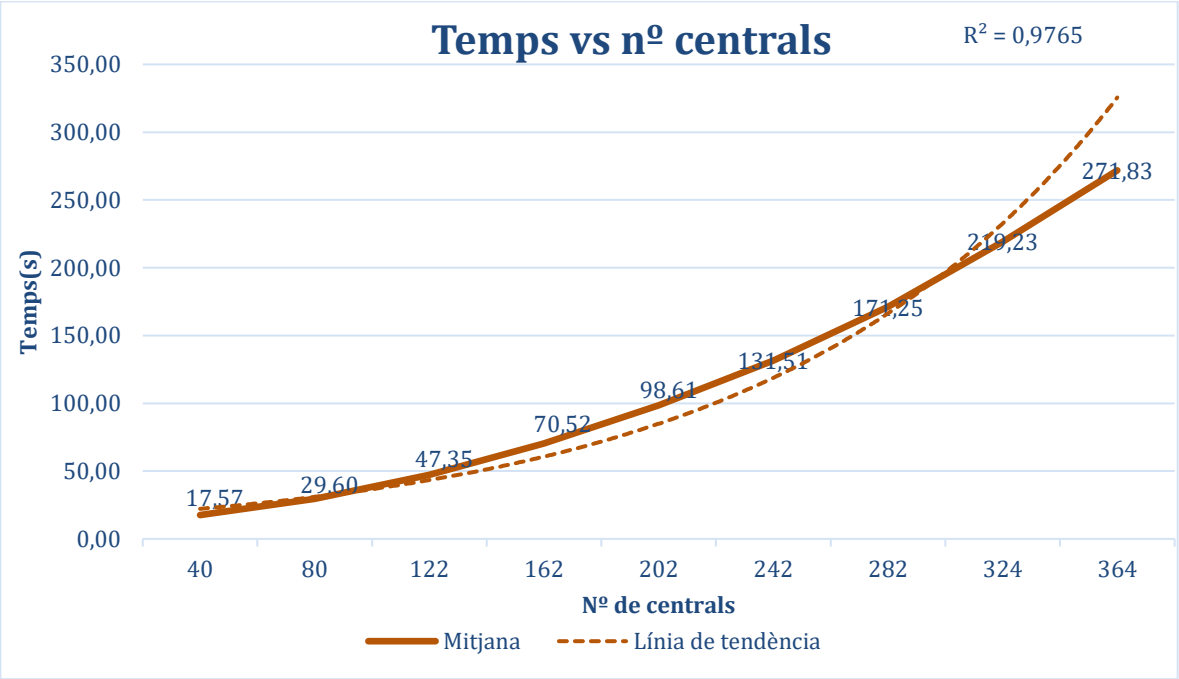
Una vegada obtinguda la taula amb l'increment de centrals, s'ha de fer un plot per veure la tendència general.



4.3.A) Gràfic de línies amb la mitjana i la desviació de l'experiment 4

Com podem observar, la tendència que surt és exponencial, amb un R^2 de 0.9991, de relació amb la línia de tendència exponencial, per tant, podem confirmar que es tracta d'una tendència exponencial i per consegüent no hem realitzat una bona hipòtesi a l'inici de l'experiment. Aquest fet implica que, encara que l'algorisme *Hill Climbing* tingui un creixement exponencial de per si, amb la representació que hem fet del problema ens apareix una tendència exponencial, que és d'estranyar, ja que per deducció, pensaríem que hauria de seguir una tendència lineal, però acabem de veure que no és així.

Ja que veiem estrany que seguís una tendència exponencial, hem realitzat el mateix experiment amb els tres operadors de *Insert Client*, *Move Client i Swap State*, podem veure el resultat en la figura 4.3.B.



4.3.B) Gràfic de línies amb la mitjana de l'experiment addicional de l'experiment 4

Podem veure com amb els operadors de *Insert Client* + *Move Client* + *Swap State*, també veiem una tendència exponencial entre el temps d'execució i el nombre de centrals. Comprovat que no és un error de l'ús dels operadors, pels seleccionats en l'experiment 1, hem de confirmar que hi ha una tendència exponencial.

Per tant, **queda demostrat empíricament que la relació entre el nombre de centrals i el temps és potencial.**

Cal destacar que aquest experiment ha tingut un alt cost d'execució i el haver-ho fet fins a 606 centrals ens ha suposat molt de temps de còmput. En el primer experiment, si sumem totes les mitjanes i ho multipliquem per 10, que és el nombre de llavors i per 10 un altre cop que són les iteracions per cada llavor, tenim un temps total de 91320,2 segons, o el que és el mateix, 25 hores i 20 minuts aproximadament. Pel segon experiment, tenim un temps total de 105746,1 segons, o de 29 hores i 25 minuts. En conseqüència, un temps total d'experimentació de 54 hores i 45 minuts.

6.5. Experiment

6.5.1. Característiques de l'experiment

Observació	Si donem una penalització en la heurística cap a aquells estats on no hi hagi subministrament de garantits, potser pot existir un rang de valors en la penalització on es compleixi una solució vàlida
Plantejament	A partir d'un estat inicial on no hi hagi cap subministrament, amb els operadors agafats, mirarem si aconsegueix obtenir una solució on tots els garantits estiguin servits amb diferents penalitzacions en l'heurística.
Hipòtesis	Podem col·locar qualsevol valor en la penalització i donarà un valor vàlid (H0) o existeixen un rang de valors on els quals no es compleixi aquesta condició.
Mètode	● Escollirem n valors de penalització. ● Executarem 10 experiments per valor de penalització distints i anotarem si dona una solució vàlida i quantes persones no subministra energia. ● Elegirem 10 llavors aleatòries per cada experiment, que s'utilitzen per la generació dels clients i les centrals. ● Escollirem la funció generadora que indica l'enunciat de l'experiment, és a dir, no subministrar a cap client. ● Utilitzarem els següents paràmetres: ○ proporció de centrals[A,B,C] = [1, 3, 5] ○ Nombre de clients = 100 ○ Proporció de clients[XG,MG,G] = [0.2, 0.3, 0.5] ○ Proporció de garantits = 0.75 ○ *Simulated annealing*: ■ k = 5 ■ lambda = 0.0001 ■ limit = 240 ● Utilitzarem l'algorisme de *Hill Climbing i simulated annealing*, amb els paràmetres obtinguts a l'experiment 3.

6.5.2. Resultats de l'experimentació

Un aspecte a tenir en compte és, que amb els operadors actuals no hem pogut obtenir cap solució vàlida. L'algorisme *Hill Climbing* agafava una solució en la que hi havia garantits en centrals que estaven apagades, da igual la penalització que és posés en la heurística. Per tant, per tal de que no existeixi aquest problema, hem modificat 2 línies de l'operador de *Swap_State*. Hem fet que quan hi hagi un garantit s'encengui la central, en canvi d'enviar un yield. A més, hem tingut un imprevist a última hora que ha fet que no ens doni temps per fer les rèpliques necessàries per mil clients, per tant, s'ha fet l'opció de 100 clients. A més, de totes les llavors, agafarem aquella que ens indica clarament quina es la cota màxima i mínima de la penalització. En aquest cas, la llavor 732.

A més, una cosa a tenir en compte és l'heurística agafada, on és la següent :

$$h1 = \sum beneficis - X * clients\ garantits\ no\ subministrats$$

Per tant, s'ha fixat el valor que li donem als beneficis i només variem la penalització de clients garantits no subministrats.

Com hem de fer aquest experiment amb els dos algorismes, esperem que amb el *hill climbing*, encara que tardi menys, ens doni més valors no vàlids, ja que pot ser es troba amb un màxim local poc òptim. A més, esperem que el *simulated annealing* ens doni més valors vàlids i millors en quant a beneficis.

6.5.3 Conclusió de l'experimentació

F. Generadora	COTA MÍNIMA	COTA MÀXIMA	TEMPS PROMIG	DIFF BENEFICIS
HILL CLIMBING	800	1600	0.07 +- 0.03	15241,8 +- 1779,6
SIMULATED ANNEALING	> 0	-	0.09 +- 0.01	16027.7 +- 1915.1

Una vegada feta l'experimentació, podem observar 2 resultats totalment diferents.

Per un costat, tenim els resultats aplicant l'algorisme de *Hill Climbing*, on la cota mínima on la qual la penalització surt efectiva és a partir d'un valor de 800, en quant a la llavor 732.En quant a la cota màxima, on dona un resultat no vàlid havent tingut vàlids anteriorment, és de 1600.

Per l'altre banda, tenim el *simulated annealing*, que ens ha sorprès bastant els resultats obtinguts. Ens hem trobat que amb les mateixes penalitzacions ens dona sempre una solució vàlida, i a més, solucions properes. Vist això, hem pensat que això podria ser possible deguda a la natura del propi algorisme, on al fer diferents iteracions evita quasi sempre aquelles solucions que no són vàlides, és també per això que no existeix una cota màxima. A més, també pensem que el factor d'aleatorietat pot afectar directament a l'elecció d'aquests estats no vàlids.

Finalment, partint de la mateixa solució inicial, hem obtingut que *Hill climbing* i *Simulated Annealing* donen resultats molt semblants en quant a temps i diferència de benefici. Per tant, farem 2 t-test per veure quin dels dos és millor.

Tenim la hipòtesi de que *Simulated annealing* dona millor diferència de beneficis i millor temps, per tant, farem 2 t-test per comprovar aquesta hipòtesi alternativa.

```
        Welch Two Sample t-test

data:  sa and hc
t = 7.4745, df = 134.45, p-value = 4.457e-12
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.01837053        Inf
sample estimates:
mean of x mean of y
   0.0941    0.0705
```

5.3.A) T-test per comparar els temps

```
        Welch Two Sample t-test

data:  sad and hcd
t = 3.0064, df = 196.94, p-value = 0.001494
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 353.9181      Inf
sample estimates:
mean of x mean of y
 16027.73  15241.77
```

5.3.B) T-test per comparar les diferències de beneficis

En la figura 5.3.A, el p-value és menor a 0.05, per tant, tenim suficient informació per rebutjar la hipòtesi nul·la. Llavors, sabem, ara sí que sí, que en quant a temps és millor *Simmulated annealing*.

Per altre costat, en la figura 5.3.B, el p-value és una mica menor que 0.05, indicant que podem rebutjar la hipòtesi nul·la, però sabem que més o menys poden ser iguals. Llavors *Simulated annealing* és millor en quant a diferència de beneficis.

En conclusió, si partim des d'una solució inicial buida, la millor opció en quant a temps i diners, encara que *Hill Climbing* té valors similars, és la d'agafar l'algorisme de *Simulated annealing*.

7. Conclusions finals

Finalment, després de concloure amb tots els experiments hem pogut analitzar de forma empírica quines eren les combinacions que ens permeten arribar a solucions finals més òptimes tant per el *Hill Climbing* com per el *Simulated Annealing*.

Hill Climbing:

Representació de l'Estat:

Abans de començar la pràctica ja vàrem valorar les diferents opcions que teníem en quant a la representació d'estats i després de valorar les diferents opcions que teníem ens vàrem decantar per un diccionari que la seva clau és el índex de la central i el seu valor és un set amb el índex dels clients que hi ha en la central, tal i com hem especificat a l'apartat del punt 2 de la pràctica.

Operadors:

Al principi de la pràctica vàrem pensar en diversos operadors que podien servir per arribar a obtenir una major quantitat de beneficis. Per tal de decidir quina combinació d'aquests ens generaria més beneficis en un temps raonable vàrem fer experiments amb diferents llavors i diferents combinacions d'operadors. A partir dels resultats que vam obtenir, de seguida vàrem descartar l'operador d'Exchange ja que el temps d'execució era massa gran i per tant mai acabava l'execució i no arribàvem a la solució final.
Per altra banda, els dos operadors que ens retornaven millors resultats en termes de benefici eren "*Insert + Move + Swap*" i "*Insert + Swap*" però hi havia una diferència molt clara entre aquestes dues combinacions. El temps d'execució amb l'operador *Swap* era molt major ja que aquest tenia un cost temporal molt major i la diferència de beneficis era de tan sols 30 euros mentre la diferència de temps era d'un 4084% més, el qual suposa una diferència molt gran tant per les centrals com per fer els experiments de la pràctica.
Per aquests motius finalment ens vàrem quedar amb els dos operadors *"Insert" i "Swap"*.
Després de veure a partir d'experiments quins operadors acabàvem utilitzant, vàrem decidir quina solució inicial seria la més apropiada junt amb aquests operadors

Solucions Finals Definitives Continuació:

Per decidir quina solució inicial era la més adient amb els operadors que vàrem triar vàrem utilitzar diverses llavors i les dues representacions.
Amb els resultats finals podem veure que la funció generadora "Ordered" comença amb beneficis inicials molt elevats el qual no dona gaire llibertat a la funció generadora per trobar estats molt millors. Per altra banda veiem clarament que la funció generadora "Only Granted" comença amb uns beneficis inicials molt menors però la funció heurística acaba arribant a una solució final de forma més ràpida i la diferència de beneficis entre la inicial i la final és molt major.
Això ens ha fet decidir-nos per la segona funció generadora ja que busquem que la funció heurística ens generi el màxim benefici amb el mínim temps possible.

Per acabar de verificar entre aquestes dues, vàrem fer un t-test per mirar si les mitjanes s'assemblen entre elles. D'aquesta forma vàrem veure que el p-value és menor que el nivell de significació (0,05) i per tant rebutgem que son iguals i acceptem així que "Only Granted" és millor que la de "Ordered".

Simulated Annealing:

Paràmetres:

Un cop ja vàrem verificar l'estat inicial i els operadors més òptims per el Hill Climbing vàrem agafar aquests i els vàrem utilitzar per veure quins paràmetres eren els més adients junt amb els operadors i l'estat inicial triat.

Després de fer l'experiment amb 10 llavors diferents i 10 combinacions de paràmetres diferents per cada llavor vàrem poder deduir quins paràmetres eren els que ens podien aportar el resultat més òptim en el menor temps possible.

En aquest cas primer vàrem experimentar amb la mateixa constant i lambda i vàrem canviar el límit i ens vàrem adonar que arriba un punt en el qual tots els clients queden subministrats i el benefici arriba a un màxim i totes les iteracions posteriors fan saltar un error a causa de que ja no queden més clients per subministrar i els operadors no poden fer res més.

Per aquest motiu vàrem establir un límit de 240 iteracions ja que d'aquesta manera, tot i no arribar al màxim de beneficis, per totes les llavors de l'experiment podíem executar-ho fins el final.

Per altra banda podem veure que a mesura que augmenta la constant k, el número d'iteracions en que la probabilitat d'acceptar estats pitjors es alta va augmentant. I si augmenta la lambda, la velocitat en que aquesta probabilitat accepta estats pitjors es alta va augmentant i disminueix el número d'iteracions que necessitem per arribar al punt on ja no acceptem estat pitjors.

Per aquests motius finalment ens vàrem quedar amb una lambda més elevada i una constant més alta per tal d'arribar al punt en que ja no acceptem estats pitjors de forma més prolongada i en un recorregut menor i arribant a majors beneficis per l'insignificant augment en els beneficis.

Comparació d'algorismes:

Finalment, gràcies a l'experiment número 5 descrit a l'enunciat, vàrem poder comprovar quin algorisme era millor partint d'una solució buida. Havent vist les conclusions en el seu apartat corresponent, a mètode de resum, tenim que *Hill climbing* dona resultats en un temps molt baix, però els seus beneficis no són dels més òptims i en canvi, *Simulated annealing* dona resultats en un temps més baix i, a més, la seva bondat de solució és pitjor. Un aspecte a destacar és que, degut a la aleatorietat de *Simulated annealing*, hi ha molt poca possibilitat de poder agafar aquella solució on s'apliqui la penalització, per tant, la penalització, en aquest cas no serveix per guiar l'algorisme si partim des de la solució buida. En canvi, *Hill climbing* si que permet explorar aquelles solucions on es penalitzi, degut a que sempre agafa aquell successor que és millor que els altres i no té cap aleatorietat.

8.Apèndix

8.1. Codi de la representació de l'estat

```
class StateRepresentation(object):
    def __init__(self, clients:Clientes, centrals:Centrales, dict:dict, states:list, left:list = []):
        self.clients = clients
        self.centrals = centrals
        self.dict = dict
        self.states = states
        self.left = left
```

8.1.A) Codi de la representació de l'estat.

```
def copy(self):
    new_dict = {x:self.dict[x].copy() for x in self.dict}
    new_left = [x for x in self.left]
    new_states = [x for x in self.states]
    return StateRepresentation(self.clients,self.centrals,new_dict,new_states, new_left)
```

8.1.B) Codi del mètode de còpia

8.2 Codi dels operadors

8.2.1 Insert Client

```
# InsertClient
if len(self.left) > 0:
    cl = self.left[0]
    minim = VEnergia.loss(distance((self.clients[cl].CoordX, self.clients[cl].CoordY),
                                   (self.centrals[0].CoordX, self.centrals[0].CoordY)))
    c = 0
    for i in self.dict:
        d = VEnergia.loss(distance((self.clients[cl].CoordX, self.clients[cl].CoordY),
                                   (self.centrals[i].CoordX, self.centrals[i].CoordY)))

        if minim > d and power_left(i, self.dict, self.clients, self.centrals) >= \
                clients_power(cl, self.dict, self.clients, self.centrals, i):
            minim = d
            c = i

    yield InsertClient(cl, c)
```

8.2.A) Codi de l'operador *InsertClient*.

8.2.2 Move Client

```
# Move client
for cl in range(len(self.clients)):
    c_fin = None
    c_init = None
    if cl not in self.left:
        cons_cl = clients_power(cl, self.dict, self.clients, self.centrals)

        for c in self.dict:
            if cl in self.dict[c]:
                c_init = c

            cons_cl_fin = clients_power(cl, self.dict, self.clients, self.centrals, c)

            if cons_cl > cons_cl_fin:
                cons_cl = cons_cl_fin
                c_fin = c

    if c_fin != None and c_init != None:
        yield MoveClient(cl, c_init, c_fin)
```

8.2.B) Codi de l'operador *MoveClient*.

8.2.3 Swap State

```
# Swap State
for c in self.dict:
    exist_granted = False
    for cl in self.dict[c]:
        if self.clients[cl].Contrato == 0 and not exist_granted:
            exist_granted = True
            # If exists granted i central, it MUST be ON
            self.states[c] = True

    if not exist_granted:
        gains = 0
        ind = 0
        for cl in self.dict[c]:
            gains += VEnergia.tarifa_cliente_no_garantizada(self.clients[cl].Tipo) * self.clients[cl].Consumo
            ind += VEnergia.tarifa_cliente_penalizacion(self.clients[cl].Tipo) * self.clients[cl].Consumo

        if self.states[c]:
            gains -= VEnergia.daily_cost(self.centrals[c].Tipo)
            gains -= VEnergia.costs_production_mw(self.centrals[c].Tipo) * self.centrals[c].Produccion
        else:
            gains -= VEnergia.stop_cost(self.centrals[c].Tipo)

        if gains < 0:
            if not self.states[c]:
                pass
            else:
                coste_enc = VEnergia.daily_cost(self.centrals[c].Tipo) + (
                        VEnergia.costs_production_mw(self.centrals[c].Tipo) * self.centrals[c].Produccion)
                coste_ap = VEnergia.stop_cost(self.centrals[c].Tipo) + ind

                if coste_ap < coste_enc:
                    yield SwapState(c, False)
        else:
            yield SwapState(c, True)
    else:
        if not self.states[c]:
            yield SwapState(c, True)
```

8.2.C) Codi de l'operador *SwapState*.

8.2.4 Exchange clients

```
# Echange clients
for central in self.dict:
    for client in self.dict[central]:
        for sec_central in self.dict:
            if sec_central != central:
                for sec_client in self.dict[sec_central]:
                    if sec_client != client:
                        pl1 = power_left(sec_central, self.dict, self.clients, self.centrals)
                        pl2 = power_left(central, self.dict, self.clients, self.centrals)
                        if clients_power(client, self.dict, self.clients, self.centrals) < pl1 \
                                and clients_power(sec_client, self.dict, self.clients,
                                                  self.centrals) < pl2 and pl1 > 0 and pl2 > 0:
                            yield SwapClients(client, central, sec_client, sec_central)
```

8.2.D) Codi de l'operador *ExchangeClients*.

8.3. Codi de l'heurística

```
def heuristic(self) -> float:
    self.gains = 0

    for c in self.dict:
        for cl in self.dict[c]:
            client = self.clients[cl]
            type = client.Tipo
            consump = client.Consumo
            deal = client.Contrato

            if self.states[c] == False:
                self.gains -= VEnergia.tarifa_cliente_penalizacion(type) * consump
                self.gains += VEnergia.tarifa_cliente_no_garantizada(type) * consump
            else:
                if deal == 0:
                    self.gains += VEnergia.tarifa_cliente_garantizada(type) * consump
                else:
                    self.gains += VEnergia.tarifa_cliente_no_garantizada(type) * consump

        if self.states[c] == False:
            self.gains -= VEnergia.stop_cost(self.centrals[c].Tipo)
        else:
            self.gains -= VEnergia.daily_cost(self.centrals[c].Tipo)
            self.gains -= VEnergia.costs_production_mw(self.centrals[c].Tipo) * self.centrals[c].Produccion

    for cl in self.left:
        self.gains -= VEnergia.tarifa_cliente_penalizacion(self.clients[cl].Tipo) * self.clients[cl].Consumo
    #print(self.gains)
    return self.gains
```

8.3.A) Codi de la funció heurística.

8.4. Codi de la generació de solució inicial

8.4.1. Gen_initial_state_only_granted

```python
def gen_initial_state_only_granted(params: Parameters) -> StateRepresentation:
    """
    Function for generating the initial state.
    Assigns all guaranteed clients to one of its nearest centrals. Not guaranteed keeps out in left[list]
    Guaranteed clients must fit on all centrals.
    Quite greedy.
    """
    clients = Clientes(params.n_cl, params.propc, params.propg, params.seed)
    centrals = Centrales(params.n_c, params.seed)
    state_dict = {i: set() for i in range(len(centrals))}
    states = [True for _ in range(len(centrals))]
    # granted expression for guaranteed clients, only abbreviation stuff
    clients_granted = []
    clients_no_granted = []

    for cl in range(len(clients)):
        if clients[cl].Contrato == 0:
            clients_granted.append(cl)
        else:
            clients_no_granted.append(cl)

    for cl in clients_granted:
        lossers = []
        c = 0
        while c <= len(centrals) - 1:
            loss = VEnergia.loss(
                distance((clients[cl].CoordX, clients[cl].CoordY), (centrals[c].CoordX, centrals[c].CoordY)))
            lossers.append(loss)
            c += 1

        central = lossers.index(min(lossers))
        state_dict[central].add(cl)

    return StateRepresentation(clients, centrals, state_dict, states, clients_no_granted)
```

8.4.A) Codi de la funció generadora *gen_initial_state_only_granted*.

8.4.2. Gen_initial_state_ordered

```python
def gen_initial_state_ordered(params: Parameters) -> StateRepresentation:
    """
    Function for generating the initial state.
    Assigns all guaranteed clients in entrance order, then with not guaranteed clients.
    Guaranteed clients must fit on all centrals, if not an exception is raised.
    Not so greedy, but assigning all clients we get a fine initial solution.
    """
    clients = Clientes(params.n_cl, params.propc, params.propg, params.seed)
    clients_granted = []
    clients_no_granted = []
    centrals = Centrales(params.n_c, params.seed)
    state_dict = {i: set() for i in range(len(centrals))}
    state = [True for _ in range(len(centrals))]

    for cl in range(len(clients)):
        if clients[cl].Contrato == 0:
            clients_granted.append((cl, clients[cl]))
        else:
            clients_no_granted.append((cl, clients[cl]))

    end = False
    while len(clients_granted) > 0 and not end:
        c = 0
        placed = False
        while c < len(centrals) and not placed:
            if power_left(c, state_dict, clients, centrals) < \
                    clients_power(clients_granted[0][0], state_dict, clients, centrals, c):
                c += 1
                if c == len(centrals) - 1:
                    end = True
            else:
                state_dict[c].add(clients.index(clients_granted[0][1]))
                c += 1
                placed = True
                clients_granted.pop(0)
```

Continua a la següent pàgina...

```
    end = False
    while len(clients_no_granted) > 0 and not end:
        c = 0
        placed = False
        while c < len(centrals) and not placed:
            if power_left(c, state_dict, clients, centrals) < \
                    clients_power(clients_no_granted[0][0], state_dict, clients, centrals, c):
                c += 1
                if c == len(centrals) - 1:
                    end = True
            else:
                state_dict[c].add(clients.index(clients_no_granted[0][1]))
                c += 1
                placed = True
                clients_no_granted.pop(0)

    if clients_granted:
        raise Exception("Not a valid initial state")
    if clients_no_granted:
        aux = []
        for cl in clients_no_granted:
            aux.append([cl[1].Consumo, cl[0], cl[1]])
        aux.sort()

        for i in aux[::-1]:
            clients_no_granted.remove((i[1], i[2]))  # clients_no_granted.index((i[1],i[2]))
            clients_no_granted.insert(0, i[1])

        return StateRepresentation(clients, centrals, state_dict, state, clients_no_granted)

    return StateRepresentation(clients, centrals, state_dict, state)
```

8.4.B) Codi de la funció generadora *gen_initial_state_ordered.*

8.5. Resultats de l'experimentació

8.5.1 Resultats experiment 1

Rèplica	Llavor	Beneficis Inicials(€)	Beneficis finals(€)	Temps mitjà(s)
1	399	64245	106675	18.63
2	289	44754	89494	21.98
3	393	66781	108051	26.06
4	387	67235	114470	24.85
5	410	51036	98146	24.53
6	591	62099	99839	22.68
7	906	57689	93274	20.63
8	986	52649	97724	25.08
9	31	58729	108058	26.98
10	51	50899	90614	21.39

8.5.1A) Taula de resultats de la combinació d'operadors: Insert Client + Swap State + Move Client

Rèplica	Llavor	Beneficis Inicials(€)	Beneficis finals(€)	Temps mitjà(s)
1	399	64245	103260	21.24
2	289	44754	86634	22.95
3	393	66781	108051	26.17
4	387	67235	111340	21.4
5	410	51036	94311	23.65
6	591	62099	98244	25.56
7	906	57689	90534	22.97
8	986	52649	94799	26.04
9	31	58729	101304	24.04
10	51	50899	86279	20.07

8.5.1B) Taula de resultats de la combinació d'operadors: Insert Client + Move Client

Rèplica	Llavor	Beneficis Inicials	Beneficis finals	Temps mitjà
1	399	64245	106675	0.66
2	289	44754	89494	0.51
3	393	66781	108051	0.49
4	387	67235	114470	0.56
5	410	51036	98061	0.67
6	591	62099	99839	0.52
7	906	57689	93274	0.49
8	986	52649	97724	0.66
9	31	58729	107793	0.58
10	51	50899	90614	0.51

8.5.1C) Taula de resultats de la combinació d'operadors: Insert Client + Swap State

Rèplica	Llavor	Beneficis Inicials	Beneficis finals	Temps mitjà
1	399	64245	67800	0.28
2	289	44754	47614	0.21
3	393	66781	66781	0.11
4	387	67235	70385	0.6
5	410	51036	55026	0.52
6	591	62099	63769	0.26
7	906	57689	60429	0.43
8	986	52649	55634	0.51
9	31	58729	65793	0.57
10	51	50899	55234	0.55

8.5.1D) Taula de resultats de la combinació d'operadors: Move Client + Swap State

Rèplica	Llavor	Beneficis Inicials	Beneficis finals	Temps mitjà
1	399	64245	64245	190,71
2	289	44754	44754	252,00
3	393	66781	66781	137,65
4	387	67235	67235	214,94
5	410	51036	51036	219,50
6	591	62099	62099	264,61
7	906	57689	57689	210,89
8	986	52649	52649	199,52
9	31	58729	58729	129,69
10	51	50899	50899	303,91

8.5.1E) Taula de resultats de la combinació d'operadors: Exchange Clients

8.5.2. Resultats experiment 2

Llavor	Beneficis inicials	Beneficis finals	Diff beneficis	Temps promig
245	47550	90325	42775	0,64
39	57694	99759	42065	0,62
678	60555	108870	48315	0,73
1345	53973	90703	36730	0,59
239	54936	103596	48660	0,76
29	52292	93937	41645	0,57
568	53149	106379	53230	0,64
1422	69687	113187	43500	0,56
991	52866	96886	44020	0,68
132	55771	105511	49740	0,68

8.5.2A) Taula de resultats de la funció generadora Only Granted

Llavor	Beneficis inicials	Beneficis finals	Diff beneficis	Temps promig
245	88405	99810	11405	1,1
39	99759	106219	6460	1,13
678	69890	110875	40985	2,53
1345	89933	105923	15990	1,1
239	99186	110481	11295	1,1
29	93302	107317	14015	1,17
568	98864	111654	12790	1,33
1422	89747	114882	25135	2,01
991	93026	108221	15195	1,12
132	96411	108811	12400	1,56

8.5.2B) Taula de resultats de la funció generadora Ordered

8.5.3 Resultats experiment 3

Constant Proporcional (k)	Exponent (lam)	Num Iteracions (limit)	Beneficis Inicials	Beneficis Finals	Diferencia Beneficis	Temps promig (10 iteracions)
5	0.0001	200	53233	76013	22780	0,766
70	0.5	10	53233	57188	3955	134
50	0.1	50	53233	59388	6155	0,451
35	0.05	87	53233	62238	9005	0,519
30	0.01	128	53233	66488	13255	0,671
20	0.01	170	53233	71263	18030	0,972
20	0.01	200	53233	76223	22990	1,293
10	0.001	220	53233	79328	26095	0,72
5	0.0001	240	53233	82493	29260	0,755
20	0.0001	200	53233	76013	22780	0,823
5	0.01	200	53233	76223	22990	0,735

8.5.3.A) Taula de resultats per la llavor 22

Constant Proporcional (k)	Exponent (lam)	Num Iteracions (limit)	Beneficis Inicials	Beneficis Finals	Diferencia Beneficis	Temps promig (10 iteracions)
20	0.5	200	57315	70520	13205	1,049
70	0.5	10	57315	62770	5455	0,126
50	0.1	50	57315	63790	6475	0,352
35	0.05	87	57315	65200	7885	0,495
30	0.01	128	57315	66030	8715	0,907
20	0.01	170	57315	69810	12495	1,557
20	0.01	200	57315	70120	12805	1,684
10	0.001	220	57315	71860	14545	0,843
5	0.0001	240	57315	73030	15715	0,863
5	0.0001	200	57315	70840	13525	0,972
5	0.001	200	57315	70300	12985	0,831

8.5.3.B) Taula de resultats per la llavor 1234

Constant Proporcional (k)	Exponent (lam)	Num Iteracions (limit)	Beneficis Inicials	Beneficis Finals	Diferencia Beneficis	Temps promig (10 iteracions)
20	0.5	200	55415	77625	22210	1,226
70	0.5	10	55415	61185	5770	0,141
50	0.1	50	55415	63385	7970	0,42
35	0.05	87	55415	66095	10680	0,595
30	0.01	128	55415	70315	14900	0,864
20	0.01	170	55415	74685	19270	1,039
20	0.01	200	55415	77220	21805	2,059
10	0.001	220	55415	79110	23695	0,767
5	0.0001	240	55415	80845	25430	0,812
5	0.0001	200	55415	77895	22480	1,296
5	0.001	200	55415	77490	22075	0,785

8.5.3.C) Taula de resultats per la llavor 52

Constant Proporcional (k)	Exponent (lam)	Num Iteracions (limit)	Beneficis Inicials	Beneficis Finals	Diferencia Beneficis	Temps promig (10 iteracions)
20	0.5	200	66781	87576	20795	0,686
70	0.5	10	66781	67331	550	0,128
50	0.1	50	66781	69531	2750	0,407
35	0.05	87	66781	73356	6575	0,563
30	0.01	128	66781	77746	10965	0,789
20	0.01	170	66781	82606	15825	1,251
20	0.01	200	66781	87576	20795	1,855
10	0.001	220	66781	93246	26465	0,943
5	0.0001	240	66781	102976	36195	0,898
5	0.0001	200	66781	77895	11114	1,32
5	0.001	200	66781	87576	20795	0,841

8.5.3.D) Taula de resultats per la llavor 393

Constant Proporcional (k)	Exponent (lam)	Num Iteracions (limit)	Beneficis Inicials	Beneficis Finals	Diferencia Beneficis	Temps promig (10 iteracions)
20	0.5	200	54553	91195	36642	0,724
70	0.5	10	54533	70770	16237	0,141
50	0.1	50	54533	72970	18437	0,376
35	0.05	87	54533	76770	22237	0,457
30	0.01	128	54533	81000	26467	0,531
20	0.01	170	54533	87025	32492	1,286
20	0.01	200	54533	91685	37152	2,847
10	0.001	220	54533	93645	39112	0,953
5	0.0001	240	54533	98510	43977	0,805
5	0.0001	200	54533	91440	36907	0,822
5	0.001	200	54533	91685	37152	0,938

8.5.3.E) Taula de resultats per la llavor 387

Constant Proporcional (k)	Exponent (lam)	Num Iteracions (limit)	Beneficis Inicials	Beneficis Finals	Diferencia Beneficis	Temps promig (10 iteracions)
20	0.5	200	52165	71765	19600	0,796
70	0.5	10	52165	54790	2625	0,31
50	0.1	50	52165	56990	4825	0,422
35	0.05	87	52165	59165	7000	0,696
30	0.01	128	52165	63355	11190	0,804
20	0.01	170	52165	67755	15590	1,584
20	0.01	200	52165	71765	19600	1,688
10	0.001	220	52165	75365	23200	0,86
5	0.0001	240	52165	80425	28260	0,792
5	0.0001	200	52165	71765	19600	0,682
5	0.001	200	52165	71765	19600	0,843

8.5.3.F) Taula de resultats per la llavor 672

Constant Proporcional (k)	Exponent (lam)	Num Iteracions (limit)	Beneficis Inicials	Beneficis Finals	Diferencia Beneficis	Temps promig (10 iteracions)
20	0.5	200	59641	76651	17010	0,824
70	0.5	10	59641	64901	5260	0,135
50	0.1	50	59641	67091	7450	0,517
35	0.05	87	59641	70016	10375	0,549
30	0.01	128	59641	72756	13115	0,901
20	0.01	170	59641	74816	15175	1,824
20	0.01	200	59641	75016	15375	0,865
10	0.001	220	59641	77056	17415	0,808
5	0.0001	240	59641	77326	17685	0,776
5	0.0001	200	59641	75571	15930	0,79
5	0.001	200	59641	75126	15485	0,871

8.5.3.G) Taula de resultats per la llavor 812

8.5.4 Resultats Experiment 4

	Llavors											
Nºcentrals	134	34	556	9012	4321	12	346	762	987	222	Mitja	Desv
40	0,7	0,65	0,44	0,48	0,56	0,49	0,46	0,48	0,45	0,49	0,52	0,863611
80	0,98	0,83	0,82	0,71	0,85	0,67	0,66	0,81	0,7	0,75	0,778	1,476377
122	1,88	2,18	1,84	2,36	1,69	1,86	1,86	2	1,63	1,8	1,91	1,25209
162	4,81	5,15	4,13	4,61	4,78	3,9	4	4,58	3,92	4,14	4,402	1,619691
200	9,19	10,14	8,99	9,17	9,35	7,97	8	9,21	8,27	8,95	8,924	2,681201
242	16,45	17,23	15,32	16,24	15,56	14,48	14,42	15,97	15,28	15,63	15,658	0,86425
282	25,53	27,03	24,71	25,76	24,75	23,32	23,17	25,03	22,95	25,25	24,75	1,288332
324	38,2	40,28	36,88	38,13	36,62	35,19	35,2	37,78	34,58	37,27	37,013	1,721957
364	55,49	55,47	51,42	54,31	50,93	48,82	51,84	51,84	49,61	53,09	52,282	2,292669
404	70,86	74,03	68,44	70,76	67,69	63,91	65	67,07	65,24	71,44	68,444	3,274831
444	94,15	97,85	88,73	92,73	88,03	84,19	85,12	87,37	83,75	88,04	88,996	4,583825
484	112,48	116,77	108,37	112,38	107,03	104,19	106,16	108,29	105,55	110,53	109,175	3,856142
526	139,76	145,2	134,88	139,7	132,93	130,22	132,46	134,34	131,85	137,81	135,915	4,622035
566	169,17	175,28	163,69	168,92	160,99	159,62	163,24	165,83	163,01	169,71	165,946	4,786043
606	207,32	210,51	195,35	201,74	191,81	189,69	194,08	198,03	196,58	199,78	198,489	6,58201

8.5.4.A) Taula de resultats de l'experiment 4

	Llavors											
Nºcentrals	134	34	556	9012	4321	12	346	762	987	222	Mitja	Desv
40	16,92	18,26	17,87	17,64	16,12	17,07	18,89	17,11	17,12	18,67	17,57	0,86
80	30,9	33	29,17	30,77	27,96	29,11	29,12	27,96	28,96	29,28	29,60	1,48
122	48,19	50,09	46,61	48,54	46,9	45,97	46,79	46,64	46,35	47,44	47,35	1,25
162	71,54	73,89	68,84	71,49	69,68	68,23	69,84	70,75	69,77	71,17	70,52	1,62
202	102,49	103,36	97,27	99,54	97,23	94,82	96,23	99,11	97,14	98,89	98,61	2,68
242	133,77	137,22	130,75	133,8	129,85	128,01	129,27	132,06	128,39	131,95	131,51	2,86
282	174,23	177,89	170,37	173,84	169,53	167,21	168,79	171,14	167,34	172,13	171,25	3,36
324	222,28	226,57	218,35	222,67	217,24	214,86	215,37	219,12	214,58	221,26	219,23	3,94
364	275,67	280,85	271,27	275,88	269,17	265,96	267,82	271,43	266,61	273,64	271,83	4,73

8.5.4.B) Taula de resultats de l'experiment 4 extra

8.5.5 Resultats experiment 5

8.5.5.1 Hill climbing

Penalització	Vàlid	Persones no subministrades	Temps promig	diff beneficis
0	False	100	0,03	14020
200	False	7	0,07	14305
400	False	6	0,08	14980
600	False	6	0,08	14980
800	False	6	0,07	14980
1000	False	7	0,09	14305
1200	False	7	0,08	14305
1400	False	6	0,08	14980
1600	False	7	0,11	14305
1800	False	11	0,10	12165

8.5.5.A) Taula de resultats llavor 274

Penalització	Vàlid	Persones no subministrades	Temps Promig	Diff beneficis
0	False	100	0,03	15402
200	False	6	0,07	16065
400	False	6	0,08	16065
600	False	6	0,06	16065
800	False	6	0,07	16065
1000	False	6	0,07	16065
1200	False	6	0,07	16065
1400	False	97	0,09	14187
1600	False	97	0,12	14187
1800	False	97	0,12	14187

8.5.5.B) Taula de resultats llavor 691

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	100	0,02	14861
200	False	9	0,07	15520
400	False	9	0,07	15520
600	False	9	0,03	15520
800	False	99	0,03	14216
1000	False	99	0,03	14216
1200	False	99	0,02	14216
1400	False	99	0,03	14216
1600	False	99	0,03	14216
1800	False	99	0,03	14216

8.5.5.C) Taula de resultats llavor 356

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	100	0,02	14557
200	False	4	0,08	15190
400	False	4	0,08	15190
600	False	4	0,08	15190
800	True	0	0,08	18250
1000	True	0	0,11	18250
1200	True	0	0,09	18250
1400	True	0	0,11	18250
1600	False	99	0,08	13057
1800	False	99	0,13	13057

8.5.5.D) Taula de resultats llavor 732

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	100	0,03	15164
200	False	100	0,03	15164
400	False	100	0,03	15164
600	False	100	0,03	15164
800	False	100	0,04	15164
1000	False	100	0,04	15164
1200	False	100	0,04	15164
1400	False	100	0,03	15164
1600	False	100	0,03	15164
1800	False	100	0,03	15164

8.5.5.E) Taula de resultats llavor 285

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	100	0,03	17017
200	False	8	0,08	17253
400	False	8	0,08	17253
600	False	8	0,06	17253
800	False	8	0,08	17253
1000	False	8	0,08	17253
1200	False	8	0,08	17253
1400	False	3	0,09	20408
1600	True	0	0,08	22783
1800	False	96	0,04	12463

8.5.5.F) Taula de resultats llavor 22

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	100	0,03	12751
200	False	10	0,06	13415
400	False	10	0,07	13415
600	False	10	0,08	13415
800	False	12	0,07	12535
1000	False	12	0,09	12535
1200	False	12	0,08	12535
1400	False	12	0,10	12535
1600	False	12	0,09	12535
1800	False	9	0,11	13955

8.5.5.G) Taula de resultats llavor 467

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	100	0,02	15008
200	False	11	0,08	15340
400	False	11	0,06	15340
600	False	11	0,07	15340
800	False	11	0,06	15340
1000	False	11	0,07	15340
1200	False	11	0,08	15340
1400	False	14	0,07	13750
1600	False	14	0,10	13750
1800	False	18	0,10	11870

8.5.5.H) Taula de resultats llavor 89

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	100	0,04	13149
200	False	6	0,07	13355
400	False	6	0,08	13355
600	False	6	0,09	13355
800	True	0	0,10	17105
1000	True	0	0,10	17105
1200	True	0	0,11	17105
1400	True	0	0,11	17105
1600	True	0	0,11	17105
1800	True	0	0,14	17105

8.5.5.I) Taula de resultats llavor 123

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	100	0,03	14664
200	False	2	0,08	15195
400	False	2	0,06	15195
600	False	1	0,07	16050
800	False	1	0,07	16050
1000	False	1	0,10	16050
1200	True	0	0,08	16905
1400	True	0	0,12	16905
1600	True	0	0,09	16905
1800	True	0	0,09	16905

8.5.5.J) Taula de resultats llavor 34

8.5.5.2 Simulated annealing

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	63	0,08	12930
200	False	74	0,11	13495
400	False	62	0,08	14655
600	False	59	0,09	13320
800	False	74	0,08	15530
1000	False	52	0,11	14110
1200	False	63	0,11	16490
1400	False	50	0,10	13335
1600	False	68	0,08	14045
1800	False	60	0,11	13950

8.5.5.A) Taula de resultats llavor 34

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	50	0,08	18952
200	False	54	0,08	16934
400	False	48	0,10	17857
600	False	57	0,10	18272
800	False	55	0,10	17177
1000	False	50	0,11	17657
1200	False	55	0,09	18472
1400	False	55	0,08	17177
1600	False	52	0,10	18772
1800	False	66	0,07	17452

8.4.6.B) Taula de resultats llavor 691

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	54	0,10	17511
200	False	62	0,10	16681
400	False	61	0,07	16811
600	False	56	0,10	16526
800	False	70	0,10	16541
1000	False	62	0,08	16681
1200	False	57	0,07	17211
1400	False	63	0,09	16581
1600	False	58	0,11	17111
1800	False	61	0,10	17401

8.4.6.C) Taula de resultats llavor 356

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	60	0,10	15447
200	False	61	0,09	15347
400	False	53	0,11	17627
600	False	57	0,10	15757
800	False	50	0,07	17907
1000	False	61	0,10	15347
1200	False	61	0,10	15347
1400	False	56	0,08	15857
1600	False	54	0,09	16057
1800	False	61	0,10	16847

8.4.6.D) Taula de resultats llavor 732

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	59	0,07	15224
200	False	61	0,07	17944
400	False	48	0,11	15494
600	False	65	0,07	17564
800	False	54	0,10	14128
1000	False	58	0,10	18224
1200	False	63	0,09	17744
1400	False	61	0,08	14994
1600	False	63	0,10	17744
1800	False	64	0,08	17664

8.4.6.E) Taula de resultats llavor 285

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	56	0,10	18778
200	False	59	0,10	18498
400	False	62	0,11	18178
600	False	62	0,10	19407
800	False	52	0,11	20307
1000	False	65	0,09	19127
1200	False	64	0,08	19227
1400	False	65	0,10	19127
1600	False	58	0,10	18102
1800	False	60	0,08	16703

8.4.6.F) Taula de resultats llavor 22

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	56	0,11	10102
200	False	63	0,08	13911
400	False	65	0,08	13138
600	False	63	0,11	15551
800	False	75	0,08	12228
1000	False	62	0,10	15651
1200	False	69	0,11	15011
1400	False	72	0,11	14731
1600	False	65	0,08	15371
1800	False	62	0,08	13448

8.4.6.G) Taula de resultats llavor 467

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	66	0,10	14976
200	False	62	0,10	17658
400	False	73	0,08	14256
600	False	69	0,11	14772
800	False	66	0,09	17318
1000	False	70	0,08	16918
1200	False	70	0,09	16918
1400	False	73	0,10	16638
1600	False	72	0,10	14336
1800	False	55	0,11	14661

8.4.6.H) Taula de resultats llavor 89

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	49	0,11	13564
200	False	60	0,09	13999
400	False	57	0,11	14339
600	False	53	0,10	14719
800	False	56	0,09	14439
1000	False	55	0,10	12934
1200	False	66	0,09	15089
1400	False	59	0,09	14109
1600	False	68	0,10	13179
1800	False	56	0,07	14439

8.4.6.I) Taula de resultats llavor 123

Penalització	Vàlid	Persones no subministrades	Temps promig	Diff beneficis
0	False	57	0,10	14599
200	False	64	0,11	16904
400	False	66	0,11	13699
600	False	50	0,08	15279
800	False	56	0,10	14699
1000	False	62	0,10	17104
1200	False	55	0,10	17764
1400	False	53	0,10	17944
1600	False	63	0,10	17004
1800	False	49	0,10	14019

8.4.6.J) Taula de resultats llavor 34