PRÀCTICA DE CERCA

Bernat Borràs Civil, Alexandre Ros i Roger i Oscar Ramos Núñez

Índex

1 Introducció al problema

En aquesta pràctica d'Intel·ligència Artificial, se'ns demana dissenyar i implementar un algorisme de cerca per a tal d'optimitzar un problema proposat. A l'enunciat de la pràctica s'explica en detall els costs i detalls del problema, així que nosaltres presentarem un breu resum d'aquest problema més enfocat per a la justificació de les decisions d'implementació que hem pres.

En resum, tenim un conjunt de *Clients* i *Centrals* amb posicions enteres sobre un tauler *100x100.* El nostre deure és assignar els clients a les centrals amb l'objectiu de maximitzar el benefici final. A més a més, se'ns presenten diferents tipus de clients i de centrals, que modifiquen l'energia que consumeixen, que generen, i també els beneficis i pèrdues que provoquen.

D'entre aquestes classificacions, la més important amb diferència tracta sobre la garantia dels clients. En una solució del problema, cada client és o bé *Garantit* o no. Si ho és, llavors aquest client no pot quedar-se sense central. En canvi, els clients *no-Garantits* poden quedar-se sense centrals. En aquest cas, a canvi d'una indemnització.

També destaquem el fet que una central, si subministra a qualsevol nombre de clients, aquesta produeix tota l'energia que pot produir independentment del nombre de clients o del consum dels clients. Per tant, cal aprofitar tota aquesta energia intentant que les centrals estiguin o bé quasi plenes, o bé completament buides. Així, si una central no està subministrant cap client, aquesta pot estalviar-se el cost de tenir-la encesa.

Finalment, volem comentar que si assignem una central molt llunyana a un client, hi ha una pèrdua d'energia proporcional i la central haurà de produir més per a aquell client. Això evidentment suposa una pèrdua de capacitat total i pot arribar a ser problemàtic a l'hora de generar una solució inicial, com veurem més endavant. Per tant, idealment voldriem evitar aquesta situació.

Plantejar aquest problema com a un problema de cerca local és adequat i ideal, doncs ens demanen trobar una solució final maximitzant un criteri, el del benefici, sense haver-nos de preocupar per si trobem l'òptim o no. A més a més, com veurem més endavant, tota la informació del problema podrà ser representada fàcilment i eficientment per a la generació d'altres solucions i l'avaluació de l'heurístic.

2 Representació d'un estat

Evidentment d'alguna manera haurem de representar tota la informació que descriu un cert estat del problema. El nostre primer i principal atribut que descriurà l'estat és una *array* d'enters, on *array[c]* = *k* és la central a la qual està assignada el client *c*. Si *k* = *-1* llavors el client *c* no està assignat a cap central. Aquesta array s'anomena *state* al nostre codi.

Els identificadors enters de centrals i clients són l'*offset* d'aquests en els objectes *Clientes* i *Centrales*. Per conveni propi, al nostre codi, emprarem la variable *c* per a referir-nos al client *c* i la variable *k* per a referir-nos al client *k.*

Aquesta informació és suficient per a la representació d'un estat. Tot i així, quan calculem l'heurístic, ens interessarà mantenir en cada moment informació clau per a no haver-la de recalcular cada vegada que generem un nou estat; això seria molt ineficient. Així, afegirem més atributs que actualitzaran els seus valors quan s'executin els operadors.

Per tant, també mantindrem els següents atributs que descriurem i justificarem seguidament:

- *double benefici:* El benefici actual de l'estat.

- *double distance*: La suma de les distàncies de cada client a la seva central. Si un client no està assignat a cap central, per tal de penalitzar aquest fet, sumarem la distància entre dues cantonades del tauler (és a dir, $100\sqrt{2}$ unitats)

- *double energy*: La suma de les energies restants / sobrants per a cada central. Així i tot, per tal d'incentivar que hi hagi centrals buides i estalviar-nos els costos de mantenir-les enceses, les centrals sense clients no es tindran en compte.

- *int [ ] clientsXcentral:* El nombre de clients actuals en una certa central. Aquest atribut ens serà útil per a determinar si una central estava buida i s'hi afegeix el primer client, o bé si després d'una desassignació una central deixa d'estar operativa.

- *double [ ] energyleft:* L'energia restant / sobrant en una certa central. Si una central està buida l'energia restant serà tota la disponible. Aquest atribut és vital i imperiós per tal de decidir amb un cost raonable si un operador es pot aplicar o si, al aplicar-lo, es vulneraria alguna restricció del problema *(per exemple, si la central no pot suportar un altre client, l'energia restant seria inferior a zero).*

- *int nonAssignedG:* Nombre de clients amb contracte "Garantit" que, de moment, no tenen central. Aquest enter, tot i que sembla ser inútil doncs una solució sempre hauria de tenir *nonAssignedG = 0,* ens serà útil quan implementem estats inicials que no siguin solucions i penalitzem fortament aquest fet en l'heurístic. Per a l'experiment 5 aquest atribut serà essencial.

  Donat que tenim en total *C* clients i *K* centrals, no és difícil de comprovar que l'espai de solucions està limitat per $O((K+1)^C)$. Tot i així, moltes solucions segurament no seran solucions, doncs poden violar les restriccions del problema.

3 Operadors de cerca

3.1 Els operadors i les seves funcions

Per a moure-n's dins l'espai de solucions, necessitem aplicar operadors als nostres estats que ens permetin explorar tot l'espai de solucions. Aquests operadors també hauran de cohesionar bé amb els heruístics que dissenyarem posteriorment.

Al primer experiment de tots, s'analitzarà quina combinació d'aquests operadors és la millor segons no només el benefici final, sinó també el cost temporal i factor de ramificació de cada operador. Nosaltres hem implementat dos operadors (en veritat són tres) que, si s'empren conjuntament, permeten perfectament explorar tot l'espai de solucions:

- *assign (int c, int k)*: L'operador més bàsic de tots; el d'assignar el client *c* a la central *k.* Al aplicar-se l'operador, el client *c* passarà d'estar assignat de la seva central anterior (possiblement nul·la) a la central *k*. Aquest operador és doble, doncs si *k = -1,* es tradueix en un *deassign(int c)* que deixarà el client *c* sense central.

  Aquest últim operador, tot i que sembla contradictori amb els nostres objectius (que és el de maximitzar el benefici) i completament inútil amb Hill Climbing, podria arribar a prendre's si, per exemple, el client *c* és l'últim en una central i deixar el client sense central augmenti el benefici, i el client no es pugui assignar a cap altre central.

  S'ha implementat la funció *boolean canAssign(int c, int k)*, que serveix de precondició per a aplicar l'operador *assign* i serà feta servir per les classes *SuccessorFunction*. Si després de fer l'*assign* es violés alguna restricció del nostre problema, el *canAssign* retornaria *false.* També poda alguns casos sense sentit, com per exemple assignar a un client la central que té actualment assignada.

El factor de ramificació de l'operador *assign* (+ *deassign)* és de l'ordre $\Theta(CK + C)$, on *C* és el nombre de clients totals i *K* el nombre de centrals.

- *swap (int c1, int c2):* El client *c1* passa a tenir la central de *c2*, i el client *c2* passa a tenir la central de *c1*. Al aplicar-se aquest operador, immediatament els clients s'intercanvien les centrals, que poden ser nul·les (*-1)*. Si, per exemple, el client *c1* no tenia central i el client *c2* tenia una central *k*, el client *c1* passarà a estar assignat a la central *k* i el client *c2* passarà a no tenir central.

  Aquest operador ens serà útil i necessari per tal de moure clients a diferents centrals sense empitjorar l'heurístic a mig camí, sobretot amb l'heurístic que dependrà d'*energy*. També és necessari per si volem explorar tot l'espai de solucions, doncs es podria donar el cas on moure qualsevol client sobrepassés l'energia total d'una central, però moure'n dos alhora no ho fes i fos millor.

  Similarment, s'ha implementat la funció booleana *canSwap(int c1, int c2)*, que comprovarà si l'operador *swap* vulneraria alguna restricció del nostre problema. També poda alguns swaps que no tenen sentit, com per exemple un swap on *c1* = *c2*, o un swap on els dos clients tenen la mateixa central.

  El factor de ramificació d'aquest operador és de $\Theta(\frac{C\,(C-1)}{2})$, doncs fer un *swap* entre els clients *i, j* és el mateix que fer un *swap* entre els clients *j, i*.

3.2 Les funcions d'aplicabilitat

Abans de poder aplicar un operador i modificar l'estat, la funció successora haurà de comprovar, mitjançant les nostres funcions d'aplicabilitat, si aquests operadors es poden dur a terme sense violar cap restricció del nostre problema. Les nostres funcions booleanes *(true* si es pot aplicar, *false* si no es pot aplicar) són les següents:

- *boolean canAssign(int c, int k):* Funció d'aplicabilitat de l'assignació (assign) i la desassignació (deassign). Aquest operador retornarà *cert* excepte en els casos on:
    - El client *c* té un contracte *Garantit* i se'l vol desassignar (*k* = -1).
    - La planta *k* no tingui suficient energia sobrant per a mantenir el client *c.*
    - El client *c* no té cap central actualment i li volem desassignar.
    - El client *c* ja està assignar a la central *k*.

- *boolean canSwap(int c1, int c2):* Funció d'aplicabilitat del *swap* / intercanvi entre dos clients. Aquest operador retornarà *cert* excepte en els casos on:
    - Ni *c1* ni *c2* tenen una central actualment assignada.
    - El client *c1* és *Garantit* i *c2* no té cap central assignada.
    - El client *c2* és *Garantit* i *c1* no té cap central assignada.
    - El client *c1* i el client *c2* estan assignats a la mateixa central.
    - Alguna de les (màxim) dues centrals, després del *swap*, sobrepassarà la seva producció d'energia.

4 Estratègies de generació de solucions inicials

Per a aquest problema nosaltres hem plantejat diverses estratègies per a la generació de solucions inicials, de les quals hem decidit implementar-ne finalment dues després de descartar-ne algunes que finalment no generaven sempre solucions. Tot i així, com veurem més endavant, és possible que una de les nostres funcions generadores de solucions no sigui capaç de generar una solució sencera. Això ho penalitzarem més endavant amb l'heurístic.

Cal recordar que una solució inicial haurà de preservar les restriccions del problema, és a dir, haurà d'assegurar que:

- Tots els clients *Garantits* tenen assignada una central.
- Cap central sobrepassa la seva capacitat màxima.

Començarem comentant l'estratègia menys informada que hem implementat, i després explicarem perquè aquesta estratègia pot no donar solució i seguidament introduïrem la que nosaltres creiem que probablement és la millor.

La primera funció s'anomena *randomAssignGarantit* (en el codi, *initialState2).* Aquesta primera implementació assigna centrals aleatòries només als clients *Garantits*. El codi és simple i fàcil d'implementar. En pseudicodi es podria descriure de la següent manera:

```
For every client c:
     Let k be a randomly generated power plant
     If k can provide sufficient energy to c:
          assign(c, k)
     Else, generate a new k and check again
```

A simple vista, pot semblar que aquesta estratègia bàsica sigui adient, doncs al final de tot els operadors optimitzaran el benefici i aquesta estratègia ens genera qualsevol solució vàlida. També és una funció prou eficient, doncs si ho implementem de tal forma que *k* no es pugui generar dues vegades en una mateixa iteració, té un cost temporal en el cas pitjor de $O(GK)$ on *G* és el nombre total de clients *Garantits* i *K* el nombre de centrals.

Tot i així, té un greu problema que haurem d'atacar i que ens motivarà a implementar la nostra segona funció generadora de solucions inicials.

El problema del *randomAssignGarantit* és que, com que assignem els clients a centrals completament aleatòries, segurament hi hagi pèrdues d'energia importants i poc desitjades. Això a priori no seria un greu problema. El problema important sorgeix quan, a causa de les pèrdues i de la sobreproducció d'energia per part de les centrals, arribem a un punt on a un client *Garantit* no li podem assignar cap central sense que aquesta sobreeiximeixi la seva producció màxima.

Aquest efecte es pot observar amb el primer expermiment de tots, on el nombre d'energia produïda total és bastant limitada i la majoria dels clients són amb contracte *Garantit*. La funció assignarà centrals a clients aleatòriament, fins que arribi un client *Garantit* que desgraciadament no podem servir. Amb una distància de 75 unitats al tauler (aproximadament mig tauler), tenim pèrdues del 40%. En els pitjors casos aquestes pèrdues signifiquen que els clients *Garantits* inicialment restaran de les centrals el doble d'energia de la que en veritat necessiten.

Per tant, proposem la segona funció generadora de solucions, aquesta vegada més informada: la *closestAssignGarantit* (o *initialState* en el codi). Aquesta funció assigna, per a cada client *Garantit*, la central disponible que té més aprop. D'aquesta manera, eliminem completament el problema de les pèrdues.

Tot i així, això requereix ordenar, per a cada client, totes les centrals de més properes a més llunyanes. Per aquesta raó inicialitzem la matriu d'enters *closestCentrals*. En aquesta matriu, *closestCentrals [c] [i] = k* si la central *k* és la i-èsima més propera al client *c*. Per a fer l'ordenació hem implementat el *mergesort*, doncs un algorisme d'ordenació lineal com el *counting sort* no es pot emprar al haver-hi distàncies no enteres.

En pseudocodi, la funció *closestAssignGarantit* es descriuria de la següent manera:

```
For every client c:
     Sort closestCentrals[c] by their distance from c
     Let i = 0
     While not canAssign(c, closestCentrals[c][i]):
          Increment i

     assign(c, closestCentrals[c][i])
EndFor
```

El cost temporal en cas pitjor de tot l'algorisme de generació és el d'ordenar, $O(G\,K\,log\,K)$, on *G* és el nombre de clients *Garantit* i *K* és el nombre de centrals.

Aquesta funció evidentment resol els problemes de les pèrdues, doncs al assignar la central més propera al client aquestes pèrdues són mínimes i, per tant, no correm el risc de generar una solució a mitges.

Això no significa, però, que no podem utilitzar *randomAssignGarantit*, doncs a l'heurístic hem implementat una penalització significant depenent del nombre de clients *Garantits* sense assignar, i doncs podem fer una exploració per solucions parcials també. La comparació entre les dues estratègies es farà al segon experiment.

5 Funcions Heurístiques

Probablement el factor més important a l'hora de dissenyar un algorisme de cerca local és la tria de l'heurístic i el seu funcionament amb els operadors dissenyats. Nosaltres hem, durant l'etapa d'implementació, dissenyat i provat diferents heurístics que depenen dels següents atributs, explicats també a la pàgina cinc d'aquest document:

- *double benefici:* El benefici actual de l'estat.

- *double distance*: La suma de les distàncies de cada client a la seva central. Si un client no està assignat a cap central, per tal de penalitzar aquest fet, sumarem la distància entre dues cantonades del tauler (és a dir, $100\sqrt{2}$ unitats)

- *double energy*: La suma de les energies restants / sobrants per a cada central. Així i tot, per tal d'incentivar que hi hagi centrals buides i estalviar-nos els costos de mantenir-les enceses, les centrals sense clients no es tindran en compte.

- *int nonAssignedG:* Nombre de clients amb contracte "Garantit" que, de moment, no tenen central.

Jugant amb aquests quatre atributs, que s'actualitzen quan s'aplica un operador, podrem generar diferents heurístics, alguns dels quals generen solucions millors que altres heurístics.

Segudiament, analitzarem tres diferents heurístics que hem provat amb *Hill Climbing* i comentarem els punts forts i febles dels tres.

- *Heurístic Benefici:* Aquest és l'heurístic naïf, doncs és el mateix que la funció calitat del problema. L'heurístic es calcula com `-benefici`, doncs cal recordar que l'heurístic serà s'intenta minimitzar, i volem maximitzar el benefici. Aquest heurístic no és gens òptim, però, doncs arriba un punt on amb el *Hill Climbing* no es pot aplicar cap operador. Això és degut a que un *swap* sempre conservarà el mateix benefici.

- *Heurístic Multiplicació:* Aquest heurístic considera l'energia sobrant de les centrals i també el benefici. L'heurístic es calcula com `energy * (-benefici)`, doncs cal recordar que volem maximitzar el benefici.

  Aquest heurístic és més útil i informat que l'anterior, doncs ens permet aplicar els operadors de *swap*. Tot i així, multiplicar dos símbols qualssevol que volem minimitzar és probable que no funcioni, i com veiem experimentalment el benefici final és menor que el que obtindrem amb el tercer heurístic. A més a més, tant aquest heurístic com l'anterior no penalitzen l'atribut *nonAssignedG*, que és necessari si permetem que a l'estat inicial hi hagi clients *Garantits* no assignats.

- *Heurístic Logarítmic i Penalitzador:* Aquest heurístic té en compte les distàncies client-central, l'energia i penalitza estats que no són solucions. L'heurístic es calcula amb la fórmula:

  ```
  (distance * log(distance) + A*energy)*(nonAssignedG + 1)
  ```

  Seguidament justificarem perquè aquest heurístic és bastant millor que els dos anteriors. Primer de tot, el penalitzador (`nonAssignedG + 1`) ens permet multiplicar l'heurístic tantes vegades com clients *Garantits* no assignats (més 1, pel cas on no hi hagi clients no assignats). D'aquesta manera, clarament l'heurístic penalitza aquest fet i prioritza assignar clients *Garantits* sense central.

En segon lloc, aquest heurístic té com a paràmetres *distance* i *energy*. Donat que els dos tenen diferents dimensions, per tal d'evitar que un paràmetre completament cal afegir-hi una constant a almenys a un símbol. Hem provat diferents valors, i experimentalment hem comprovat que, pel cas *A = 1*, els beneficis solen ser òptims, o almenys màxims respecte la resta de constants.

El logaritme de *distance* l'hem fet servir per a prioritzar l'intercanvi dels clients que tenen centrals llunyanes a centrals més properes, i també per a donar més pes al símbol *distance*. Experimentalment emprar el logarítme de *distance* ens dona major benefici.

6 Experimentació

6.1 Experiment 1: Sobre els Operadors

A la nostra pràctica hem implementat els operadors necessaris per a cercar tot l'espai de solucions, el *swap* i l'*assign*, on també es pot desassignar centrals (*deassign*) amb l'*assign*. Així, per a aquest experiment hem pres tres possibles combinacions que poden resultar interessants:

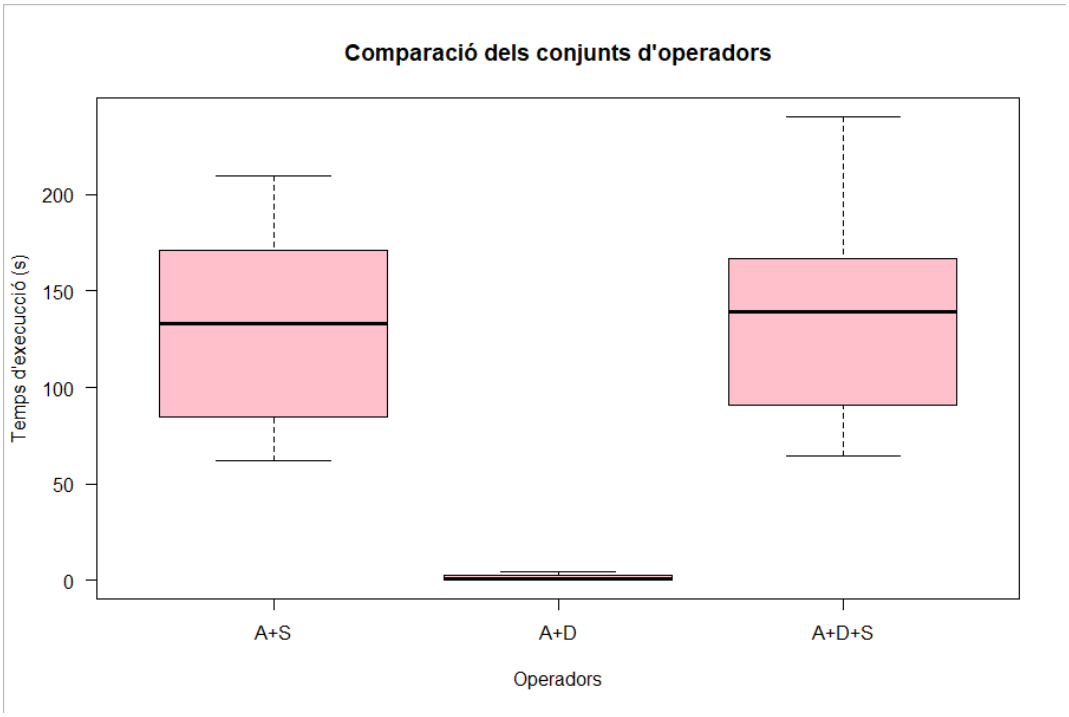
- *Assign* + *Swap* (A+S)
- *Assign* + *Deassign* (A+D)
- *Assign* + *Swap* + *Deassign* (A+S+D)

Els experiments els hem pres amb el mateix heurístic (2) explicat al capítol anterior. A més a més hem escollit l'estratègia d'inicialització (1), que assigna només els clients garantits a les centrals més properes. Les execucions s'han dut a terme al processador *Intel i7-8750H* amb 24 GiB de memòria RAM.

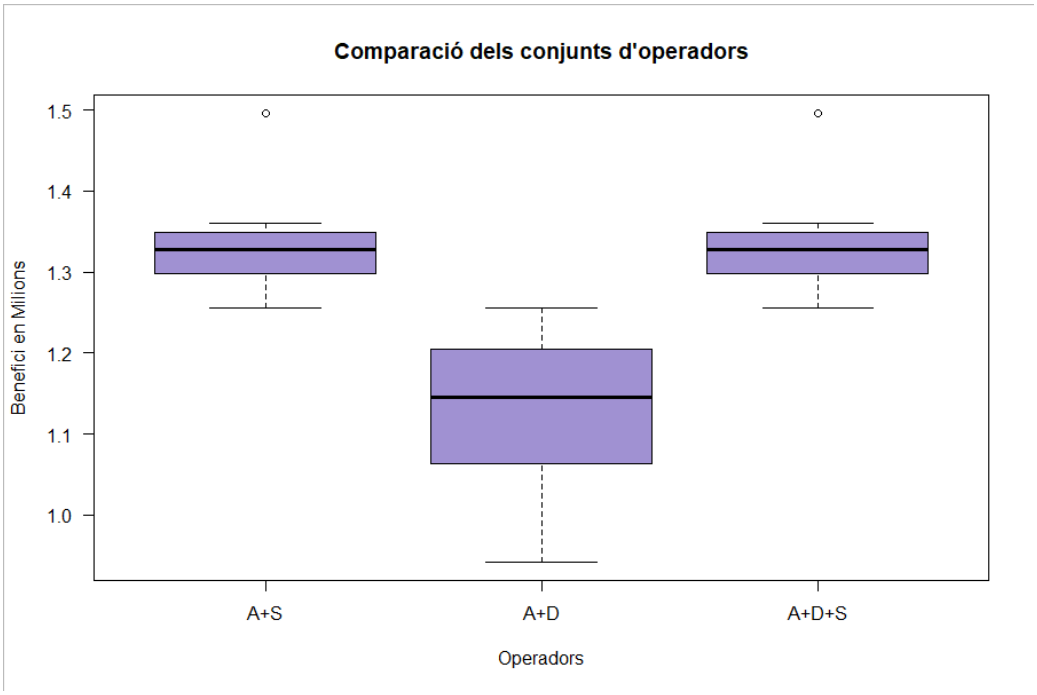
Observació	Pot haver-hi operadors no necessaris o poc funcionals
Plantejament	Escollim tres diferents combinacions d'operadors i observem les seves solucions i temps d'execució
Hipòtesis	El conjunt d'operadors A+S+D és el millor (H0) o n'hi ha un de millor
Mètode	● Escollirem 10 *seeds* aleatòries, una per rèplica. ● Executem 1 experiment per a cada conjunt d'operadors. ● Utilitzem Hill Climbing, heurístic (3) i funció d'estat inicial (1). ● Seguirem els paràmetres proposats a l'enunciat de l'experiment 3.6.1. ● Mesurarem els diferents paràmetres necessaris per a fer la comparació.

Els resultats recollits, amb detall a la fulla de càlcul *experimentacio.ods*, són els següents:

Operadors	*Benefici Mitjà*	*Nodes Expandits*	*Temps Mitjà (s)*
A+S	1 334 667	964	133.91
A+D	1 130 694	173	001.52
A+S+D	1 334 667	964	139.98



Gràfic 1: *Boxplot* dels temps d'execució segons el conjunt d'operadors



Gràfic 2: *Boxplot* del benefici final segons el conjunt d'operadors

Podem observar que la combinació d'operadors A+D obté solucions on el benefici és força inferior que els altres dos. Això és degut a que sense l'operador de *swap,* no es poden moure clients entre centrals. La única manera de fer-ho amb els operadors *assign* + *deassign* seria desassignant un client d'una central i assignant-la a una altra. Al desassignar, el valor de l'heurístic empitjora en la majoria dels casos, i per tant serà molt difícil moure clients amb aquest conjunt d'operadors.

D'altra banda, podem observar que els conjunts d'operadors A+S i A+S+D obtenen solucions on el benefici és idèntic. Llavors, quan ens fixem en quin dels dos conjunts és més eficient en temps, veiem que el conjunt A+S és una mica més ràpid ja que el factor de ramificació és menor que el del conjunt A+S+D.

El *deassign* no es produirà ja que a l'estat inicial només estem assignant centrals a clients *Garantits*. Això fa que només es pugui desassignar clients no garantits que hem assignat quan es millorava l'heurístic, un fet que no es produirà mai.

Tot i que s'observa un *speedup,* estadísticament no podem assegurar que A+S sigui millor que A+S+D, doncs no podem rebutjar la nostra hipòtesis nul·la:

```
One Sample t-test

data:  datos1$TEMPS.2 - datos1$TEMPS
t = 1.3775, df = 9, p-value = 0.2017
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
      -3.882557 15.972957
sample estimates:
mean of x
      6.0452
```

6.2 Experiment 2: Sobre les estratègies de generació d'estats inicial

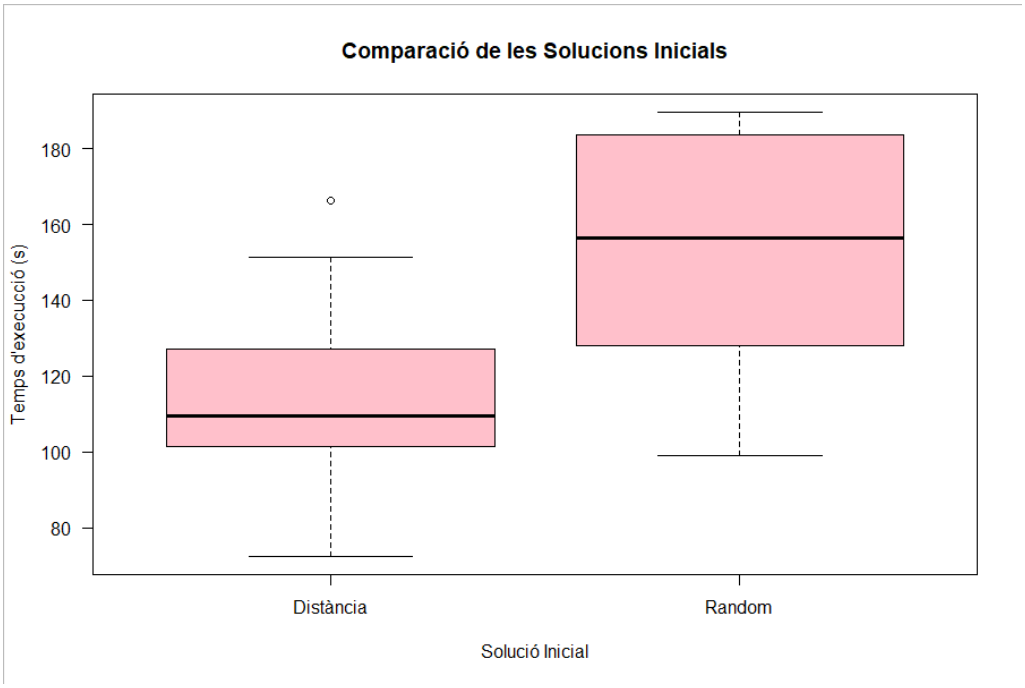
En aquesta pràctica hem implementat dos funcions generadores d'estats inicials diferents, explicades en detall al capítol d'estratègies de generació de solucions inicials. Al final ens hem quedat amb les següents:

1. *Closest Guaranteed:* Només assigna els clients *Garantits*, a les centrals més pròximes a aquests.
2. *Random Guaranteed:* Només assigna els clients *Garantits*, a centrals aleatòries.

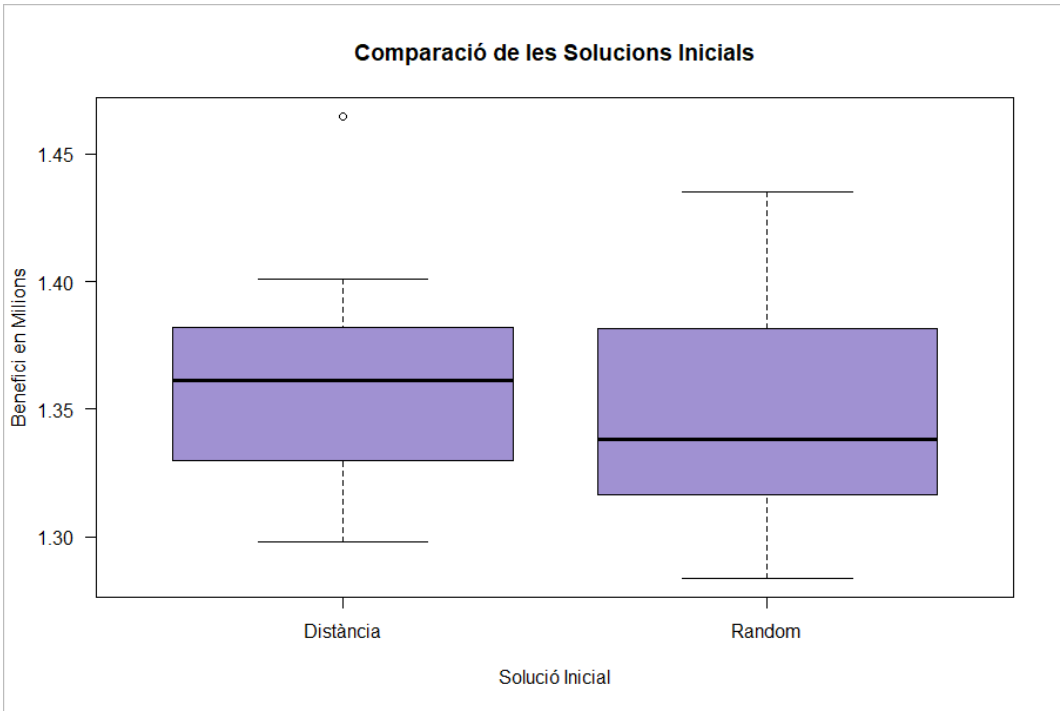
Observació	Hi ha estratègies de generació de solucions inicials millors que altres
Plantejament	Escollim les dues estratègies per generar la solució inicial i comparem el temps d'execució, el benefici final i el nombre de nodes expandits
Hipòtesis	Els dos són igual de millors (H0) o n'hi ha un més eficient que l'altre
Mètode	● Escollirem 10 *seeds* aleatòries, una per rèplica. ● Executem 1 experiment per a cada estratègia generadora. ● Utilitzarem Hill Climbing i l'heurístic (2), i el conjunt d'operadors A+S. ● Seguirem els paràmetres proposats a l'enunciat de l'experiment 3.6.1. ● Mesurarem els diferents paràmetres necessaris per a fer la comparació.

Similarment, les dades recollides són les següents:

	Benefici Mitjà	*Nodes Expandits*	*Temps Mitjà (s)*
Closest Guaranteed	1 362 296	957	113.2
Random Guaranteed	1 346 775	1 041	155.7



Gràfic 3: *Boxplot* dels temps d'execució segons l'estat inicial



Gràfic 4: *Boxplot* dels beneficis finals segons els estats incials

En aquest experiment podem veure que les dues estratègies pel càlcul de l'estat inicial obtenen solucions on el benefici és força similar. Hem de comentar que l'estratègia *Closest Guranteed* té uns beneficis lleugerament superiors que *Random Guaranteed,* tot i que haurem de fer un estudi estadístic per a poder afirmar-ho.

Quan comparem els temps d'execució de les dues estratègies, observem que *Closest Guaranteed* és una mica més ràpid, doncs amb la primera estratègia ens acostem més a una solució bona. La nostra funció heurística bonifica que els clients estiguin assignats a centrals properes, per tant *Closest Guaranteed* convergirà més ràpidament cap a un estat final.

```
data:  datos2$BENEFICI - datos2$BENEFICI.1
t = 4.8061, df = 9, p-value = 0.0009655
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
      8215.346 22825.954
sample estimates:
mean of x
      15520.65
```

El *p-value* de la diferència dels beneficis ens mostra que clarament el benefici obtingut amb *Closest / Distància* és major que el benefici del *Random.* Podem veure que també és més ràpid *Closest / Distància* que *Random* en el segon estudi. És just el que ens esperàvem: començar des d'una solució millor ens porta a millors temps d'execució i beneficis lleugerament millors. Ergo, rebutgem l'hipòtesis nul·la.

```
One Sample t-test

data:  datos2$TEMPS - datos2$TEMPS.1
t = -4.2919, df = 9, p-value = 0.002014
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
      -55.81408 -17.28512
sample estimates:
mean of x
      -36.5496
```

6.3 Experiment 3: Sobre els paràmetres del Simulated Annealing

L'algorisme del Simulated Annealing utilitza quatre paràmetres a l'hora de fer la cerca:

- *Steps:* El nombre d'iteracions total
- *Sitter:* El nombre d'iteracions per a cada càlcul de temperatura
- *K*: La temperatura inicial
- λ: Mesura del descens de temperatura

Per defecte els paràmetres prenen valor:

- *Steps = 10 000*
- *Sitter = 100*
- *K = 20*
- λ *= 0.005*

Els valors per els quals provarem cadascuna de les 4 variables seran:
*(Variable: valor_ini * 0.01, valor_ini * 0.1, valor_ini, valor_ini * 10, valor_ini * 100)*

Steps	100	1 000	10 000	100 000	1 000 000
Sitter	1	10	100	1 000	10 000
K	1	20	100	1 000	10 000
Lambda (λ)	5e-5	5e-4	5e-3	0.05	0.5

Amb l'experiment podrem veure quins són els valors més apropiats per cada variable per tal d'obtenir la millor eficiència (segons el temps d'execució i el benefici final). Els valors subratllats amb taronja són els valors per defecte de cada variable.

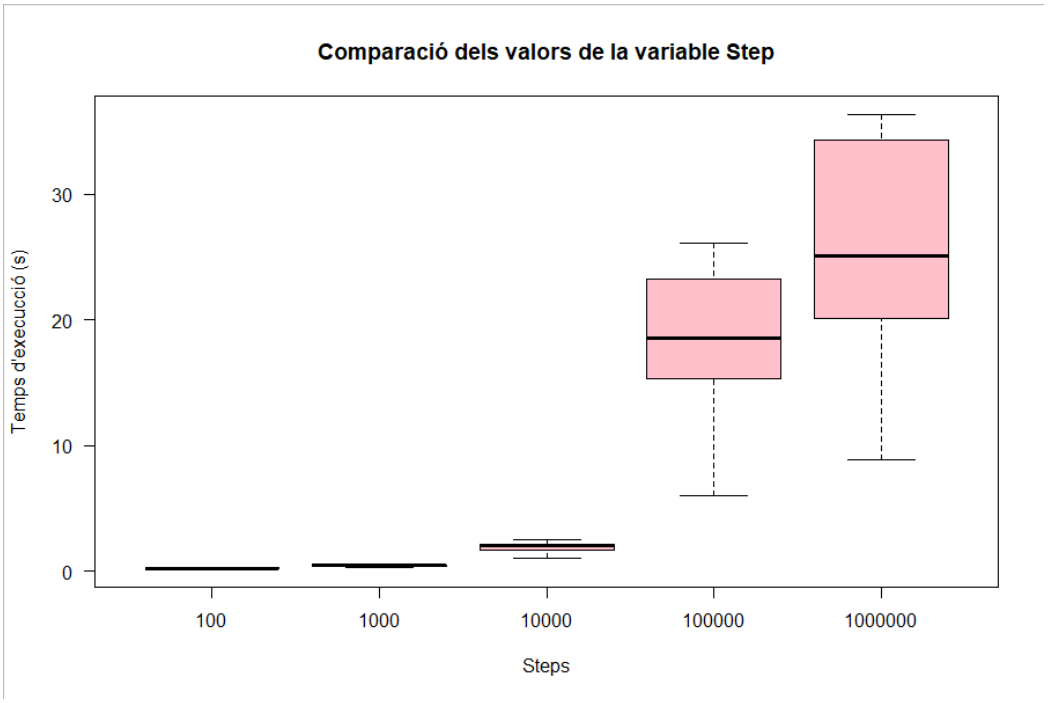
Observació	Els paràmetres del *Simulated Annealing* intervenen en els resultats.
Plantejament	Mesurarem el resultat de multiplicar per 0.01, 0.1, 1 *(per defecte)*, 10 i 100 cada paràmetre del *Simulated Annaling.*
Hipòtesis	Els valors per defecte dels paràmetres del *Simulated Annealing* són els millors, o hi ha una altra combinació que aconsegueix millors resultats
Mètode	● Farem 10 rèpliques per cada modificació plantejada per cada paràmetre. ● S'estudiaran els paràmetres per l'ordre en el qual es passen a la funció. ● Un cop hem près les mesures per 1 paràmetre, agafarem el valor que doni millors resultats i el fixarem. ● Utilitzarem *Simulated Annealing*, l'heurístic (2), el conjunt d'operadors A+S i l'estat inicial generat per *ClosestGuaranteed*.

Els resultats, més detallats en el full de càlcul *experimentacio.ods*, els hem representat en vuit diferents *BoxPlot* (pàgines 23-26): dos per a cada variable (un per a analitzar el temps d'execució i un altre pel benefici).

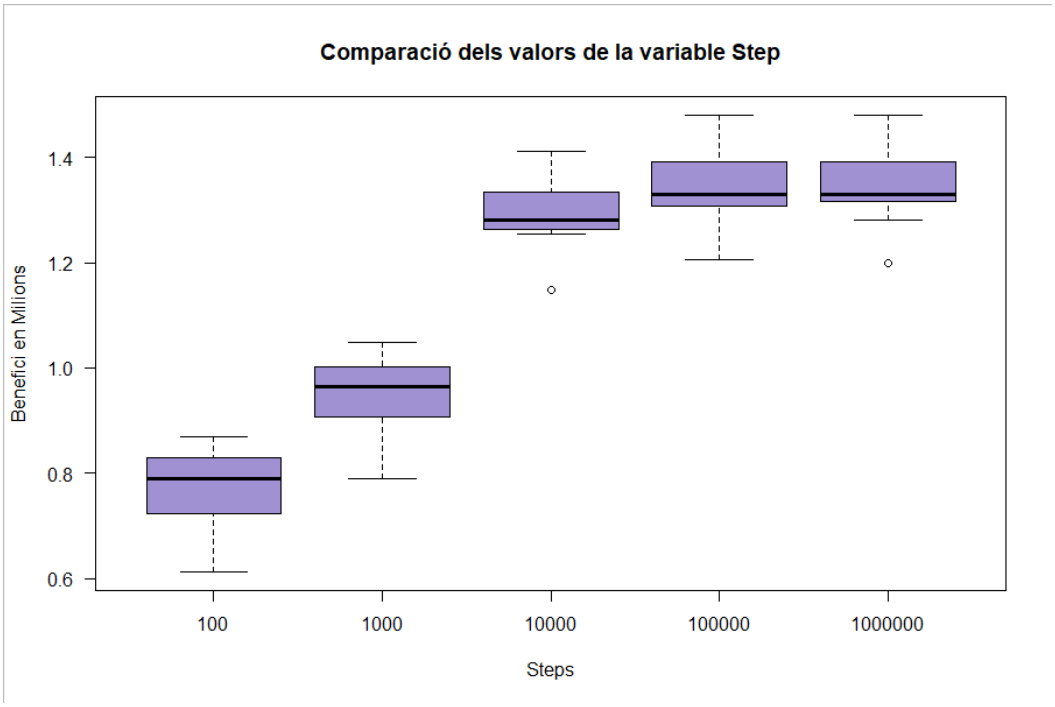
Podem observar als gràfics 5 i 6 que, al augmentar la variable *Step* (nombre d'iteracions total), el temps d'execució augmenta de forma linial (l'eix X està en una escala logarítmica) mentre que el benefici augmenta de forma quasi logarítmica fins a arribar al màxim benefici possible del problema. Això és just el que ens esperàvem; doncs el temps d'execució depèn del nombre d'iteracions total. Per tant, per a aquest problema, en comptes de *10 000* iteracions podriem emprar-ne *100 000*, que garanteixen els màxims beneficis, a canvi d'uns segons més d'execució.

Així i tot, per a les altres variables, no hem observat una gran diferència entre els diferents resultats obtinguts. El valor de temperatura *K = 100* (per defecte) sembla ser visualment un punt òptim, però no hem observat cap altre canvi substancial.

VARIABLE STEP:

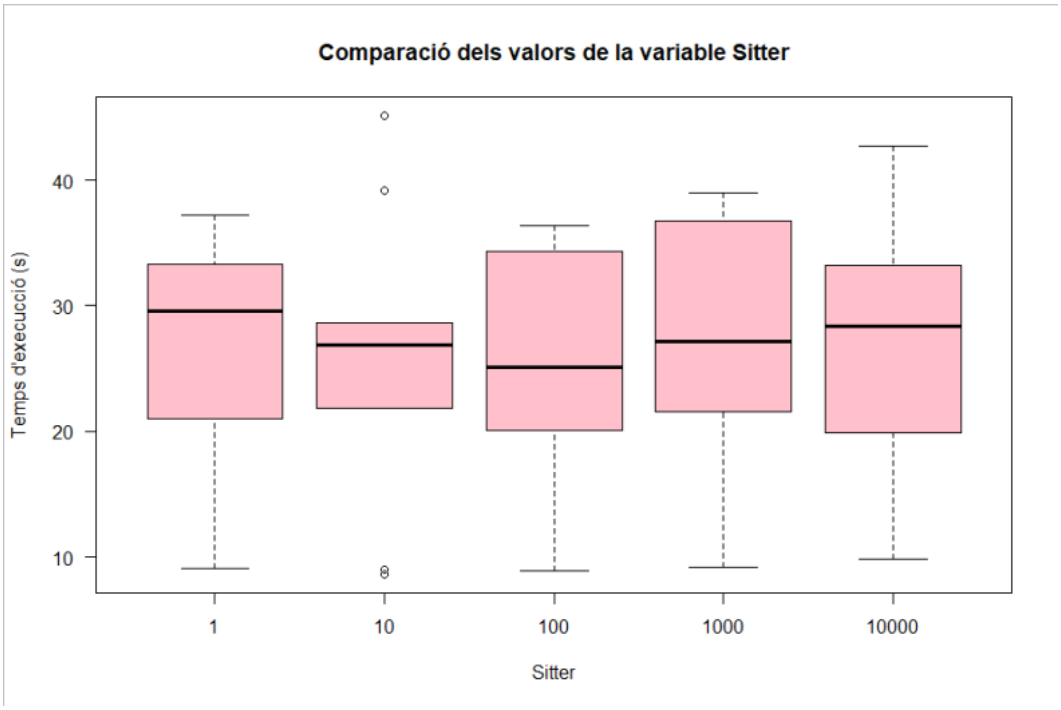


Gràfic 5: *Boxplot* dels temps d'execució segons el paràmetre *step*.

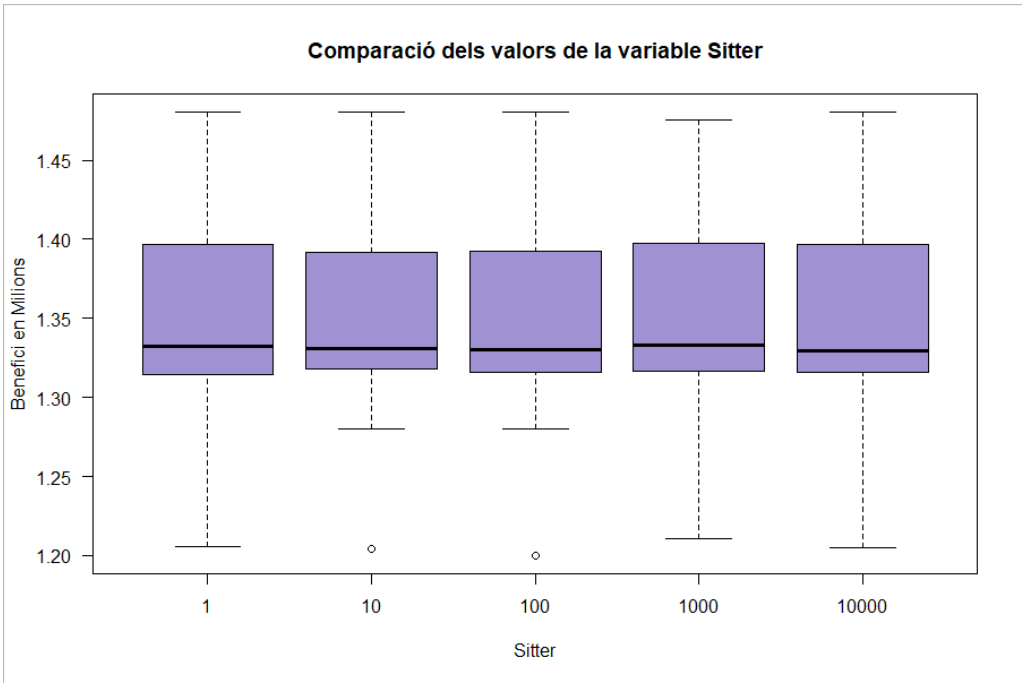


Gràfic 6:*Boxplot* dels beneficis segons el paràmetre *step*.

VARIABLE SITTER:

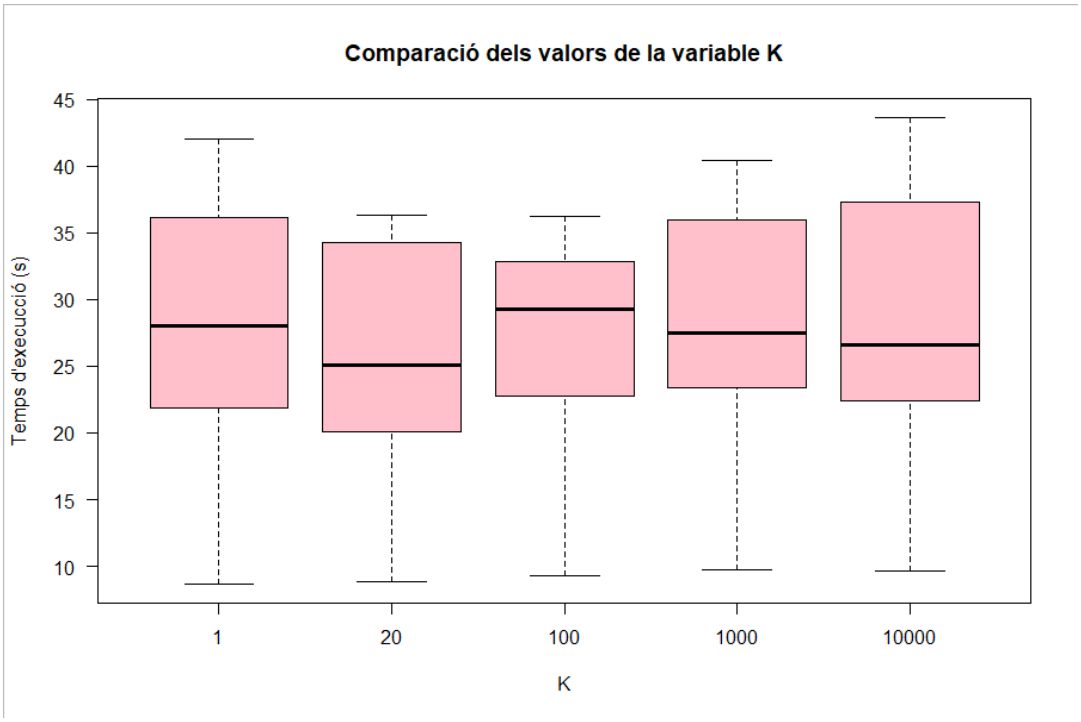


Gràfic 7: *Boxplot* dels temps d'execució segons el paràmetre *sitter*.

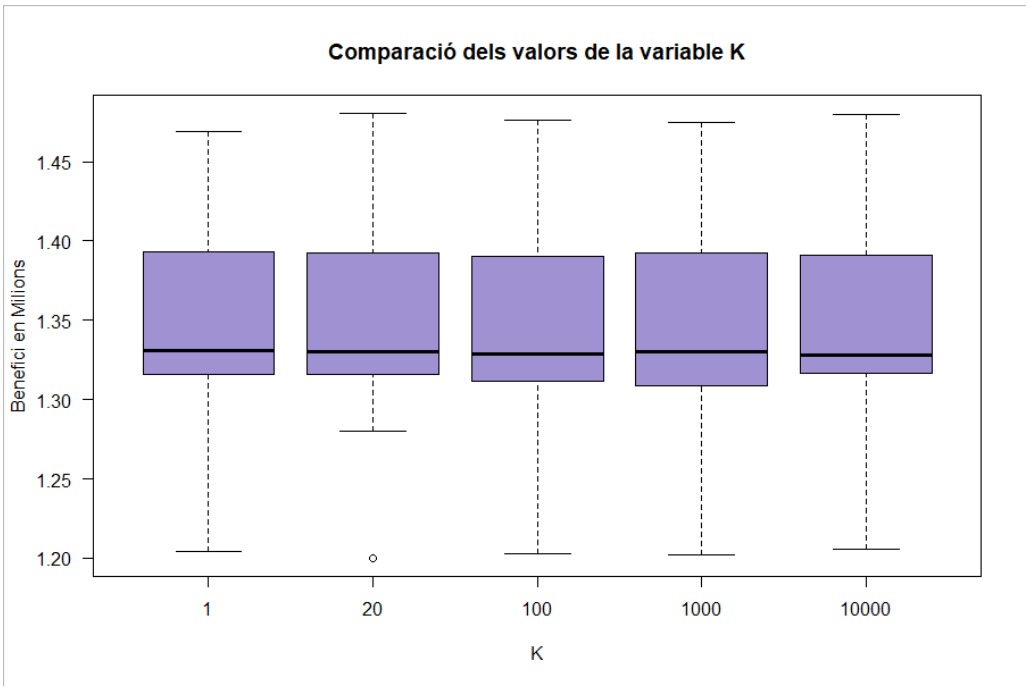


Gràfic 8: *Boxplot* dels beneficis segons el paràmetre *sitter*.

VARIABLE K:

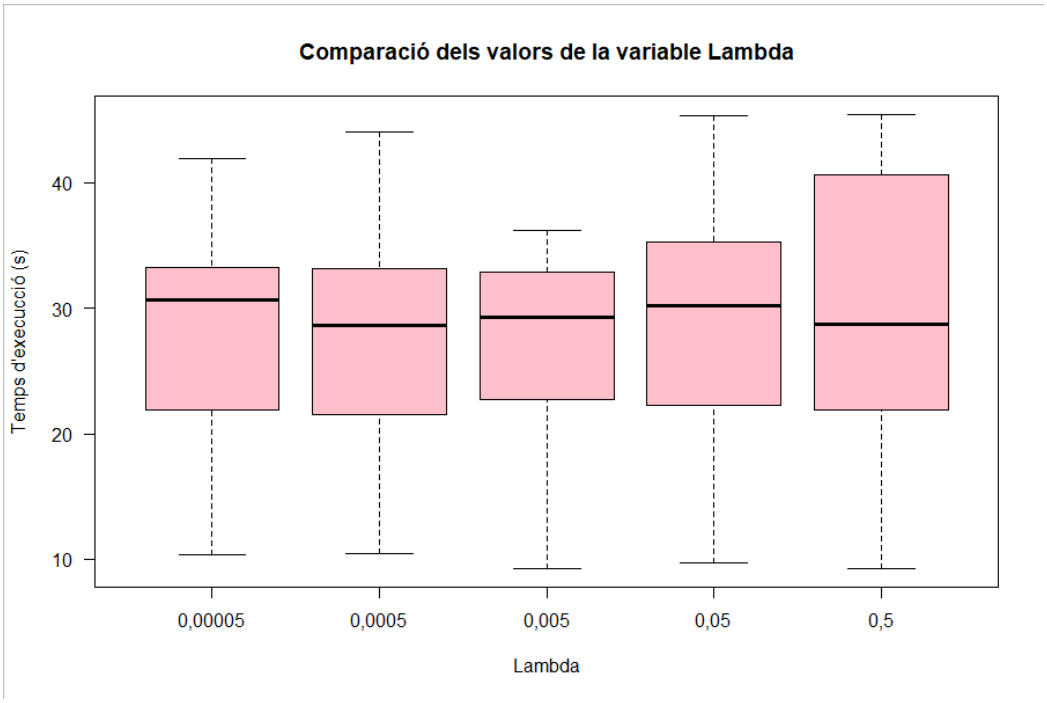


Gràfic 9:*Boxplot* dels temps d'execució segons el paràmetre *temperatura inicial*.

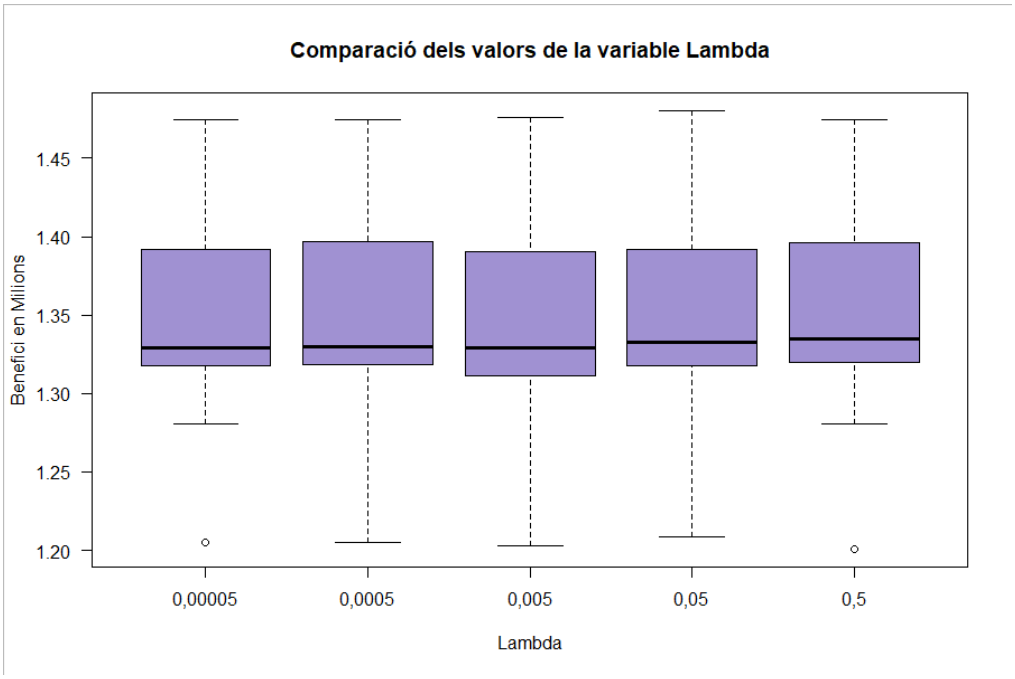


Gràfic 10:*Boxplot* dels beneficis segons el paràmetre *temperatura inicial*.

VARIABLE LAMBDA:



Gràfic 11:*Boxplot* dels temps d'execució segons el paràmetre *descens de temperatura*.



Gràfic 12:*Boxplot* dels beneficis segons el paràmetre *descens de temperatura*.

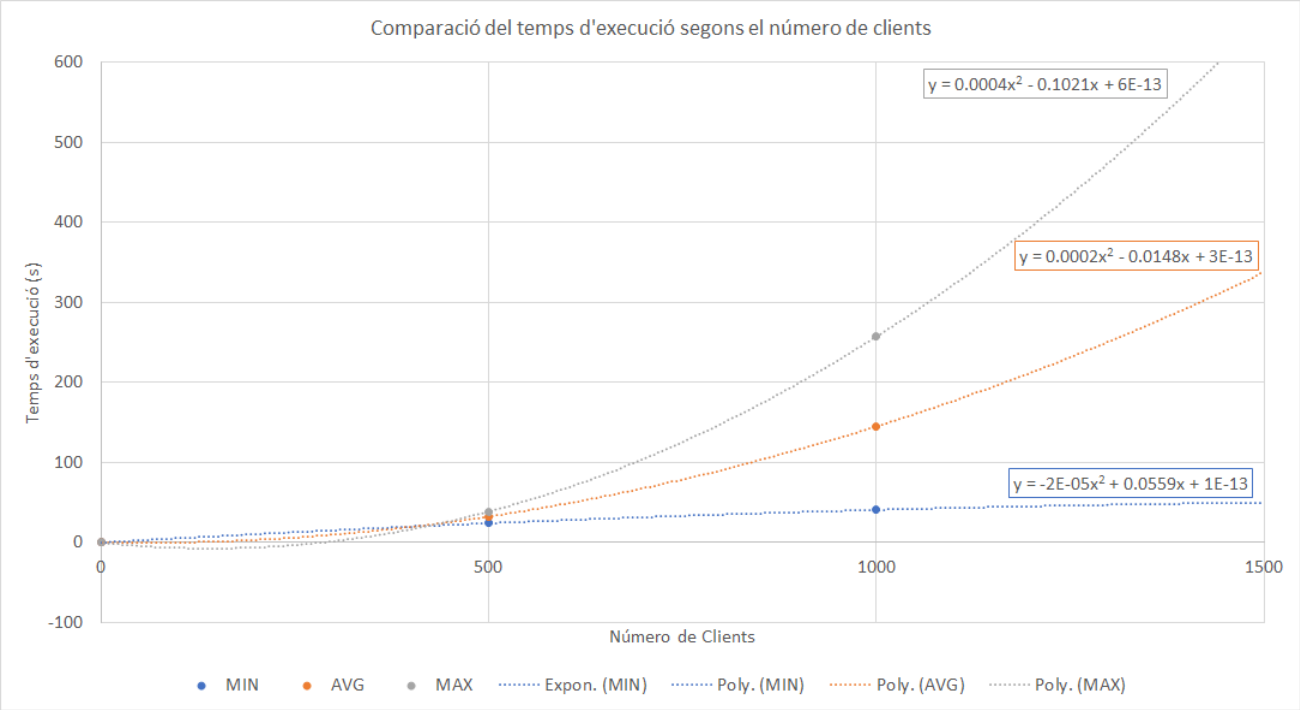
6.4 Experiment 4: Sobre el nombre de centrals i de clients al problema

Donats els resultats del primer experiment, hem conclòs que els operadors òptims per a atacar el problema són *l'assign* i el *swap*.

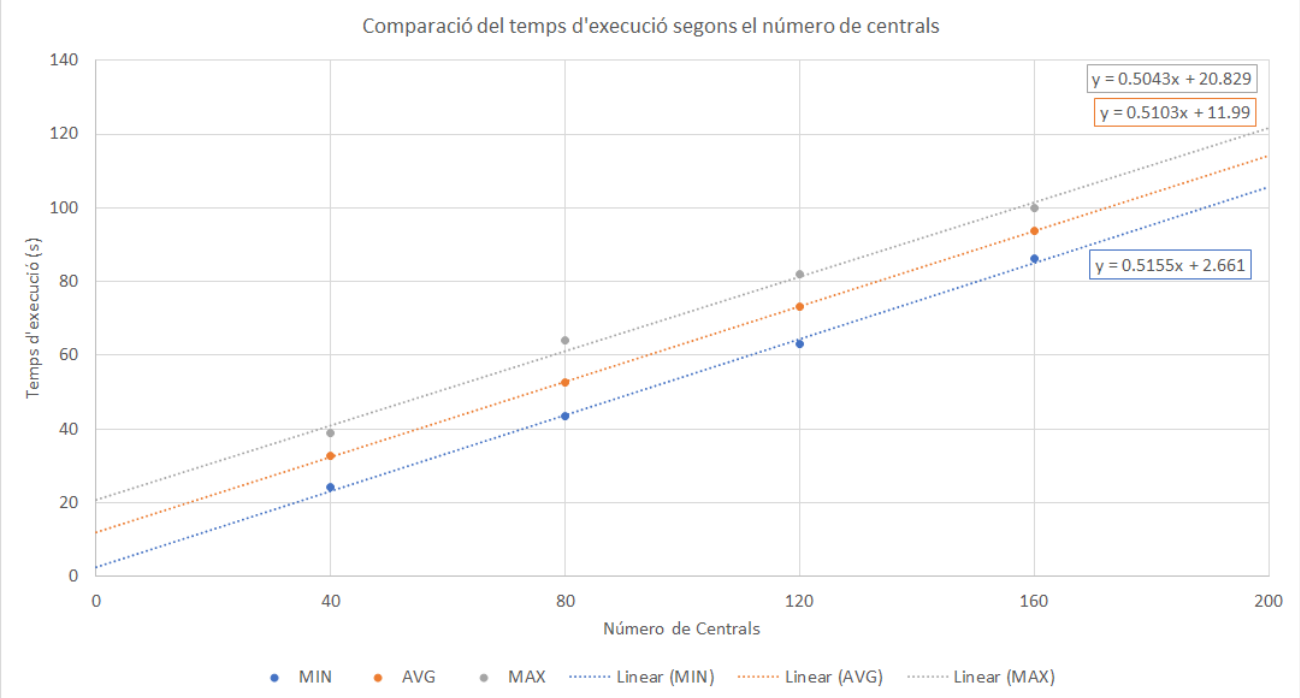
Com s'ha comentat anteriorment, l'operador *assign* té cost $\Theta(CK)$ i, per l'altra banda, l'operador swap té un cost major, $\Theta(C * (C - 1) / 2)$, on *C* és el nombre de clients i *K* el número de centrals del problema.

Degut a que ambdòs costos dels operadors depenen proporcionalment del nombre de clients i centrals quants més clients assignables o més centrals disponibles tinguem el temps d'execucció s'incrementarà.

Observació	A més centrals i/o clients al problema, el temps d'execució incrementa.
Plantejament	Escollim un número de clients fix i anem augmentant el número de centrals per cada iteració fins a veure un patró i viceversa.
Hipòtesis	El temps d'execució s'incrementa linealment d'acord amb el número de centrals, i de manera exponencial d'acord amb el número de clients.
Mètode	● Executem 10 rèpliques de l'experiment, cadascuna amb una seed random diferent, per cada número de clients i centrals. ● Utilitzarem HIll Climbing, l'heurístic (2), el conjunt d'operadors A+S i l'estat incial generat per *ClosestGuaranteed*. ● Seguirem els paràmetres proposats a l'enunciat de l'experiment 3.6.1. ● Mesurarem els diferents paràmetres necessaris per a fer la comparació.



Gràfic 13: *Scatterplot* del temps d'execució segons el número de clients.



Gràfic 14:*Scatterplot* del temps d'execució segons el número de centrals.

Abans de res, hem de remarcar que a l'experimentació on comprovem com augmenta el temps d'execució respecte el nombre de clients, no hem pres mostres més enllà de 1 000 clients, ja que les solucions finals donades no són vàlides al no poder assignar tots els clients *Garantits* a centrals.

Els gràfics presentats anteriorment interpolen tres funcions: una per el temps d'execució menor, una pel major i una pel mitjà.

En ell primer experiment, podem observar que duplicant el número de clients estem augmentant aproximadament en un 650% el temps d'execució en el cas pitjor. Es pot observar que el temps d'execució augmenta de manera **quadràtica** respecte el número de clients.

Això té sentit, ja que el temps d'execució depèn sobretot del factor de ramificació de l'operador *swap*, que és quadràtic respecte el nombre de clients. Quan obtenim les equacions de les funcions que representen el màxim i el mínim temps, podem predir que el temps d'execució de la majoria de les execucions es trobarà entre aquestes dues. Les equacions són les següents:

Temps d'execució millor	$y = 0.0004x^2 - 0.1021x + 6E\text{-}13$
Temps d'execució mitjà	$y = 0.0002x^2 - 0.0148x + 3E\text{-}13$
Temps d'execució pitjor	$y = -2E\text{-}05x^2 + 0.0559x + 1E\text{-}13$

En el segon experiment hem pogut agafar més mostres, ja que un número major de centrals no ens produeix solucions invàlides. Podem observar que, al augmentar el nombre de

centrals, el temps d'execució augmena de manera **lineal**. Això té sentit; ja que l'operador *assign* té un factor de ramificació de *num_clients * num_centrals*.

D'aquesta manera, com que mantenim el total de clients constant, l'únic operador que incrementa en temps d'execució és l'*assign*, però el *swap* no. Quan obtenim les equacions de les funcions que representen el màxim i el mínim temps, podem predir que el temps d'execució de la majoria de les execucions es trobarà entre aquestes dues. Les equacions són les següents:

Temps d'execució millor	y = 0.5043x + 20.829
Temps d'execució mitjà	y = 0.5103x + 11.99
Temps d'execució pitjor	y = 0.5155x + 2.661

La nostra hipòtesi on el temps d'execució augmenta de forma exponencial respecte el número de clients es rebutja, i hem vist que segueix una funció quadràtica. En canvi, podem acceptar la hipòtesis on el temps d'execució és lineal segons el número de centrals.

6.5 Experiment 5: Sobre la penalització per a garantir un estat final vàlid.

Una solució del problema plantejat a la pràctica ha de tenir, com a mínim, tots els clients *Garantits* assignats. Per això, un cas on al final de l'execucció quedin clients *Garantits* pendents d'assignar no és solució i no s'hauria de considerar correcte.

Per a aconseguir-ho, com hem vist anteriorment, tenim dues estratègies: o bé no permetre-ho mai amb les funcions d'aplicabilitat i assegurant una solució inicial que ho compleixi, o bé aplicar un factor penalitzador a l'heuristic per tal de que, al final, s'arribi a una solució vàlida.

Si el factor penalitzador a l'heurístic no és suficientment dur, podriem arribar a un estat final que no acabi de complir la restricció que volem penalitzar. Així i tot no podem aplicar un factor penalitzador de +∞**,** doncs és probable que des d'una solució parcial no poguem arribar a una solució vàlida aplicant només un operador, i s'arribaria a una "*meseta"* o altiplà.

Haurem de modificar el nostre heurístic per tal d'admetre paràmetres que poguem modificar i experimentar. L'heurístic que emprarem serà:

```
distance * log(distance) + energy + ð * nonAssignedG
```

on ð serà la constant penalitzadora.

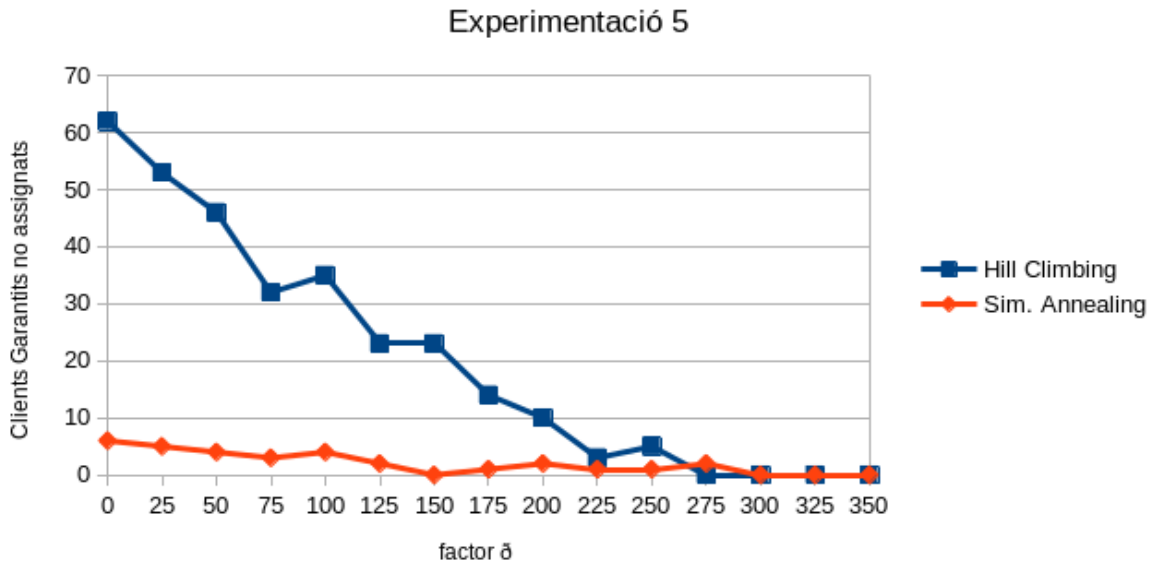
Observació	Podem assegurar una solució vàlida partint d'un estat buit amb un factor penalitzador a la funció heurística.
Plantejament	Escollirem un valor inicial per a la constant ð i l'anirem incrementant.
Hipòtesis	Existeix un cert valor ß on, ∀ð > ß, es garanteix una solució final vàlida.

Mètode	• Es parteix del valor ð = 0, i s'anirà incrementant de 25 en 25 unitats. • Per a cada factor ð s'executa un experiment en *Hill Climbing* i un en *Simulated Annealing*, anotant el número de clients *Garantits* no servits per a cada ð. • S'emprarà sempre la mateixa *seed.*

Els resultats es poden observar en el gràfic següent. Podem veure, a partir dels resultats, que si ð **≥** 275, llavors amb *Hill Climbing* es garanteix una solució final vàlida. Per a *Simulated Annealing*, necessitem ð **≥** 300 per a garantir-ho.

Concloem, doncs, que sí existeix un cert factor penalitzador on, a partir d'aquest, es pot assegurar una solució final vàlida. Això és el que nosaltres suposavem. Tot i així, també hem observat que el *Simulated Annealing*, sense cap o amb poc penalitzador, és més eficient assignant el màxim nombre de clients a centrals que no pas el *Hill Climbing*.

Sense penalitzador, *Hill Climbing* deixa *nonAssignedG = 62*, metre que el *Simulated Annealing* en deixa sis. *Hill Climbing*, però, convergeix més ràpidament a una solució final.



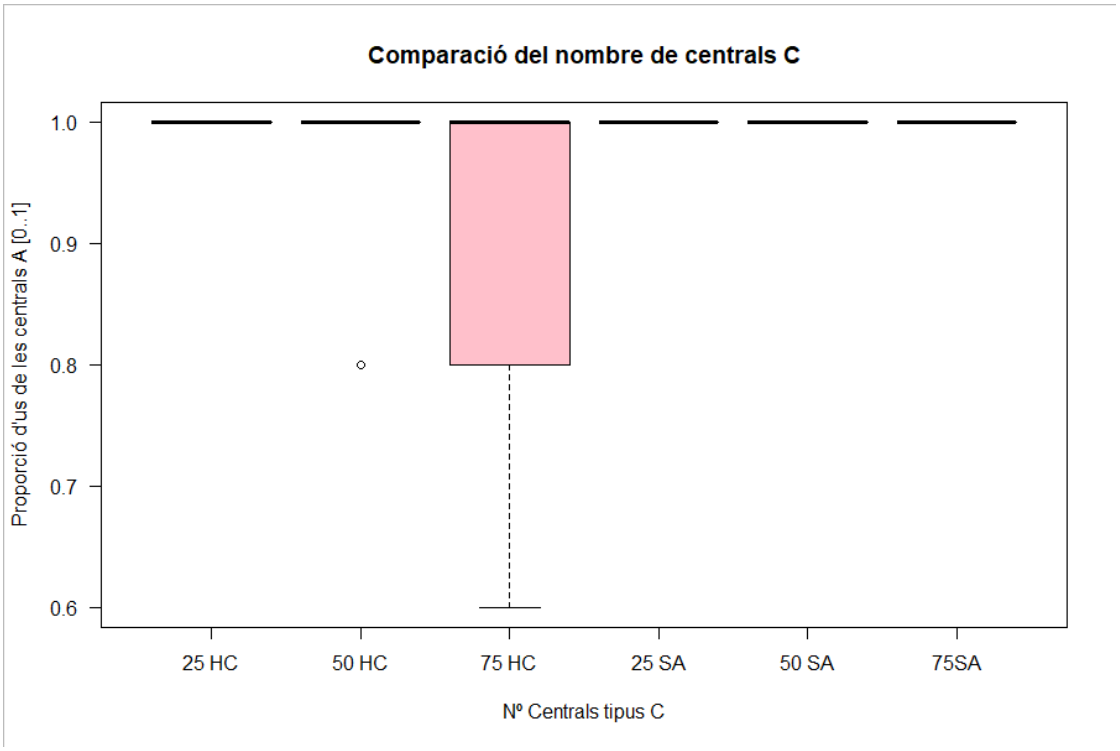
Gràfic 15: *Nombre total de clients Garantits no assignats segons el factor ð*

6.6 Experiment 6: Sobre el nombre de centrals de tipus *C* en el problema

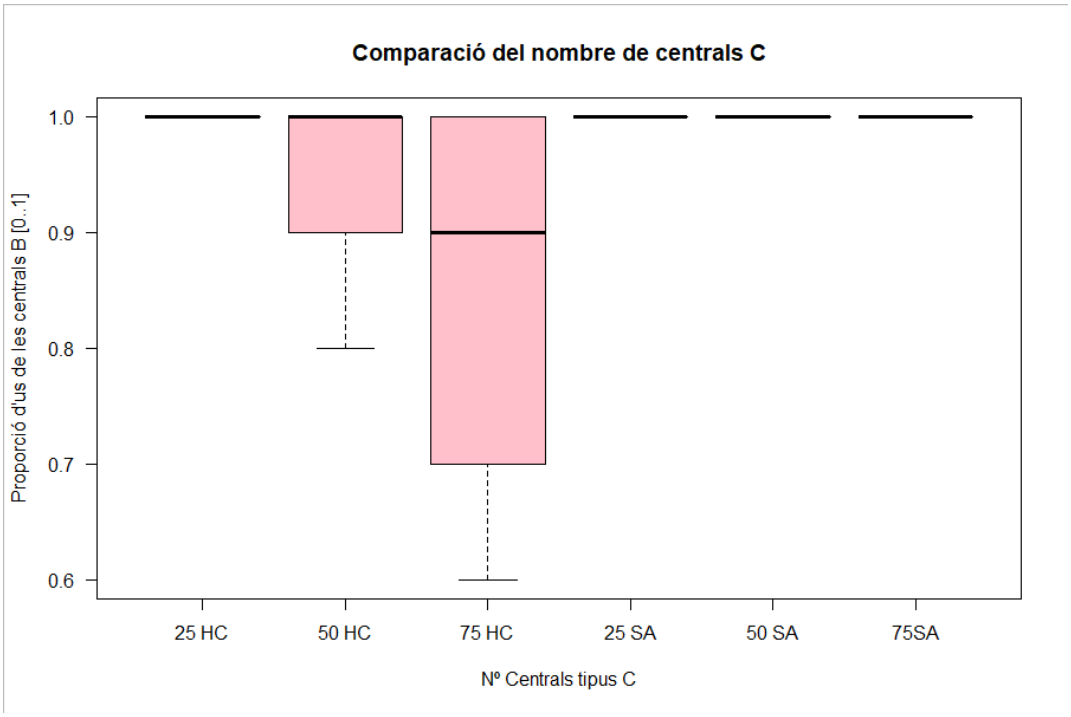
L'estratègia més eficient per a assignar clients a centrals és assignar-los a la central més propera la qual tingui suficient capacitat restant. Per aquest motiu, al tenir més centrals, l'assignació hauria de ser més eficient i ràpida.

Tot i això, com que ara tenim més centrals, els operadors tindran un cost més elevat i el temps es podria veure afectat (recordem que els costos són $\Theta(CK)$ per *l'assign* i $\Theta(C * (C - 1) / 2)$ pel *swap* on *C* és el total de client i *K* el total de centrals).

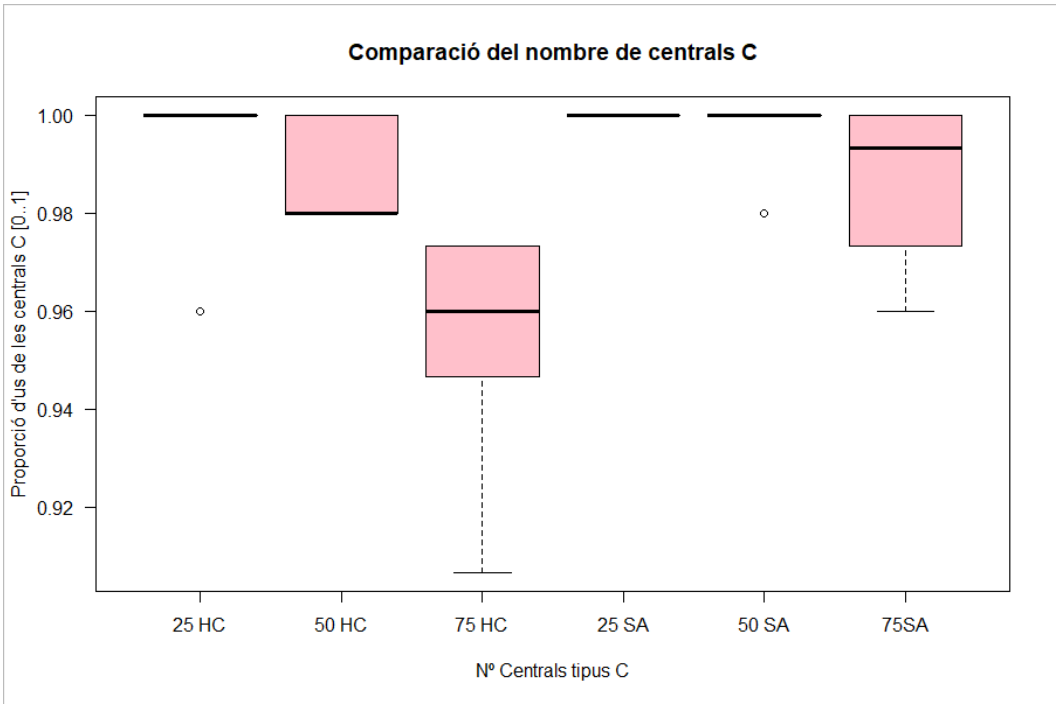
Observació	Tenir més centrals redueix la distància mitjana mínima per l'assignació de cada client i, per tant, s'incrementa l'eficiència energètica.
Plantejament	Anem multiplicant el número de centrals C i comparem l'utilització de les centrals de tipus A i B.
Hipòtesis	A més centrals de tipus C, menor temps d'execució.
Mètode	● Prendrem 10 mesures pel cas inicial, 10 més amb el doble de centrals inicials i 10 més pel triple de centrals. ● Utilitzarem *Simulated Annealing* i *Hill Climbing*, l'heurístic (2), el conjunt d'operadors A+S i l'estat incial generat per *ClosestGuaranteed*. ● Seguirem els paràmetres proposats a l'enunciat de l'experiment 3.6.1. ● Mesurarem els diferents paràmetres necessaris per a fer la comparació.



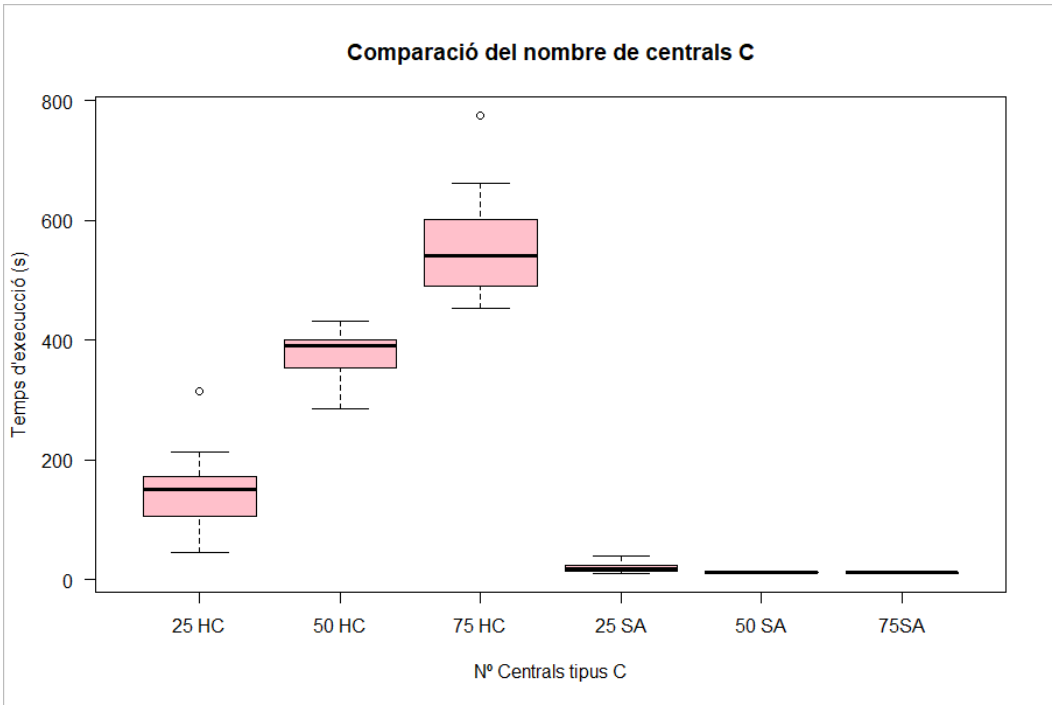
Gràfic 16: Boxplot de la proporció del les centrals tipus A utilitzades per clients (en tant per 1)



Gràfic 17: Boxplot de la proporció del les centrals tipus B utilitzades per clients (en tant per 1)



Gràfic 18: *Boxplot* de la proporció del les centrals tipus C utilitzades per clients (en tant per 1)



Gràfic 19: *Boxplot* dels temps d'execució segons el *nombre de centrals tipus C*.

En aquest experiment es proposava estudiar la variació del temps d'execucció del programa i del nombre de centrals de tipus *A* i tipus *B* que s'utilitzen a mesura que s'incrementen les de tipus *C*.

Com es pot veure en el gràfic 19, amb *Simulated Annealing* el temps es manté constant. En canvi amb *Hill Climbing* sí que podem veure un increment lineal del temps d'execucció d'acord amb l'increment del nombre de centrals del tipus C.

Les centrals del tipus A s'utilitzen sempre totes excepte en el cas on hi ha 75 centrals tipus C amb *Hill Climbing*, ja que al haver-hi més aquestes estaran més distribuïdes i és més probable que una central de tipus C sigui la més propera per a un client.

Experimentalment hem pogut observar (vegem el gràfic 17) que les centrals B segueixen una proporció semblant a les de tipus A, encara que s'utilitzen lleugerament menys. Per últim, les centrals C comencen utilitzant-se totes; però a mesura que hi anem posant més n'hi ha algunes que queden redundants i es queden sense ser la central més propera de cap client, cosa que provoca una petita baixada en la proporció (un 10%).

Pensem que els resultats d'aquesta experimentació no són els que hauríen de donar. Tenim el mateix nombre de clients amb la mateixa proporció segons el tipus, i anem augmentant el número de centrals.

En teoria, el número de centrals hauria de ser molt similar en els tres casos, però no és així. Creiem que el factor més important és que la nostra funció heurística és adient en casos on el nombre de centrals és més ajustat, però poc addient altrament. El nostre heurístic bonifica massa el fet d'assignar un client a una central molt propera. Per tant, les mostres preses en aquest experiment prioritzen l'assignació de clients a centrals properes en comptes d'aprofitar al màxim les centrals ja en funcionament.