Pràctica de cerca local
Intel·ligència Artificial

Bernat Farrero Badal
Jordi Romero de Villalonga

12 d'Abril de 2010

Índex

Capítol 1

Introducció

En aquesta pràctica es planteja optimitzar l'organització de les rutes d'autobús del poble de *Squaretown*. Es tracta d'una ciutat urbanitzada de forma reticular quadrada (quadrícula de 20 x 20). Per facilitar l'anàlisis del problema, i sense pèrdua de generalitat, d'ara en endavant parlarem d'un taulell quadrat, amb files, columnes i caselles. Es dóna la ubicació fixa de les dues úniques cotxeres d'autobusos que hi haurà disponibles, d'on unívocament començarà o acabarà qualsevol ruta. Aquestes es troben a les caselles (1,1) i (19,19). La nostra primera decisió consisteix en determinar que la generació de rutes es farà sempre des de la cotxera (1,1) a la (19,19). Òbviament, per qualsevol escenari real això seria fàcilment canviable paramètricament.

El problema plantejat és definir K ϵ {2-10} rutes d'autobús que connecten P ϵ {10-50} parades obtingudes aleatòriament. Tenim dos criteris a optimitzar. El primer és la minimització del recorregut total de les rutes. El segon és la maximització de la similitud de les distàncies entre parades.

Una restricció addicional (i opcional) assenyala que totes les rutes tinguin almenys $\frac{P}{2*K}$ parades.

Els algoritmes de cerca local que usarem per realitzar aquests càlculs són el **Hill Climbing** i el **Simulated Annealing**. Amb ells haurem d'experimentar modificant paràmetres, usant diferents configuracions inicials, aplicant o traient restriccions i provant els diferents heurístics per veure quins són els que ofereixen millors resultats.

Capítol 2

Representació de l'estat

El primer a plantejar-se és què és el que considerem **estat** en el context del nostre problema.

Un estat és qualsevol configuració vàlida (o solució candidata) de rutes, recordem que qualsevol problema de cerca local es mou sempre dins l'espai de solucions i no en tot l'espai de camins, per tant, tot estat és solució.

Definim una configuració com a vàlida quan compleix que totes les rutes comencen per la casella (1,1) i acaben a la (19, 19). A més, totes tenen almenys una parada assignada i, només en el cas que s'apliqui la restricció addicional, cada ruta té almenys $\frac{P}{2*K}$ parades.

2.1 Espai de cerca

L'espai de cerca és molt gran tenint en compte la quantitat de combinacions possibles que es poden donar. Precisament per això, s'encara el problema mitjançant una estratègia de cerca local, que evita visitar gran part de l'espai de cerca. En general, l'espai de cerca s'obté recorrent totes les possibles permutacions de parades dins d'una mateixa ruta per cada configuració nova de parades de la ruta. Això és així, evidentment, sempre que la nova configuració sigui un estat vàlid.

Es pot comprovar que l'espai de cerca creix ràpidament segons els paràmetres P i es redueix en funció del número de rutes K. Per calcular-lo considerem les possibles combinacions de P en K grups i després les permutacions de les P parades dins de cada grup. En particular, és de l'ordre següent:

$$\Theta(C^k_{p-k}P(P)) = \Theta(\binom{P-K}{K}P!) = \Theta(\frac{(P-K)!P!}{K!(P-2K)!})$$

Val a dir que el nombre de combinacions no es fa sobre P sinó sobre (P - K) per ser coherents amb la restricció que cada parada ha de tenir almenys una ruta. En cas d'aplicar la restricció addicional, l'espai de cerca es redueix parcialment, ja que el nombre de dalt del binomi aquest cop és $P - \frac{P}{2K} = P(2K-1)$. En concret:

$$\Theta(C^k_{p(2K-1)}P(P)) = \Theta(\binom{P(2K-1)}{K}P!) = \Theta(\frac{(P(2K-1)!P!}{K!(P(2K-1)-K)!})$$

2.2 Estructures de dades

Per emmagatzemar l'estat hem dissenyat les següents estructures de dades.

2.2.1 Parada

Una parada consta d'una posició (x, y) (per les columnes i files respectivament) dins del taulell i un identificador de parada. La parada és la unitat bàsica del nostre problema, és el que s'assignarà a diferents rutes i en diferents posicions dins d'una mateixa ruta. Per facilitar la feina, i posat que la independència i reutilització de les classes no és cap prioritat en aquesta pràctica, hem inclòs la informació de la ruta actual a la parada. També hem implementat dos mètodes per calcular les distàncies entre parades.

El primer **distParadaFisica** calcula la distància física entre dos parades, que sempre és la mínima. Ho fa tal com indica l'anunciat, de la següent manera:

$$(p1, p2) = |p1_x - p2_x| + |p1_y - p2_y|$$

$p1_x$ i $p2_y$ són els valors de les coordenades (x,y) de dues parades consecutives.

El segon mètode **distParada** calcula la distància entre parades a través de les rutes. En cas que l'altra parada no es trobi a la mateixa ruta, es calcula la distància fent transbord entre rutes a la prada (1,1) o (19,19).

Ambdós mètodes calculen sempre la distància mínima, ja que la minimització del recorregut de les rutes és objectiu del problema.

2.2.2 Ruta

Una ruta consta principalment d'un vector de parades (en particular, emmagatzemem tant sols l'identificador de les parades d'aquella ruta). L'algorisme que implementa la nostra solució farà variar aquestes parades fins trobar l'òptim local entre totes les rutes. Per ajudar en l'implementació, les rutes també contenen la informació del seu nombre de parades i la seva distància (el mètode **calcularDist()** s'encarrega de calcular-la després de l'aplicació d'un operador).

Dins de la classe Ruta s'implementen també els operadors, dels quals es parlarà més endavant.

2.2.3 Vector de rutes

El vector de rutes conté totes les rutes creades en cada moment. La solució inicial les inicialitza al principi i els operadors el modifiquen fins obtenir l'òptim local.

2.2.4 Vector de parades

El vector de parades conté la configuració de parades de la ciutat. Aquesta configuració s'obté aleatòriament des d'un inici i no és modificable pel problema.

Capítol 3

Generació de la solució inicial

Hem implementat dos algorismes diferents d'obtenció de l'estat inicial per tal de poder comparar-los i veure quin porta a millors estats finals. A priori, tal com funciona la cerca local, si l'estat inicial de partença és bo hi ha més probabilitats d'obtenir un resultat òptim (o millor). En ambdós casos, els estats es troben dins l'espai de solucions i per això els anomenem solucions inicials.

3.1 Solució inicial aleatòria

Aquesta solució (mètode anomenat **solIni2**) té com a prioritat la rapidesa i l'aleatorietat de la seva obtenció i no la seva qualitat. En cada iteració es procedeix a afegir una parada escollida consecutivament a la ruta. Quan totes les parades han estat assignades a alguna ruta, l'algorisme s'atura. La configuració obtinguda no només és solució del problema, sinó que compleix amb la restricció addicional proposada d'entrada. L'assignació de parades és homogènia entre les rutes i l'aleatorietat no prové de l'algorisme en si sinó de la configuració de les parades que es genera aleatòriament per cada escenari. El cost de la funció és lineal $\Theta(P)$.

3.2 Solució inicial de qualitat

Aquesta solució (mètode anomenat **solIni1**), en canvi, intenta ser bona d'entrada amb l'esperança que facilitarà als algoritmes de cerca local arribar a un bon òptim local o bé a l'òptim global. Per fer-ho, la solució incorpora coneixement del problema a minimitzar.

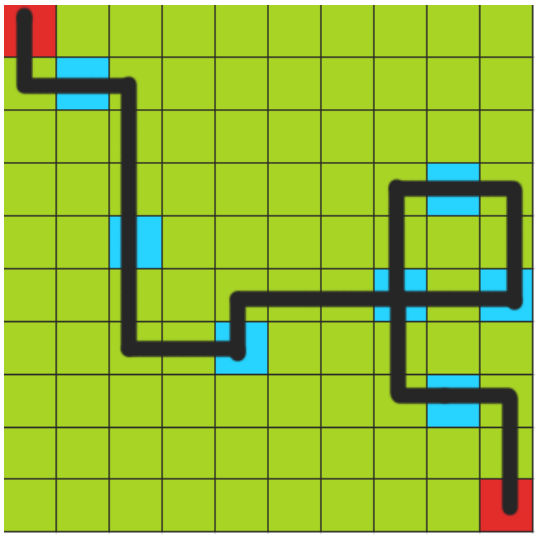
En concret, el procediment consisteix en anar afegint a cada ruta la parada més propera a l'última parada afegida i, a la vegada, més propera a la parada final. Aquesta última restricció evita que apareguin *recorreguts en cercle* (veure figures 3.1 i 3.2). Aquestes comprovacions faciliten que la solució obtinguda segueixi

el primer criteri a optimitzar (minimitzar el recorregut de les rutes), però no garanteixen explícitament millores pel que fa al segon criteri. Tanmateix, pel fet d'escollir sempre les distàncies menors entre parades estarem també reduint la diferència de la separació entre parades (que altrament seria aleatòria), així que també oferirà millors resultats pel que fa al compliment del segon criteri a optimitzar (minimitzar diferència entre separació de parades) respecte a la solució aleatòria.

A més, com que la repartició de les parades entre rutes es fa de forma homogènia, també garanteix d'entrada el compliment de la restricció addicional, posat que totes les parades tindran el mateix número de rutes o una diferència de 1.

En definitiva, aquest algorisme ofereix una solució raonablement bona amb un cost de l'ordre $\Theta(K * P)$.

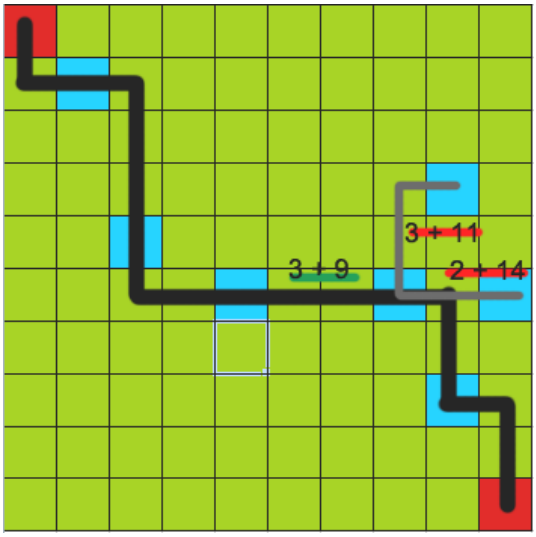
Figura 3.1: No es té en compte la distància entre la parada candidata i el final, les solucions poden caure en recorreguts en cercle.



3.3 Variant de la solució aleatòria

Malgrat que la solució aleatòria proporcionada anteriorment és perfectament vàlida, hem creat una última solució inicial (mètode anomenat **solIni3**) que és indeterminista. Per un taulell donat, els dos mètodes anteriors sempre ofereixen la mateixa solució inicial. En canvi, aquest mètode és totalment aleatori. El nombre de parades per ruta no és homogeni i compleix només la restricció que cada ruta tingui almenys una parada assignada (no compleix la restricció addicional). El cost de l'algorisme és indeterminat, depèn de les col·lisions que es produeixen en el bucle que assigna aleatòriament una parada per ruta (tampoc determinada). La creació d'aquesta solució addicional està motivada pel fet

Figura 3.2: Es tenen en compte ambdues distàncies, el problema desapareix.



d'experimentar amb l'estratègia de Hill Climbing amb reinici aleatori, explicada més endavant.

Capítol 4

Operadors

De cara a que l'algorisme de cerca local en qüestió sigui capaç de moure's per l'espai de solucions és necessari implementar uns operadors. Aquests operadors tenen com a finalitat crear un estat diferent (on els paràmetres de qualitat tenen valors potencialment diferents) veí a l'estat al que se li aplica l'operador. Això significa que aquests ens permeten obtenir un estat contigu a l'original. D'aquesta manera, l'algorisme de cerca local pot consultar tots els estats propers per veure quin és el que promet millors resultats.

El mecanisme utilitzat per la implementació d'AIMA és sol·licitar per un estat quins són tots els possibles successors partint d'ell. Per això, el que fem és generar tots els possibles estats que sorgeixen d'aplicar un operador (en un sol pas) i retornar-los a l'algorisme perquè segueixi amb la cerca.

Per la implementació del problema que hem dut a terme, els operadors mai s'apliquen a les parades corresponents a les cotxeres, i pel que fa a aquest apartat, aquestes parades les considerem especials i no entren en joc.

En el problema que tractem, hem reduit qualsevol possible canvi d'estat en l'aplicació d'un d'aquests dos operadors:

- Canviar una parada de ruta
- Moure de posició una parada dins d'una ruta

Amb una combinació d'aquests dos operadors es pot arribar a qualsevol possible que sigui solució. Com expliquem a continuació, el comportament d'aquests operadors està restringit de manera que no puguin conduir a ambigüitats del seu comportament.

4.1 Canviar de ruta

Amb aquest operador pretenem assolir que qualsevol parada pugui acabar a qualsevol ruta. Per tant, aplicant aquest operador per totes les parades i per totes les rutes tenim totes les possibles combinacions parada-ruta (o sigui C^K_{P-K}).

$$C_{P-K}^{K} = \binom{P-K}{K} = \frac{(P-K)!}{K!((P-K)-K)!}$$

La única **condició d'aplicabilitat** d'aquest operador és que:

- La ruta a la que pertany la parada en qüestió ha de tenir més d'una parada

Això ens garanteix no sortir en cap momenet de l'espai de solucions. Aquesta restricció però no implica que no es pugui arribar a tot l'espai de solucions, perquè l'únic que cal és aplicar-la en un altre estat on aquella parada no sigui la única que queda en la seva ruta.

El **factor de ramificació** d'aquest operador és $P \times K - 1$. Podem moure cada parada a qualsevol de les altres rutes.

4.1.1 Restricció addicional

En aquest operador s'aplica la restricció addicional creant una nova condició d'aplicabilitat, que és que:

- La ruta a la que pertany la parada en qüestió té al menys $\frac{P}{2*K}$ parades

4.2 Moure en ruta

Un cop vist que podem arribar a qualsevol combinació entre parades i rutes, l'únic que falta per poder cobrir l'espai de solucions complet és que les parades dins d'una ruta puguin estar ordenades de qualsevol manera. Això significa generar totes les permutacions de les parades d'una ruta, o sigui P_{P_i} on P_i és el número de parades de la ruta i.

$$P_{P_i} = P_i!$$

En la implementació de l'operador l'hem simplificat de manera que compleixi el mateix objectiu però aplicant sempre el mateix moviment:

- Intercanviar la parada en qüestió amb la propera parada de la seva ruta, menys quan es tracta de la darrera parada que aleshores s'intercanvia amb la primera.

Amb aquesta simplificació del moviment podem igualment assolir qualsevol ordenació possible (P_{P_i}) però amb més control de quin és el moviment que s'està efectuant.

Aquest operador no té **condicions d'aplicabilitat**, ja que es pot usar sempre, fins i tot quan es tracta de l'única parada de la seva ruta, doncs l'intercanvi amb si mateixa és semànticament correcte, tot i que evitem utilitzar-lo en aquest cas perquè no es genera un estat diferent. El seu **factor de ramificació** és P, ja que podem escollir moure en una posició qualsevol parada.

Capítol 5

Funció de qualitat

5.1 Definició

Per tal d'avaluar la qualitat d'una solució cal fer un càlcul respecte certs paràmetres de l'estat.

Segons l'enunciat, la qualitat d'una solució ve definida pels següents factors:

- El recorregut total de les rutes ha de ser mínim
- La distància del recorregut entre qualsevol parell de parades ha de ser el més similar possible

En el context del problema, amb el primer factor es pretén estalviar combustible, mentre que amb el segon es pretén crear recorreguts equitatius per viatjar entre quasevol parell de parades de la ciutat.

En termes directament aplicables a la nostra implementació de l'estat això es pot transformar en:

- La suma de les distàncies totals de totes les rutes ha de ser mínima
- La suma de desviacions estàndard de les distancies entre tots els parells de rutes ha de ser mínima

5.2 Heurístics

5.2.1 Heurístic 1: Recorregut total de les rutes

Per tal d'obtenir aquest valor heurístic, com hem explicat, sumarem les distàncies totals de cada ruta, càlcul que podem fer només consultant per cada ruta un camp on s'emmagatzema aquest valor.

$$h_1 = \sum_{i=1}^{K} \sum_{j=1}^{P_i - 1} dist(parada_j, parada_{j+1})$$

On K és el número de rutes, P_i és el número de parades que té la ruta i (incloent origen i final) i $dist(a, b)$ retorna la distància entre la parada a i la b anant en autobús.

5.2.2 Heurístic 2: Desviació estàndard de distàncies entre parades

En aquest cas el que ens interessa és minimitzar la diferència de distàncies entre un parell qualsevol de parades, per tant el que fem és calcular la desviació estàndard de totes les distàncies entre parells de parades. Això ho fem amb la fórmula següent

$$h_2 = \sum_{i=1}^{P} \sum_{j=i+1}^{P} \|dist(parada_i, parada_j) - \overline{X}\|$$

On P és el número total de parades, $dist(a, b)$ retorna la distància entre la parada a i la b anant en autobús i $\overline{X}$ és el promig de totes les distàncies entre parells de parades, calculat així:

$$\overline{X} = \frac{2}{P^2 - P} \sum_{i=1}^{P} \sum_{j=i+1}^{P} dist(parada_i, parada_j)$$

5.2.3 Heurístic 3: Combinació dels anteriors

Per tal de trobar una solució al problema que potencii els dos paràmetres de qualitat definits, cal provar una funció heurística que combini les dues anteriors. Aquesta combinació s'ha de fer ponderant els valors d'aquestes per tal de compensar el pes d'una i l'altra.

El valor d'aquesta constant que hem anomenat $k_{h_{21}}$ l'hem obtingut a partir d'experimentar amb diferents possibilitats tal com s'explica a l'apartat 6.5.

$$h_3 = (1 - k_{h_{21}}) \times h_1 + k_{h_{21}} \times h_2$$

Anomenem $k_{h_{21}}$ al valor que utilitzem per ponderar el segon heurístic respecte el primer.

Capítol 6

Experimentació

Per tal de justificar decisions preses durant el transcurs de la pràctica, o bé per comprovar-ne l'eficàcia, s'han anat duent a terme experiments que a continuació llistem i mostrem en profunditat.

El primer que es demanava era calibrar l'algorisme de *Simulated Annealing*, i per tant provar possibles valors pels 4 paràmetres. Es pot consultar a l'apartat 6.1.

A continuació a la secció 6.2 es comparen les diferents solucions inicials, observant-ne la qualitat i el temps d'execució necessari.

També s'analitza la influència del paràmetre K (nombre de rutes) en el cost i la qualitat de la cerca a l'apartat 6.3.

En quant als paràmetres de la cerca, comencem comparant els efectes d'usar l'heurístic de la distància total contra el de les distàncies entre parades semblants (6.4). Després calculem el valor òptim de la ponderació d'aquests dos en un tercer heurístic (6.5). Finalment comparem els dos algorismes de cerca: Hill Climbing i Simulated Annealing (amb els paràmetres òptims experimentats) a l'apartat 6.6.

Com a afegit, provem quin és el resultat si es prenen consideració la restricció addicional (6.7).

6.1 Paràmetres de Simulated Annealing

L'algorisme *Simulated Annealing* pren quatre paràmetres per definir el seu comportament. Aquests serveixen per indicar de quina manera ha d'efectuar l'*escalfament* que orientarà l'algorisme.

Els paràmetres són:

- Iteracions
- Iteracions per cada pas de temperatura
- Paràmetre K
- Paràmetre λ

A continuació mostrem els resultats d'experimentar amb possibles valors de cadascun d'ells. Per aquests experiments s'ha triat fixar tres dels valors amb els valors per defecte que assigna AIMA si no s'especifiquen (10000, 100, 20 i 0.045 respectivament) i en el valor experimentat provar un ampli ventall de possibilitats, per observar gràficament quina és la opció que ofereix millor resultats.

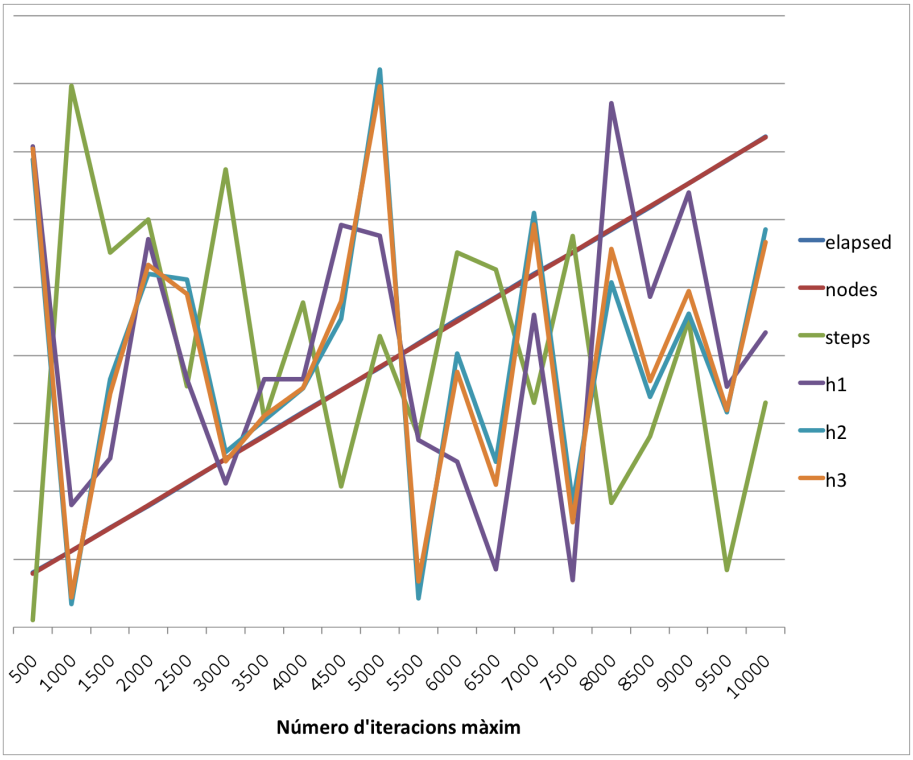
En els gràfics que es mostren a continuació tots els valors s'han normalitzat per tal de poder observar si l'efecte és positiu o negatiu en cada un dels paràmetres observats. Simplement convé observar quins valors creixen i quins decreixen amb diferents possibilitats en els paràmetres de *Simulated Annealing*.

Per a executar aquestes proves hem escollit la solució inicial de qualitat. S'ha executat 10 vegades la cerca per cada tauler i valor del paràmetre en concret, amb 10 taulers diferents. Els gràfics estan fets amb el promig d'aquests 100 resultats (per valor per paràmetre).

6.1.1 Iteracions màximes

Amb aquest paràmetre definim el número màxim d'iteracions, de certa manera limitant tota la resta de factors obtinguts per l'algorisme perquè no desemboqui en massa càrrega de treball.

Figura 6.1: Influència del paràmetre iteracions màximes



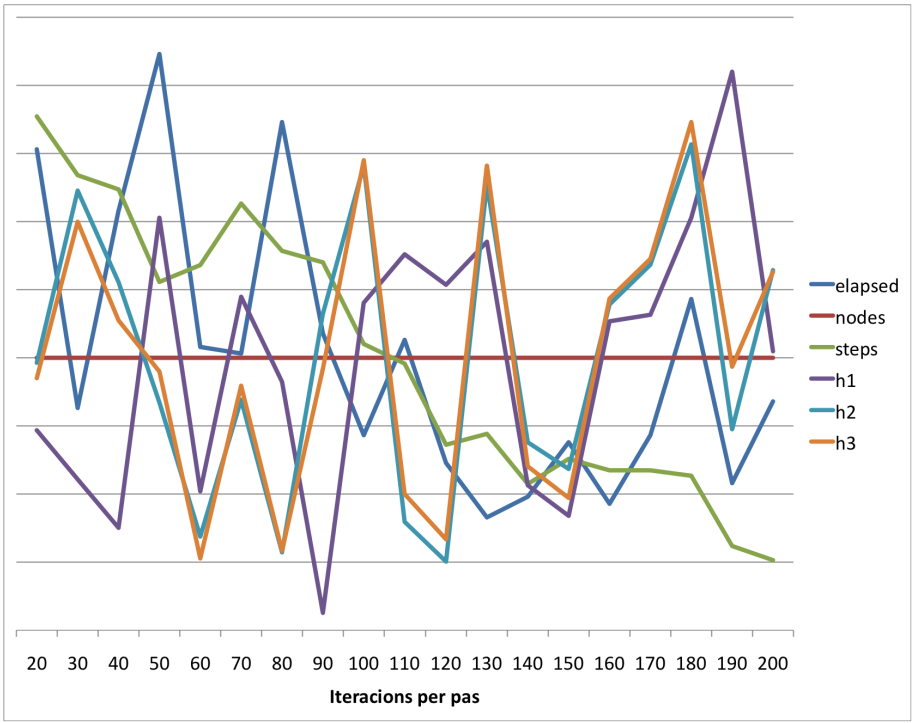
A la figura 6.1 observem que a mesura que el número màxim d'iteracions creix (en un rang entre 500 i 10000) creix també linealment el número de nodes expandits i l'*elapsed time*. D'una manera molt més erràtica i menys clara es pot veure que els valors dels tres heurístics es mouen d'una forma similar, i el número de passos fins a la millor solució trobada també segueix una mica el mateix patró. Sense que sigui una diferència molt gran, al punt de les **5500** iteracions s'observa una davallada dels heurístics i el número de passos, i en la zona fins a les 7500 iteracions és on s'observen els millors resultats globals.

Per aquest fet s'ha escollit que el valor òptim d'aquest paràmetre és **5500** iteracions.

6.1.2 Iteracions per cada pas de temperatura

L'algorisme de *Simulated Annealing* va variant el valor de la temperatura que empra per guiar-se, i en cada temperatura diferent realitza un número d'iteracions. Aquest és el paràmetre que estudiem ara. En el codi per defecte d'AIMA s'utilitza un valor de 100 iteracions, i per aquest motiu hem decidit provar els valors entre 20 i 200 amb increments de 10 per escollir-ne l'òptim.

Figura 6.2: Influència del paràmetre iteracions per pas



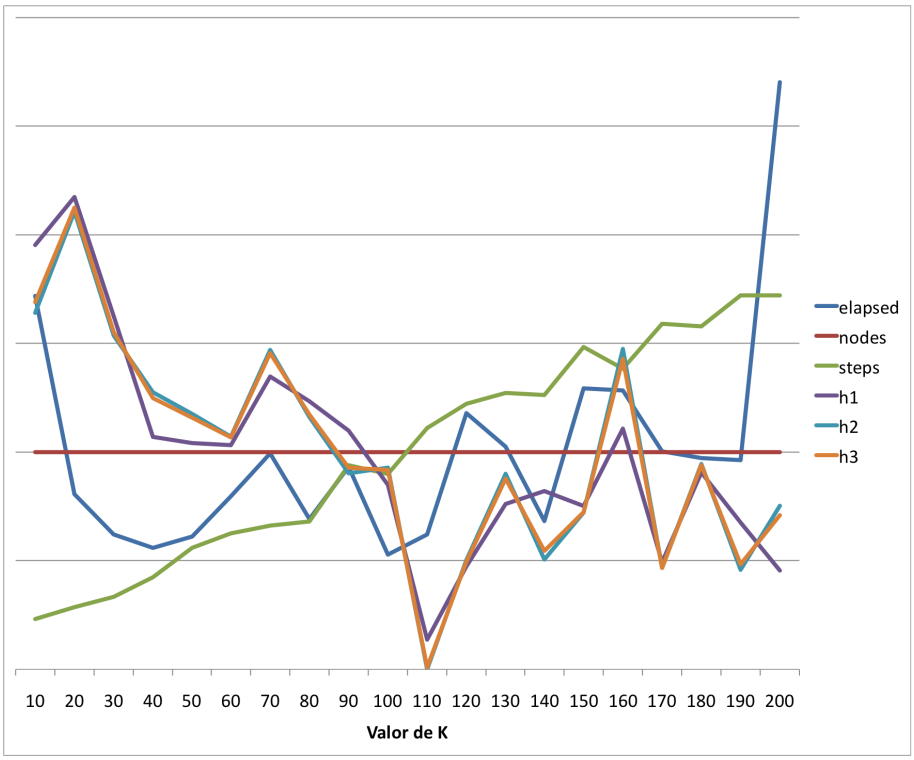
En aquest cas no tenim cap increment lineal sino un valor constant, que és el número de nodes expandits. Aquest factor no és afectat pel número d'iteracions per pas. Per altra banda, observem que el número de passos sí que disminueix a mesura que el paràmetre augmenta, i l'*elapsed time* també tendeix a decréixer. Els heurístics tenen un comportament menys clar, tot i que en aquesta gràfica s'observa que al punt de les **150** iteracions tots els valors coincideixen en un mínim que fa clara l'elecció a prendre.

Tot i que el valor dels heurístics assoleix un mínim absolut a la zona dels 60-80, la combinació de tots els factors és clarament guanyadora en el punt dels 150. Si l'objectiu fos conseguir solucions amb un número de passos petit, es podria escollir el número més gran que minimitza aquest valor. En el nostre cas ens és irrellevant el camí per arribar a la millor solució.

6.1.3 Paràmetre K

El valor de K determina quant triga la temperatura en començar a descendir. El valor per defecte d'AIMA és de 20 i nosaltres hem provat totes les possibilitats entre 10 i 200 de 10 en 10.

Figura 6.3: Influència del paràmetre K



Podem comprovar que a els heurístics mostren una tendència a decréixer a mesura que augmenta K, i en canvi el número de passos incrementa. El valor de l'*elapsed time* es comporta d'una manera irregular, tot i que amb els valors dels extrems aquest augmenta molt.

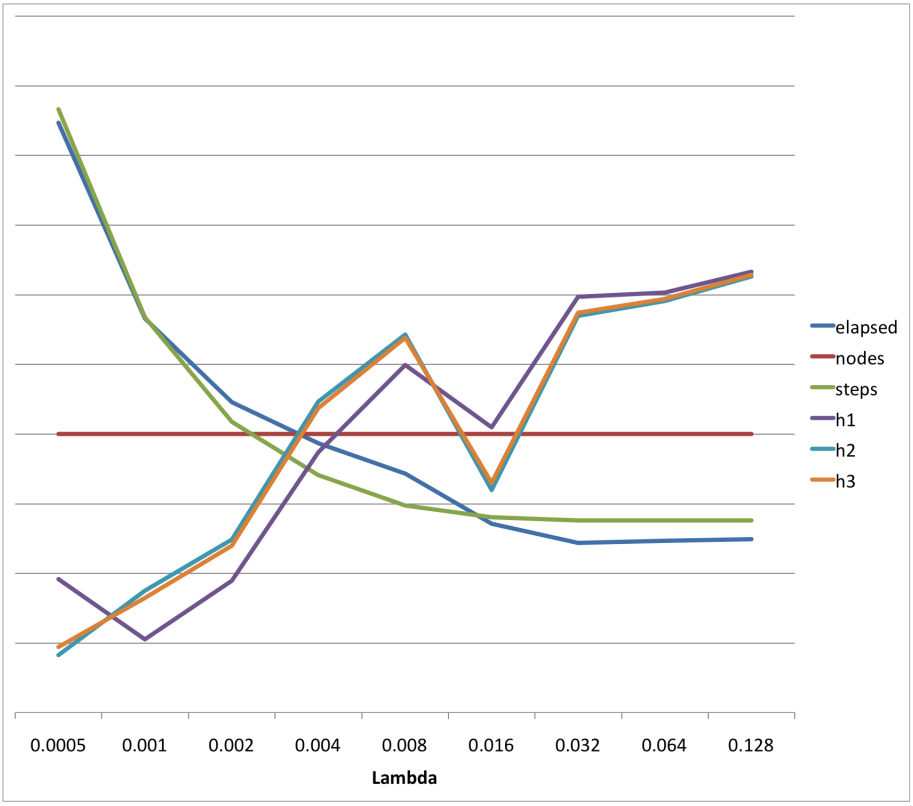
Si el que es vol és millorar la qualitat dels heurístics, el punt òptim es troba quan $k = 110$ ja que aquests assoleixen un mínim, mentre que el número de passos tampoc és molt gran i l'*elapsed time* també és moderat. Altres valors positius són el 120, 140 i 170, amb uns valors similars.

Escollim com a valor per defecte del paràmetre K **110**.

6.1.4 Paràmetre λ

El paràmetre λ és el que marca la pendent del refredament de la funció, o sigui com n'és de ràpid aquest refredament un cop comença. A partir dels gràfics observats a les transparències sobre AIMA i el valor per defecte que aquesta classe defineix per λ, decidim provar els possibles valors entre 0.0005 i 0.128 incrementant a base de multiplicar per dos. D'aquesta manera podem contemplar el comportament de la cerca amb un ampli ventall de valors però sense haver de provar-ne milers. Els resultats són bastant clars en el següent gràfic:

Figura 6.4: Influència del paràmetre λ



Degut a la naturalesa exponencial dels valors de l'eix X s'han d'interpretar els resultats exponencialment. S'observa que l'*elapsed time* i el número de passos fins a la solució decrementa molt amb els primers valors de λ, i que tots els heurístics prenen millors valors amb les λ més petites. Per trobar u ncompromís es pot escollir una λ de **0.002** que obté molt bons valors en els heurístics i l'*elapsed time* ha caigut bastant respecte els valors més petits.

6.1.5 Resum dels paràmetres de Simulated Annealing

Després de comentar un per un els paràmetres requerits per l'algorisme de Simulated Annealing, mostrem les conclusions d'aquests experiments: Hem partit de la informació bàsica proporcionada pels apunts d'AIMA i el seu propi codi, on hem observat quins valors són raonables aplicar a aquest algorisme. A partir d'aquí hem decidit provar-ne tants com fos oportú per tenir la certesa de trobar l'òptim. Normalment hem provat 20 valors diferents per cada paràmetre, i ens hem quedat amb el que assolia millors resultats tant en el valor dels heurístics com (en menys mesura) els valors de l'*elapsed time* i el número de passos.

Figura 6.5: Taula amb els valors dels paràmetres per Simulated Annealing

	Valor AIMA	**Valors**	**Resultat**
Iteracions màximes	10000	500 .. 10000	5500
Iteracions per pas	100	20 .. 200	150
K	20	10 .. 200	110
λ	0.045	0.0005 .. 0.128	0.02

Donada la precisió amb la que hem intentat fer els experiments (amb 10 iteracions per cada possible valor) i repetint tot l'experiment en 10 taulers diferents (per tant 100 execucions per cada valor de cada paràmetre), amb un total de 8000 cerques per elaborar aquests quatre gràfics (evidentment automatitzades amb el codi de la classe `Main`), creiem que la diferència entre els resultats obtinguts i els paràmetres per defecte d'AIMA (notable tot i que dins dels paràmetres raonables) han de ser deguts a la peculiaritat del nostre problema. Tots els valors que hem trobat han estat dins del que cabia esperar i han millorat la qualitat de la cerca.

6.2 Influència de les solucions inicials

6.2.1 Comparació entre solució inicial 1 i 2

Tal com indica la teoria de la cerca local, obtenir una bona solució inicial fa més fàcil l'obtenció de millors resultats. Ho hem comprovat realitzant 10 execucions amb una mateixa configuració de parades. En la següent taula es pot veure la mitjana d'aquestes 10 execucions. En aquest cas, només s'intenta maximitzar l'heurístic 1, tal com demana l'enunciat.

Recordem que la solució inicial 1 és aquella que incorpora coneixement del problema i intenta optimitzar d'entrada la configuració inicial. La solució 2 és aquella generada immediatament a partir de la configuració aleatòria de parades.

Valor de l'heurístic 1 de la solució inicial 1: **288.0**

Valor de l'heurístic 1 de la solució inicial 2: **384.0**

Experiment	Elapsed	H1
HC - Solució Inicial 1	8	256,00
HC - Solució Inicial 2	2	308,00
SA - Solució Inicial 1	1209	229,00
SA - Solució Inicial 2	1215	237,20

HC Hill Climbing
SA Simulated Annealing

L'ús de la solució inicial 1 millora els resultats de la solució 2 en un **17%** en el cas de l'algorisme Hill Climbing i en un **4%** en el cas de Simulated Annealing. D'això s'infereix que una bona solució inicial porta millors resultats pel Hill Climbing que pel Simulated Annealing.

En els altres experiments, amb diferents paràmetres, amb diferents números de parades i rutes hem observat que la solució inicial de qualitat sempre millora els resultats notablement. Hem extret la conclusió que val la pena usar sempre una solució voraç.

6.2.2 Experiment addicional: Hill Climbing amb reinici aleatori

Hem volgut comprovar com es podia millorar els resultats del Hill Climbing mitjançant una estratègia de reinici aleatori (tal com s'ha explicat a classe). Per això hem executat l'algorisme de Hill Climbing amb 100 reinicis aleatoris, cadascuna amb una solució inicial aleatòria diferent (per això hem creat la solució inicial 3) per trobar el valor mínim de l'heurístic 1 que s'obtenia. Aquest experiment l'hem realitzat 10 vegades i n'hem fet la mitjana en la següent taula:

Experiment	H1 en HC	Elapsed
Solució Inicial 1 HC	130,00	0
Solució Inicial 2 HC	150,00	1
Solució Inicial 3 HC	147,00	0
Solució Inicial 1 SA	121,00	631
Solució Inicial 2 SA	118,00	625
Solució Inicial 3 SA	122,00	637
RRHC - Sol. Inicial 3 x 50	122,00	77

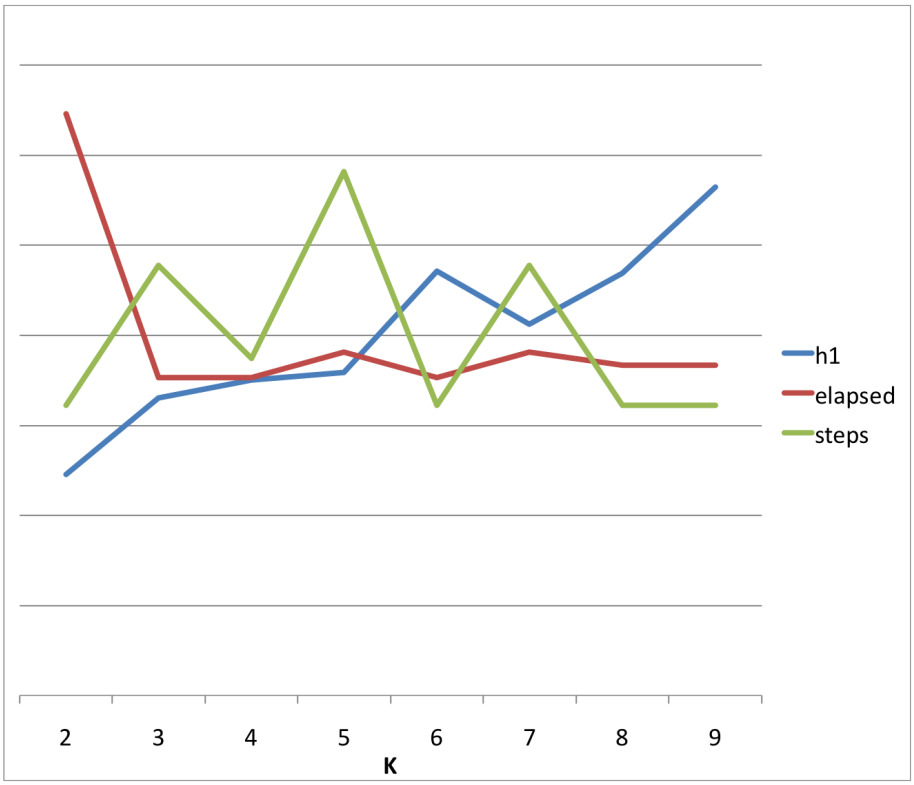
HC Hill Climbing
SA Simulated Annealing
RRHC Random Restarting Hill Climbing

Es pot comprovar que la solució de qualitat aquí segueix millorant les aleatòries però quan usem l'estratègia de reinici aleatori amb 100 reinicis obtenim per un cost, molt més econòmic que el de la solució de Simulated Annealing, resultats semblants a aquest. En particular, obtenim un resultat 8 punts millor que la solució obtinguda amb Hill Climbing + solució inicial de qualitat i tant sols 4 punts pitjor que Simulated Annealing + solució inicial de qualitat, la mateixa solució obtinguda amb Simulated Annealing i solució aleatòria. Cal fixar-se el temps d'execució dels algorismes de Simulated Annealing, de l'ordre de 630 u.t. mentre que el HC amb Reinici Aleatori té un cost de l'ordre dels 80 u.t.

6.3 Com varia la dificultat del problema en funció de K?

Per comprovar com afecta el número de rutes en la dificultat del problema hem aplicat 10 vegades els algorismes de Hill Climbing i Simulated Annealing sobre una configuració inicial aleatòria de 30 parades i n'hem obtingut la mitjana. Els resultats es mostren als següents gràfics (que han estat normalitzats).

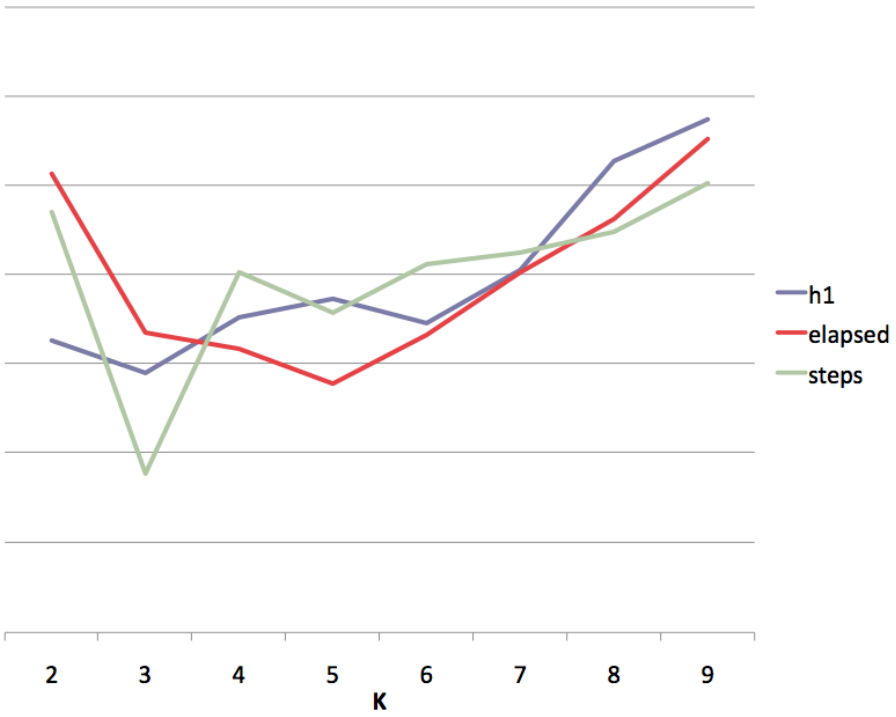
Figura 6.6: Heurístic 1, temps d'execució (*elapsed*) i passos (*steps*) de l'algorisme Hill Climbing en funció de K.



En el gràfic de la Figura 6.6 es pot comprovar com evolucionen els paràmetres en l'algorisme de Hill Climbing a mesura que va augmentant el nombre de rutes (K). La primera observació és el pic del temps d'execució de l'algorisme quan la K és 2 (es troba en l'ordre de 30 u.t.) i es redueix progressivament a partir de 3 fins mantenir-se estable per la resta de valors de K. Els passos de l'algorisme canvien relativament poc (es mouen en l'interval de 10 a 15) i el seu màxim es troba en la K = 5, que realitza 15 passos. El recorregut de les rutes (Heurístic 1) augmenta progressivament a mesura que la K augmenta, del qual se'n dedueix que li resulta més fàcil a l'algoritme trobar valors millors quan es tenen menys rutes.

En el cas del Simulated Annealing (Figura 6.7), s'observa altre cop un pic en el

Figura 6.7: Heurístic 1, temps d'execució (*elapsed*) i passos (*steps*) de l'algorisme Simulated Annealing en funció de K.



temps d'execució de l'algorisme per K = 2 (en particular 4344 u.t.). El temps d'execució baixa progressivament fins que veu el seu mínim en K = 5 (3094 u.t.) i torna a pujar fins quasi 5000 u.t. per K = 10. Els passos segueixen més o menys el mateix patró de creixement que el temps d'execució. El recorregut de les rutes, però, creix igual que el Hill Climbing progressivament a mesura que s'afegeixen més rutes al problema. Tanmateix, per qualsevol K, sempre millora el valor de l'algorisme de Hill Climbing per un valor entre 75 i 125.

Així doncs, s'extreu com a conclusió que la K augmenta la dificultat del problema en ambdós casos i empitjora els resultats en els dos algorismes, tot i que el temps d'execució augmenta de forma més evident en el Simulated Annealing.

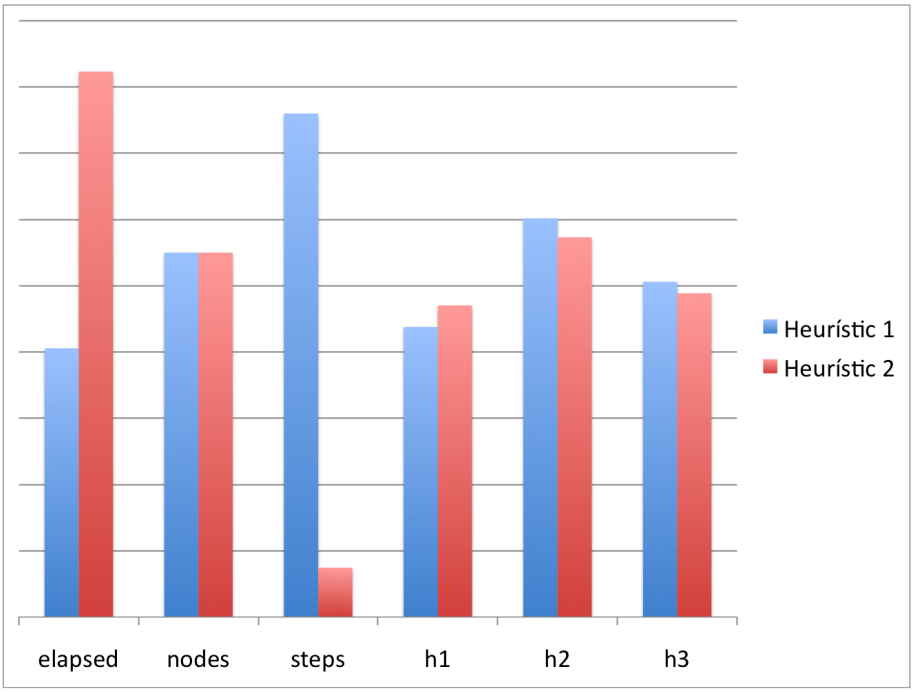
6.4 Comparació dels heurístics

Per tal de poder analitzar el comportament de l'algorisme amb els dos heurístics (sense combinar), comprovem quins resultats obtenim fent-ne servir cadascun d'ells.

En el següent experiment hem fet servir 10 taulers diferents, i per cada tauler hem fet 10 iteracions per cada heurístic, combinant les dades fins tenir-ne dues series.

La solució inicial escollida ha estat la de qualitat i l'algorisme Simulated Annealing amb els paràmetres comentats anteriorment.

Figura 6.8: Resultats amb h1 i h2



A la Figura 6.8 s'observa que com és d'esperar les execucions amb la cerca guiada per l'heurístic 1 obtenen un valor d'aquest més baix que les que han estat guiades per l'heurístic 2, tot i que la diferència no és molt gran. En aquest sentit, escollir un heurístic no perjudica el valor de l'altre.

Per altra banda, observem un efecte molt contundent en aquest gràfic, i és que l'heurístic 1 consegueix resultats amb molts més passos per arribar a la solució però la meitat del temps d'execució. Finalment, comentar que l'heurístic 2 és el que consegueix (per poca diferència) el millor valor per l'heurístic 3, que no s'ha emprat per guiar la cerca sino només per il·lustrar.

6.5 Paràmetre de ponderació dels heurístics

Com que volem una funció heurística que combini els dos paràmetres de qualitat esmentats (apartat 5.2.3), cal saber com conbinar-los.

Per tal de calcular una constant de ponderació $k_{h_{21}}$ hem realitzat un experiment que consisteix en observar el valor dels heurístics 1 (recorregut total de les rutes) i 2 (distància entre rutes homogènia) depenent de com els ponderéssim, i evidentment fent que l'algorisme intenti minimitzar la combinació d'ambdós.

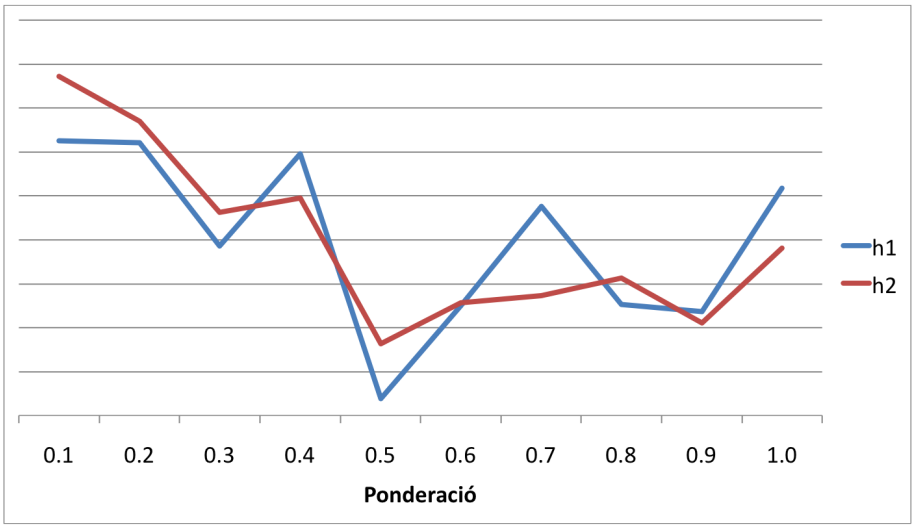
Hem utilitzat els paràmetres que han donat millor solucions fins al moment per dur a terme aquest experiment, o sigui:

- La solució inicial de qualitat
- L'algorisme de Simulated Annealing amb els paràmetres trobats a l'apartat 6.1
- Tots els valors de ponderació entre 0.1 i 1.0
- 10 possibles taulers i 10 iteracions per cada tauler

La fórmula per ponderar els heurístics h_1 i h_2 és la següent:

$$h_3 = (1 - k_{h_{21}}) \times h_1 + k_{h_{21}} \times h_2$$

Figura 6.9: Valor dels heurístics 1 i 2 amb diferents ponderacions



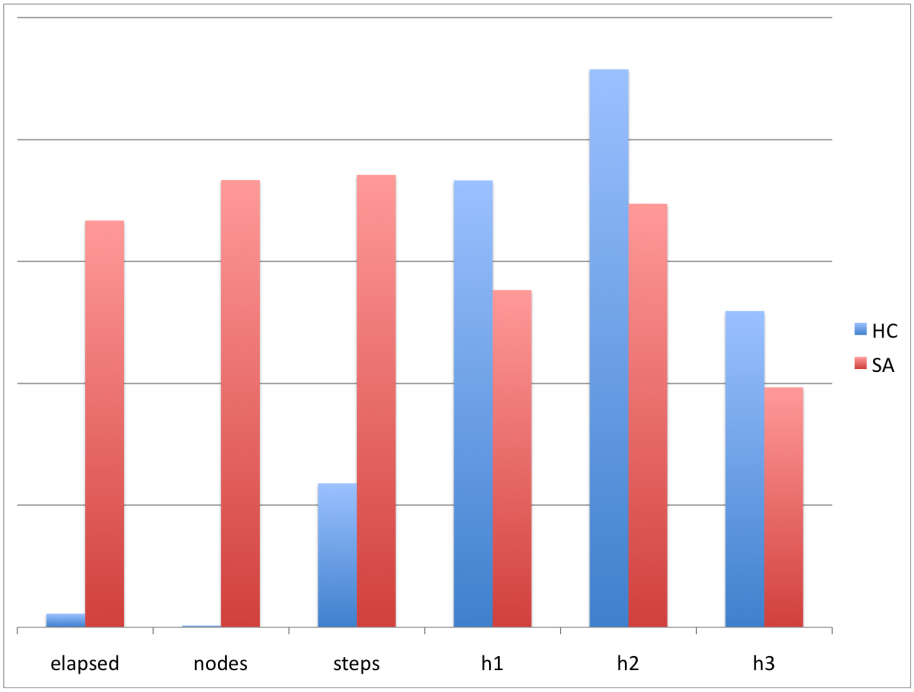
Es pot observar a la figura 6.9 que clarament els dos heurístics troben un punt mínim a la zona mitja del gràfic, on $k_{h_{21}} \in \{0.5..0.6\}$. Per això s'ha fixat per a la resta de proves que el valor ideal de $k_{h_{21}}$ sigui **0.5**.

6.6 Comparació d'algorismes

En aquest experiment hem provat en 10 taulers diferents el resultat de fer 10 cerques amb l'algorisme Hill Climbing i 10 amb el de *Simulated Annealing*. El resultat que s'observa és el promig de totes les execucions en tots els taulers. S'ha escollit usar els paràmetres òptims de Simulated Annealing (6.1), la solució de qualitat i l'heurístic combinat.

Es pot observar en la figura 6.10 que en els tres heurístics donen un millor resultat les cerques amb *Simulated Annealing*, i la diferència entres els heurístics és bastant equivalent, el que mostra gràficament que la ponderació entre h_1 i h_2 està ben calibrada.

Figura 6.10: Hill Climbing contra Simulated Annealing



A més de veure quin és l'algorisme que proporciona més qualitat a la solució, també observem que *Hill Climbing* és moltm és resolutiu, ja que assoleix un resultat prou similar amb una fracció molt petita de passos, i sobretot és més eficient perquè la diferència de nodes expandits i *elapsed time* entre els algorismes és aclaparadorament millor en l'algorisme de *Hill Climbing*.

6.7 Com varia el problema aplicant la restricció addicional?

Se'ns demana estudiar el comportament dels algorismes quan s'aplica la restricció addicional descrita a l'enunciat. Aquesta consisteix en limitar l'espai de cerca per tal que totes les rutes tinguin almenys $\frac{P}{2*K}$ parades.

L'experiment es basa en l'execució sobre un taulell generat aleatòriament per la solució inicial aleatòria i on s'apliquen els algorismes de Hill Climbing i Simulated Annealing amb i sense restriccions. Els algorismes s'executen 10 vegades sobre el taulell per extreure'n la mitjana dels resultats obtinguts. En la següent taula es pot veure la informació del temps d'execució, els nodes expandits, els passos i el recorregut total de les rutes (heurístic 1). L'algorisme optimitza l'heurístic 1.

Experiment	Elapsed	Nodes	Steps	H1
HC Rest	19	4	3	348,00
SA Rest	3626	5501	233	305,20
HC No Rest	1	2	1	298,00
SA No Rest	3209	5501	150	293,00

```
HC Rest: Hill Climbing amb o sense rest. addicional.
SA Rest: Simulated Annealing amb o sense rest. addicional.
```

Com es pot comprovar, lluny de facilitar el problema el complica. Tant l'algorisme de Hill Climbing com el de Simulated Annealing empitjoren els seus resultats. Mentre el temps d'execució, els passos i els nodes empitjoren en el Hill Climbing, en el cas de Simulated Annealing els nodes expandits es mantenen invariables, tot i que els passos i el temps d'execució també creixen quan s'aplica la restricció. Deduïm doncs que en aquest cas resulta més convenient permetre que es generin sempre tots els successors sense la restricció en l'espai de cerca.

Capítol 7

Conclusions

Després de tota la feina que s'ha fet de implementar la cerca, calibrar els diferents paràmetres i observar el comportament de tots els escenaris proposats, la primera conclusió és que el que a priori eren les millors condicions han estat efectivament les que han ofert millors resultats.

Pel que fa a l'algorisme, entre *Hill Climbing* i *Simulated Annealing* ha sortit guanyador l'algorisme de *Simulated Annealing*, tot i que hem fet entrar en joc un tercer algorisme, modificació del primer, que és *Hill Climbing* amb reinici aleatori. Aquest ha mostrat sorprenentment resultats molt similars al de *Simulated Annealing*, tot i que amb molt menys temps. Aquest és l'únic cas on el resultat ens ha sorprès ja que no ha complert amb el que esperavem (que fos simlement una lleu millora a la seva versió original).

Dins de la disputa entre algorismes, la recerca de paràmetres òptims per a l'algorisme de *Simulated Annealing* ha estat costosa, tot i que concloent. Aquest fet pot fer descartar aquest algorisme quan no es té la capacitat de fer un anàlisi acurat com el que s'ha dut a terme en aquest document.

En quant als heurístics, els dos es comporten de manera similar, cadascun optimitzant el seu objectiu, i hem trobat que el tercer heurístic que combina ponderadament els anteriors aconsegueix uns resultats òptims per als dos heurístics originals. Sens dubte és el que ens permet obtenir millors resultats globals.

Pel que fa a les solucions inicials també la lògica ha manat: la solució inicial de qualitat començava amb millors valors pels heurístics i acabava amb millors valors pels heurístics que la solució aleatòria. En aquest punt hi ha un cas especial que és la solució totalment aleatòria que hem necessitat per implementar el *Hill Climbing* amb reinici aleatori. Aquesta en combinació amb l'algorisme ha proporcionat molts bons resultats, però el mèrit és de l'algorisme.

Finalment, la restricció addicional proposada ha mostrat uns resultats manifestament pitjors a la proposta original, tant en temps d'execució com en la qualitat dels heurístics.

Capítol 8

Annex: Funcionament de la pràctica

El codi principal de la pràctica es troba en la classe *SquareBoard.java*. Per tal d'efectuar els experiments, hem creat un programa principal que s'executa amb la comanda *make run* i prova els diferents escenaris.

Codi 8.1: Comandes

```
make # Compila els .java
make run # Executa el programa principal
make clean # Esborra fitxers font
```

El menú es mostra de la següent manera:

```
--------------------- Selecció d'experiments -------------
Experiment 1: Comparació solucions inicials
Experiment 2: Estudi creixement de K
Experiment 3: Efectes de la restricció addicional
Experiment 6: Ponderació entre heurístics
Experiment 7: Parametre It. Totals de SimulatedAnnealing
Experiment 8: Parametre It. per pas de SimulatedAnnealing
Experiment 9: Parametre K de SimulatedAnnealing
Experiment 10: Parametre Lambda de SimulatedAnnealing
Experiment 11: Comparació heurístics h1 i h2
-----------------------------------------------------------
Introdueix el número d'experiment desitjat (de 1 a 11)
```

Tant sols cal introduir el número d'experiment i prémer Intro per executar el codi de proves. En alguns casos es genera un taulell sobre el qual es realitzen 10 simulacions i s'extreu la mitjana, en altres casos es genera més d'un taulell. Hem escollit cada escenari de proves segons hem cregut que reflexa més fidelment els valors mitjans.