Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Primera pràctica ABIA: Azamon

Grau en intel·ligència artificial

ABIA

Pablo Chacón Martin, Mikel Lecumberri Arteta i
Elies Aragonès Larrañaga

11/11/2024

Índex

1 Introducció

Aquest treball es centra en la implementació i l'anàlisi d'algorismes de cerca local per resoldre un problema logístic de distribució de paquets. La companyia fictícia Ázamon, que envia paquets a ciutats determinades, ens ha plantejat el repte de dissenyar un sistema que assigni els enviaments de manera eficient, minimitzant els costos associats tant al transport com a l'emmagatzematge dels paquets. A més, es considera la felicitat del client com a factor de satisfacció, relacionat al compliment dels terminis de lliurament.

L'objectiu principal és optimitzar aquests aspectes utilitzant algorismes de cerca local, concretament, Hill Climbing i Simulated Annealing, avaluant-ne l'eficàcia i comparant-ne els resultats en diferents escenaris experimentals. Aquest projecte inclou una fase de disseny experimental, amb la qual es busca obtenir conclusions sobre l'eficàcia de les estratègies d'inicialització de solucions, els operadors de transformació d'estats i les funcions heurístiques utilitzades.

Així, aquest treball ofereix una exploració detallada de com els algorismes de cerca local poden aplicar-se en problemes logístics reals, amb la finalitat de proporcionar solucions pràctiques i eficients a problemes molt presents a nombroses empreses avui dia.

2 Representació de l'estat

Observant el problema podem deduir que principalment hi haurà dos elements que intervindran en el problema: els diferents paquets amb els seus respectius temps d'entrega i pes i les ofertes, amb els seus dies d'arribada i la càrrega màxima. Per tant, aquests dos elements tindran una classe cada un amb els atributs comentats.

Per tant, un estat que sigui solució serà aquell que hagi assignat cada paquet a una oferta. Per ser vàlida, s'hauran de complir dues condicions, que la suma dels pesos dels paquets a un camió (oferta) sigui superior a la seva càrrega màxima i que tots els paquets siguin entregats dins de la seva prioritat, o abans.

Per tal de representar els estats hem decidit utilitzar una llista de sets, ja que això ens permet emmagatzemar les dades de forma eficient. Cada un dels sets representa una de les ofertes, als quals es va afegint els índexs dels paquets que es van assignant a les respectives ofertes. Vam triar aquesta estructura perquè, a part que no necessitem tenir un ordre a dins de cada un, els conjunts són molt eficients per certes operacions, com ara la cerca i l'eliminació, que tenen un cost molt baix gràcies a l'ús de la funció *hash*.

3 Elaboració del codi

3.1 Solucions inicials

El primer pas va consistir en l'elaboració del codi encarregat de generar les solucions inicials, ja que és el punt de partida per després poder aplicar els algoritmes. Teòricament, aquesta part hauria de tenir el cost més baix possible, ja que el pes de l'execució haria d'estar amb els operadors. En aquest apartat vam aconseguir crear dos codis que generen solucions inicials vàlides, però amb un cost una mica més elevat del que ens hagués agradat.

3.1.1 Primer generador de la solució inicial

El primer generador de solucions inicial que ens va venir al cap va ser el guiar-nos per assignar als paquets l'oferta de menor preu possible i que alhora fos assignable. Ens vam fixar en la generació subòptima del codi proporcionat per la pràctica, traient l'aleatorietat i aplicant-la al que ens interessava.

Aquesta assignació és una funció formada per dues funcions auxiliars que es criden durant el codi. La primera és *assignable*, que retorna un booleà si aquell paquet es pot assignar a aquella oferta mirant el criteri de la prioritat. A *assignable* li passem com arguments la prioritat del paquet i l'oferta. La segona funció auxiliar és la que determina la nostra aproximació. *Precio min* calcula l'index de la oferta més barata tenint en compte la funció anterior (criteri prioritat), on els arguments de la funció són la llista de id d'ofertes, la llista d'ofertes i la prioritat del paquet. Es crida des de la següent part del codi que ara explicarem.

El primer que fem és generar tres llistes, una amb tants espais com ofertes per col·locar el pes que hi ha, una altra amb tants espais com paquets i una última buida que omplirem amb els id d'ofertes de la llista original que ens proporciona el problema. Després iterarem per a cada paquet i amb un booleà determinarem si el paquet està assignat o no. Mentre no estigui assignat entrarem a un bucle *while*, que el primer que farà serà cridar a *precio min* per obtenir el id de la millor oferta quant a preu i que sigui assignable segons la prioritat. Després mirarem si el paquet sobre el qual estem iterant cap a l'oferta que hem obtingut. Si obtenim èxit i el paquet cap a la oferta, sumem el pes d'aquest paquet a la llista de pesos per oferta que hem creat l'inici i assignem a la llista d'ofertes el paquet en el seu respectiu id d'oferta. Després d'això passem a *TRUE* el boolèa de paquet assignat i passem al següent. Per l'altra banda, si el paquet no cap en aquesta oferta traiem el id de l'oferta de la llista de id's d'ofertes i tornem a buscar la següent millor oferta, ja que la condició del *while* seguirà sent certa. Ens apropa massa a la solució òptima, per la qual cosa li deixem molt poca feina a Hill Climbing i Simulated Annealing. En treure l'oferta si no cap a un paquet ens pot deixar espai lliure, aquí es on entra el paper del operados dels algoritmes de cerca local.

3.1.2 Segon generador de la solució inicial

La idea darrera de la creació del segon codi per a la generació de la solució inicial és molt més senzilla i la vam arribar a comentar també a classe amb l'exemple del *binpacking*. El primer pas consisteix a transformar la llista de paquets original en una tupla de *len* 3, on en cada posició hi ha una llista amb els paquets que tenen la mateixa prioritat. A més, en tot moment guardem les instàncies de *Paquet* en una tupla juntament amb el seu índex oringial, per tal e no perdre'l després d'ajuntar els paquets.

A continuació, a la funció principal ens trobarem una seqüència de bucles per tal de fer l'assignació. Els dos primers serveixen per recórrer cada una dels paquets de la tupla generada anteriorment. Finalment, hi ha un tercer que serveix per, quan tenim un paquet, recórrer les ofertes per assignar-lo a la primera disponible. Per fer-ho comprovarem dues condicions: la primera, que el pes del paquet sigui menor o igual al pes disponible que té l'oferta i, la segona, es comprova amb una funció auxiliar que els dies d'entrega de l'oferta estiguin dins de l'interval associat a la prioritat del paquet.

Ara bé, fent això només ens vam trobar amb un error: depenent de l'ordre amb el qual assignàvem els paquets podia haver-hi casos en els quals quedaven paquets per assignar, degut que no cabien als paquets que complien els seus temps d'entrega. Per solucionar això només vam haver de fer un petit canvi: un cop accedim a la tupla i tenim la llista de paquets, en comptes de seleccionar-los per l'ordre predeterminat, a cada iteració s'agafa sempre el que té el pes més gran. Amb això vam aconseguir emplenar al màxim cada un dels paquets i assegurar-nos que tots cabessin.

Per acabar, una vegada es troba la primera oferta disponible es duen a terme les següents accions: es treu el paquet de la llista de paquets, s'afegeix l'índex del paquet (segon element de la tupla) al set corresponent de la llista resultat i s'actualitza el pes disponible de l'oferta. Posem un *break* per deixar de recórrer les ofertes.

A més a més, al final del codi també es comprova que tots els paquets hagin sigut assignats, mirant si la suma de les longituds de tots els sets és igual a la longitud de la llista de paquets inicial.

3.2 Operadors

Els operadors seran els que ens permetran moure'ns per l'espai de solucions per tal de dur a terme els algoritmes de Hill Climbing i Simulated Annealing. En un inici vam decidir que els nostres operadors serien el de *moure_paquet* i el de *intercanviar_paquet*, ja que vam considerar que amb aquests dos ja podríem explorar tot l'espai de solucions. Posteriorment, en vam crear algun d'addicional, el qual provaríem més endavant a l'apartat d'experimentació per tal de comprovar si s'obtenen millors resultats amb aquests. Cal mencionar que els criteris d'aplicabilitat no es comproven en aquest apartat, sinó que en el següent, en el moment de generar les accions.

3.2.1 *move*

Aquest és l'operador més senzill, pel fet que l'únic que fa és canviar un paquet d'una oferta a una altra. Per fer-ho, se li passen tres arguments, l'índex de l'oferta, el número de l'oferta a la qual estava originalment el paquet i el número de l'oferta on volen posar el paquet. Pel que fa al factor de ramificació, si tenim n paquets en m ofertes, en el pitjor dels casos, és a dir, en el cas que tots els paquets puguin ser moguts a qualsevol oferta (exceptuant en la que està), el factor de ramificació seria:

$$\text{Factor de ramificació move} = n \times (m - 1)$$

3.2.2 *swap*

Tot i que podria semblar que aquest operador no és estrictament necessari per tal d'explorar tot l'espai de solucions, això no és així. Si féssim un *swap* utilitzant un doble *move* podria suposar haver de passar per una solució pitjor abans d'arribar a una de millor, el qual no és possible amb Hill Climbing. Per això, sense aquest oeperador no podríem explorar tot l'espai. El factor de ramificació per a swap en el pitjor dels casos serà el nombre de parelles úniques de paquets:

$$\text{Factor de ramificació swap} = \frac{n \times (n - 1)}{2}$$

3.2.3 *swap múltiple*

En aquest cas el que es fa és substituir un sol paquet per més d'un, concretament hi ha l'opció de fer-ho per 2, 3 o 4. Per fer-ho, es basarà en qual la suma del pes de tots els paquets canviats han de ser més petita que el pes del paquet. Gràcies a això podrem explorar noves solucions, sobretot amb el primer generador de solucions inicials, pel fet que aquest assigna en funció del preu, no del pes com el segon generador. Per a aquest operador hem de calcular totes les possibles combinacions úniques de x paquets, $x \in \{2, 3, 4\}$, multiplicat pel nombre de paquets. El factor de ramificació resultant és:

$$\text{Factor de ramificació swap múltiple} \approx \sum_{i=2}^{4} n \times \binom{n}{i}$$

3.3 Generació d'accions

Aquest apartat del codi consisteix en la generació d'accions on creem totes les possibles solucions que ens ofereixen els nostres operadors. El que fa és, primer de tot, explorar quines accions es poden dur a terme. En aquest pas també es fan totes les comprovacions per tal que es compleixin totes les condicions quan apliquem els operadors. Això ens permet que, amb els operadors, ens moguem sempre per l'espai de solucions.

Trobem una altra funció anomenada *generate_one_action*, la qual utilitzarem quan vulguem aplicar el Simulated Annealing. El que fa aquest és molt similar a l'anterior: aplica les operacions amb els operadors, però no sense abans fer les comprovacions necessàries per tal que això no ens porti a no-solucions. També cal remarcar que aquesta funció retorna un sol possible moviment escollit totalment a l'atzar. Això es degut a la definició de l'algoritme de Simulated Annealing.

A més a més, en aquest apartat també és on es pot controlar la selecció d'operadors. El conjunt que vulguem utilitzar es passarà com un paràmetre del problema, i a dins d'aquest codi de generació d'accions es faran les comprovacions per veure quins operadors es poden utilitzar i quins no.

3.4 Heurístiques

El primer què tal tenir en compte a l'hora d'elaborar les heurístiques és què demana l'enunciat. En aquest cas demana dos casos diferents: un on es minimitzen costos (o, el que seria el mateix, es minimitzen els costos) i una altra on es demana el mateix afegint la maximització també de la felicitat. Cal mencionar també que vam implementar una funció heurística basada en l'entropia, tal com s'havia comentat amb el *bin packing*, però a l'hora de fer els experiments, les dues que hem utilitzat al final ens proporcionaven resultats molt millors.

3.4.1 Primera heurística: maximitzar benefici

Per tal de fer aquesta heurística vam fer una funció auxiliar que serveix per calcular el cost d'una solució. El que fa, bàsicament, és recórrer cada paquet i anar multiplicant el pes d'aquest pel preu per quilogram associat a l'oferta on ha estat assignat, el resultat de cada una d'aquestes operacions és a sumat a una variable. També es suma el cost de l'emmagatzematge, si és necessari.

El que fa aquesta primera heurística és, per tant, anar calculant el cost de cada solució a la qual s'arriba amb els operadors per intentar minimitzar-lo.

3.4.2 Segona heurística: maximitzar benefici i felicitat

En aquesta heurística vam utilitzar dues funcions auxiliars. La primera és la que hem comentat anteriorment i la segona és una que serveix per comptar quants dies abans s'han entregat els paquets, és a dir, per quantificar la felicitat. Aquesta segona la vam elaborar de la següent manera: es recorren tots els paquets i es comprova si han sigut assignats a una oferta que s'entrega abans de la seva prioritat, i si és així, quants dies. Aquests dies es van sumant a una variable i és el que retorna al final la funció.

En aquest cas, el que fa l'heurística és calcular el cost i restar-li la felicitat. Ara bé, com tenen unitats diferents, és important assignar un pes a cada un.

4 Experimentació

Un cop ja teníem tots els elements preparats, ja podíem passar a l'experimentació. El que vam fer en aquest apartat va ser executar els algoritmes canviant els operadors, mètode de generació inicial, etc., i recol·lectant les dades més interessants en taules i gràfics per veure quina combinació ens proporcionaria millors resultats.

A més a més, el nostre objectiu era aconseguir un codi que estigués totalment automatitzat, de manera que es puguin triar els paràmetres per a cada experiment des de la mateixa terminal durant l'execució del programa. També, com *VisualStudio*, la plataforma que hem utilitzat per programar, a vegades dona problemes la visualització de gràfic de *matplotlib* utilitzant la funció *plt.show()*, ha sigut necessari desar cada un dels gràfics. Per això, també hem dissenyat un menú que permet a l'usuari seleccionar quins gràfics voldrà descarregar.

4.1 Primer experiment: Operadors

4.1.1 Explicació i hipòtesi

El primer experiment consistia a buscar quina combinació d'operadors és la millor per tal de dur a terme els experiments. Per fer-ho, seleccionarem un generador de solucions inicials dels dos i farem les proves amb aquest, utilitzant 50 paquets i una proporció de pes d'1,2. En el nostre cas, vam seleccionar el segon generador, ja que proporciona estats que, segons la primera heurística, estan més lluny del que considera una bona solució. Aquestes proves consistiran en, tenint una llista de 10 *seeds* executar l'algoritme de Hill Climbing 10 vegades amb cada una.

Pel que nosaltres creiem, el conjunt d'operadors que millors resultats tindrà serà el de *Swap* i *Move*, pel fet que ens permetrà arribar a una bona solució inicial, i trigant menys que si ho féssim amb tots 3.

4.1.2 Experiments

Primer de tot el programa et demana quins gràfics voldràs obtenir i, a continuació, hi ha una llista amb 10 *seeds* aleatòries, on cada una s'executa amb l'algoritme Hill Climbing un total de 10 vegades. Per a cada *seed*, es guarda la mitjana de les 10 iteracions de diferents dades: passos que ha fet, el temps que ha trigat, el cost final, etc. Abans de tot, per tal d'entendre els gràfics cal saber que el número 1 equival a *move*, el 2 a *swap* i el 3 al *swap múltiple*, i la resta de nombres són combinacions. Per exemple, el cas amb *move* i *swap* es representa amb un 12.

Primer utilitzarem els gràfics 1, 2 i 3, que són gràfics de línies on es veuen els resultats del temps d'execució, cost final i número de passos per a les 10 *seeds* i per a les 7 possibles combinacions d'operadors. Començarem amb el gràfic que mostra el cost final de cada combinació d'operadors. Ho veiem a la figura 1.

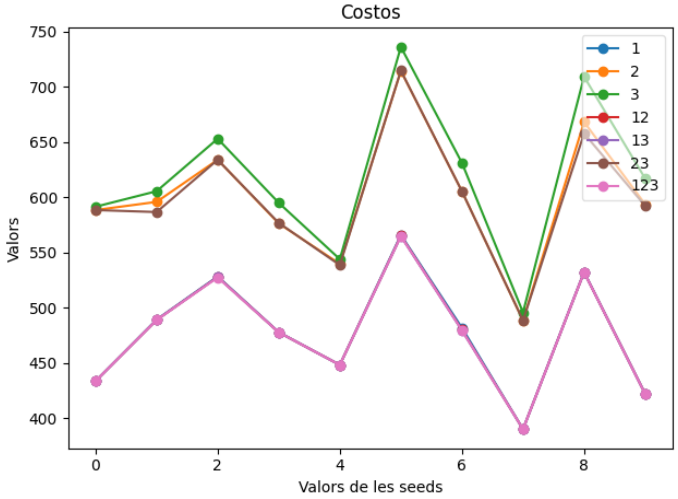


Figura 1: Cost final dels diferents operadors

Primer de tot cal mencionar que encara que no es pugui apreciar a primera vista, les línies que no apareixen al gràfic és perquè estan sobreposades, totes amb la mateixa forma que la rosa. Veiem que les línies verdes, taronja i marró, que tenen una solució final pitjor que la resta, tenen totes una cosa en comú: no tenen l'operador *Move*, així que descartarem aquestes combinacions d'operadors. Per veure això amb més claredat veiem la figura 2

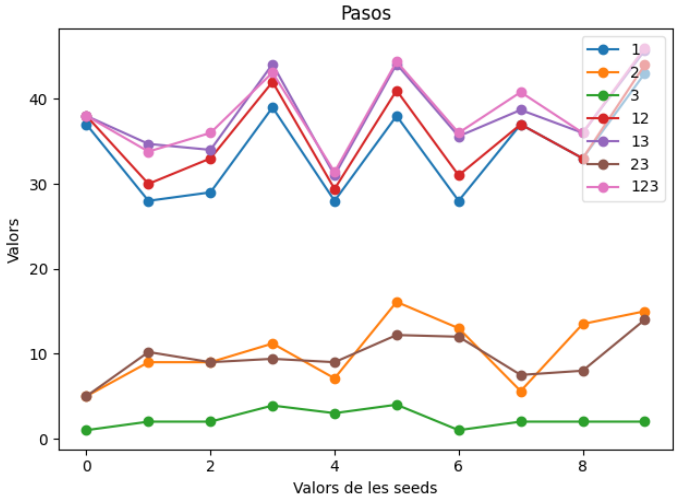


Figura 2: Número de pasos dels diferents operadors

Observem clarament com, aquelles combinacions que abans ens havien donat pitjor solucions finals, ara són les que tenen un nombre de passos més baix. Això ens pot fer pensar que sense

l'operador *Move*, no s'explora tot l'espai de solucions. Per tant, tenint 4 combinacions d'operadors restants, mirarem el gràfic de la figura 3.

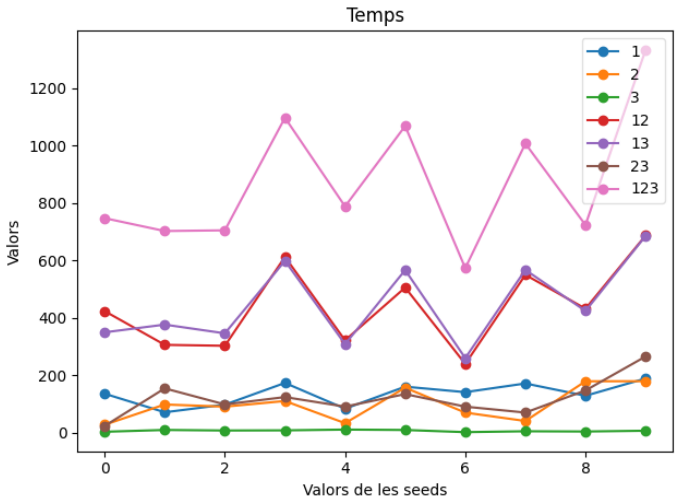


Figura 3: Temps (en ms) d'execució dels diferents operadors

En aquest últim gràfic veiem els temps d'execució en mil·lisegons per cada combinació d'operadors. El primer que veiem és que n'hi ha una que triga molt més que la resta, i aquesta és la que inclou tots els operadors, així que aquesta va quedar pràcticament descartada. A continuació veiem un parell de combinació amb un temps mitjà, que són les dues combinacions del *Move* amb els altres operadors. Finalment, veiem un grup de quatre línies que, si bé és veritat que tres d'elles no ens interessen perquè les hem descartat anteriorment, n'hi ha una que sí: la 1, és a dir, la del *Move* sol.

Si bé és veritat que podria semblar que l'opció 1 és la millor definitivament, cal tenir en compte que a l'estar treballant amb gràfics i tenir unes escales altes, sobretot amb la del cost final, aqeustes conclusions seran poc precisses. Llavors, el que farem serà analitzar les taules que genera el nostre mateix codi amb l'operador *Move* i també les seves combinacions amb els altres dos, ja que això ens permetrà veure les dades amb més detall.

Seed	T. assig (ms)	Pasos	Cost inicial (€)	Cost (€)	Diferencia (€)	Felicitat	Temps (ms)
1234	0.37	37.0	591.48	433.8	157.68	0.0	135.78
5678	0.15	28.0	605.81	489.4	116.4	0.0	59.72
1357	0.14	29.0	653.0	528.4	124.6	0.0	76.89
2468	0.15	39.0	595.76	478.01	117.75	0.0	135.63
1122	0.15	28.0	544.29	448.4	95.89	0.0	68.58
3344	0.15	38.0	736.24	565.95	170.28	0.0	124.24
5566	0.16	28.0	631.14	481.95	149.2	0.0	78.62
7788	0.14	37.0	495.98	390.29	105.69	0.0	96.93
9910	0.15	33.0	709.73	532.47	177.26	0.0	73.21
7777	0.13	43.0	617.73	422.35	195.39	0.0	103.39

Figura 4: Taula amb les dades de l'operador Move

Seed	T. assig (ms)	Pasos	Cost inicial (€)	Cost (€)	Diferencia (€)	Felicitat	Temps (ms)
1234	0.14	38.0	591.48	433.78	157.7	0.0	357.83
5678	0.15	30.0	605.81	489.06	116.75	0.0	275.92
1357	0.15	33.0	653.0	527.95	125.05	0.0	321.45
2468	0.17	42.0	595.76	477.87	117.89	0.0	587.67
1122	0.14	29.2	544.29	448.4	95.89	0.0	333.5
3344	0.15	41.0	736.24	565.2	171.03	0.0	590.87
5566	0.26	31.0	631.14	480.12	151.03	0.0	321.16
7788	0.13	37.0	495.98	390.29	105.69	0.0	558.2
9910	0.24	33.0	709.73	532.23	177.5	0.0	376.91
7777	0.13	44.0	617.73	422.21	195.52	0.0	764.13

Figura 5: Taula amb les dades de l'operador Move i Swap

Seed	T. assig (ms)	Pasos	Cost inicial (€)	Cost (€)	Diferencia (€)	Felicitat	Temps (ms)
1234	0.15	38.0	591.48	433.75	157.73	0.0	463.48
5678	0.3	33.5	605.81	489.05	116.76	0.0	438.01
1357	0.26	34.0	653.0	527.86	125.14	0.0	330.44
2468	0.15	44.7	595.76	477.87	117.89	0.0	535.08
1122	0.15	31.0	544.29	448.4	95.89	0.0	302.5
3344	0.15	44.4	736.24	564.86	171.38	0.0	450.02
5566	0.15	35.5	631.14	479.68	151.46	0.0	245.41
7788	0.14	41.2	495.98	390.29	105.69	0.0	448.05
9910	0.15	36.0	709.73	532.23	177.5	0.0	308.54
7777	0.13	45.9	617.73	422.16	195.57	0.0	577.85

Figura 6: Taula amb les dades de l'operador Move i Swap múltiple

4.1.3 Resultats

Gràcies a aquestes taules podem veure com, efectivament, els costos de les solucions obtingudes per a cada una de les combinacions és pràcticament el mateix. Llavors, tenint en compte que les diferències de cost no són significatives, ens basarem únicament en el temps d'execució. Tal com hem observat abans als gràfics, i com podem veure ara a les taules, l'operador *Move* sol té un temps d'execució bastant més baix que els altres dos. Per tant, rebutjarem la nostra hipòtesi inicial, ja que l'únic operador que utilitzarem serà el de *Move*

4.2 Segon experiment: Generació solució inicial

4.2.1 Explicació i hipòtesi

Un cop ja sabem quin operador utilitzarem, toca decidir quin és el millor generador de solucions inicials. Per fer-ho, utilitzarem el mateix escenari que abans: primera heurística, 50 paquets i proporció de pes 1,2.

Per la nostra hipòtesi inicial, nosaltres creiem que el millor generador d'estats inicials serà el primer, pel fet que justament intenta aconseguir solucions amb costos baixos, el qual ens permetrà arribar abans a aquestes.

4.2.2 Experiments

Igual que abans, el primer que fa el codi és demanar un input triant quins gràfics es voldrà desar. A continuació, executarem l'algoritme Hill Climbing amb les dues solucions inicials amb 5 *seeds* diferents. En aquest cas, no caldrà fer més d'una iteració per *seed*, ja que, en estar treballant amb generadors pseudoaleatoris, sempre obtindrem els mateixos resultats.

Un cop executem el codi obtindrem els gràfics seleccionats, que poden ser: un gràfic de línies amb el temps de la generació de la solució inicial, un gràfic de línies amb el temps d'execució del Hill Climbing i una taula amb més informació. Començarem analitzant el primer, el veiem a la figura 7. Cal saber que la generació 1 és la primera, on es prioritza el cost, mentre que la 2 és la segona, on es dona més pes a la felicitat.

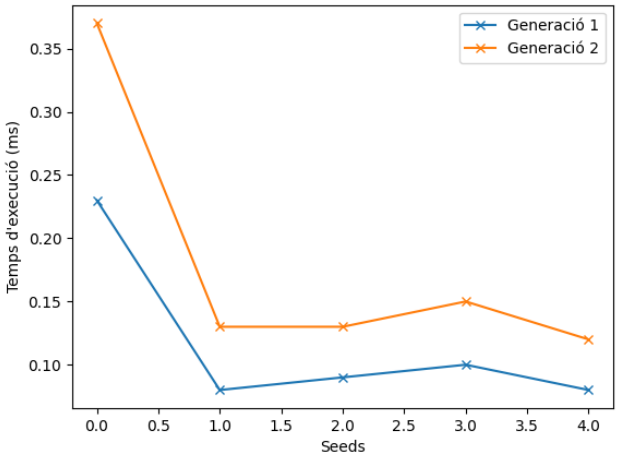


Figura 7: Gràfic de línies amb el temps de la generació de la solució inicial

Veient el gràfic podem afirmar que el generador 1 triga menys en crear les solucions inicials en tots els casos estudiats. Si bé és veritat que és un punt favorable, no és res prou significatiu per a poder treure conclusions. Per aquest motiu, analitzarem la resta de gràfics. El següent és el del temps d'execució de l'algoritme amb una solució inicial o l'altre. El veiem a la figura 8.

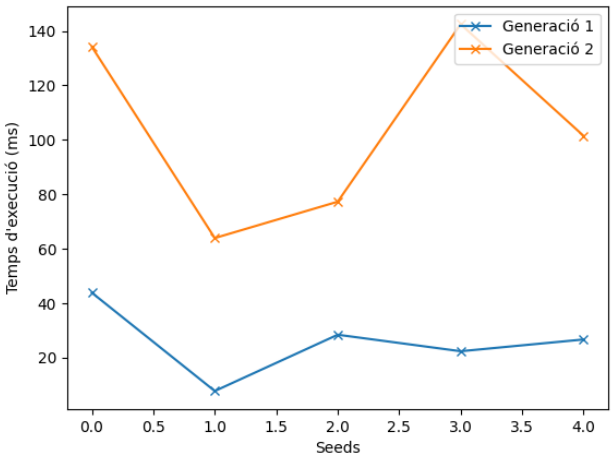


Figura 8: Gràfic de línies amb els temps d'execució amb les solucions inicials de cada generador

Observant aquest segon gràfic podem veure que el temps d'execució del segon generador és molt més alt que el del primer, en aquest cas amb una diferència molt més gran. Podem destacar, per exemple, com a la quarta *seed*, el primer generador triga aproximadament 140 ms i el segon 20 ms, el qual és 7 vegades menys. Per acabar de fer la nostra conclusió, mirarem la taula per tenir més informació. Veiem la figura

Generador solució	Mitjana t. ini.	Mitjana t. ex.	Cost ini. (€)	Cost final (€)	Diferència (€)
Primer	0.12	25.84	450.4	448.64	1.76
Segon	0.18	103.88	544.29	448.4	95.89

Figura 9: Taula amb les dades dels dos generadors d'estats inicials

4.2.3 Resultats

Veient la taula podem confirmar el que havíem vist als gràfics anteriors: la mitjana dels temps per fer la solució inicial amb el primer generador és més baix i el temps d'execució del Hill Climbing és també molt més baix amb el primer, pel fet que, segons aquest primer criteri, la qualitat de les solucions inicials del primer generador és molt millor. Ho podem veure també amb la diferència entre el cost final i l'inicial, on la del primer generador és de menys de 2 i la del segon és de més de 95.

Un cop vistos els resultats, vam decidir que seria millor utilitzar el segon generador, ja que, en estar tan a prop de la solució final amb el primer, el pes del treball dels operadors és molt baix, el qual no ens interessa. Per tant, tornem a rebutjar la nostra hipòtesi inicial.

4.3 Tercer experiment: Paràmetres Simulated Annealing

4.3.1 Explicació i hipòtesi

Podem deduir de primeres que amb lambdes més petites i K més grans obtindrem millors resultats tal com està definida la funció d'acceptació de Simulated Annealing, ja que ens permet explorar més. No obstant el temps d'execució es dispara a mesura que augmentem o disminuim aquests paràmetres. Buscarem un compromís entre expansió de l'algoritme i temps d'execució. El que farem en aquest experiment és executar i veure els resultats de les diferents combinacions de K i Lambda. Treballarem amb K=[1,5,25,125] i lambda [1,0.01,0,001], en tots els experiments posem un límit d'iteracions de 300000 per tal que totes les combinacions convergeixen. Els resultats obtinguts per dues llavors i executant diversos cops cada combinació, ja que aquest algoritme pot variar els resultats de major manera. Obtenim aquests resultats:

4.3.2 Experiments

Aquest experiment tracta d'executar les 12 diferents combinacions 4 cops per cada llavor, 1234 i 5678 en aquest cas. Obtenint resultat de la nostra funció heurística i del temps.

	Lambda		
K	1	0,01	0,001
1	435,38 €	434,34 €	434,17 €
5	437,99 €	435,15 €	434,25 €
25	435,08 €	435,43 €	434,16 €
125	435,83 €	434,79 €	434,07 €

	Lambda		
K	1	0,01	0,001
1	491,35 €	489,31 €	490,57 €
5	492,30 €	491,83 €	491,28 €
25	492,82 €	493,86 €	493,35 €
125	491,23 €	489,57 €	489,59 €

	Lambda		
K	1	0,01	0,001
1	438,07 €	435,15 €	434,87 €
5	435,13 €	434,75 €	434,07 €
25	434,81 €	434,62 €	434,42 €
125	436,06 €	435,57 €	434,63 €

	Lambda		
K	1	0,01	0,001
1	491,11 €	491,75 €	492,49 €
5	492,14 €	493,83 €	489,85 €
25	491,58 €	496,05 €	489,28 €
125	492,29 €	489,30 €	489,53 €

	Lambda		
K	1	0,01	0,001
1	439,76 €	434,45 €	433,95 €
5	436,01 €	435,14 €	434,09 €
25	441,04 €	434,50 €	434,00 €
125	436,14 €	435,79 €	434,44 €

	Lambda		
K	1	0,01	0,001
1	492,60 €	489,40 €	489,29 €
5	493,00 €	494,72 €	492,23 €
25	492,04 €	494,37 €	490,20 €
125	491,94 €	494,08 €	493,01 €

	Lambda		
K	1	0,01	0,001
1	434,50 €	435,06 €	434,21 €
5	435,03 €	435,08 €	434,28 €
25	434,94 €	434,64 €	434,25 €
125	435,50 €	434,25 €	434,30 €

	Lambda		
K	1	0,01	0,001
1	492,38 €	492,19 €	489,25 €
5	490,32 €	492,06 €	489,07 €
25	493,79 €	491,94 €	489,41 €
125	491,24 €	491,34 €	491,74 €

Figura 10: Cost final amb els diferents paràmetres

Podem observar que la nostra teoria és certa, K més altes i Lambdes més petites poden arribar a millor solució. No obstant el temps d'obtenir-les és molt alt. També cal remarcar que les diferències són relativament baixes i el que consistentment més afecta a l'optimització del resultat és una lambda més petita. Les taules de l'esquerra son amb llavor 1234 i el de la dreta 5678. Tenim 4 execucions per

llavor. Tenint en compte que els resultats de Hill Climbing per aquests dos escenaris són de 433.8€ i 489.06€ per 1234 i 5678 respectivament, podem veure que cap arriba millorar-lo, però sí que estan molt a prop. Això podria donar-se gràcies a l'altra randomització dels resultats en SA, ja que podem observar que els resultats són sempre diferents. Els temps d'execució amb un límit alt per tal que totes convergeixin, és molt alt quan les lambdes són petites. A més varia molt entre llavors. Podem deduir de les execucions que segueixen un patró. Per la llavor 1234 amb Lambda: 1 està a prop d'1 segon, amb lambda 0.01 al voltant de minut, 2 minuts i amb lambda 0.001 apunta més cap a 8 o 10 minuts. En canvi, per la llavor 5678 el temps és quatre cops més ràpid. Veiem que els temps depenen molt de les llavors, de què tan a prop de la solució estem i del factor random. Tots els valors de temps cost i paràmetres estaran a un fitxer al zip de l'entrega. On es podran veure algunes execucions aïllades amb límit 200000 on Simulated Annealing millora Hill Climbing però el temps és molt elevat. Aquest experiment demostra que no són consistentment millors i que no sempre Simulated Annealing millora Hill Climbing, tot i que a vegades sí per aquest problema en particular.

4.3.3 Resultats

Dit això, obvservant el grafics la combinació de paràmetres que ens permet fer el balanç entre temps d'execució i la que considerem com a millor elecció és K = 125 i lambda = 0.01. En general sol ser així però hi ha un gran marge de variació. Tenint en compte que en general els millors resultat residiran a la lambda més petita possible i K més gran possible, dins de la randomització dels resultats i evitant un temps d'execució massa alt.

4.4 Quart experiment: Evolució del temps d'execució

4.4.1 Explicacó i hipòtesi

Per a aquest experiment, dividim l'anàlisi en dues parts per observar els efectes de canviar els paràmetres de pes total de les ofertes en relació als preus dels paquets i el nombre de paquets:

1. **Increment progressiu del pes total de les ofertes:** En aquesta primera part, es comença amb un pes de 1.2 per a les ofertes i es va incrementant en intervals de 0.2 fins a observar com impacta en els resultats de l'algorisme. A mesura que augmenta el pes de les ofertes, esperem veure una tendència on el cost baixi, ja que amb més ofertes o amb més pes en cada oferta, en teoria es poden arribar a trobar millors solucions respecte a les quals tenen menys pes.

2. **Augment de la quantitat de paquets:** En la segona part, partim amb una quantitat inicial de 50 paquets i anem incrementant fins a arribar a 150. Això ens permet veure com l'algorisme maneja una major varietat i complexitat en les opcions, ja que augmentar els paquets implica un major cost computacional per calcular les possibles solucions. Podem anticipar que, amb més paquets, l'algorisme tindrà una major capacitat per optimitzar, però també es veurà afectat en termes de temps de càlcul i complexitat de les decisions, ja que es requereix una major exploració.

Aquest experiment permet explorar quins paràmetres poden oferir un equilibri òptim entre la maximització del valor de les ofertes i la complexitat del problema, així com les tendències en els resultats per obtenir una heurística efectiva que tingui en compte tant l'eficiència en el cost com la qualitat de les solucions obtingudes.

4.4.2 Experiments

Per dur a terme aquest experiment utilitzarem el Hill Climbing i una heurística basada únicament en el cost. Utilitzem 10 seeds i realitzem 10 repeticions per a cada cas per a obtenir uns resultats més fiables. A la figura 11 veiem com varien els resultats amb l'increment progressiu en la proporció de pes transportable.

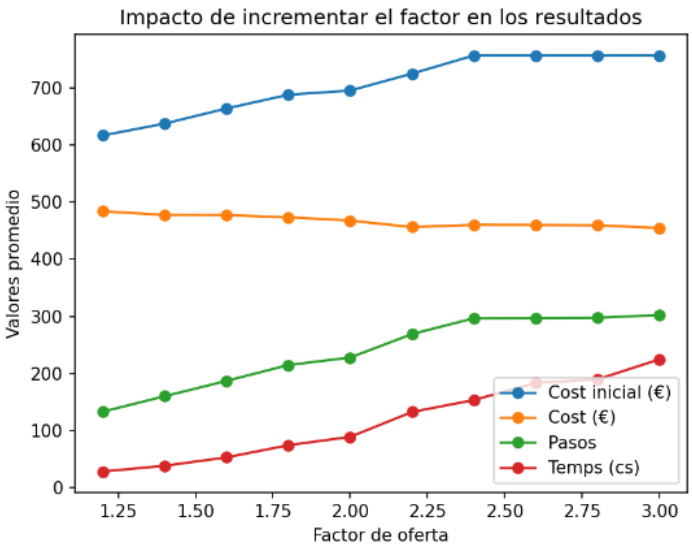


Figura 11: Impacte del factor del pes al resultat

El gràfic mostra quatre mètriques clau per poder observar les tendències en aquest experiment: el cost de la solució inicial, el cost de la solució final, el nombre de passos de l'algorisme i el temps d'execució en centèsimes de segon.

1. **Cost Inicial:**Veiem com el cost inicial augmenta gradualment a mesura que anem incrementant el factor, com sabem, la solució inicial es basa a col·locar paquets en ofertes per prioritat sense tenir en compte el cost, perquè és lògic que el cost augmenti cada cop més el cost inicial.

   En els valors més alts veiem que el cost inicial es manté constant. Això pot ser degut al fet que amb un factor tan alt, el programa ja és capaç de posar els 50 paquets en molt poques ofertes bones i que tinguin molt de pes, fent que el cost inicial es quedi igual.

2. **Cost Final:** En augmentar la proporció de pes transportable el cost final va disminuint gradualment de forma lleugera. En tenir més espai i/o més ofertes, el programa és capaç de trobar configuracions on el cost sigui cada cop menor.

3. **Pasos:** El nombre de passos augmenta fins a arribar a un factor de 2.4, on els valors es comencen a estabilitzar. L'increment en el nombre de passos es deu al fet que, com es pot veure en el cost inicial, la solució inicial es guia per la prioritat dels paquets. Això provoca que, a mesura que el factor de pes en les ofertes augmenta, el procés requereixi més passos per ajustar-se a aquesta priorització.

   L'estabilització dels passos en els valors més alts dels factors podria estar relacionada amb el mateix fenomen que es produeix en el cost inicial, ja que ocorre en el mateix punt. Això té sentit perquè, si el cost inicial és el mateix (tenint una solució inicial semblant per a cada cas), el nombre de passos també serà igual.

4. **Temps:** Si el factor de pes augmenta, això significa que els paquets tindran més opcions a on moure's, el que resulta amb més opcions que valorar pel programa. Això ho veiem traduït en la gràfica en un major temps cada cop que el factor de pes augmenta.

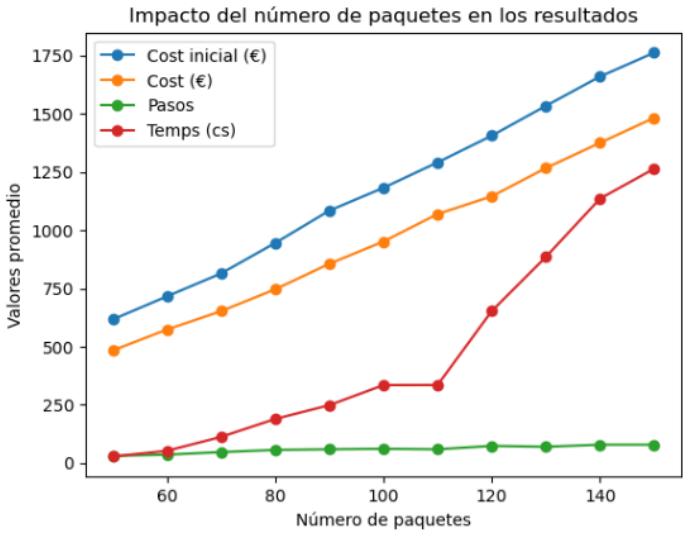


Figura 12: Imapcte del número de paquets al resultat

A la figura 12 observem representada la mateixa informació que a l'anterior, però aquesta vegada veient com varien els resultats en funció del nombre de paquets utilitzats. A continuació, comentem cada una per separat:

1. **Cost inicial i cost final:** Com es pot observar a la gràfica, tant el cost inicial com el cost final augmenten de forma consistent a mesura que s'incrementa el nombre de paquets. Aquest augment en ambdues variables es deu al fet que cada paquet comporta un cost d'enviament, i, per tant, un augment en el nombre de paquets comporta un increment proporcional en el cost total.

   D'altra banda, també es pot observar que la diferència entre el cost inicial i el cost final es fa més gran a mesura que augmenta el nombre de paquets. Això es deu al fet que la solució inicial

no té en compte l'optimització del cost, de manera que hi ha més marge per a la millora a mesura que augmenta el nombre de paquets.

2. **Pasos:** Tot i l'increment del nombre de paquets, el nombre de passos necessaris per arribar a una solució es manté relativament estable. Això pot indicar que l'algorisme troba solucions de manera eficient en el mateix nombre de moviments, fins i tot quan l'espai de cerca s'amplia.

   El normal seria pensar que contra més paquets, més passos hem de fer per arribar a una solució final, però contra més paquets tinguem, menys opcions de moure paquets tindrem, ja que hi haurà menys espai lliure. Aquest fet fa que els passos es mantingui gairebé constants.

3. **Temps:** El temps d'execució augmenta notablement amb el nombre de paquets. Aquest comportament és esperat, ja que un nombre més gran de paquets implica un espai de cerca més gran per explorar, fent que l'algorisme requereixi més temps per trobar solucions òptimes.

4.4.3 Resultats

Analitzant els resultats veiem unes tendències clares en els resultats: en augmentar el factor del pes el cost inicial augmenta, però el cost final disminueix, l'espai de cerca augmenta fent que el temps sigui més alt i la solució inicial fa que cada cop necessitem més passos per aconseguir la solució final. En l'altre experiment veiem que contra més paquets posem, més diferents seran els costos inicials respecte als costos finals. El temps d'execució augmenta gradualment pel fet que té més opcions que valorar i els passos es mantenen iguals.
Pot arribar un punt on només amb el move no puguem realitzar cap acció, són casos extrems, però afegint l'operador de swap podem prevenir aquest problema, a canvi d'un temps d'execució bastant més alt.

4.5 Cinquè experiment: Anàlisi

Analitzant els resultats de l'experiment en què s'augmenta la proporció de pes de les ofertes, s'observa que el cost inicial de transport i emmagatzematge augmenta inicialment fins a un punt en què es manté estable, mentre que el cost final disminueix gradualment. Això indica que augmentar la proporció de pes transportable pot ser beneficiós per trobar solucions més econòmiques, ja que permet a l'algorisme disposar de més opcions per optimitzar.

Tot i que el cost de transport i emmagatzematge es redueix amb més pes disponible a les ofertes, l'augment del temps d'execució i el nombre de càlculs necessaris pot comportar un cost computacional significatiu. Per tant, val la pena incrementar el nombre d'ofertes o el pes disponible fins a un cert punt, però cal tenir en compte que l'eficiència pot disminuir si l'augment en el temps de càlcul supera el benefici obtingut en la reducció del cost final.

4.6 Sisè experiment: Felicitat amb Hill climbing

4.6.1 Explicació i hipòtesi

En aquest experiment, volem estudiar com l'augment del pes assignat a la variable happiness pot influir en els resultats obtinguts en el procés de Hill Climbing. En un primer moment, esperarem que en donar-li més importància (augmentant el pes) el model pugui maximitzar millor la satisfacció en els resultats finals, però possiblement amb un increment en el cost o en els passos necessaris per arribar a una solució òptima.

En aquest experiment, realitzarem diverses execucions utilitzant valors de pes per a happiness en l'interval de [0, 5], amb increments de 0.5. Això ens permetrà identificar una tendència clara en el comportament del model segons aquest factor, i veure fins a quin punt augmentar la importància de happiness impacta els altres indicadors (cost, temps i nombre de passos) de manera favorable o desfavorable.

4.6.2 Experiments

Figura 13: Increment del pes de happiness

Com no acaben de tallar-se, decidim aumentar a 1 els intervals i que el pes de 'happiness' arribi fins a 10.

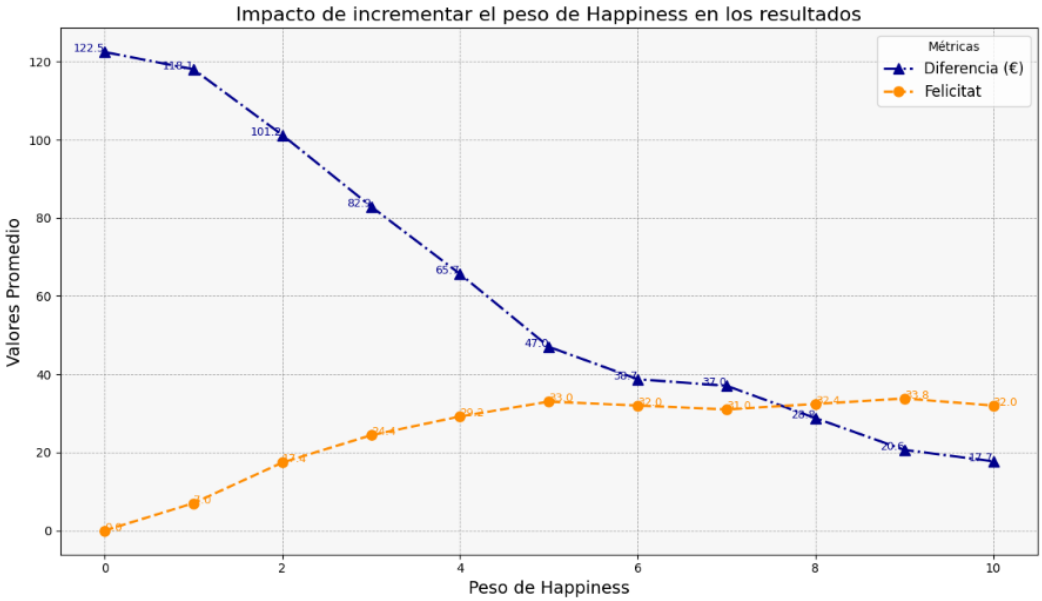


Figura 14: Increment del pes de happiness amb nous pesos

Podem observar que la nostra hipòtesi inicial es confirma en els resultats obtinguts: augmentar el pes de 'happiness' provoca una disminució progressiva en la diferència (€), és a dir, cada cop troba una solució més cara a mesura que el pes incrementa, indicant una tendencia a clara en els resultats. En canvi, la felicitat (felicitat mitjana obtinguda) augmenta moderadament a mesura que augmenta el pes assignat a 'happiness' fins a un punt on els resultats es mantenen bastant similars, inclús en algun punt, disminueix una mica el 'happiness'. A continuació, veiem una taula amb els temps d'execució:

Peso de Happiness	0	1	2	3	4	5	6	7	8	9	10
Tiempo Promedio (s)	0.502	0.7	0.696	0.632	0.638	0.57	0.442	0.54	0.326	0.322	0.276

Figura 15: Temps d'execució en funció del pes de la felicitat

Observant els temps d'execució, veiem que contra més valor li donem a 'happiness' menys temps triga el programa.

4.6.3 Resultats

Existeix una relació inversa entre aquestes dues variables quan assignem més importància a la felicitat: l'algorisme busca solucions que maximitzin 'happiness', però ho fa a costa de sacrificar en part l'eficiència en el cost. No hi ha un pes a 'happiness' que ens doni una solució perfecta, sinó que dependrà de quin paràmetre prioritzem en funció dels objectius del problema.

Si volem una heurística que tingui un equilibri entre 'happiness' i cost, hauríem d'escollir un pes entre el 4 i el 6, on el pes no ha augmentat tant, i, en canvi, la felicitat no augmenta gaire més amb pesos més alts.

El fet que el temps d'execució disminueixi contra més augmentem el pes de 'happiness' ho podem relacionar a què la solució inicial comença amb una 'happiness' molt alta, perquè assignem els paquets per prioritat tenint en compte les restriccions del problema.

Vistos els resultats, podríem explorar més opcions de configuracions en una heurística on es tingui en compte la felicitat dels clients, com per exemple, realitzar un ràtio entre el cost i el 'happiness', veient que un el volem maximitzar (happiness) i un altre el volem minimitzar (cost).

4.7 Setè experiment: Felicitat en Simulated Annealing

4.7.1 Explicació i hipòtesi

En aquest experiment realitzarem el mateix procediment que en el anterior pero amb Simulated Annealing i els paràmetres que hem escollit amb anterioritat. Mirarem com afecta el pes que li donem a la felicitat en l'heurística que combina el cost i aquesta mateixa. Les condicions son les mateixes amb una petita diferència, executarem per 5 llavors 5 cops cada llavor per cada pes que li donem a la felicitat en la nostra nova funció heurística. Primer ho pujarem de 0.5 en 0.5 fins a 5 i després de 1 en 1 fins 10. La nostra hipòtesi, és que es comportarà semblant a Hill Climbing, és a dir, a mesura que augmentem el pes de la felicitat el cost aumentarà o el que és el mateix què la diferència entre cost inicial i cost final sera més petita. Realitzem aquest dos experiments i obtenim aquestes gràfiques. També calcularem el temps d'execució mitjà en cada diferent pes de la felicitat .

4.7.2 Experiments

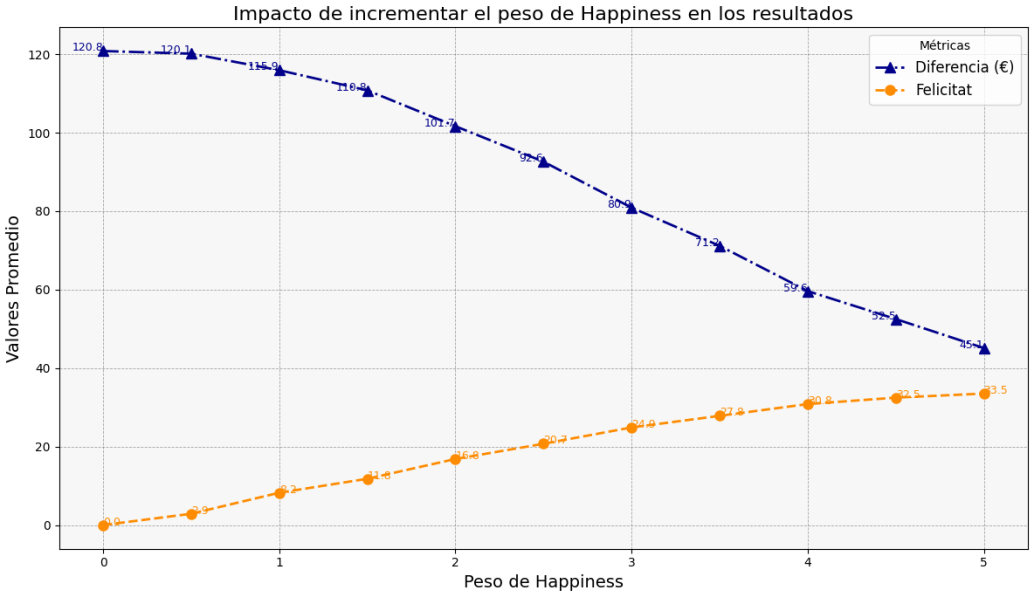


Figura 16: Relació diferència preus i felicitat de 0.5 en 0.5

Veiem la tendència inversament proporcional, però no acaben de tallar-se, per això decidim arribar més lluny amb el següent gràfic.

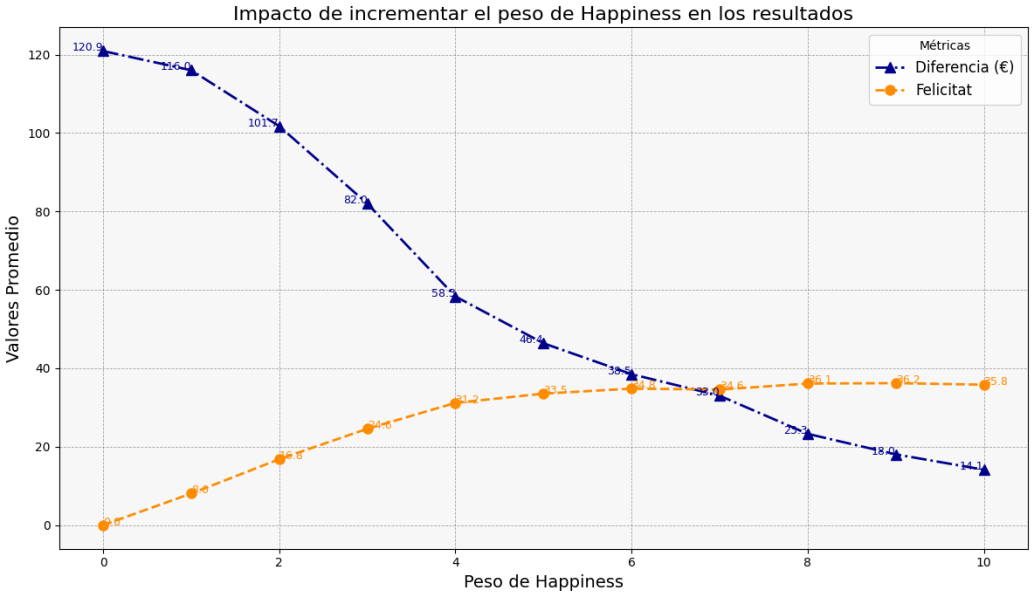


Figura 17: Relació diferència preus i felicitat de 1 en 1

Aquí veiem més clara la tendència.

Peso de Happiness	0	1	2	3	4	5	6	7	8	9	10
Tiempo Promedio (s)	102.25	126.9	126.64	126.695	149.505	312.855	292.895	333.1	326.955	350.56	338.08500000000004

Figura 18: Temps d'execucio mitjà en cada pes

4.7.3 Resultats

Un a cosa a remarcar d'aquest experiment és el temps d'execució elevat que té. Per altra banda veiem les mateixes tendències que en Hill Climbing, la diferència baixa a mesura que augmenta el pes de la felicitat. També podem observar que el ritme és semblant que a Hill Climbing o que inclús baixa la diferència més ràpid. Pel que fa al temps, a comparació de l'altre algoritme de cerca el temps va augmentant a mesura que pujem el pes de la felicitat, que és just al revés en aquest problema. El que és important aquí no són el temps obtinguts en si, sinó la tendència a l'alça que veiem. Això pot ser ocasionat per l'ampliació de l'espai de solucions a l'afegir la felicitat amb més pes o simplement per la diferència de acceptació de solucions que tenen els dos algoritmes. L'augment de pes podria arribar a provocar un augment d'energia, el que produirà un augment de temps de búsqueda. Tot això en aquest problema concret. Arribem a aquestes conclusions o suposicions per les dades que hem obtingut.

4.8 Vuitè experiment

En el vuitè experiment es demana deduir com canviaran les solucions si augmentem o disminuim el cost d'emmagatzematge dels paquets, que fins ara era de 0,25€/quilogram. Cal recordar que aquest cost s'aplicarà en funció dels dies que passin fins que el paquet és entregat al seu respectiu camió, així que això no dependrà de les prioritats de cada paquet, si no que dependrà de les assignacions. En concret, els paquets assignats a camions que arribin el dia 3 o 4 se'ls afegirà un cost de 0.25€ multiplicat pel seu pes, mentre que si estan assignats a camions del dia 5, hauran de passar dos dies al magatzem, suposant un cost afegit de 0.5€ multiplicat pel seu pes.

Tenint això en compte, si augmentem aquest cost suposarà que els paquets enviats el primer i el segon dia tindran exactamentel mateix preu, mentre que la resta veuran com el seu preu augmentaran significativament, pel fet que l'afegit al cost per l'emmagatzematge serà més alt. En comptes, si disminuim aquest cost, el preu dels paquets entregats en 3 dies o més disminuirà significativament.

Per tant, pel que fa al criteri del cost, si augmentem el cost d'emmagatzematge segurament farà que les primeres ofertes s'emplenin més que anteriorment, pel fet que, tot i que les primeres ofertes tenen preus més elevats, hi haurà casos on serà més barat entregar algun paquet a una oferta ante-

rior per tal d'estalviar-nos el cost d'emmagatzematge. Si, a més a més, tenim en compte la segona heurística, on la felicitat també és un factor important, llavors veurem com els paquets es decantaran encara més per les primeres ofertes per dos motius: menor cost d'emmagtazematge i major felicitat.

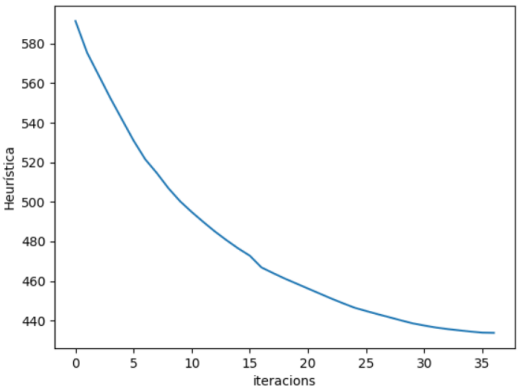
Per altra banda, si disminuïm aquest cost d'emmagatzematge, i tenint només en compte el criteri del cost, passarà el contrari: segurament els paquets tendiran a anar a les últimes ofertes que puguin amb les seves prioritats. Si també tenim en compte la felicitat, segurament es trobarà una solució equilibrada: el cost voldrà assignar els paquets el més tard possible al ser menys costós i la felicitat voldrà posar-los a les primeres ofertes, per tal de maximitzar-la. Òbviament, si es decanta més cap a un costat o cap a l'altre, dependrà del pes que li posem a la felicitat.

4.9 Experiment extra: Personalitzat

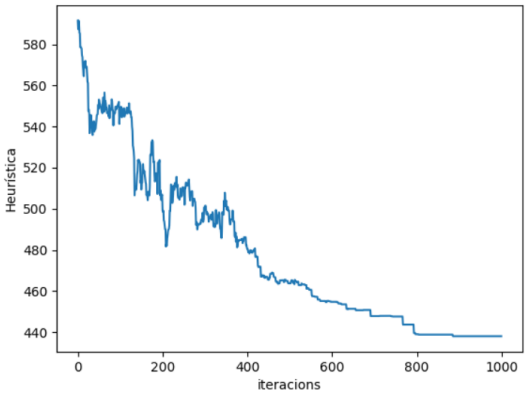
A més a més, un cop acabats tots els experiments, vam decidir afegir al nostre fitxer un últim experiment. Aquest serveix per fer experiments personalitzats, és a dir, podent definir cada un dels paràmetres un mateix. Això ens va servir, a més a més, per fer l'experiment demanat per l'obtenció del punt extra. Per altra banda, també ens ha servit per veure altres casos concrets i generar algun gràfic, com els que es troben a l'apartat següent.

5 Comparació entre resultats de Hill Climbing i Simulated Annealing

En aquest problema en particular hem arribat a la conclusió que Hill Climbing pot arribar a ser més adient que Simulated Annealing. Ja sigui pel temps de còmput o perquè els resultats amb el segon, si arriben a millorar solució, són considerablement molt poc millors. També es pot comentar que Hill Climbing sempre dona el mateix resultat en un mateix escenari o la gran majoria de vegades, en canvi, Simulated Annealing sempre arriba a un punt diferent a causa de la randomització. El punt positiu d'això és que ho pot arribar a millorar, però com hem comentat, molt poc en aquest problema en particular. També hem detectat que la diferència de resultats per cada combinació de paràmetres, fet a l'experiment número 3 de l'apartat anterior, és relativament baixa i que hi ha gran variabilitat en quant quina és la millor de totes. Tot i això, es pot veure alguna tendència.



(a) Evolució dels valors de l'heurística amb Hill Climbing



(b) Evolució dels valors de l'heurística amb Simulated Annealing)

Figura 19

A més a més, utilitzant el codi de l'experiment extra, vam poder generar els gràfics de les heurístiques utilitzant Hill Climbing i Simulated Anealing, tots dos a l'escenari de 50 paquets i 1234 de *seed*. Podem observar els resultats a la figura 19. Per una banda, veiem el gràfic de Hill Climbing, on observem com sempre busca estats on el valor de l'heurística és menor, per això sempre decreix. Veiem, també, que a les primeres iteracions el valor de l'heurística disminueix més ràpidament que a les últimes. Passant ara al Simulated Anealing, observem com la seva aleatorietat per escollir estats pitjors provoca que no sempre decreixi. Ara bé, es veu com, a partir de les 500 iteracions, aproximadament, la temperatura ja és molt baixa i només selecciona solucions millors.

Aquesta comparació ha sigut molt interessant per entendre millor aquests dos algoritmes de cerca local, ja que veure com evoluciona el valor de l'heurística al llarg de les iteracions permet visualitzar millor el seu comportament.

6 Resultats i conclusions

Aquest treball ha consistit en la resolució d'un problema de logística d'una situació real. Per fer-ho, ens hem basat en dos algoritmes de cerca local: Hill Climbing i Simulated Anealing. El seu funcionament, de manera molt senzilla i resumida, consisteix a explorar tot l'espai de solucions mitjançant operadors per trobar la millor i guiant-se amb una funció heurística. Ara bé, aquests algoritmes compten amb un gran nombre de paràmetres que cal triar curosament per tal de trobar la millor solució i en el mínim temps.

Vam començar utilitzant Hill Climbing. El primer que cal decidir són els operadors que s'utilitzaran. En el nostre cas vam decidir quedar-nos només amb el de *Move*, ja que obtenia els millors resultats junt amb altres combinacions, però amb el temps més baix amb diferència. Gràcies al següent experiment, vam poder triar també quin generador de solucions utilitzaríem. L'elecció final va ser el segon generador, ja que obté solucions que, segons el criteri del cost, estan lluny de ser bones, així que posem més pes als operadors, el qual és més interessant a l'hora de fer experiments.

Passant ara al Simulated Anealing, quan s'executa aquest algoritme és necessari introduir alguns paràmetres, els quals vam determinar amb el tercer experiment. A l'hora de fer això, cal tenir en compte les solucions obtingudes, però també el temps d'execució. Per tant, els paràmetres que aconseguien un balanç en aquests dos aspectes eren K = 125 i lambda = 0.01.

Fins aquest punt, havíem estat treballant només amb el criteri del cost, però ara afegirem també la felicitat. Com es tracta de dues unitats diferents vam fer un experiment per veure quin pes donar-li a la felicitat. Vam arribar a la conclusió que, per tal de tenir un equilibri entre cost i felicitat, el millor pes seria un valor entre 4 i 6. Després, vam executar-ho amb Simulated Anealing i vam obtenir uns resultats molt similars.

Gràcies a aquests experiments vam poder trobar tota la informació necessària per resoldre el problema plantejat en un inici de la millor manera, com ara quin algoritme utilitzar, quins operadors o quin generador de solucions inicials, entre altres.