INTEL·LIGÈNCIA ARTIFICIAL

PRÀCTICA de CERCA LOCAL

DAVID GARCÍA, DAVID HIDALGO, JOAN MARC COLL

30 d'octubre 2023

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona
FIB

Índex

1 Introducció

Realització de la primera pràctica de laboratori, on l'objectiu és familiaritzar-se amb diferents tècniques de resolució de problemes, basades en cerca local; concretament treballarem amb els algorismes de *Hill Climbing* i *Simulated Annealing*.

També ens familiaritzarem amn l'ús de les llibreries AIMA de Java, en les quals trobem els dos algorismes mencionats, centrant-nos així en com optimitzar la resolució del problema sense tenir en compte l'implementació d'aquests.

Després de raonar sobre els elements necessaris per a resoldre el problema, determinar els recursos que utilitzem i molt esforç, aquests han estat els nostres resultats.

2 Problema

2.1 Descripció del problema

El problema consisteix en ajudar a l'empresa Bicing a resoldre un problema de demandes en les seves estacions de bicicletes. Per fer-ho, hem de tenir en compte diversos aspectes, com la previsió de quantes bicicletes no seran utilitzades en una estació durant una hora especifica i que poden ser mogudes a una altra estació, amb el primordial objectiu de cobrir la demanda prevista.

La modelització del problema es basa en entendre la ciutat com un quadrat de 10x10 km i que els carrers formen una quadrícula on cada illa té 100x100 metres. Les estacions Bicing es trobaran en els creuaments entre carrers, però no necessàriament hi haurà una estació en cada creuament.

Per poder realitzar la pràctica, tenim un número E d'estacions de bicicletes a la ciutat i un número B de bicicletes en total. Aquestes bicicletes estan distribuïdes entre totes les estacions i no podem treure o afegir bicicletes al sistema. Assumirem que el nombre de bicicletes que caben en una estació és il·limitat. Per altra banda, tenim dos escenaris diferents de demanda: la demanda equilibrada, i l'hora punta, on la concentració de demanda de bicicletes per estació varia.

Per optimitzar la distribució de bicicletes a les estacions l'empresa té com objectiu traslladar bicicletes d'una estació a una altra per aproximar-se a la demanda prevista. Hem de tenir en compte que ens pagaran un euro per cada bicicleta que es transporti que apropi una estació a la seva demanda prevista. Per contra, ens cobraran un euro per cada bicicleta que es transporti i que allunyi una estació de la seva previsió.

Per dur a terme el trasllat de bicicletes, comptem amb una flota de F furgonetes, cadascuna capaç de transportar 30 bicicletes com a màxim. Cada furgoneta en una hora només pot fer un únic viatge, transportant bicicletes d'una estació a com a màxim 2 estacions i no pot haver-hi dues furgonetes que agafin bicicletes de la mateixa estació. Tampoc cal utilitzar totes les furgonetes en cas que no sigui necessari.

L'origen de la furgoneta, els destins que tindrà (dos com a màxim) i en quin ordre els recorrerà, així com quantes bicicletes deixarà en cada destí depenen de la solució del problema per a una hora concreta.

Donat que cal optimitzar diversos criteris i el camí per arribar a la solució no ens interessa, és interessant realitzar un problema de cerca local per trobar una solució prou òptima, amb un cost temporal i espacial raonable.

2.2 Representació del problema

Per resoldre problemes amb algorismes de cerca local és imprescindible ser capaç d'encapsular tota la informació necessària per representar l'estat del problema en cada instant de temps. Però no només això, cal fer-ho d'una forma eficient en quant a memòria, donat que el nombre de nodes que es poden arribar a tractar alhora és potencialment gran. És per això que hem implementat la classe BicingBoard.

2.2.1 Classe BicingBoard

La classe BicingBoard té com a objectiu representar l'estat del problema, és a dir, com es troba el món en moment determinat. Més concretament, ens informa d'on estan localitzades cadascuna de les estacions, on comença cada furgoneta i quin trajecte faran. Atributs:

- *estaciones:* instància de la classe Estaciones. A grans trets, array de Estacion que conté tota la informació referent a les estacions.
- *num_est:* enter que representa el nombre d'estacions del problema.
- *nfurgos*: enter que representa el nombre de furgonetes del problema.
- *bicisXfurgo1:* array d'enters que indica el nombre de bicis que transporta la furgoneta i-ésima al primer destí.
- *bicisXfurgo2:* array d'enters que indica el nombre de bicis que transporta la furgoneta i-ésima al segon destí.
- *origenFurgo:* array d'enters que indica l'identificador de l'estació on comença el trajecte la furgoneta i-ésima.
- *esOrigen:* array de booleans que indica si l'estació i-ésima és l'origen d'alguna furgoneta.
- *estacionDest1:* array d'enters que indica l'identificador de la primera estació que visita la furgoneta i-ésima.
- *estacionDest2:* array d'enters que indica l'identificador de la segona estació que visita la furgoneta i-ésima.
- *bicisLibres:* array d'enters que indica quantes bicis disponibles té l'estació i-ésima.

Els atributs *num_est* i *nfurgos* estan declarats com static, donat que, com mai canvien, no cal emmagatzemar-los en cada instancia d'estat. A més, tenir l'atribut *bicisLibres*

ens permet declarar també *estaciones* com static, ja que només l'utilitzem per consultar la posició de les estacions.

A més dels atributs esmentats, la classe BicingBoard implementa una serie d'altres mètodes per treballar amb els atributs esmentats.

3 Generació d'una solució inicial

Per tal de generar la solució inicial, inicialment assignem les furgonetes a aquelles estacions amb el major nombre de bicicletes sobrants. D'altra banda, hem plantejat diverses maneres d'assignar destins a les nostres furgonetes. Aquestes són l'assignació de destins per mitjà del criteri de l'algorisme, assignació aleatòria a estacions on com a mínim calgui una bicicleta, o per últim, a les estacions on calgui el major nombre de bicicletes.

3.1 Solució inicial no assignada

La primera d'aquestes és l'assignació de destins per mitjà del criteri de l'algorisme. En executar el programa, després d'haver inicialitzat les variables pertinents per la representació de l'estat inicial, el que fem és cridar a la funció *asignaDestinos()*, que s'encarregarà de posar a -1 tots els destins de cadascuna de les F furgonetes, ja que no els volem inicialitzats a una estació en concret. També assignarà que cada furgoneta portarà zero bicicletes inicialment, doncs volem carregar-les quan s'indiqui el destí.

Per l'assignació de destins i les bicicletes carregades per cadascuna d'aquestes furgonetes, l'algorisme utilitzarà els operadors que cregui necessaris per tal de modificar els nostres arrays de destins i nombre de bicis carregades. Segurament, s'hagin d'expandir més nodes que amb les altres assignacions, però així ens assegurem que posem les furgonetes en la millor posició.

3.2 Solució inicial aleatòria

A continuació, farem esment a la solució inicial aleatoritzada. El que busquem amb aquesta assignació és buscar els destins on calgui com a mínim una bicicleta, ja que no ens interessa col·locar com a destí una estació qualsevol, perquè potser l'estació escollida no té demanda de bicicletes, i per tant no tindria sentit assignar-la com a destí. La funció encarregada és *asignaDestinosRandom()*.

En cas de trobar una estació on calguin bicicletes, el que fem serà assignar-la com a primer destí i carregarem les bicis pertinents a *bicisXfurgo1*, essent aquest l'array que

carrega les bicis pel primer destí. El nombre de bicicletes que carregarà serà el mínim entre les bicicletes que l'estació necessita per cobrir la demanda i les bicicletes que pot carregar del seu origen. Anàlogament, pel segon destí.

3.2 Solució inicial més demandada

Per últim, però no menys important, tenim l'assignació fixada a les estacions amb major demanda de nombre de bicicletes. El que volem amb aquest plantejament és cobrir la demanda d'aquelles estacions per aprofitar i deixar el major nombre de bicicletes d'una tirada. La funció encarregada per aquest darrer cas és la *asignaDestinosMax()*.

Aquesta última assignació serà semblant a la random respecte al fet que agafa com a destins les estacions on falten bicicletes, però la diferència és que agafa aquelles on faltin més.

4 Operadors

4.1 Descripció d'operadors

Els operadors ens permeten moure'ns entre els estats del nostre problema.Inicialment, vam plantejar la idea d'incloure operadors per posicionar i treure furgonetes, o carregar i descarregar bicicletes d'aquestes; no vam trigar, però, en adonar-nos que aquests operadors o bé només ens interessen a la solució inicial, o la seva implementació només empitjoraria el factor de ramificació de cada estat. És per això que vam decidir no incloure'ls. Els operadors que vam decidir implementar són:

a) **ChangeOrigen(int f, int est)**

- Descripció: Estableix *est* com a nova estació d'origen de la furgo *f*.
- Condicions d'aplicabilitat: Que l'estació *est* no sigui origen d'una altra furgoneta (*origenFurgo*), i que li sobrin bicicletes (*bicisLibres*).
- Factor de ramificació: O(*nfurgos* · *nest*)

b) **ChangeDest1(int f, int est)**

- Descripció: Canvia el primer destí de *f* per l'estació *est*.
- Condicions d'aplicabilitat: Que l'estació *est* tingui demanda de bicicletes (*bicisLibres*)
- Factor de ramificació: O(*nfurgos* · *nest*)

c) ChangeDest2(int f, int est)

- Descripció: Canvia el segon destí de la furgoneta *f* per l'estació *est*.
- Condicions d'aplicabilitat: Que la furgoneta *f* tingui assignat un primer destí (*estacionDest1*) i que *est* tingui demanda de bicicletes (*bicisLibres*).
- Factor de ramificació: O(*nfurgos* · *nest*)

d) SwapDest1(int f1, int f2)

- Descripció: Intercanvia els primers destins de *f1* i *f2*.
- Condicions d'aplicabilitat: Que *f1* i *f2* tinguin assignats un primer destí (*estacionDest1*).
- Factor de ramificació: O(*nfurgos* · *nfurgos*)

e) SwapDest2(int f, int est)

- Descripció: Intercanvia els segons destins de *f1* i *f2*.
- Condicions d'aplicabilitat: Que *f1* i *f2* tinguin assignats un segon destí (*estacionDest2*).
- Factor de ramificació: O(*nfurgos* · *nfurgos*)

f) SwapDestPropios(int f)

- Descripció: Intercanvia l'ordre dels destins de la furgoneta *f*.
- Condicions d'aplicabilitat: La furgoneta f té assignat un primer i segon destí (*estacionDest1* i *estacionDest2*)
- Factor de ramificació: O(*nfurgos*)

Amb aquests operadors explorarem una gran part de l'espai de solucions, sempre apropant-nos a una millor solució, adaptant-se implícitament a mesura que canvien els estats; per exemple, al modificar el destí d'una furgoneta aquesta passarà a carregar les justes i necessàries per a la ruta que està traçant (*bicisXfurgo1* i *bicisXfurgo2* indiquen les bicicletes que dipositarà en cadascun dels seus destins).

Cal comentar que segons l'estratègia que utilitzem per generar la solució inicial, alguns operadors tindran més o menys rellevància, com és el cas de *changeOrigen*.

4.2 Conjunt d’operadors

4.2.1 Conjunt *Change*

Aquest conjunt serà la base del nostre problema, ja que *changeDest1* i *changeDest2* s'encarreguen d'inicialitzar els destins de les furgonetes quan triem una estratègia de solució inicial que no ho fa. Es compon de *changeDest1*, *changeDest2*, *changeOrigen* i *swapDestPropios*. Aquests són els operadors que hem considerat bàsics per a qualsevol estratègia generadora de solució inicial.

4.2.1 Conjunt *Swap*

Aquest segon conjunt d'operadors, es compondrà del conjunt d'operadors Change més els operadors *swapDest1* i *swapDest2*. Ja que consideràvem que podrien afegir una major exploració de solucions a l'algorisme, atès que els operadors *changeDest1* i *changeDest2* era poc probable que tinguessin en compte estacions que ja havien estat assignades, perquè la demanda d'aquestes ja havia estat en part suplida.

5 Generació d’estats successors

Una part important a l’hora de resoldre un problema utilitzant algorismes de cerca local és la generació dels estats successors mitjançant operadors. Per fer-ho, cal estudiar com aplicar aquests operadors i cal controlar en quins escenaris es poden o no utilitzar.

5.1 Funció generadora d’estats

Amb l'objectiu d'aconseguir una bona exploració de l'espai de solucions, cal aplicar els operadors de totes les formes possibles. No obstant, això pot provocar generar estats poc prometedors o estats que es troben fora de l'espai de solucions. És per aquesta raó que cal assegurar-se que es compleixen unes condicions d'aplicabilitat abans d'utilitzar aquest operadors.

Amb aquest objectiu hem implementat una funció per cadascun dels operadors que comprova si es compleix la condició d'aplicabilitat mencionades a l'apartat anterior.
Per altra banda, de cara als experiments, hem implementat una classe anomenada 'BicingSuccesorFunctionNoSwaps.java' per provar el nostre programa de manera més ràpida i que es salti moltes solucions.

6 Heurística

En quant a l'heurística, hem definit tres tipus. Dues heurístiques per separat, i una tercera que és la combinació de les anteriors. Les dues heurístiques individuals serveixen per calcular el cost i la demanda, respectivament. La tercera és la minimització del cost i la maximització de la demanda, ja que el nostre propòsit és abastir el requisit de bicicletes de totes les estacions de la forma menys costosa.

6.1 Heurística Cost

Pel que fa a l'heurística del cost, hem de tenir dos aspectes molt presents. Siguin *nb1*, *nb2* dos variables double que ens serviran per definir les bicicletes portades a destí 1 i 2, respectivament. De la mateixa manera, tenim definides *km1* i *km2*, essent aquests els quilòmetres que recorren les furgonetes per arribar a cada destí. Ambdues variables per cada destí estaran inicialitzades a zero.

Seguidament, hem de calcular aquestes distàncies per mitjà de la distància de Manhattan. La fórmula emprada és la següent:

$$d(i,j) \;=\; \left|i_x - j_x\right| \;+\; \left|i_y - j_y\right|$$

on i_x , i_x són les coordenades $\{x, y\}$ en metres del punt i a la quadrícula.

Partirem des de l'origen i anirem fins a l'estació de destí 1, la qual ja estarà assignada independentment del mètode que agafem inicialment. El càlcul corresponent es guardarà a *km1*, i les bicicletes transportades al primer destí *nb1* seran les bicis que ha de portar a destí 1 més les bicis que la furgo ha de portar al segon destí, perquè un cop surti del primer destí, s'anirà cap al segon. Per tant, també hem de tenir en consideració aquestes bicicletes del segon destí.

Anàlogament, fem el mateix pel segon destí. Però hem de tenir en compte que aquest segon destí pot estar assignat o no, ja que potser no cal anar-hi. El que sí que sabem és que una furgoneta no pot tenir assignat un segon destí sense tenir assignat el primer, però pot tenir assignat un primer destí sense tenir assignat el segon.

Aleshores, un cop tenim assignat el segon destí, calculem la distància partint del primer fins al segon, i la guardem a *km2*. Ara, però, el nombre de bicis transportades *nb2* al destí seran únicament les que calguin portar, doncs les del primer destí ja les haurem deixat prèviament.

Ara bé, un cop calculades les variables, farem el càlcul de l'heurístic, sent aquest el sumatori del càlcul del cost dels dos destins. Els *km1*, *km2* que estaven en metres, ara hauran de passar a ser quilòmetres. La fórmula del càlcul del cost serà la següent:

$$C_f = C_{dest1} + C_{dest2} \qquad \text{on} \quad C_{dest\alpha} = km\alpha \cdot \frac{nb\alpha + 9}{10}\text{, sent } \alpha \in \{1, 2\}$$

Repetirem el càlcul per cadascuna de les furgonetes, i el resultat l'acumularem a una variable, que serà la que finalment retornem. El cost representaran euros.

6.1 Heurística Demand

Aquest heurístic en comparació amb l'anterior no comporta tants càlculs. El que farem serà, per cada estació, calcular les bicicletes que hem mogut per apropar-la a la seva demanda prevista. Això ho farem fent la diferència entre la demanda de bicicletes que tenia una estació inicialment i la demanda que té actualment a bicisLibres. No cal tenir en compte res més, ja que mai allunyarem una estació de la seva demanda prevista.

Al final retornarem la suma de tots aquests valors negada, perquè la diferència entre dues demanades es negativa.

6.1 Heurística Profit

Aquesta heurística serà la que més utilitzem, doncs tenir les dues heurístiques anteriors combinades serà un factor clau. Com hem comentat anteriorment, el que es basa aquesta heurística és minimitzar el cost de transport i maximitzar el compliment de la demanda.

En cas que ens indiquin que el cost de transport és gratuït, simplement haurem de retornar el cost multiplicat per zero i cridar a la funció per calcular la demanda.

7 Experimentació

7.1 Conjunt d'operadors

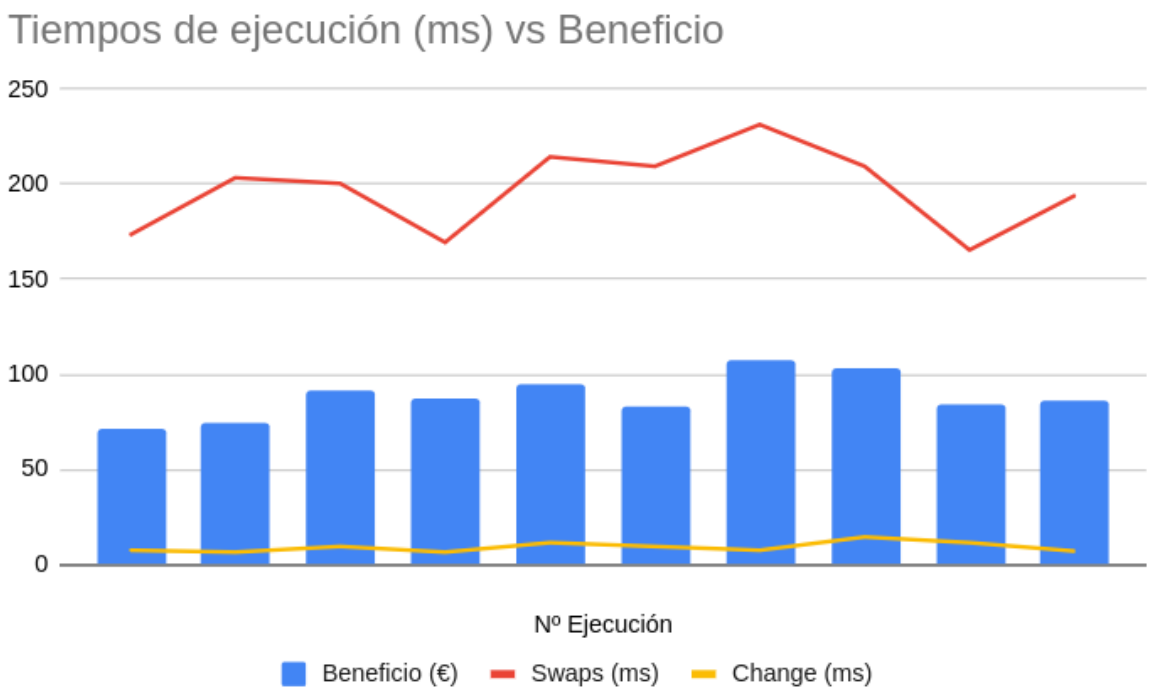
Observació	La decisió de quin conjunt d'operadors utitilitzar pot afectar al resultat.
Plantejament	Observar els resultats obtinguts amb cadascun dels conjunts d'operadors.
Hipótesis	El benefici no canviarà donat que en ambdós casos exploren el mateix espai de solucions, però el conjunt swap acabarà abans.
Mètode	- Inicialitzem el problema amb 25 estacions, 5 furgonetes, 1250 estacions i demanda equilibrada. - Utilitzarem Hill Climbing amb la heurística de demanda sense cost. - Com a solució inicial utilitzarem la buida, és a dir, no assignarem cap destí. - Escollim a l'atzar 10 seeds diferents. - Per cada seed, executem l'algorisme amb els dos conjunts d'operadors. - Per cada execució extraiem el benefici i el temps d'execució.

Resultats a continuació. Taula amb els beneficis i temps per execució depenent del conjunt d'operadors:

	Conjunto change		**Conjunto swaps**	
Nº Ejecución	**Tiempo(ms)**	**Beneficio**	**Tiempo(ms)**	**Beneficio**
Exec 1	7	71	173	71
Exec 2	6	75	203	75
Exec 3	9	91	200	91
Exec 4	6	87	169	87
Exec 5	11	95	214	95
Exec 6	9	83	209	83
Exec 7	7	107	231	107
Exec 8	14	103	209	103

Exec 9	11	84	165	84
Exec 10	7	86	194	86
Promedio	**8,7**	**-**	**196,7**	**-**

Gráfica que exposa la diferència en temps d'execució per cada conjunt d'operadors:



Conclusions: La hipòtesi inicial es parcialment correcta. Es compleix que el benefici és el mateix tant pel conjunt change com pel conjunt swap. No obstant, però, els temps d'execució resulten significativament més grans pel conjunt d'operadors que utilitza swaps. Això es pot deure a que aquest segon conjunt no és més que una ampliació del primer i que, per tant, té més operadors i genera més estats. A més, els operadors swaps que s'agreguen al segon conjunt tenen una complexitat cuadrática respecte el nombre d'estacions, un ordre potser massa gran per un algorisme de HC.

7.2 Solució inicial

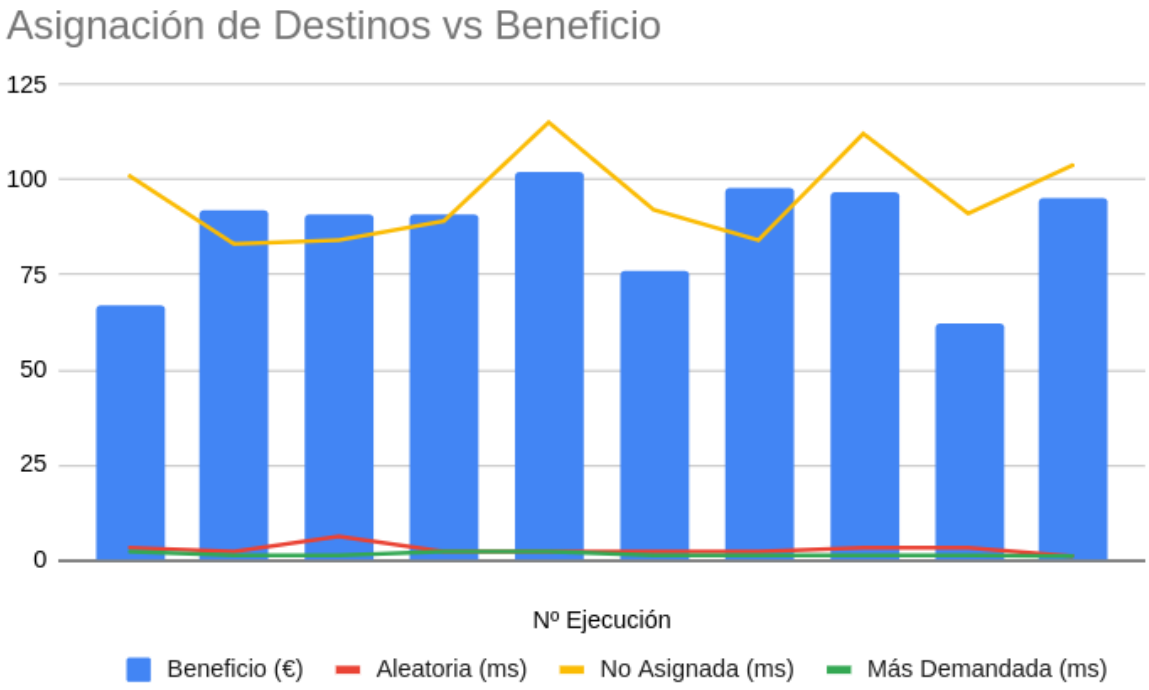
Observació	La decisió de quina solució inicial utitilitzar pot afectar al resultat.
Plantejament	Observar els resultats obtinguts amb cadascuna de les solucions inicials.
Hipótesis	La solució buida (no assigna destins) pot donar en promig un major benefici donat que té potencial per explorar un major espai de solucions que la resta.
Mètode	- Inicialitzem el problema respectant les proporcions d'estacions, furgonetes i bicicletes, i amb una demanda equilibrada. - Utilitzarem Hill Climbing amb la heurística de demanda sense cost. - Com a conjunt d'operadors hem escollit el conjunt change. - Escollim a l'atzar 10 seeds diferents. - Per cada seed, executem l'algorisme amb les diferents solucions inicials. - Per cada execució extraiem el benefici i el temps d'execució.

Resultats a continuació. Taula amb els beneficis i temps per execució depenent de la solució inicial escollida:

	Generación Destinos no Asignada		**Generación Destinos Aleatoria**		**Generación Destinos Máxima Demanda**	
Nº Exec	**Tiempo (ms)**	**Beneficio**	**Tiempo (ms)**	**Beneficio**	**Tiempo (ms)**	**Beneficio**
Exec 1	101	67	3	67	2	67
Exec 2	83	92	2	92	1	92
Exec 3	84	91	6	91	1	91
Exec 4	89	91	2	91	2	91
Exec 5	115	102	2	102	2	102
Exec 6	92	76	2	76	1	76
Exec 7	84	98	2	98	1	98
Exec 8	112	97	3	97	1	97

Exec 9	91	62	3	62	1	62
Exec 10	104	95	1	95	1	95
Promedio	**95,5**	-	**2,6**	-	**1,3**	-

Gráfica que exposa la diferència en temps d'execució amb les diferents solucions inicials:



Conclusions: La hipótesi inicial no es compleix. Es pot observar que el benefici obtingut es el mateix per cadascuna de les solucions inicials. La raó darrere d'aquest resultat podria ser que, al utilitzar l'heurístic que no té en compte el cost, existeixen molts màxims locals iguals i tots acaben convergint en un d'aquests. Com a observació, podem extreure també que quan utilitzem la solució buida el temps d'execució és significativament més gran. Probablement es deu a que les altres dos ja estan força a prop d'un màxim local.

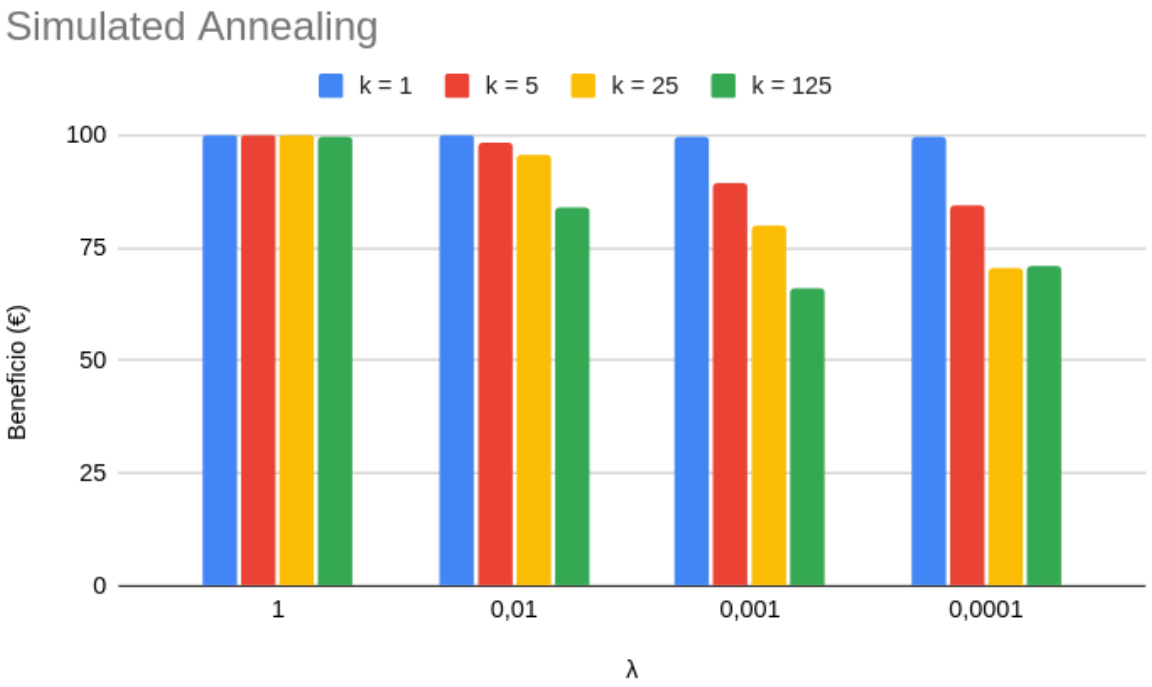
7.3 Paràmetres Simulated Annealing

Observació	Els paràmetres triats per l'algorisme afectaran en gran mesura el rendiment d'aquest.
Plantejament	Observar els resultats obtinguts amb cadascun dels paràmetres triats.
Hipótesis	Una vegada fixat un numero d'iteracions altes per tal de que l'algorisme tingui marge per baixar la seva temperatura, els valors òptims de k i λ que obtindrem, seran una k ni molt gran ni molt petita per que l'algorisme tingui períodes significatius de temperatures altes i baixes, i una λ relativament gran per tal de contrastar la diferencia de temperatura.
Mètode	- Inicialitzem el problema amb els paràmetres d'estat utilitzats als anteriors experiments. - Fixem un numero d'iteracions altes per l'algorisme, entorn les 100000, ja que observem que aquest mai passa de les 76000. - Escollim a l'atzar 5 seeds que es repetiran per a tots el experiments i fer les mitjanes. - Per cada seed, executem l'algorisme amb els valors de k i λ corresponents. - Per cada execució extraiem el benefici.

Mitjana execucions per als diferents valors de **k** i **λ**.

k \ λ	**1**	**0,01**	**0,001**	**0,0001**
1	100	100	99,6	99,6
5	100	98,2	89,3	84,3
25	100	95,6	80	70,5
125	99,5	84,1	65,9	70,8

*Màxim benefici es 100, quan l'algorisme arriba a totes les solucions òptimes.



Conclusions: La hipótesi inicial no es compleix. Es pot observar que l'algorisme maximitza el benefici quan λ pren un valor gran, i millora amb una k petita. Observem que té sentit, ja que l'algorime de *Simulated Annealing* en un problema amb un comportament tan greedy com aquest no té gaire sentit; i per tant, al maximitzar λ i minimitzar k, la temperatura baixa de seguida, aconseguint un comportament molt similar al de *Hill Climbing*.

7.4 Temps d'execució amb Hill Climbing

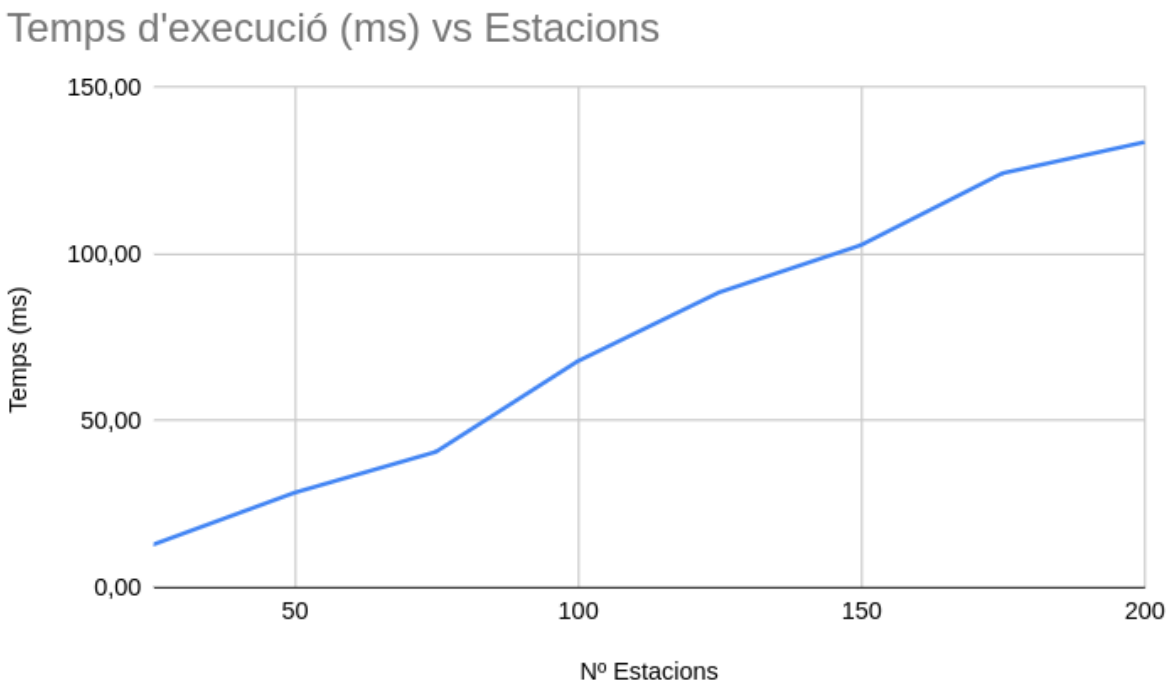
Observació	S'ha d'establir una proporció 1 a 50 entre estacions i bicicletes, i 1 a 5 entre furgonetes y estaciones.
Plantejament	Anirem augmentant el nombre d'estacions seguint la relació establerta fins a trobar la tendència del temps d'execució.
Hipótesis	El temps d'execució amb Hill Climbing seguirà una tendència lineal (H_0), per cada valor que augmentem de forma proporcional.
Mètode	- Escollim a l'atzar 7 seeds diferents. - Establim els valors d'estacions, bicicletes i furgonetes. - Executem 1 experiment per cada seed usant Hill Climbing. - Guardem el temps d'execució. - Augmentem les furgonetes en múltiples de 5, sumem 25 estacions i 1250 bicicletes per cada experiment. - Tornem a repetir el procediment. Un cop tenim totes les execucions fetes: - Fem la mitjana de les execucions per cada seed, depenent del número d'estacions, bicicletes i furgonetes. - Tracem el gràfic corresponent i veiem la tendència.

Taula dels temps d'execució, depenent del nombre de furgonetes, estacions i bicicletes, respectivament:

Seed	5 - 25 - 1250	10 - 50 - 2500	15 - 75 - 3750	20 - 100 - 5000	25 - 125 - 6250	30 - 150 - 7500	35 - 175 - 8750	40 - 200 - 10000	60 - 300 - 15000
s1	13	27	39	69	95	125	114	119	241
s2	10	24	43	73	83	84	107	122	246
s3	24	29	50	64	83	111	119	143	282
s4	11	28	42	73	94	99	112	138	298
s5	12	29	40	63	82	102	126	142	291
s6	7	24	33	59	76	100	125	144	269
s7	11	36	36	72	105	96	165	127	300

Taula de mitjanes de temps d'execució T, en ms, depenent del nombre d'estacions:

Nº Est.	**25**	**50**	**75**	**100**	**125**	**150**	**175**	**200**	**300**
T (ms)	12,57	28,14	40,43	67,57	88,29	102,43	124,00	133,57	275,29



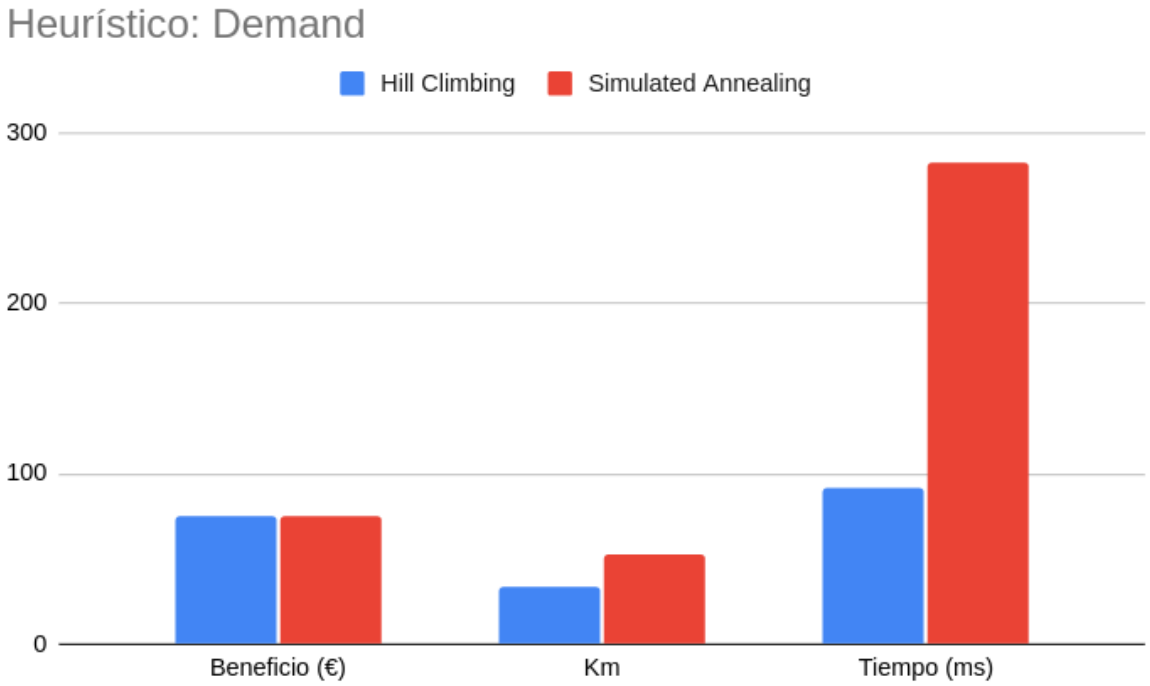
Conclusions: La nostra hipòtesis inicial és certa perquè es pot veure perfectament com la gràfica segueix una tendència lineal a mesura que anem incrementant el número de furgonetes, estacions i bicicletes.

7.5 Diferències entre algorismes de cerca local

Observació	S'han d'agafar els valors experimentals anteriors que millor funcionin per poguer analitzar les diferencies en cas millor.
Plantejament	Executarem els dos algorismes amb les heurístiques de la demanda i el profit, que té en compte el cost, per trobar quin algorisme proporciona més benefici.
Hipótesis	El millor benefici el proporcionarà Simulated Annealing perquè no s'estancarà localment i tornarà la solució més ràpid (H_0).
Mètode	- Escollim a l'atzar 5 seeds diferents. - Assignem un dels heurístics per fer l'experiment. - Executem 1 experiment per cada seed usant Hill Climbing. - Executem un altre experiment usant Simulated Annealing. - Guardem el benefici, quilometratge i temps d'execució. - Tornem a repetir el procediment, però escollint l'altre heurístic. Un cop tenim totes les execucions fetes: - Fem la mitjana del benefici, distància recorreguda i temps d'execució per cada seed, tant per Hill Climbing com per Simulated Annealing. - Tracem els gràfics corresponents i observem les diferències.

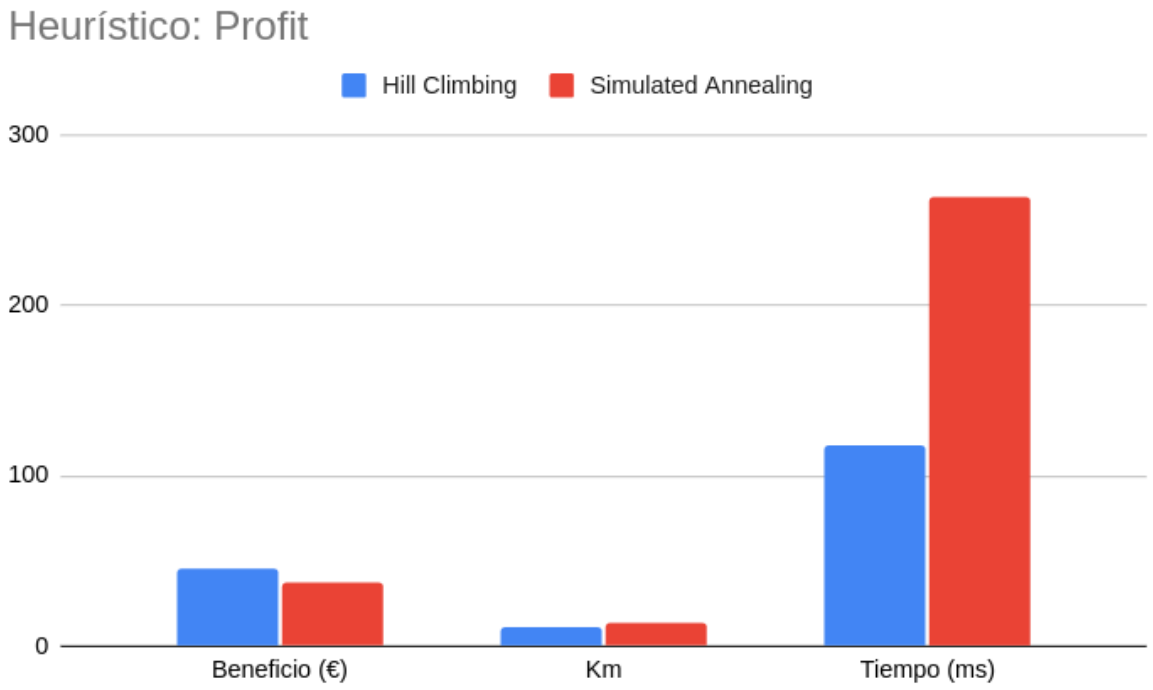
Taula del benefici, quilometratge i temps d'execució, depenent de cada algorisme, utilitzant l'heurística de la demanda, que s'encarrega de mirar les bicis a moure:

	Demand					
	Hill Climbing			**Simulated Annealing**		
Seed	**beneficio(€)**	**km**	**tiempo(ms)**	**beneficio(€)**	**km**	**tiempo(ms)**
s1	96	31,5	94	96	51,8	254
s2	73	44,6	96	73	49,5	259
s3	54	33,4	92	54	64,7	259
s4	89	28	92	89	48,3	280
s5	63	29	86	63	51,6	360
Media	**75**	**33,3**	**92**	**75**	**53,18**	**282,4**



Taula del benefici, quilometratge i temps d'execució, depenent de cada algorisme, utilitzant l'heurística de profit, que té en compte el cost de transport:

	Profit					
	Hill Climbing			**Simulated Annealing**		
Seed	**beneficio(€)**	**km**	**tiempo(ms)**	**beneficio(€)**	**km**	**tiempo(ms)**
s1	57,33	11,2	110	45,97	12,6	256
s2	35,56	12,9	115	30,15	19,5	273
s3	35,72	7,6	116	26,75	6,1	278
s4	65,21	11,8	129	53,22	14,7	242
s5	35,82	13,4	119	33,11	13,9	269
Media	**45,93**	**11,38**	**117,8**	**37,84**	**13,36**	**263,6**



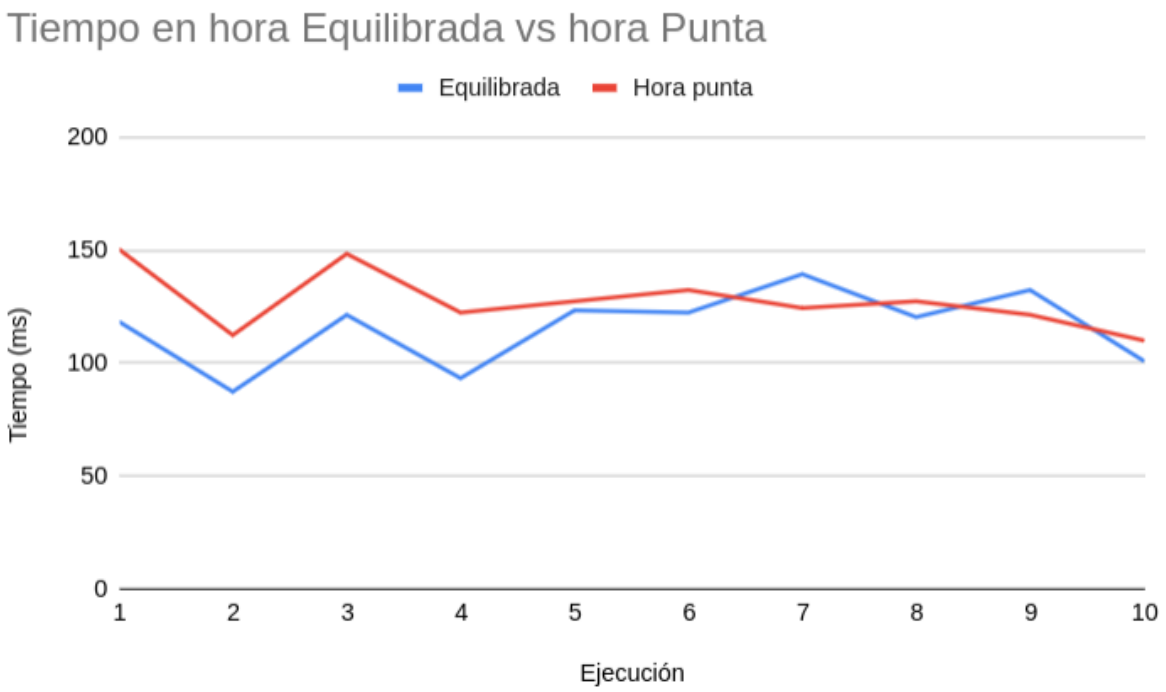
Conclusions: La nostra hipòtesis inicial és incorrecta perquè en un primer moment creiem que les furgonetes amb Simulated Annealing recorrerien menys distància, però com hem pogut comprovar, es recorre més distància per intentar trobar una millor solució. Això, per tant, implica que es trigarà més temps si executem Sim. Annealing.

Veiem que el benefici amb l'heurística de la demanda és el mateix, perquè les bicis a moure son les mateixes en ambdós casos, però quan intervé el cost de transport, tenim millor benefici amb Hill Climbing perquè com hem comentat, amb Simulated Annealing es recorre més distància, i per tant, menys benefici.

7.6 Tipus de demanda

Observació	S'han d'agafar els valors experimentals anteriors que millor funcionin per poguer analitzar les diferencies en cas millor.
Plantejament	Executarem l'algorisme de Hill Climbing amb diferents seeds, observant com varia el temps d'execució al variar el tipus de demanda.
Hipótesis	El temps d'execució per a les dues demandes serà similar.
Mètode	- Escollim a l'atzar 10 seeds diferents. - Assignem una de les demandes per a l'experiment. - Executem 1 experiment per cada seed. - Fem el mateix canviant la demanda. - Guardem el temps d'execució. Un cop tenim totes les execucions fetes: - Fem la mitjana del temps d'execució per a les dues demandes. - Tracem els gràfics corresponents i observem les diferències.

Seed	Tiempo Equilibrada (ms)	Tiempo Hora punta (ms)
1	118	150
2	87	112
3	121	148
4	93	122
5	123	127
6	122	132
7	139	124
8	120	127
9	132	121
10	101	110
	115,6	**127,3**



Conclusions: La nostra hipòtesis inicial és correcta, observem que hi ha una diferència en quant a temps d'execució rellevant a les 4 primeres execucions, però això pot ser a causa de factors externs per al rendiment del portàtil. Per la resta d'execucions veiem com la diferència de temps es negligible, la qual cosa té sentit donat que l'algorime només redistribueix les bicicletes al canviar la demanda, la qual cosa fa que el temps d'execució sigui pràcticament el mateix.

7.7 Furgonetes òptimes

Observació	S'han d'agafar els valors experimentals anteriors que millor funcionin per poguer analitzar les diferencies en cas millor.
Plantejament	Executarem l'algorisme amb els dos tipus de demanda, augmentat el número de furgonetes en cada experiment, per comparar el número òptim per a maximitzar el benefici en cada cas.
Hipótesis	Caldran més furgonetes per arribar al màxim benefici quan la demanda es troba en hora punta.
Mètode	- Escollim a l'atzar 5 seeds diferents. - Assignem un tipus de demanda. - Executem 1 experiment per cada seed usant els valors experimentals esmentats. - Executem un altre experiment amb l'altre tipus de demanda. - Guardem el benefici. - Tornem a repetir el procediment, però augmentant en 5 el número de furgonetes. Un cop tenim totes les execucions fetes: - Fem la mitjana del benefici en cada grup d'execucions. - Tracem els gràfics corresponents i observem les diferències.

Taula de la demanda equilibrada:

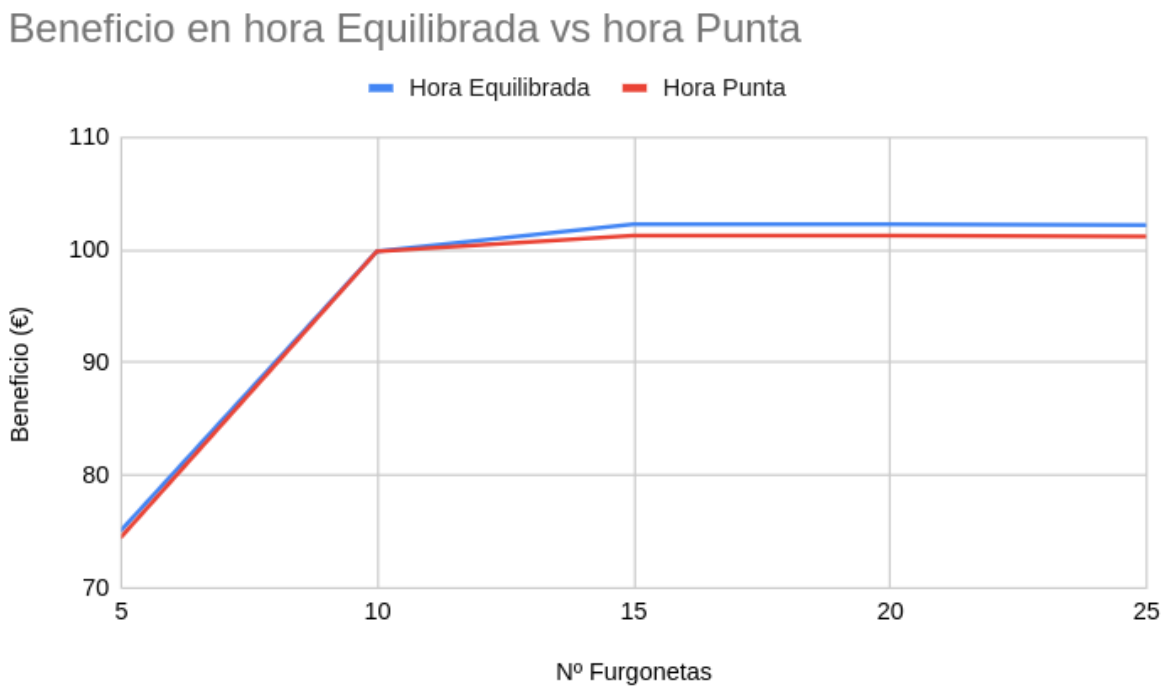
Demanda equilibrada					
F\Seed	**1**	**2**	**3**	**4**	**5**
5	96	73	54	89	63
10	151	95	79	108	66
15	162	96	79	108	66
20	162	96	79	108	66
25	162	96	79	108	66

Taula de la demanda en hora punta:

	Hora punta				
F\Seed	**1**	**2**	**3**	**4**	**5**
5	90	83	60	84	55
10	134	107	88	103	67
15	138	108	89	104	67
20	138	108	89	104	67
25	138	108	89	104	67

Taula de mitjanes en hora equilibrada i hora punta:

F\Tipus Dem	Hora Equilibrada	Hora Punta
5	75	74,4
10	99,8	99,8
15	102,2	101,2
20	102,2	101,2
25	102,2	101,2



Conclusions: La nostra hipòtesis inicial és incorrecta perquè en un primer moment creiem que en demanda d'hora punta, al haver-hi estacions amb un número de bicicletes necessàries major al màxim que pot transportar una furgoneta, creiem que caldrien més furgonetes.

A la gràfica podem observar que la diferència que existeix entre la relació del benefici i el número de furgonetes per cada demanda és molt petita o gairebé inexistent.