Intel·ligència Artificial

Pràctica 1: Cerca local

Aleix Dalmau
Víctor Vallejo
Pau Núñez

2017/2018-Q2

ÍNDEX

1. El problema

1.1 Descripció, característiques i elements

El problema de la pràctica consisteix en organitzar un equip d'helicòpters de rescat en una situació d'emergència en la que cal salvar la vida de persones.
L'objectiu és planificar sortides de rescat pels diversos helicòpters per tal d'aconseguir salvar tots els grups de gent en el menor temps possible.

El problema té les següents característiques i restriccions:

- **Àrea de rescat:** Per simplicitat assumim una àrea de rescat rectangular de $50km^2$ amb orígen de coordenades a la cantonada inferior esquerra.

- **Elements sobre el terreny:** Aquests són els elements que intervenen en el problema i que participen en la definició de l'estat:
    - *C* centres de rescat situats a les vores de l'àrea.
    - *G* grups a rescatar escampats per tot el mapa.
        - Els grups estan formats per entre 1 i 12 persones.
        - Si el grup té ferits serà de prioritat 1, si no en té cap, serà de prioritat 2
        - Tots els grups han de ser rescatats.
    - *H* helicòpters de rescat que poseeix cada centre. Tenim (*C*H*) helicòpters en total.
        - Els helicòpters viatgen a 100km/h
        - Els helicòpters poden transportar fins a un màxim de 15 persones
        - Tarden *np* minuts en recollir un grup de prioritat 2 i *2np* minuts en recollir un grup de prioritat 1. (On *np* = #persones del grup)
        - Quan un helicòpter torna al centre a deixar les persones rescatades tarda 10min en poder sortir a rescatar més grups.
        - Un helicòpter ha de rescatar grups sencers, no es poden fraccionar grups.
        - Un helicòpter pot portar fins a un màxim de 3 grups en un moment donat qualsevol.

1.2 Per què utilitzar la cerca local?

La cerca local s'ajusta correctament a les característiques del problema mentre que la resta de famílies d'algorismes de cerca no ho fan:

La cerca no informada i la cerca heurística troben una única solució, l'òptima.
Ambdues són capaces d'explorar tot l'espai de cerca, visitant tots els nodes possibles. Guardar tota aquesta informació a memòria per a segons quins paràmetres del problema (*G,C* i *H*) i els operadors pot ser inviable perquè l'espai de cerca creix ràpidament. Trobar l'òptim sol ser massa car tant en temps com en memòria i no compensa si no és per espais de cerca petits.

La cerca per satisfacció de restriccions no permet optimitzar una variable i al nostre problema nosaltres necessitem minimitzar certes variables.

La cerca local en canvi, és una bona tria pel nostre problema:
Es mou per l'espai de solucions, és a dir, cada node és una solució. A partir d'una solució inicial relativament poc costosa es van expandint nodes solució que no necessàriament seran òptimes, però seran una solució al problema. D'aquesta manera si aturem l'execució del programa, aquest sempre ens pot retornar una solució tot i no ser la millor. A més a més, no tenim el problema de memòria tan accentuat ja que no es guarden els nodes visitats.

2. Estat i representació del problema

2.1 Part invariant

Aquesta part és invariant, no canvia entre execucions del problema ni tampoc entre estats (solucions) diferents.

Principalment es tracta del mapa de 50km x 50km. Representat implícitament amb un sistema de coordenades 2D (x,y)
$(x,y) \in [0,49999]$ en metres.

Repartits pel mapa hi ha centres (només a les vores) i grups representats com a arrays en classes proporcionades pels professors. Tant centres com grups tenen assignats coordenades (x,y) per tal que puguin ser localitzats en aquest mapa implícit.

2.2 Part variant

Aquesta part varia entre execucions i també entre estats diferents. Els operadors són els encarregats de modificar-la.
Un estat ve definit per un conjunt de sortides assignades als diferents helicòpters. (estat que ha de complir les restriccions exposades a la descripció del problema)

Nosaltres hem triat una matriu irregular (mida de les files pot variar) per representar l'estat. Concretament és un *ArrayList<ArrayList<Sortida>>*.

En aquesta matriu *i (files)* són els diferents helicòpters ordenats. L'identificador de l'helicòpter és implícit en la seva posició *i:*
Els *H* primers helicòpters pertanyen al centre 0, els següents *H* al centre 1, etc
Per determinar a quin grup pertany l'helicòpter cal fer (*i/C*)
La matriu sempre té *(C*H)* files, una fila per cada helicòpter.

La *j (columnes)* recorre la llista de Sortides de l'helicòpter *i*.
Una Sortida està formada per:

- Un array de 3 posicions. En una posició pot haver-hi un nombre $0 \leq n < grupos.size()$ que indica un grup rescatat o bé -1 que indica absència de grup (per Sortides que no tenen 3 grups)
- Un valor que indica la distància que es recorre en aquella sortida en total (anar, recollir els grups i tornar).
- Un valor que indica el temps que tarda l'helicòpter en efectuar la sortida.

Les sortides estan ordenades, de manera que les de *j* menor es fan abans que les de *j* major.

Per tant la posició (*i,j)* de la matriu representa la sortida número *j* de l'helicòpter *i*.
La matriu és irregular ja que cada helicòpter pot tenir un nombre diferent de sortides i també ens estalviem el cost espacial d'una matriu quadrada amb moltes posicions buides en cas que *(C*H)* i/o *G* siguin molt grans.

2.3 Espai de cerca

Tenim un mapa amb un total de *G* grups i un total de *(C*H)* helicòpters amb els que podem rescatar aquests grups.
L'espai de cerca serà tan gran com estats (solucions) existeixin pel problema.
Les solucions possibles són les diferents maneres d'assignar tots els grups als helicòpters.

No totes les solucions són possibles ja que hem de complir les restriccions exposades prèviament, però com que volem una cota superior, les considerarem negligibles.

Ens trobem davant d'una permutació:

- sense repeticions de grup
- on l'ordre dels grups sí que importa

$\frac{G!}{(G-g)!}$ on *G* és el nombre total de grups i *g* el nombre que agafem, però com que en una solució cal salvar tots els grups, tenim que $g = G$ i per tant:

$\frac{G!}{(G-G)!} = \frac{G!}{0!}$ amb 0! = 1 ens queda només $G!$

Aquesta permutació es pot assignar a *(C*H)* helicòpters de manera que ens queda un espai de cerca de mida:

$O(G! \cdot C \cdot H)$ que assimptòticament és $O(G!)$

3. Operadors

3.1 Moure grup

Aquest operador mou un determinat grup d'un helicòpter a una altra Sortida, que pot ser del mateix helicòpter o a un helicòpter diferent. D'aquí en surten les 2 versions de moure grup:

- **Moure Grup al Mateix Helicòpter**: Rep un helicòpter *h*, un grup *g* i una sortida destí *s* i mou *g* a la sortida *s* del mateix *h*. El grup es mou de sortida però segueix sent rescatat pel mateix helicòpter.
- **Moure Grup a Diferent Helicòpter**: Rep dos helicòpters (*h1,h2*), un grup *g* i una sortida *s.* El grup *g* d'*h1* és portat fins a la sortida *s* d'*h2*. En aquest cas el grup es mou a una sortida d'un helicòpter diferent.

Abans d'aplicar qualsevol operador Moure cal comprovar les següents restriccions:

- Que *h1* $\neq$ *h2* en el segon cas de Moure
- Que a la sortida destí no hi ha ja 3 grups
- Que afegint *g* a la sortida *s* l'helicòpter no transporta més de 15 persones en aquella sortida

Després d'aplicar l'operador cal reordenar la sortida destí per tal que la ruta que fa l'helicòpter per rescatar els grups sigui la més eficient i també eliminar la sortida orígen en cas que el grup mogut fos l'únic grup que tenia la sortida original.
El **primer cas de moure** té un factor de ramificació força petit ja que només mou el grup dins el mateix helicòpter. El factor de ramificació és:

$$\sum_{h=0}^{(C \cdot H)} 3 \cdot (\#Sortides \in h) - (\#Grups \in h)$$

És a dir, el nombre total de llocs lliures dins de totes les sortides de l'helicòpter *h*, que nosaltres internament representem, com ja hem mencionat a l'explicació de què és una sortida, amb un -1. D'aquesta manera només cal sumar els -1 d'*h* per trobar les opcions que tenim per moure el grup.

Pel **segon cas de moure** sabem que un grup es pot moure a qualsevol Sortida de qualsevol helicòpter mentre a la Sortida destí hi hagi almenys 1 lloc lliure i no es violin les restriccions, per tant, el factor de ramificació és:

$$\sum_{h1=0}^{(C \cdot H)} \left(\sum_{h2=0, h1 \neq h2}^{(C \cdot H)} \left(3 \cdot (\#Sortides \in h2) - (\#Grups \in h2) \right) \right)$$

3.2 Swap grups

Aquest operador intercanvia la posició de dos grups ja sigui en el mateix helicòpter o a respecte helicòpters diferents. De forma similar que a l'operador moure, tenim dues versions del swap:

- **Swap grups al mateix helicòpter**: Rep un helicòpter *h* i dos grups *(g1,g2)* i intercanvia les seves posicions.
- **Swap grups de diferents helicòpters**: Rep dos helicòpters *(h1,h2)* i dos grups *(g1,g2)*. A l'inici tenim $g1 \in h1$ *i* $g2 \in h2$ i després del swap ens queda $g1 \in h2$ *i* $g2 \in h1$

Abans d'aplicar qualsevol operador Swap cal comprovar les següents restriccions:

- Que posant *g1* a la Sortida on estava *g2*, la Sortida en qüestió no suma més de 15 persones entre tots els grups que té.
- Que posant *g2* a la Sortida on estava *g1*, la Sortida en qüestió no suma més de 15 persones entre tots els grups que té.

Després d'aplicar l'operador cal reordenar els grups dins les dues sortides que s'han vist afectades pel swap per tal de recalcular la ruta més eficient que ha de fer l'helicòpter propietari d'aquella sortida.

El **primer cas de swap** té un factor de ramificació força petit ja que només intercanvia grups dins el mateix helicòpter. El seu factor de ramificació és:

$\sum_{h=0}^{(C\cdot H)} (\#Grups \in h \cdot ((\#Grups \in h) - 1))$, assimptòticament $\sum_{h=0}^{(C\cdot H)} (\#Grups \in h)^2$

És a dir per tots els helicòpters [0..(C·H-1)] per un grup que pertany a un determinat helicòpter cal intercanviar-lo amb tots els grups d'aquell mateix helicòpter.

El **segon cas del swap** intercanvia dos grups de diferents helicòpters, genera una quantitat de successors molt superior a la primera versió. El factor de ramificació és:

$$\sum_{h1=0}^{(C\cdot H)} \left(\sum_{h2=0, h1\neq h2}^{(C\cdot H)} (\#Grups \in h1) \cdot (Grups \in h2) \right)$$

3.3 Moure sortida

Aquest operador mou una sortida *s* que pertany a un helicòpter *h1* a un altre helicòpter diferent *h2*.
No cal comprovar cap restricció abans d'aplicar aquest operador ni tampoc cal ajustar res després de fer-ho. Això es degut a que la granularitat de l'operador és superior a la dels operadors que treballen amb grups, ara movem tota una Sortida i *s* internament no canvia, només passa a estar assignada un altre *h*.
Només en el cas que *s* tingui algun grup amb prioritat 1 caldria recol·locar-la cap al principi de les sortides d'*h2*.
El seu factor de ramificació és:

$$\sum_{h1=0}^{(C\cdot H)} \left(\sum_{h2=0, h1\neq h2}^{(C\cdot H)} \#Sortides \in h1 \right)$$

3.4 Swap sortides

Aquest operador intercanvia dues sortides *s1,s2* de dos helicòpters diferents *h1,h2*.
Intercanviar dues sortides dins el mateix helicòpter té sentit ja que la suma de temps de les sortides queda exactament igual. De nou, tenim granularitat Sortida i no grup, per tant internament *s1,s2* no s'alteren de cap manera. Com abans, és possible que haguem de recol·locar *s1* i/o *s2* cap al principi del seu helicòpter destí si tenen algun grup amb prioritat 1. El seu factor de ramificació és:

$$\sum_{h1=0}^{(C\cdot H)} \left(\sum_{h2=0, h1\neq h2}^{(C\cdot H)} \left((\#Sortides \in h1) \cdot (Sortides \in h2) \right) \right)$$

3.5 Explicació de l'elecció dels operadors

Hem triat aquest conjunt d'operador perquè ens ha semblat que eren els necessaris per a poder explorar en la seva totalitat l'espai de solucions.

Tots els operadors de moure son imprescindibles, ja que sense un d'ells podem afirmar que algun dels nodes successors d'un estat no seria visitat. Vam veure que realment els moure a diferent helicòpter i els moure a mateix helicòpter podrien fusionar-se i que nomes existís el cas de moure grup i moure sortida, i que es pogués cridar amb el mateix helicòpter com a origen i destí. Però vam optar per deixar-ho en operadors separats, ja que així el nostre codi seria més modular, i en el cas de que es vulgues prescindir d'algun d'aquests casos, es podria fer de forma més simple. L'única penalització que té deixar-los separats es que fas més línies de codi, però els casos que s'executen són exactament els mateixos. En el cas de Moure Sortida només contemplem que sigui per a helicòpters diferents, ja que moure sortida al mateix helicòpter no ens oferia cap millora, almenys pel primer Heurístic, que és amb el que ens van demanar que penséssim els operadors.

Un cop teníem els operadors de moure, vam pensar que com que el nostre moure grup en cas de no trobar lloc lliure no crearia una nova sortida, ens calia tenir alguns operadors que permetessin avançar al Successor Function en cas de que tots els helicòpters tinguessin sortides plenes. I d'aquesta idea van néixer els nostres operadors de swap grup, que s'estalvien el tenir una sortida auxiliar per a poder intercanviar dos grups d'una sortida completa a una altre. Aquests operadors també permetien superar possibles altiplans que es podien generar al pas intermedi de moure un grup a un lloc per tal de deixar lliure la seva posició a la sortida per a un altre grup.
De la mateixa manera va aparèixer l'operador de Swap Sortides, que també ens oferia l'ultim avantatge esmentat, però que potser el cost de generar-lo no és rentable al sí poder arribar al mateix estat utilitzant l'operador moure Sortida.

Comprovarem a l'apartat 6.1 si aquest conjunt d'operadors és òptim o podem prescindir d'algun d'ells.

4. Estat Inicial

Per generar la solució inicial, vam triar 3 estratègies diferents, una molt simple, una altra similar a la primera però notablement més realista, i una tercera que intenta tenir una solució més propera a la final.

4.1 Solució inicial 1

És la solució inicial més senzilla, que consisteix en crear una sortida nova per a cada grup a rescatar i que totes aquestes sortides les porti a terme el primer helicòpter de la llista d'helicòpters, que en el nostre cas serà l'helicòpter 0 del centre 0.
No cal fer cap comprovació, ja que un sol grup mai excedirà els 12 integrants i per tant complirà la restricció d'espai al helicòpter.

Es genera en temps lineal: $O(G)$, on G es el número de grups que s'han de rescatar.

Vam triar aquesta solució inicial degut a que té el mínim cost de generació que pot tenir el problema i perquè tot i que a simple vista es veu clarament que es una opció molt dolenta, en quant a solució final, això dóna molt marge de treball als operadors per tal de trobar una solució que estigui més a prop de l'òptima. Als experiments veurem més clar si val la pena.

4.2 Solució inicial 2

La segona solució inicial és molt similar a la primera, però en aquest cas repartirem la feina entre tots els helicòpters dels diferents centres que disposem.
Per tant, igual que abans, tenim una sortida per grup, però ara anirem assignant cada sortida a un helicòpter diferent de forma equilibrada. Donat un grup, per saber a quin helicòpter li tocarà rescatar-lo, farem el següent càlcul:

Donats *g* i *C·H,* on *g* és un valor que va de [0..grupos.size()-1] i *C·H* és el número total d'helicòpters que tenim:

l'Helicòpter que durà a terme la sortida serà, $h = g\%(C \cdot H)$.

Com és normal, el cost de generació d'aquesta solució, serà el mateix que la primera: $O(G)$, on *G* es el número de grups que s'han de rescatar.

La idea d'aquesta solució va néixer com a millora de la primera, que tingués un cost de generació similar, però que fes és dels helicòpters que tenim a disposició. En aquest cas estem fent servir el màxim d'helicòpters possibles donats un número de grups, ja que a un helicòpter no li programem una segona sortida fins que tots els altres ja en tenen una.

4.3 Solució inicial 3

La tercera solució és la més complexa, la intenció era apropar la distribució de grups i helicòpters a una possible solució final que tingui en compte quin és el centre que té més a prop cada grup. Per tal de fer això primer recorrem els grups i anem comparant a quina distància es troba un grup de cada centre. Un cop hem fet això, tenim un vector de mida *G* on cada índex representa l'id d'un grup, i el contingut d'una posició és l'id del Centre més proper. És a dir:
aQuinC [g] = c, vol dir que el grup *g* és menys distant a *c* que a qualsevol altre centre.
Ara el que fem és, per cada grup, accedim al seu centre proper, i dins d'aquest mirem quin helicòpter té menys sortides programades i intentem afegir el grup a aquesta sortida. En el cas de que tots els helicòpters estiguin igual de carregats en quan a sortides (al inici per exemple, que tots en tenen 0), agafa el primer helicòpter. Si afegint aquest grup a la sortida es viola algun tipus de restricció, que ja hi hagin 3 grups o que afegint el nou grup l'helicòpter hauria de transportar més de 15 persones, s'afegeix una nova sortida per aquest helicòpter amb només el nou grup per recollir. La penúltima sortida es queda com estava, en el cas de que violava el màxim número de passatgers, deixem la tercera posició del vector de grupsRecollits de la penúltima sortida sense grup assignat i mai l'omplirem.
Aquest últim fet el podíem modificar i que en comptes d'intentar afegir el grup a l'última sortida abans, comprovés si es podia afegir a alguna altre sortida que hagués quedat amb només dos grups assignats, però vam decidir que no valia la pena degut al cost i la complexitat de l'algorisme i perquè el fet de deixar espais a algunes sortides li donava joc als operadors per poder fer la seva feina.

El cost de generar aquesta solució inicial és: $O(G \cdot H + G)$, que assimptòticament és $O(G \cdot H)$, on *G* = número de grups, i *H* = número d'helicòpters dels que disposa cada centre.

Veiem que el cost d'aquest últim algorisme és notablement superior als dels dos anteriors, però creiem que val la pena, ja que s'apropa bastant a una possible solució final. I tot i que si un centre *c* es troba més a prop que tots els grups, aquests s'assignaran a *c*, estem agrupant a sortides grups que es troben bastant a prop.

Aquests dos fets faran que els operadors no hagin d'expandir tants nodes, i per tant millorarà el temps d'execució del programa. Comprovarem si estem equivocats o no quan efectuem el segon experiment.

5. Heurístics

El primer heurístic que es demana a l'enunciat està pensat per minimitzar la suma de tots els temps empleats pels helicòpters en rescatar a tots els grups, es pot entendre d'una altre manera com a un heurístic per a minimitzar l'ús de gasolina total, ja que temps implica gasolina. En aquest cas doncs, parlar de temps total i de gasolina feta servir, és el mateix.
És un heurístic senzill, el que fa bàsicament és sumar el temps que empleen tots els helicòpters en realitzar cada una de les seves sortides, i la penalització de temps entre sortides. Quan major sigui aquesta suma, menys valuosa serà la solució.

Observem que no és un heurístic gaire ètic, ni intel·ligent. Ja que creiem que en una situació de desastre natural, la principal prioritat ha de ser rescatar en el mínim temps possible a totes les persones que es troben a la zona afectada, sense donar-li molta importància al consum de gasolina. En tot cas, es podria ponderar el consum de gasolina i el temps que es triga en rescatar a l'últim grup.
Un altre indicador de que no és un heurístic intel·ligent, per exemple, és que si tenim 2 helicòpters per centre, és probable que la solució final que obtinguem li doni molta càrrega a un helicòpter i a l'altre no. Degut a que aquest heurístic no considera millor solució que els helicòpters d'un mateix centre es reparteixin la càrrega de sortides a que un d'ells l'assumeixi (això es produeix un cop cada helicòpter del centre ja te una sortida assignada, i per tant els 10 min d'espera entre sortides els patirà un o altre helicòpter).

En quant al segon heurístic, a més de minimitzar la suma de tots els temps empleats pels helicòpters en rescatar tots els grups (primer Heurístic), també minimitza el temps que passen sense ser rescatats tots els grups amb prioritat 1.

Aquí hem hagut de ponderar la importància que li donàvem al fet que hi haguessin grups de prioritat 1 sense ser rescatats o al fet "d'estalviar gasolina".

6. Experiments

6.1 Selecció d'operadors

En el nostre cas, hem considerat 6 operadors diferents, i ara ens disposem a comprovar si hi ha algun d'ells del que podem prescindir, ja que sospitem que algun d'ells genera estats als que podríem arribar utilitzant un dels altres 5 operadors.

La nostra hipòtesis és que l'operador SwapSortida és prescindible, ja que podem generar el mateix node successor amb MoureSortida amb dos moviments en comptes d'un. En quant als altres swaps no tenim aquesta seguretat, ja que no podem moure un grup d'una sortida a una altre si no hi ha lloc en aquesta.

Per tal de comprovar la nostra teoria, primer de tot hem triat 10 *seeds* diferents sense cap tipus de criteri, i sabent que cada operador significa afegir temps d'execució del programa, hem comprovat si el nostre algorisme veia fortes pitjores en quant a trobar una bona solució final que justifiquessin la no eliminació d'aquest operador.

Per tal de comprovar-ho simplement hem calculat el temps execució i la solució en minuts (suma de tots els temps empleats pels helicòpters per tal de rescatar tots els grups), amb l'operador swap sortida i sense ell. Per tal de fer aquests càlculs hem utilitzat la solució inicial 3.

Ja amb les dues taules generades, podem confirmar que el prescindir de l'operador swap no només no perjudica a l'hora de trobar una bona solució, sinó que redueix molt considerablement el temps d'execució del nostre programa. Per tant ja podem deixar inoperatiu permanentment el nostre operador SwapSortida

Un cop teníem això, hem volgut comprovar si el nostre programa seguia millorant al deixar d'utilitzar un altre operador Swap. Hem triat comparar-ho amb el SwapGrupMateixHeli (la variant diferent heli veuria unes millores molt similars ja que té el mateix handicap) i al fer-ho hem pogut observar que tot i que el temps d'execució, com és natural, redueix, el programa no és capaç d'arribar a una solució tan bona com amb els operadors anteriors i per tant no surt rentable prescindir d'aquest operador.

D'aquesta manera ens quedem amb el conjunt inicial d'operadors, exceptuant l'operador SwapSortida. Per tant confirmem que la nostra hipòtesis era certa.

Tot seguit adjuntem la taula on es pot observar clarament quina es la millor opció, ressaltada en color verd.

	Temps Solucio (minuts)			Temps Execució (s)		
Seed	*Tots els operadors actius*	*Tots menys Swap Sortida*	*Tots menys Swap Sortida i SwapGrupMHeli*	*Tots els operadors actius*	*Tots menys Swap Sortida*	*Tots menys Swap Sortida i SwapGrupMHeli*
32809	1369.99	1369.99	1441.79	1.922	1.046	0.61
67121	1567.19	1567.19	1579.16	1.564	1.201	0.83
89101	1195.01	1195.01	1298.69	2.26	1.373	1.118
1234	1337.05	1337.05	1414.05	1.354	1.205	0.772
1997	1356.25	1356.25	1387.51	2.24	1.204	0.933
9997	1290.73	1290.73	1346.24	1.403	0.836	0.554
05240	1562.27	1562.27	1636.25	1.121	0.852	0.489
1872	1265.83	1265.83	1426.28	1.871	1.149	0.722
37562	1380.77	1380.77	1450.40	1.537	1.061	0.514
87523	1196.74	1196.74	1325.02	1.07	0.758	0.463

6.2 Determinar solució inicial

Hem programat tres estratègies diferents per tal de generar la solució inicial. Ara hem de comprovar quina d'elles obté millor resultat en un mateix escenari.

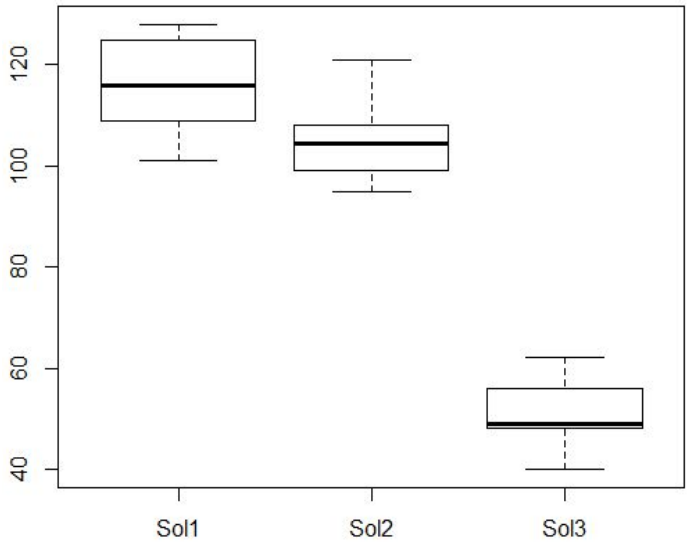
Observació: Hi poden haver mètodes d'inicialització que obtenen millors solucions**.**
Plantejament: Agafem els nostres tres mètodes d'inicialització i observem les seves solucions.
Hipòtesis: Tots els mètodes d'inicialització són iguals.
Mètode: Agafem 10 seeds diferents (les mateixes que a l'apartat anterior) i per cada una executem el programa amb les 3 estratègies d'inicialització.

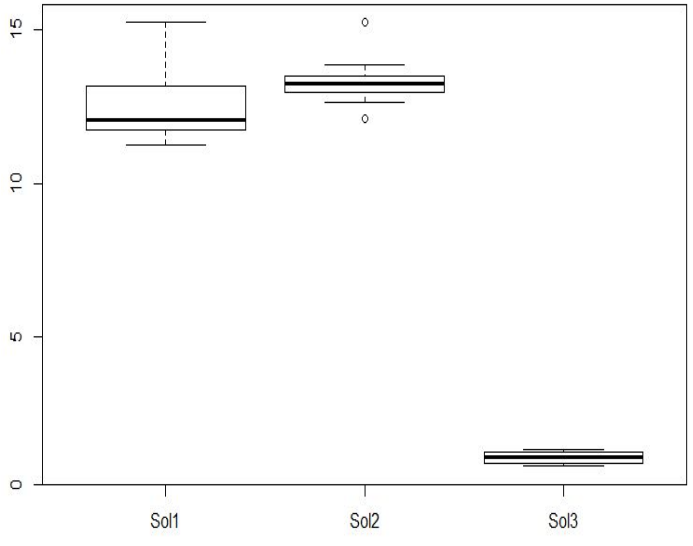
Els resultats d'aquestes execucions el veiem plasmats a la següent taula, on es representa per cada una de les 3 solucions inicials el temps de la Solució final, els nodes que s'expandeixen i el temps que triga en executar-se el programa.

	Temps Solució (minuts)			Nodes Expanded			Temps Execució (s)		
Seed	**Sol1**	**Sol2**	**Sol3**	**Sol1**	**Sol2**	**Sol3**	**Sol1**	**Sol2**	**Sol3**
32809	1351.88	1429.38	1369.99	124	99	52	11.326	12.124	1.008
67121	1480.03	1549.30	1567.19	118	97	60	12.061	13.014	1.257
89101	1156.61	1187.44	1195.00	109	110	62	12.908	15.238	1.334
1234	1322.74	1308.27	1337.05	101	103	48	13.190	13.388	1.096
1997	1301.32	1356.30	1356.25	112	102	56	11.732	13.133	1.310
9997	1224.26	1293.90	1290.73	114	95	49	11.237	12.631	0.875
05240	1516.01	1540.03	1562.27	128	108	40	12.040	13.403	0.851
1872	1232.08	1292.29	1265.83	127	106	49	15.239	13.497	1.119
37562	1323.62	1379.83	1380.77	109	108	45	12.035	12.951	1.055
87523	1259.65	1249.21	1196.74	125	121	49	15.151	13.851	0.770

Valor de la solució,nodes expandits i temps emprat per a trobar-la.

Per tal d'interpretar tots aquests números, generarem els boxplot de cada una de les 3 mesures que estem estudiant.
Primer de tot compararem els boxplot dels nodes expandits i del temps d'execució:





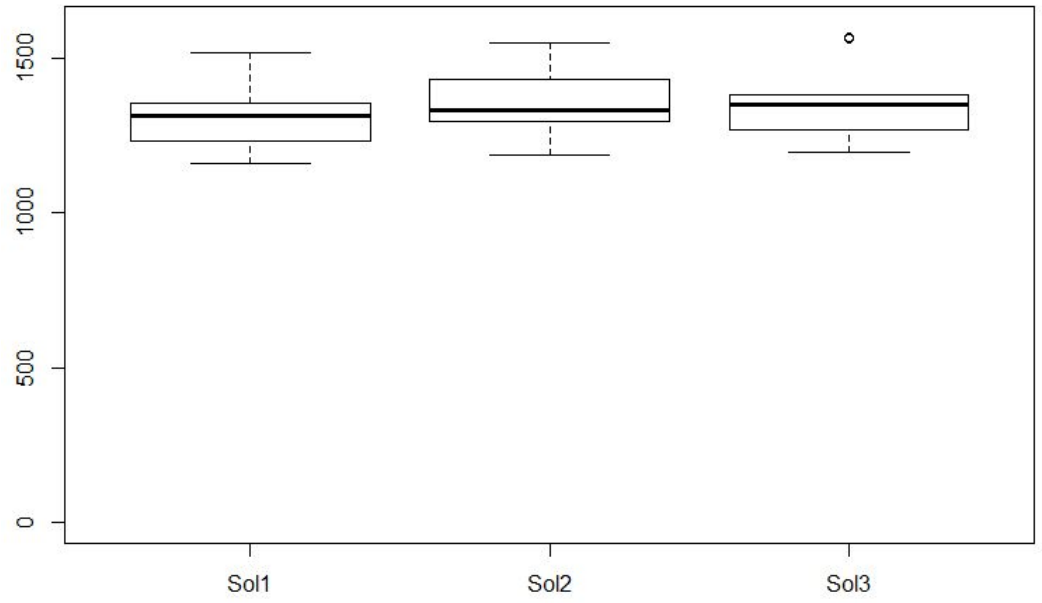
Distribució dels nodes expandits i del temps d'execució.

A simple vista podem deduir que la manera de generar la solució inicial, Sol3, ha de fer molts més moviments, i triga unes 10 vegades menys en executar-se. Per comprovar la probabilitat matemàtica de que sigui Sol3 l'estratègia més adient, la compararem primer amb Sol1, i després amb Sol2.

Dels 10 experiments fets, 10/10 vegades Sol3 ha d'expandir menys nodes que Sol1, i 10/10 vegades Sol3 triga menys temps en executar el programa que Sol1. Tenint en compte que per hipòtesis hauria d'haver la mateixa probabilitat de que Sol3 acabes d'executar-se abans que Sol1, el fet de que els 10 cops sigui Sol3 qui ho fa, te una probabilitat de 0,001. Passa exactament el mateix per a expanded nodes. Per tant, podem refutar l'hipòtesis inicial, ja que la probabilitat es prou minúscula com per afirmar que Sol3 és un millor generador de solució inicial que Sol1, en quant a expansió de nodes i temps d'execució.

Ara que sabem això, caldria comprovar quin és millor entre Sol3 i Sol2, però ens trobem exactament amb la mateixa situació que entre Sol3 i Sol1, per tant podem afirmar que altre vegada es Sol3 qui queda com a millor opció entre aquestes dues, i anàlogament entre les 3.

L'únic argument possible que ens faria no escollir Sol3 com a generador de la nostra solució inicial, seria que aquesta no fos capaç d'arribar a una solució final semblant a les altres dues estratègies, i en canvi en trobés una de molt pitjor. Això no tindria gaire sentit al tenir els mateixos operadors a les 3, però podria passar si generés una solució inicial tan restrictiva que no donés marge de treball als operadors. Per tant hem generat un altre boxplot per comparar les solucions finals que troba cada estratègia d'inicialització.



Distribució del valor de la solució

Com podem observar, és cert que de mitjana Sol3 troba solucions finals pitjors que els altres generadors, i aquest podria ser un motiu per descartar Sol3. Però la diferència és tan petita que les dades obtingudes al observar les altres dues gràfiques son raons de pes suficients perquè Sol3 sigui la nostre millor opció com a estratègia de generació de solució inicial.

Quan vam pensar les 3 diferents estratègies, estàvem convençuts que la millor seria la Sol2, ja que ens semblava que repartia la feina equitativament i donava molt espai als operadors per tal de trobar una bona solució final. Però a diferencia del que pensàvem, Sol3 ha suposat molt poc cost de generació i s´ha apropat prou a una solució final bona, cosa que creiem que es veuria sacrificada en aquest cas.

Això demostra que no es poden fer supòsits i s'han de prendre les decisions a partir dels resultats dels experiments efectuats.

6.3 Determinar paràmetres per Simulated Annealing

Per determinar els paràmetres per al Simulated Annealing, primer em de conèixer com funciona, a diferencia del Hill Climbing, la funció que genera els successors només en genera un aleatori de tots els successors possibles. Si aquest successor és millor que el seu pare l'accepta, si no l'acceptarà amb una certa probabilitat depenent de la iteració en que ens trobem i dels diferents paràmetres que li haguem assignat. Aquests paràmetres que hem de determinar són:

- Màxim d'iteracions: és el numero màxim d'iteracions que farà el Simulated Annealing. Aquí haurem de veure a partir de quina iteració l'algorisme ja no accepta successors pitjors i deixa pas al Hill Climbing. El valor que li donem haurà de ser necessàriament superior a aquest, si no el Simulated Annealing no funcionarà a ple rendiment.
- λ i k: En funció d'aquests dos paràmetres l'algorisme decidirà si accepta un estat pitjor o no. Quan major sigui la lambda més ràpidament es deixarà d'acceptar estats pitjors. Per contra, quan major sigui la k més iteracions d'acceptació d'un estat pitjor hi haurà.
- Iteracions: Aquest algorisme es divideix per fases, en cada fase els paràmetres són els mateixos. Per tant, és necessari que aquest valor sigui divisor de Màxim d'Iteracions.

Primerament determinarem els valors de k i λ. Per a fer-ho estudiarem breument la funció que inspira el Simulated Annealing, la funció per al càlcul de la temperatura del sistema:

$$F(T) = k * e^{-\lambda * T}$$

I la funció que determina l'acceptació d'un estat pitjor:

$$P(estado) = e^{(\frac{\Delta E}{F(T)})}$$

Com podem veure, es tracta d'una funció exponencial i per tant, la probabilitat d'acceptar un estat pitjor mai serà exactament 0, tot i que ajustant els valors de λ i de k podem fer que al tram final de l'execució aquesta probabilitat sigui pràcticament zero.

Per determinar quins valors de λ i de k farem servir prendrem 4 valors per k i 5 per λ de forma totalment arbitrària i executarem l'algorisme per aquests valors. Per tal que el número màxim diteracions no interfereixi, farem servir, de moment, un valor gran com per exemple 200 iteracions i 20 iteracions per fase. Amb 200 iteracions donem suficient marge λ i k trobin una bona solució, considerem que valors superiors a 200 iteracions no són el suficientment eficients comparant-ho amb els gairebé 100 nodes que expandia el Hill Climbing. Un cop executades totes les combinacions (amb 10 seeds diferents per combinació) ens quedarem amb la que hagi obtingut millors resultats i repetirem el procés amb valors ara de λ i k aproximats als obtinguts. Aquests últims seran els valors de k i λ definitius.

tx = temps d'execució (segons) | ts = temps de solució (minuts)

		λ				
		0.0001	**0.001**	**0.01**	**0.1**	**1**
k	**1**	tx = 9.477 ts = 2187.03	tx = 8.726 ts = 2175.03	tx = 13.267 ts = 2187.03	tx = 8.969 ts = 2187.03	tx = 9.861 ts = 2119.03
	5	tx = 10.449 ts = 2187.03	tx = 11.153 ts = 2170.75	tx = 10.422 ts = 2185.98	tx = 10.853 ts = 2166.58	tx = 11.876 ts = 2187.03
	25	tx = 11.046 ts = 2187.03	tx = 12.872 ts = 2187.03	tx = 10.453 ts = 2185.56	tx = 9.226 ts = 2184.25	tx = 10.085 ts = 2187.03
	125	tx = 11.531 ts = 2187.03	tx = 9.276 ts = 2170.80	tx = 8.872 ts = 2187.03	tx = 10.302 ts = 2187.03	tx = 12.032 ts = 2187.03

Un cop fixats els punts de partida de λ (0.001) i k (1)

		λ				
		0.0005	**0.001**	**0.002**	**0.0035**	**0.005**
k	**1**	tx = 9.454 ts = 2161.06	tx = 10.145 ts = 2185.43	tx = 8.519 ts = 2186.08	tx = 9.4 ts = 2187.03	tx = 10.376 ts = 2187.03
	2	tx = 8.264 ts = 2187.03	tx = 9.205 ts = 2187.03	tx = 8.023 ts = 2187.03	tx = 9.552 ts = 2173.48	tx = 8.436 ts = 2187.03
	3	tx = 9.009 ts = 2187.03	tx = 8.783 ts = 2183.91	tx = 8.963 ts = 2187.03	tx = 10.288 ts = 2187.03	tx = 9.844 ts = 2187.03

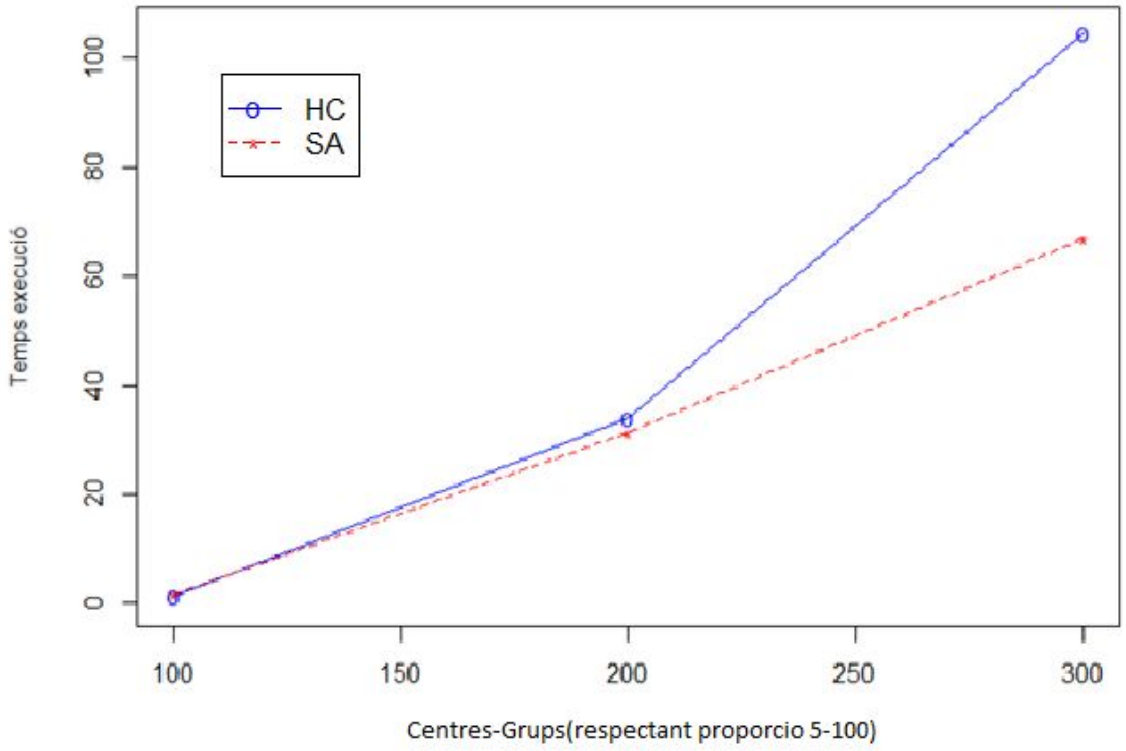
Ara amb k i lambda definides (k = 2, λ = 0.002), procedim a definir el número màxim d'iteracions i les iteracions per fase. Durem a terme el mateix procediment que hem fet servir fins ara.

		Iteracions per fase			
		5	**10**	**25**	**50**
Màxim iteracions	**50**	tx = 3.593 ts = 2187.03	tx = 3.167 ts = 2187.03	tx = 3.089 ts = 2187.03	tx = 3.535 ts = 2187.03
	100	tx = 5.601 ts = 2186.90	tx = 5.799 ts = 2187.03	tx = 5.774 ts = 2187.03	tx = 5.928 ts = 2187.03
	150	tx = 7.716 ts = 2187.03	tx = 7.606 ts = 2187.03	tx = 8.840 ts = 2187.03	tx = 11.378 ts = 2187.03
	200	tx = 11.303 ts = 2187.03	tx = 9.985 ts = 2185.68	tx = 9.091 ts = 2187.03	tx = 14.004 ts = 2187.03

Finalment els paràmetres que donen el millor resultat, que són els que escollim per l'algorsime Simulated Annealing són els següents:

- Màxim d'iteracions = 50
- Iteracions per fase = 25
- k = 2
- λ = 0.002

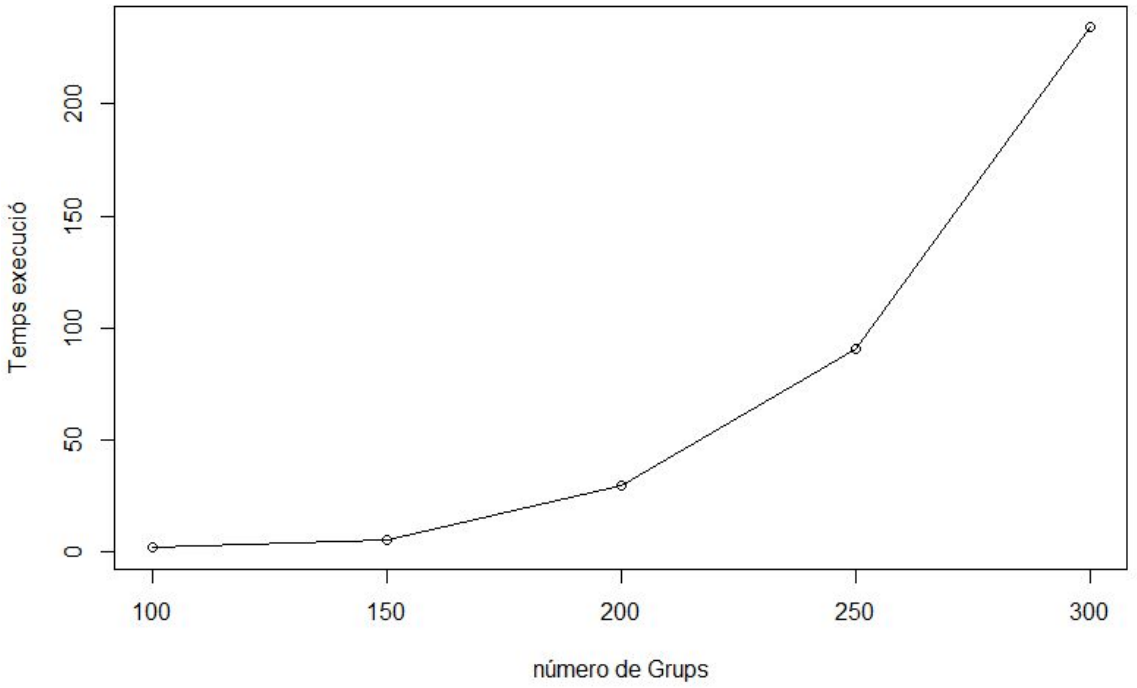
6.4 Evolució temps d'execució proporció 5:100 centros-grupos



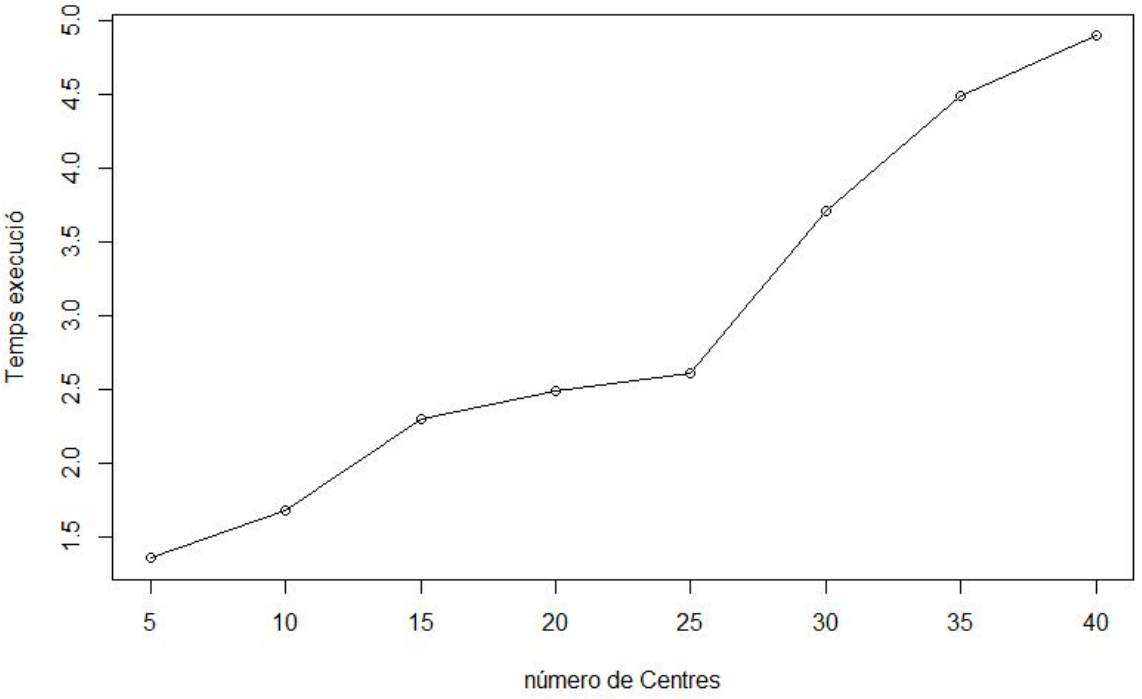
Com podem apreciar a la gràfica, HC i SA es comporten de forma molt similar fins a la cota dels 200 grups (10 centres). A partir d'aquest punt SA és més ràpid en generar una solució que HC perquè la probabilitat d'escollir un operador que genera menys estats successors existeix mentre que HC els ha de generar absolutament tots. Per a valors de *G* entre 350 i 400 el problema es torna impracticable perquè el nostre algorisme de SA és força ineficient ja que genera molts més estats dels que necessita realment. Desafortunadament ens n'hem adonat massa tard d'aquest problema d'optimització i SA, a l'igual que HC no funciona per a valors alts de *G*.

6.5 Evolució temps d'execució per a valors creixents dels parametres del problema

Per a fer aquest experiment, primer procedirem a la recollida de dades i després a graficar els resultats obtinguts, com en tots els experiments usem la mitjana de 10 execucions amb diferents *seeds* cada una:



En aquesta primera gràfica relacionem el temps d'execució amb *G*
Com nosaltres esperàvem els temps d'execució augmenta molt ràpidament amb el nombre de grups que cal salvar. Això és degut a que l'espai de cerca creix de forma factorial amb *G* i els operadors cada vegada han de generar més estats successors fins al punt en el que el problema es fa intractable, seguint l'esquema d'un augment exponencial pel que fa el temps d'execució . Aquest punt arriba cap als 400 grups.



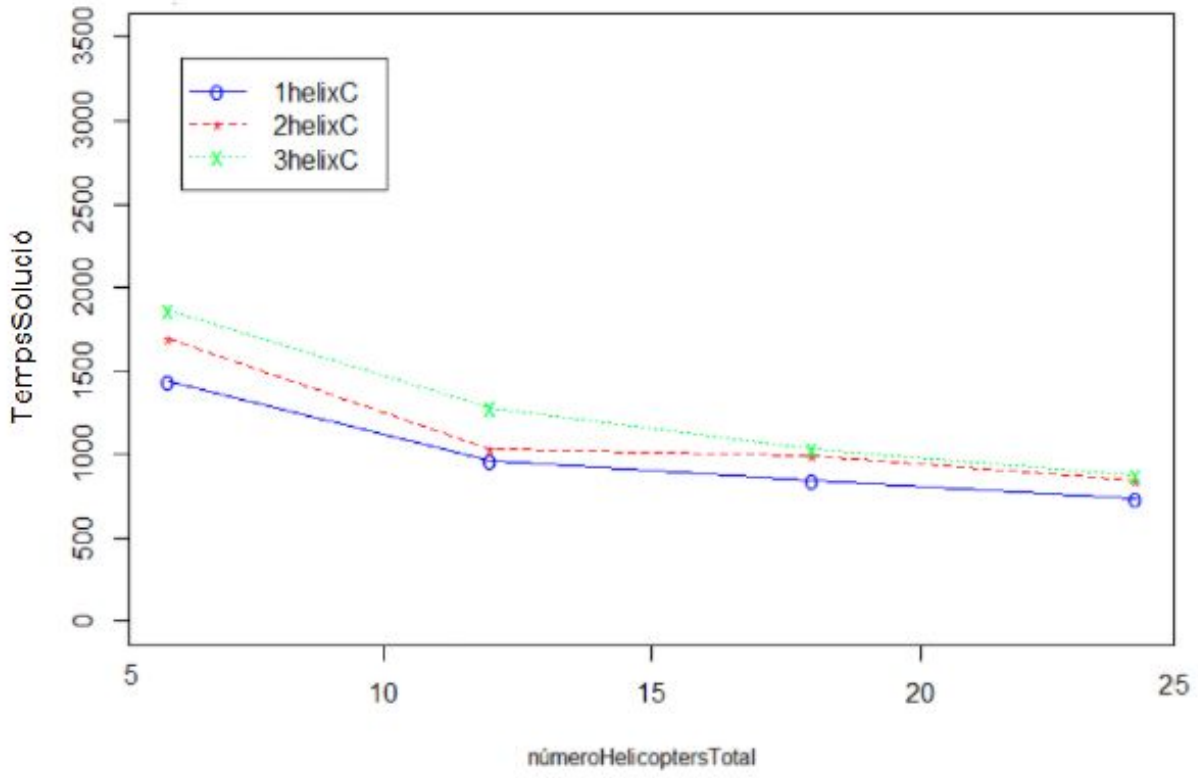
A la segona gràfica, podem veure que la relació entre el temps d'execució del programa i el número de Centres dels que disposem, és quasi lineal. Això és degut a que quants més centres participin al rescat, més espai de solucions hauran de recórrer els operadors, i per tant més temps trigarà el programa en trobar la solució final.
Tot i això, com podem observar a la gràfica, el temps d'execució amb 25 centres és quasi igual al de 20 centres, fet que trenca una mica la linealitat de la gràfica. És perfectament possible, fins i tot podríem trobar un cas en el que un número de centres tingui un temps d'execució menor a una distribució amb menys centres. I això es degut a que tot i que cada cop es disposa d'una quantitat major de nodes successors, l'heurístic pot trobar-se igualment amb la situació en que cap d'ells és millor solució que la que té en aquell moment i per tant pararia.

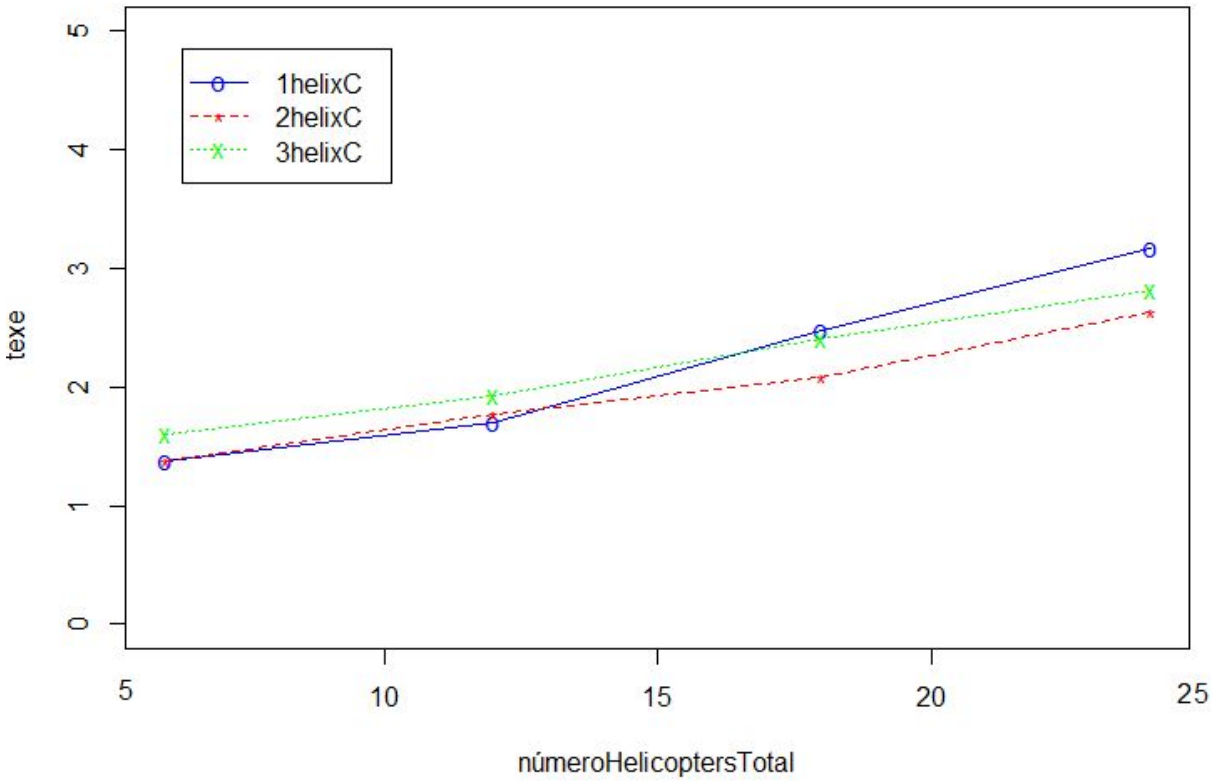
6.6 Estudi del número d'helicòpters per centre

De primeres, pel que fa a la solució que obtindrem, pensem que és millor disposar de més centres que de més helicòpters per centre, ja que l'heurístic fet servir no reparteix la càrrega dels centres entre els seus helicòpters (tal i com ho hem comentat abans a l'hora d'explicar els heurístics) aleshores s'acaben operant pocs helicòpters des de poques bases i creiem que seria millor operar amb el màxim d'helicòpters des de el màxim de bases. així doncs és millor tenir un helicòpter per centre i molts centres i que siguin els centres en funció de la seva posició en el mapa els que es vagin repartint la càrrega que assumeixen.

Per altre banda, respecte al temps d'execució de les diferents solucions, pensem que no hi hauria d'haver molta diferencia entre les solucions donades ja que el numero d'helicòpters es el mateix en tots els casos, així doncs el numero de possibles successors generats hauria de ser molt semblant en tots els casos.

Procedim a fer l'experiment:


1helixC
2helixC
3helixC
TempsSolució
0
500
1000
1500
2000
2500
3000
3500
5
10
15
20
25
númeroHelicoptersTotal

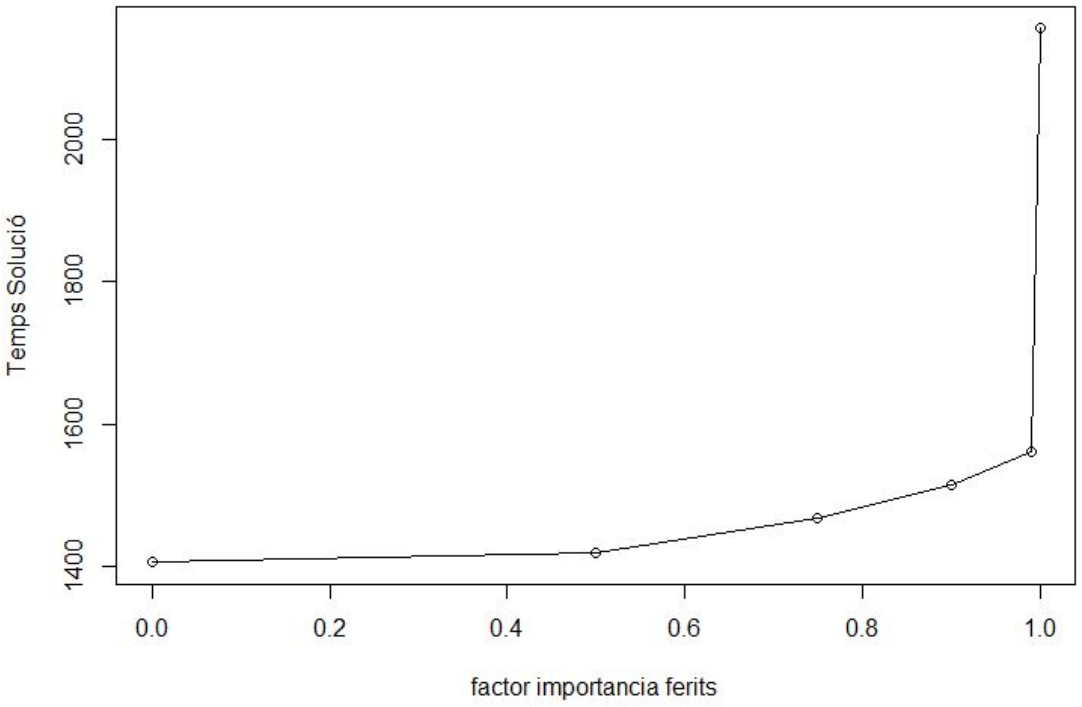

1helixC
2helixC
3helixC
texe
0
1
2
3
4
5
5
10
15
20
25
númeroHelicoptersTotal

Com podem veure a la primera gràfica i tal i com havíem previst, respecte a les solucions obtingudes, el millor dels casos es dona quan hi ha un helicòpter per centre. Degut al que hem comentat anteriorment. Tot i que creiem que la millora seria més substancial. Igualment queda demostrat que és millor tenir pocs helicòpters per centre i tenir més centres.

Respecte al temps d'execució, a partir de 15 helicòpters observem que pel cas d'un helicòpter per centre la solució obtinguda és més costosa que als altres casos. Això pot ser degut a que, com hem explicat abans, aquesta solució ha de repartir la càrrega entre tots els helicòpters. Contràriament, en els casos de més d'un helicòpter per centre, els helicòpters poden quedar solapats pel primer helicòpter de cada centre, que assumeix la major part de la càrrega destinada al centre (recordar que l'heurístic amb el que treballem no reparteix la carrega entre els diferents helicòpters del centre). Així doncs, en aquests casos, l'algorisme "juga" amb menys helicòpters que en el primer cas, que hi "juga" amb tots.

La conclusió que hi extraiem és que tot i que la solució obtingudes més costosa, la millor opció és la del primer cas, quan tenim un helicòpter per centre, ja que evitem sobrecàrregues d'helicòpters (produïdes independentment de la posició del centre) i es reparteix millor la càrrega entre tots els helicòpters.

6.7 Ponderacions 2n Heurístic



Aquest gràfic mostra el temps, en mitjana, que tarden els helicòpters en recollir tots els grups en funció de la importància que se li doni a recollir els grups que tenen ferits respecte als que no en tenen.
Tal i com podem veure en el gràfic, es distingeixen dos intervals: Quan la prioritat és de 0 a 0.99, que s'aprecia un augment, poc significatiu, lineal respecte la prioritat donada. I l'altre part quan la prioritat és de 0.99 a 1, que sí que s'aprecia un augment substancial de temps. Això és degut a que primerament es rescataran els grups que tenen ferits i un cop han estat rescatats es rescataran els grups que no en tenen.
Aquest fet, en el segon tram, fa que es produeixi un coll d'ampolla important amb els grups que tenen ferits. Solucionable, de fet, si no és crucial rescatar primer als grups amb ferits, passant al primer interval, amb valors pròxims a 1 com per exemple 0.99.
Nosaltres hem decidit, doncs, fer servir una prioritat de 0.99 per rescatar els grups amb ferits. Així atorguem una gran prioritat a rescatar als grups amb ferits, però aquesta no és absoluta i permet fer rescats a grups sense ferits abans d'haver rescatat a tots els grups amb ferits.

7.Treball d'innovació

Com a tema a estudiar en el nostre treball d'innovació hem triat el Sistema Nemesis del videojoc "Shadow of Mordor". Aquest sistema simula mitjançant IA, una jerarquia de comandament dins l'exèrcit de Sauron, on cada soldat pot augmentar el seu rang dins del mateix si aconsegueix matar el protagonista de la història controlat pel jugador.

https://www.polygon.com/middle-earth-shadow-of-war-guide/2017/10/9/16439610/the-nemesis-system-and-you

https://www.gamesradar.com/shadow-mordor-nemesis-system-amazing-how-works/

https://www.primagames.com/games/middle-earth-shadow-war/tips/shadow-war-how-nemesis-system-works

Vam triar aquest tema perquè als tres integrants del grup ens sembla molt interessant l'aplicació que pot tenir l'IA dins del món dels videojocs. Vam cercar informació de forma paral·lela i els tres ens hem trobat en que a part d'una petita explicació del funcionament del sistema dins del joc, no hi ha fonts en les que aprofundeixin realment en com intervé la IA. És comprensible ja que es tracta d'una empresa de videojocs amb un sistema revolucionari avui dia, i no està gaire interessat en compartir l'informació que poseeixen. Per tant no sabem fins a quin punt seria adient mirar altres temes a estudiar.