Documentació de la pràctica de cerca local

Federico Rubinstein Pérez,
Roger González Herrera,
Luis Oriol Soler Cruz

FIB • UPC

Quadrimestre Tardor - Curs 2019/2020

Índex:

1. Descripció del problema

El problema del bicing, al ser un problema real, s'ha adaptat millor a la nostra visió i això ens ha ajudat a tenir quantitat d'idees bones i constructives. Cal dir, però, que el fet de delimitar-ho tant ha fet que, a vegades, hàgim trobat algunes solucions que no eren adients.

El problema consta en proposar una solució que serà la distribució, la càrrega i posteriorment el transport de bicis en un mapa d'una ciutat durant un període de temps d'una hora. Per encarar-ho s'ha d'entendre la informació que ens proporciona l'empresa, en aquest cas, Bicing.

Per una part tenim: el número de furgonetes, el número de bicicletes i el número d'estacions sobre el nostre mapa, que ve delimitat per una relació entre els factors.

Per a cada furgoneta sabem que surt d'un origen (i que només pot sortir una furgoneta de cada origen) amb una càrrega màxima de 30 bicicletes, que pot tenir un destí, dos o cap, en aquest últim cas, no està operativa.

Les bicicletes simplement son l'objecte a transportar així que no en donem gaire importància.

Les estacions és d'on obtenim més informació. Sabem:

-On estan localitzades

-Les bicis que no es fan servir en aquella hora, és a dir les disponibles.

-Una previsió de les bicicletes que hi haurà al cap d'una hora si no en portem cap nosaltres.

-Una demanda que ens indica les bicicletes que es necessiten.

A partir de totes aquestes dades, l'objectiu és que la nostra solució generi un gran benefici. Aquest benefici ve estipulat per:

-Perds un euro si t'allunyes de la demanda a l'hora d'agafar una bicicleta.

-Guanyes un euro si t'apropes a la demanda a l'hora de deixar una bicicleta.

-Perds (nb+9)/10 euros per cada km recorregut on nb es el número de bicicletes.

Observant totes aquestes dades del problema ens adonem que es pot resoldre mitjançant una cerca local. Això és degut a que la mida de l'espai de solucions és massa gran com per tratar d'explorar-lo i d'obtenir la solució òptima, que és massa costosa. També, degut a que volem minimitzar o maximitzar valors de la funció heurística.

Un cop analitzat, el problema d'enteniment de l'enunciat ja queda resolt. El segon problema es el de com gestionar l'espai de memòria per a que el programa no es quedi sense

memòria, sabent que els algorismes d'Inteligencia Artificial que hem de fer servir (Hill Climbing i Simulated Annealing) generen fills i guardar moltes dades ens pot fer sobreeixir l'espai que tenim disponible.

Per tant, decidim estructurar la nostra capa de dades generalment amb 5 vectors:

-Vector que ens diu on està cada furgoneta.

-Vector que ens diu quantes bicicletes hi ha a l'estació en aquell moment (fet servir per si dues furgonetes deixen bicicletes a la mateixa estació, el càlcul de beneficis no sigui erroni)

-Vector de primers destins de cada furgoneta.

-Vector de segons destins de cada furgoneta.

-Vector de número de bicicletes a deixar a la primera estació.

-Vector de número de bicicletes a deixar a la segona estació.

A més, també utilitzem un parell d'enters on comptabilitzarem els beneficis per un cantó i les penalitzacions per errada al treure una bicicleta de més per un altre.
A més a més, utilitzem un double per als costos de transport, aprofitant així tenir-ho tot separat per, a l'heurístic, poder-ho separar o ponderar per obtenir una millor solució.

Pensem que hem de trobar els operadors justos, que permetin arribar a totes les possibles solucions. Que no n'hem de tenir ni molts ni pocs, simplement els justos. Això ho encarem de la manera de que tots els factors que entren en joc puguin tenir tots els estats posibles. És per això que obtenim operadors per a moure les furgonetes de lloc, intercanviar les estacions amb les que interactuen (ja siguin l'origen o un dels destins) i modificar el número de bicis que transporten.

Busquem doncs, amb els beneficis, maneres diferents de valorar quan una solució és millor que una altra, i doncs sabem que l'objectiu és moure les menors bicicletes posibles (ja que tenen cost de transport) i moure-les des de el lloc on en sobren fins al lloc on en falten, visitant així estacions on hi hagi molt espai entre la previsió i la demanda i sortint des de punts on hi hagi menys demanda i més bicis disponibles, i que si els dos factors juntament son més favorables voldrà dir que podem agafar més bicis.

Ens adonem que, hem de triar bé entre què té més valor, si arriscar-se a moure una bici que pot generar-te beneficis o no fer-ho. I treiem la conclusió que ho hem de provar i que el mateix algorisme ja ens donarà la millor solució. Per això no ens tanquem les portes a arribar a una solució dolenta si després aquesta en pot generar de més bones, fet que anirà bé ja que el nostre heurístic ho determinarà.

1.1. Representació del problema

La representació de l'estat del problema que hem escollit està configurada pels següents atributs:

- **estaciones:** objecte de la classe *Estaciones* que conté totes les estacions del problema (objectes de la classe *Estacion*) que s'assignaràn com estació d'origen i/o estació destí de les furgonetes que disposem.

- **asignaciones:** array d'enters que relaciona cada furgoneta amb la seva estació d'origen. Sigui '*i*' el id de la furgoneta, llavors '*asignaciones[i]*' pren com a valor -1 si la furgoneta no té cap estació assignada, és a dir, no s'utilitza, altrament pren com a valor el id de l'estació en el array list *estaciones.*

- **realBicisNext:** array d'enters que relaciona cada estació amb les bicis que hi haurà a l´hora següent. Sigui '*i*' el id de l'estació en el array list *estaciones*, llavors realBicisNext*[i]*' pren com a valor el nombre de bicis.

- **primerosDestinos:** array d'enters que relaciona cada furgoneta amb la seva primera estació destí. Sigui '*i*' el id de la furgoneta, llavors '*primerosDestinos[i]*' pren com a valor -1 si la furgoneta no té cap estació assignada, altrament pren com a valor el id de l'estació en l'array list *estaciones*.

- **segundosDestinos:** array d'enters que relaciona cada furgoneta amb la seva segona estació destí. Sigui '*i*' el id de la furgoneta, llavors '*segundosDestinos[i]*' pren com a valor -1 si la furgoneta no té cap estació assignada, altrament pren com a valor el id de l'estació en l'array list *estaciones*.

- **primerasBicisDejadas:** array d'enters que relaciona cada furgoneta amb el nombre de bicis que deixarà a la primera estació destí. Sigui '*i*' el id de la furgoneta, llavors '*primerasBicisDejadas[i]*' pren com a valor el nombre de bicis.

- **segundasBicisDejadas:** array d'enters que relaciona cada furgoneta amb el nombre de bicis que deixarà a la segona estació destí. Sigui '*i*' el id de la furgoneta, llavors '*segundasBicisDejadas[i]*' pren com a valor el nombre de bicis.

- **beneficioPorAcierto:** enter que conté el nombre d'euros que ens pagaràn transportant les bicis que facin que el nombre de bicis de l'estació s'apropi a la seva demanda.

- **penalizacionPorFallo:** enter que conté el nombre d'euros que ens cobraràn transportant bicis que facin que el nombre de bicis de l'estació s'allunyi de la seva demanda.

- **costeTransporte:** real que conté el cost de transport de bicis segons el nombre de bicis transportades i la distància recorreguda per cada furgoneta.

Hem escollit aquesta representació de l'estat del problema perquè ens permet canviar l'estació d'origen, les estacions destí i el nombre de bicis transportades en un cost temporal molt reduït, ja que simplement hem de canviar el valor de la posició corresponent als arrays d'enters *asignaciones*, *xDestinos* i *xBicisDejadas* respectivament.

La decisió de tenir *asignacions*, *xDestinos* i *xBicisDejadas* indexades pel id de la furgoneta és deguda a que ens permet accedir a tota l'informació de la furgoneta paral·lelament i de forma ordenada.

També, vam decidir incloure *realBicisNext*, tot i tenir un cost una mica més elevat, ja que ens facilita saber en cada moment el nombre de bicis que n'hi ha a cada estació, degut als trasllats de bicis de les furgonetes, per tal d'obtenir els beneficis de forma acumulativa.

Pels atributs restants *beneficioPorAcierto*, *penalizacionPorFallo* i *costeTransporte*, els vam decidir ja que ens permeten calcular i obtenir les dades que proporcionen a mesura que es desenvolupa la solució inicial i les transformacions aplicades mitjantçant els operadors, aconseguint així reduir el cost temporal. Hem decidit dividir els beneficis obtinguts (beneficioPorAcierto - penalizacionPorFallo) per tal d'obtenir uns millors heurístics.

1.2. Operadors

En quan als operadors que hem fet servir, vam estar pensant sobre quins operadors aniríen millor per a no delimitar l'espai de solucions. Per fer-ho, vam intentar resumir els elements que interveníen al problema: furgonetes, bicicletes i estacions, i com actuàven aquests dins del problema, les accions que podíen fer: agafar bicicletes d'una estació, deixar bicicletes a una estació, sortir d'un origen i arribar a un destí, tenir un destí o tenir-ne dos.

Per relacionar i delimitar amb el mínim d'operadors que fes que no se'ns reduís l'espai de cerca de la solució vam decidir crear 4 operadors: Moure furgonetes, canviar estació destí, intercanviar furgonetes i carregar furgoneta.

Cada operador, a més de proporcionar-nos les operacions que necessitem, també fa un càlcul del benefici i del cost de transport, ja que és més eficient que cada vegada que es faci un canvi es vagi actualitzant que no calcular-ho tot cada vegada. Per tant dins del codi de

cada operador hi ha funcions encarregades d'actualitzar els costos de transport, beneficis que obté i penalitzacions que se li apliquen en cas que es doni l'ocasió.

La funció "*BicingSuccessorFunction*" 1 i 2, depenent si utilitzem Hill Climbing o Simulated Annealing, crida els següents operadors:

- **moverFurgoneta:** el que fa és moure una furgoneta a una estació. Per tant la furgoneta canvia només el seu punt d'origen, conservant el destí o destins que ja tenia assignats i conservant la seva càrrega i per tant també el número de bicis que transportarà d'un lloc a l'altre. Per fer factible això comprovem que la nova estació no estigui ocupada, que la nova estació tingui prous bicicletes disponibles per agafar-les i que l'estació on la portem no sigui la mateixa des d'on ja està (per fer-ho més eficient).

  El seu factor de ramificació és O(|F|*|E|), on |F| és el nombre de furgonetes i |E| el nombre d'estacions, degut a que mourà cada furgoneta a cada estació.

- **cambiarEstacionDestino:** el que fa és modificar el recorregut d'una furgoneta, és a dir, si la furgoneta passa per l'estació A i després la B, generem solucions perquè es pugui intercanviar tant el destí 1 o 2 per un nou destí C, quedant el recorregut C i B o A i C. Per tant l'únic que canviem es el vector de destins, on substituïm l'antic pel nou, tota l'altra estructura es queda igual. També podem posar un destí a -1, per anular-lo, en aquest cas només ho podem fer al segon destí o al primer destí si no té segon. Per implementar la funció necessitem saber si el destí nou serà el seu primer o el segon, en cas de que en tingui. Per tant l'operador comprova que la furgoneta estigui posada a un origen, és a dir sigui activa, que no posem a la ruta un destí que ja té o que sigui la mateixa estació des de la que surt, comprovem també si en cas que el destí sigui el segon, que en tingui.

  El seu factor de ramificació és O(|F|*|E|), on |F| és el nombre de furgonetes i |E| el nombre d'estacions, degut a que per cada furgoneta canviarà un dels seus dos possibles destins per cada estació.

- **intercambiarFurgonetas:** és un operador que aprofita el moure furgonetas, ja que el que fa és moure'n dues. Per tant aquest fa la mateixa funció que moure furgoneta però en el doble sentit. Això ens permet intercanviar una furgoneta que no està a cap estació, és a dir que està "fora de servei", i ens permet posar-ne de noves dins del mapa o treure'n, o sigui assignar-li una estació = -1. Per tant només canviem en el vector d'assignacions la posició on es troben les noves furgonetes. Les comprovacions que hem de fer és mirar que almenys una estigui en servei i que

comprovi si les dues furgonetes poden agafar tantes bicicletes com necesiten de la estació nova que li assignem.

El seu factor de ramificació és O(|F|*|F|), on |F| és el nombre de furgonetes, degut a que intercanviarà cada furgoneta amb una altra.

- **cargarFurgoneta:** és cridat per obtenir el valor més adequat de bicis que ha de carregar una furgoneta en una estació. Per això el que fa es assignar bicicletes al primer destí i al segon. Per fer això, l'operador comprova diversos valors. En primer lloc que pugui carregar el número de bicis que se li demanen de l'estació origen en la que es troba, que no s'excedeixi ja que cada furgoneta només pot carregar 30 bicicletes, també mira que pugui deixar bicicletes al primer o segon destí, per tant comprova si té primer o segon destí, comprova també que la furgoneta estigui a una estació origen per tal de poder fer l'operació.

  El seu factor de ramificació és O(|F|*31*31), on |F| és el nombre de furgonetes, degut a que per cada furgoneta li asignarà un nombre de bicis pel primer destí i un altre pel segon destí.

1.3. Funcions heurístiques

Les funcions heurístiques del problema que hem escollit son les següents:

- **HeuristicFunction1:** té en compte només el primer criteri de maximitzar els beneficis obtinguts, de la forma:

$$penalizacionPorFallo - 10 * beneficioPorAcierto.$$

- **HeuristicFunction2:** té en compte el primer criteri i el segon criteri de minimitzar el cost de transport de les bicis, de la forma:

$$costeTransporte + penalizacionPorFallo - 10 * beneficioPorAcierto$$

Pel primer heurístic, vam decidir utilitzar aquesta resta amb ponderació per tal de diferenciar quina és millor entre dues solucions amb el mateix nombre de beneficis. Per això, donem més importància als beneficis per acertar que les penalitzacions per fallar (ponderem *beneficioPorAcierto*), i així recórrer uns nodes més si trobem un pla on tots el successors tenen el mateix nombre de beneficis.

Pel segon heurístic, vam decidir aquesta altra resta amb ponderació pel mateix motiu que el primer heurístic: volem diferenciar entre dues solucions amb el mateix valor resultant. Per això, també en aquest heurístic donem més importància als beneficis per acertar obtinguts.

En ambdós casos, ponderem els beneficis per acertar ja que vam provar amb ponderacions de les penalitzacions per fallar i pel cost de transport però ens donaven pitjors resultats. Això és així perquè acceptant un nombre de pèrdues podem obtenir un marge millor de benefici per acertar, que si ens centrem en reduir les pèrdues a costa d'un marge de beneficis per acertar reduïts.

1.4. Solucions inicials

Tal com ens demanava l'enunciat de la pràctica, hem implementat dues solucions inicials, la primera sencilla i la segona més acotada a trobar una bona solució. A continuació una explicació per cada una:

Primer generador de solucions inicials

Per fer la primera solució ens vam plantejar quin seria l'escenari des d'on els algorismes d'intel·ligència artificial podrien recorrer més àmpliament l'espai de cerca. Per això vam decidir fer una solució aleatòria perquè els algorismes trobèssin moltes solucions i així poder escollir la millor amb els heuristics.

Aquesta solució aleatòria es basa en posar cada furgoneta a una estació d'origen aleatòria on també pot ser -1, és a dir, a cap. En aquest cas, si no té estació d'origen, és a dir, fora de servei, els destins també seràn -1 i la seva càrrega 0.

A partir d'aquí les furgonetes que tinguin una estació d'origen també en podrien tenir una o dues de destí, també generades aleatòriament, amb l'única condició que per a tenir segon destí n'ha de tenir primer. El mateix passa amb les càrregues, definides aleatòriament, sempre respectant que no passin ni de la capacitat de la furgoneta ni s'excedeixin en agafar més bicicletes de les que té l'estació origen.

Amb tot això, obtenim una complexitat temporal O(|E|+|F|), ja que hem de fer les assignacions oportunes (cost O(1)) per cada furgoneta i hem de recórrer les estacions per tal d'obtenir informació respecte al nombre de bicis a l'hora següent.

Respecte a la bondat d'aquesta solució inicial random, partint d'una solució inicial relativament dolenta, arribem a una solució bona. Té un bon increment de millora.

Segon generador de solucions inicials

La segona solució, en canvi, és més específica. Per fer-la vam pensar quin seria el procés que fariem nosaltres si ens demanessin una solució bona pel problema especificat, en resum: portar bicis d'on més en sobrin cap a on més en faltin. Doncs això és el que fa la segona solució.

Per passos, crea un vector d'estacions on ordena les estacions amb més bicis disponibles, és a dir:

$$BicisDisponiblesOrigen = BicisNext - BicisDemandades$$

sempre que sigui igual o menor a BicisNoUtilizadas , sino:

$$BicisDisponiblesOrigen = BicisNoUtilizadas$$

Una vegada ordenat el procés és ben senzill, assignen les 'n' furgonetes a les primeres 'n' estacions del vector.

El procés amb els destins és molt semblant, però no es fixa amb les bicis no utilitzades, és a dir:

$$BicisDisponiblesDesti = BicisDemandades - BicisNext$$

D'aquesta manera aconseguim un vector amb les estacions que necessiten més bicis ordenat, i un cop així, per cada n furgoneta anem assignant el destí i descomptant del vector fins que aquest va tenint les posicions a 0. El guany que obtenim és que pot haver en algun moment una furgoneta que vagi al destí que en teoría es el millor, per què té més bicicletes disponibles, però que no ho sigui. Això ho hem buscat expressament perquè tingui un marge de millora i passa quan assignem a un destí una càrrega que no arriba a la seva demanda (per exemple: la demanda es de 30 i deixem aquella estació amb 28 bicicletes. La pròxima furgoneta pensarà que aquest destí és el millor quan el segon segurament tindrà un espai de més de 2 bicis per deixar).

D'aquesta manera doncs, carreguem les furgonetes amb la disponibilitat que té l'estació assignada, sense que generin beneficis negatius. Per tant la nostra solució no tindrà dèficit negatiu d'entrada en el primer node.

Comentar que la solució creix poc ja que és bastant propera a un màxim local, i per tant, s'estanca fàcilment.

Respecte al cost temporal és O(|E|log|E|) ja que necessitem ordenar les estacions segons prosperitat.

A més, sobre la bondat d'aquesta solució inicial millorada, partim d'una bona solució inicial però l'increment amb la solució final és qüasi nula ja que es troba molt a prop d'un màxim local.

2. Experiments

2.1. Determinar el conjunt d'operadors

Determinar quin conjunt d'operadors dóna millors resultats per a una funció heurística que optimitzi el primer criteri (transport gratuït) amb un escenari en el que el nombre d'estacions és 25, el nombre total de bicicletes és 1250, el nombre de furgonetes és 5 i la demanda és equilibrada. Haureu d'utilitzar l'algorisme de Hill Climbing. Escolliu una de les estratègies d'inicialització d'entre les que proposeu. A partir d'aquests resultats haureu de fixar els operadors per la resta d'experiments.

Observació	Pot ser que existeixi una combinació d'operadors que sigui millor que una altra.
Plantejament	Executem per totes les combinacions per averiguar quina és la millor.
Hipòtesis	Suposem que la millor solució serà utilitzar tots els operadors o tots menys *moverFurgoneta* o *intercanviarFurgoneta.*
Mètode	- Escollim 10 llavors aleatòries, un per cada rèplica. - Executem un experiment per cada llavor i conjunt d'operadors. - Experimentem amb problemes de 25 estacions, 1250 bicis, 5 furgonetes i demanda equilibrada. - Utilitzem el primer generador de solucions inicials, el primer heurístic i l'algorisme Hill Climbing. - Mesurem diferents paràmetres per fer la comparació.

Experiment	Beneficis			Temps d'Execució (ms)		
	Tots els Operadors	Sense *mover Furgoneta*	Sense *intercambiar Furgonetas*	Tots els Operadors	Sense *mover Furgoneta*	Sense *intercambiar Furgonetas*
1	72	48	58	254	84	32
2	46	7	55	59	47	21
3	44	3	61	102	61	16
4	60	20	29	99	68	31
5	44	13	41	45	20	12
6	53	30	72	47	40	42
7	49	12	42	73	53	13
8	36	20	43	53	46	16
9	59	9	80	65	13	49
10	59	17	41	62	49	12
Mitjana (desv. típica)	52.2 (9.958)	17.9 (12.413)	52.2 (15.039)	85.9 (59.038)	48.1 (19.922)	24.4 (12.674)

Figura 1: Valor dels beneficis i del temps d'execució

El conjunt d'operadors que ens dona millors resultats es la combinació dels operadors *cargarFurgoneta*, *moverFurgoneta* i *cambiarEstacionDestino* ja que es pot veure que la mitjana de Beneficis es la mateixa, tot i que la desviació típica es una mica més gran i per tant vol dir que els valors s'allunyen de la seva mitjana tant per sota com per sobre i no tendeixen a seguir una línea. A més, escollim aquest conjunt ja que triga molt menys en el procés del programa i és més estable que el conjunt de tots els operadors o de tots sense *moverFurgoneta*.

2.2. Determinar l'estratègia de solució inicial

Determinar quina estrategia de generació de la solució inicial dóna millors resultats per a la funció heurística utilitzada en l'apartat anterior, amb el escenari de l'apartat anterior i utilitzant l'algorisme de Hill Climbing. A partir d'aquests resultats haureu de fixar també l'estratègia de generació de la solució inicial per a la resta d'experiments.

Observació	Potser un dels generadors de solució inicial és millor que l'altre.
Plantejament	Generem solucions amb ambdós generadors i comparem resultats.
Hipòtesis	Els dos generadors són iguals (H0) o un d'ells dóna millors resultats.
Mètode	- Escollirem 10 llavors, una per cada rèplica. - Executarem 10 experiments per cada llavor i generador de solucions inicials i tindrem en compte la mitjana, per així tenir uns resultats millors i més acotats. - Experimentarem amb problemes de 25 estacions, 1250 bicis, 5 furgonetes i demanda equilibrada. - Utilitzarem el conjunt d'operacions escollit de l'experiment anterior, el primer heurístic i l'algorisme de Hill Climbing. - Mesurarem diferents paràmetres per realitzar la comparació.

Experiment	Beneficis Inicials		Beneficis Finals		Temps d'Execució (ms)	
	Generador 1	Generador 2	Generador 1	Generador 2	Generador 1	Generador 2
1	-4	71	43	72	86	28
2	0	84	53	84	30	10
3	0	79	45	79	21	12
4	-5	69	56	69	27	9
5	-4	75	55	76	24	14
6	-5	71	55	74	25	12
7	-2	76	42	76	22	6
8	1	80	52	80	24	6
9	-4	82	49	82	22	7
10	-8	81	47	81	17	6
Mitjana (desv. típica)	-3.1 (2.663)	76.8 (4.936)	49.7 (4.961)	77.3 (4.496)	29.8 (19.025)	11.0 (6.293)

Figura 2: Valor dels beneficis inicials, dels beneficis finals i del temps d'execució

Encara que es veu clarament que el generador de la solució inicial 2 obté uns beneficis finals millors que el 1, i que a més ho fa amb menys temps, decidim triar la solució que ens mostra el generador de la solució inicial 1. Això es degut a que, com es veu a la taula, el generador 2 fa una solució inicial molt semblant i de vegades idèntica degut a que hem generat un algorisme de solució inicial molt bo; fet que impedeix al Hill Climbing avançar perquè topa amb un màxim local. Tot això fa que ens decantem cap a l'elecció de la solució inicial 1, ja que sinó els experiments que venen a continuació no tindrien gaire sentit degut a la falta de dades per trobar resultats i treure'n una conclusió bona i amb fonaments.

2.3. Determinar els paràmetres del Simulated Annealing

Determinar els paràmetres que donan millor resultat per el Simulated Annealing amb el mateix escenari, utilitzant la mateixa funció heurística i els operadors i l'estratègia de la generació de la solució inicial escollits en els experiments anteriors.

Observació	Hi ha d'haver una combinació dels paràmetres de la funció que calcula el Simulated Annealing que doni millors resultats.
Plantejament	Generem solucions amb diferents paràmetres per acotar els seus valors.
Hipòtesis	Podem trobar una combinació amb Simulated Annealing que ens mostri els mateixos resultats que Hill Climbing (H0) o millors.
Mètode	- Utilitzem heuristic 1 i solució inicial 1 - Mètode Annealing amb operadors *cargarFurgoneta*, *moverFurgoneta* i *cambiarEstacionDestino* - Experiments de 1000 iteraciones per cada 10 semillas - Número màxim d'iteracions pel Simulated Annealing 1000 - Número d'iteracions per cada pas 100 - Provem diferents valors per K: 1 5 10 25 125 - Provem diferents valors per Lambda: 1 0.1 0.01 0.001

	K = 1	K = 5	K = 10	K = 25	K = 125
lambda = 1.0	89.352	85.719	80.555	85.086	83.274
lambda = 0.1	98.404	100.386	96.354	96.781	101.899
lambda = 0.01	107.333	103.973	104.995	102.256	113.442
lambda = 0.001	113.22	98.309	100.067	104.304	137.746
lambda = 0.0001	105.943	99.552	104.95	127.476	128.635

Figura 3: Valors dels paràmetres Lambda i K de l'algorisme Simulated Annealing

Hem trobat més adient fer una taula ja que a partir de gràfics no s'obtenia el resultat esperat i visualment era més complicat de justificar-ho. A partir de la taula podem observar que el benefici més òptim que s'obté a partir de l'algorisme Simulated Annealing es quan la k = 125, fent les iteracions corresponents que diem a l'apartat del mètode hem vist que els millors resultats son amb lambda = 0,001 seguits de lambda 0,0001 i després lambda = 0.01. Per tant la millor solució és clarament a lambda = 0,001.

2.4. Estudi del temps d'execució

Donat l'escenari dels apartats anteriors, estudieu com evoluciona el temps d'execució per trobar la solució en funció del nombre d'estacions, furgonetes i bicicletes assumint una proporció 1 a 50 entre estacions i bicicletes i 1 a 5 entre furgonetes i estacions. Per això, comenceu amb 25 estacions i aneu augmentant-les de 25 en 25 fins que veieu la tendencia. Utilitzeu l'algorisme de Hill Climbing i la mateixa heurística.

Observació	El temps d'execució s'incrementarà al augmentar el nombre d'estacions, de furgonetes i de bicis.
Plantejament	Generem diferents execucions augmentant de 25 en 25 el nombre d'estacions, respectant les proporcions.
Hipòtesis	El temps d'execució augmentarà segons augmenti el nombre d'estacions.
Mètode	- Utilitzem heuristic 1 i solució inicial 1 amb operadors *cargarFurgoneta*, *moverFurgoneta* i *cambiarEstacionDestino* - Fem servir Hill Climbing - Generem solució amb 25 estacions, 1250 bicicletes i 5 furgonetes - Augmentem la solució de 25 en 25 estacions i amb la proporció adequada per bicicletes i furgons - Compararem la relació benefici amb el temps - Demanda hora normal

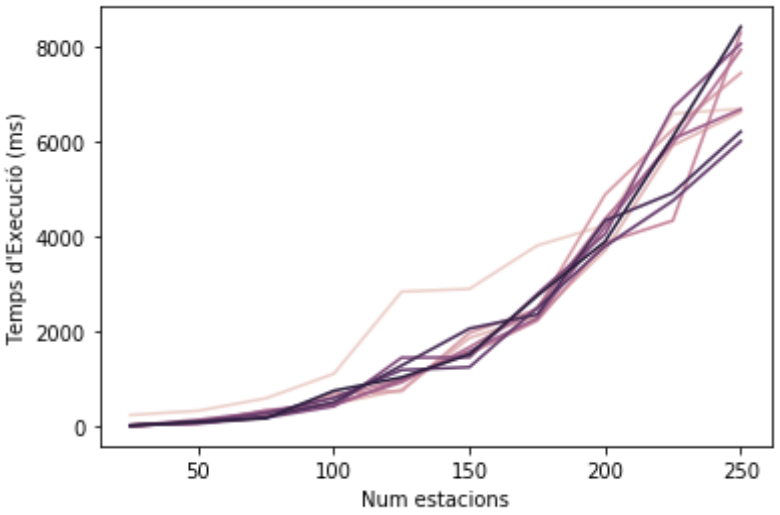

Figura 4: Variació del temps d'execució

Veiem que com més estacions té el problema més costós es la solució i per tant més temps triga. Com veiem a la grafica que hem generat, el temps es de cost cost exponencial, això és degut a que com més estacions, hi ha més bicicletes i més furgonetes per tant les possibilitats que hi ha entre operadors creixen exponencialment ja que es combinatòria i per tant es multipliquen. Per tant com més gran sigui el mapa de les estacions mes trigarà l'algorisme a trobar una solució final.

2.5. Estudi de les funcions heurístiques

Donat l'escenari del primer apartat, estimeu la diferència entre els beneficis obtinguts, la distància total recorreguda i el temps d'execució per trobar la solució amb el Hill Climbing i el Simulated Annealing per les dues heurístiques que heu implementat.

Observació	Potser un dels heurístics és millor que l'altre.
Plantejament	Generem solucions amb ambdós heurístics i comparem resultats.
Hipòtesis	El millor algoritme dependrà de la forma d'avaluació que tinguem
Mètode	- Escollirem 10 llavors, una per cada rèplica. - Executarem 10 experiments per cada llavor. - Experimentarem amb problemes de 25 estacions, 1250 bicis, 5 furgonetes i demanda equilibrada. - Mesurarem diferents paràmetres per realitzar la comparació.

Hill Climbing

Experiment	Heurístic 1			Heurístic 2		
	Beneficis	**Distància recorreguda**	**Temps d'execució (ms)**	**Beneficis**	**Distància recorreguda**	**Temps d'execució (ms)**
1	54	47900.0	117	50	18040.0	145
2	53	50520.0	65	57	18660.0	80
3	42	45690.0	32	45	13240.0	55
4	50	46300.0	27	56	15770.0	42
5	46	43210.0	20	54	13730.0	31
6	57	43760.0	25	64	15150.0	37
7	54	44360.0	21	59	14540.0	34
8	45	41620.0	20	54	14520.0	28
9	42	49030.0	19	53	15040.0	32
10	48	50980.0	22	52	22910.0	31
Mitjana (desv. típica)	49.1 (5.049)	46337.0 (3028.99)	36.8 (29.725)	54.4 (4.883)	16160.0 (2784.22)	51.5 (34.564)

Figura 5: Valor dels beneficis, de la distància recorreguda i del temps d'execució dels dos heurístics

Simulated Annealing

Experiment	Heurístic 1			Heurístic 2		
	Beneficis	**Distància recorreguda**	**Temps d'execució (ms)**	**Beneficis**	**Distància recorreguda**	**Temps d'execució (ms)**
1	88	60220.0	13	100	34960.0	7
2	99	74050.0	5	102	37680.0	5
3	102	63810.0	4	100	32550.0	4
4	102	66650.0	2	109	34390.0	2
5	110	66550.0	2	102	36280.0	2
6	100	68450.0	2	97	37790.0	2
7	105	68840.0	2	103	31080.0	2
8	92	72050.0	2	103	37980.0	2
9	104	69440.0	2	106	34060.0	2
10	87	75920.0	2	88	37250.0	2
Mitjana (desv. típica)	98.9 (7.176)	68598.0 (4440.119)	3.6 (3.292)	101.0 (5.348)	35402.0 (2270.638)	3.0 (1.673)

Figura 6: Valor dels beneficis, de la distància recorreguda i del temps d'execució dels dos heurístics

En el Heuristic 1, ens decantem clarament cap al Simmulated Annealing ja que en aquest heuristic no tenim en compte els costos de transport i els beneficis en si son el doble que en el Hill Climbing com podem observar a les dues taules. Respecte al temps, el mateix, el Simmulated Annealing triga una dècima part que el Hill Climbing.

En el cas del Heuristic 2 passa el mateix, encara que recorre el doble de distància, el Simmulated Annealing genera uns beneficis del doble que el Hill Climbing i en el temps d'execució la millora és encara més bona que amb l'Heuristic 1.

Per tant ens decantem per escollir que el Simmulated Annealing és millor que el Hill Climbing tal i com podem observar en l'experiment. Fet que confirma que el Simmulated Annealing es una optimització del Hill Climbing i per tant tant a la teoria com a la pràctica ha quedat demostrat.

2.6. Estudi del tipus de demanda

Donat l'escenari del primer apartat, genereu problemes en els que la demanda correspongui a l'hora punta i estudieu si hi ha alguna diferència en el temps d'execució per resoldre el problema. Utilitzeu l'algorisme que millor resultats os hagi donat.

Observació	Hi haurà estacions amb clarament més demanda i estacions amb menys, això afavoreix a l'hora d'obtenir beneficis
Plantejament	Generem diferents execucions augmentant de 25 en 25 el nombre d'estacions, respectant les proporcions.
Hipòtesis	L'hora punta serà més costosa en el temps.
Mètode	- Utilitzem escenari experiment 1 amb operadors *cargarFurgoneta*, *moverFurgoneta* i *cambiarEstacionDestino* - Fem servir Hill Climbing - Augmentem la solució de 25 en 25 estacions i amb la proporció adequada per bicicletes i furgons - Compararem la relació benefici amb el temps

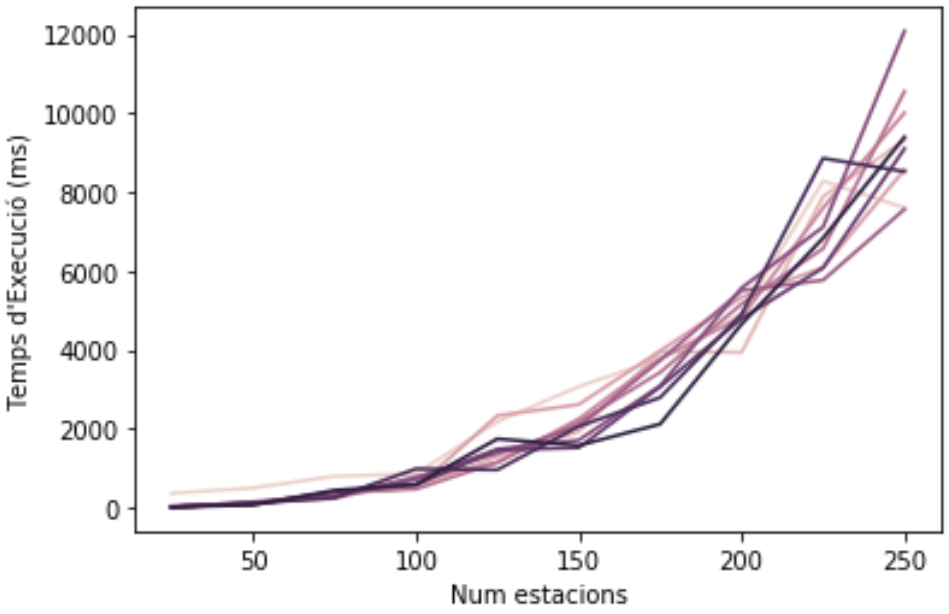

Figura 7: Variació del temps d'execució en hora punta

L'objectiu era comparar els resultats del cas de demanda normal amb els de demanda en hora punta fixant-se en el temps. Com ja figurava a la hipotesi el cost de l'algorisme en funció del temps és més elevat en l'hora punta, tal i com mostra el gràfic de dalt. Segueix sent una funció exponencial ja que com més augmentem les estacions més creix l'espai de cerca i en efecte mes combinacions de fills hi ha a cada solució. Per tant podem concloure que si estem en una hora de demanda en hora punta l'algorisme triga més que en demanda normal.

2.7. Estudi del nombre mínim de furgonetes

Donat l'escenari del primer apartat, estimeu quina és aproximadament el nombre de furgonetes que són necessàries per obtenir la millor solució. Per això comenceu amb 5 furgonetes i aneu augmentant-les de 5 en 5 fins que no hi hagi una millora significativa. Feu la prova també amb la demanda en hora punta. És diferent el nombre de furgonetes per els dos escenaris?

Observació	Hi haurà un nombre de furgonetes per la demanda on a partir d'allà la millora dels beneficis es reduirà.
Plantejament	Generem experiments començant amb 5 furgonetes i anem augmentant de 5 en 5.
Hipòtesis	La solució la trobarem aprop del número màxim de furgonetes, 25.
Mètode	- Experiments on obtenim la mitjana a partir de 100 experiments per semilla - Utilitzem heuristic 1 i solució inicial 1 amb operadors *cargarFurgoneta*, *moverFurgoneta* i *cambiarEstacionDestino* - Hill Climbing - 25 estacions, 1250 bicicletes i 5 furgonetes inicials augmentanles de 5 en 5 fins arribar a 25

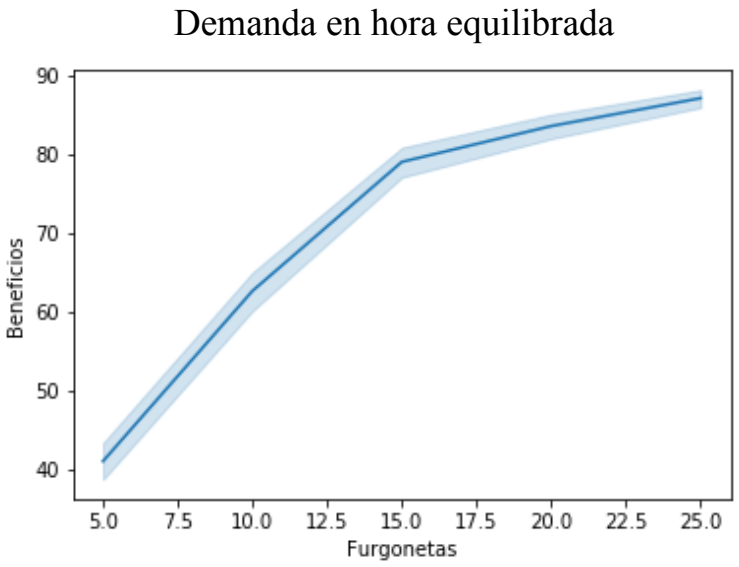

Figura 8: Variació dels beneficis segons les furgonetes utilitzades

Veiem que, en la demanda equilibrada, quan hi ha 15 furgonetes al mapa es produeix una baixada de beneficis, per tant la solució mes optima contindria 15 furgonetes per un espai de 25 estacions i 1250 bicicletes, menys furgonetes donen menys benefici i més furgonetes tindran un valor invalid que no ens comporta una millora valorable.

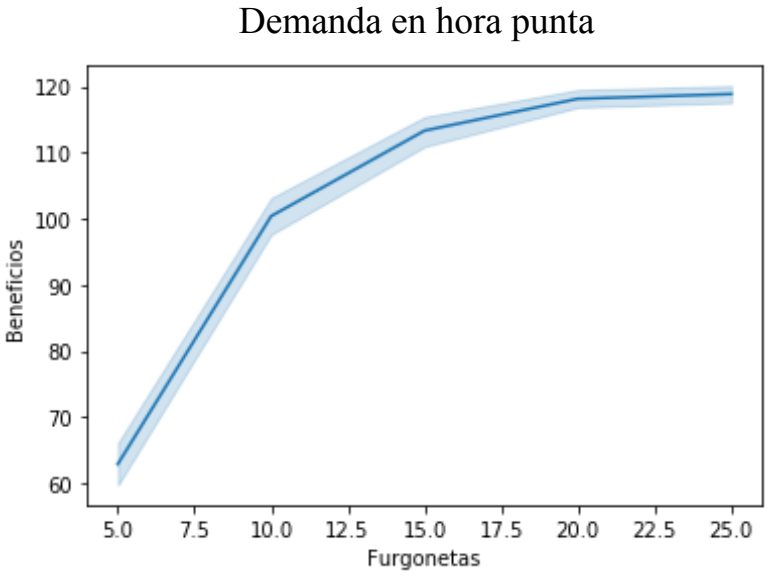

Figura 9: Variació dels beneficis segons les furgonetes utilitzades

Veiem doncs, que amb la demanda hora punta, el punt trobat amb la demanda equilibrada és més difícil de decidir, pero creiem que seria amb 20 furgonetes, això seria degut a que al tenir més estacions amb més demanda i d'altres sense tanta demanda - fet que provoca la demanda en hora punta - es necessiten més furgonetes per arribar a un benefici òptim.

3. Conclusions

Al finalitzar tots aquests experiments hem arribat a la conclusió de que l'algorisme de Simulated Annealing obté uns millors resultats, tant pels beneficis com pel temps d'execució, en comparació amb l'algorisme de Hill Climbing. Aquest fet no és molt sorprenent, ja que l'algorisme de Simulated Annealing és un Hill Climbing estocàstic i soluciona els principals problemes del Hill Climbing, com poden ser els màxims locals i els altiplans.

Ens hem adonat de la importància d'una bona funció heurística a l'hora d'utilitzar aquests algorismes, ja que els canvis han sigut molt significatius. Per altre banda, hem vist com una solució inicial molt optimitzada no és tan necessària, inclús empitjorant els resultats en algunes ocasions, ja que no deixa espai de millora als algorismes de cerca local.

A més, hem après a que les nostres idees inicials no tenen perquè ser correctes i que, si és necessari per a obtenir millors resultats, no hem de dubtar en seguir les dades que ens mostra l'experimentació, independentment del que havíem pensat inicialment.

Amb aquesta pràctica hem après la importància d'una bona planificació a l'hora de realitzar un projecte de Inteligencia Artificial, ja que una bona estructura per a dur a terme el projecte ha sigut crucial per a obtenir bons resultats.