

Intel·ligència Artificial: Pràctica de Cerca Local

Guillem Cabré, Carla Cordero, Hannah Röber

Curs 2024-25, Quatrimestre de tardor

Continguts

1	Part Descriptiva	2
1.1	Descripció del problema	2
1.2	Elements del problema	2
1.3	Definició solució	2
1.4	Espai de cerca	2
1.5	Metodologia de resolució	3
1.6	Implementació de l'Estat	3
1.7	Operadors	4
1.8	Generador de Solucions Inicials	4
1.8.1	Entrega just al dia	4
1.8.2	Maximitzar la felicitat i minimitzar els costos	5
1.9	Funció Heurística	5
1.10	Resultats	6
2	Part Experimental	7
2.1	Experiment 1	7
2.2	Experiment 2	8
2.3	Experiment 3	10
2.4	Experiment 4	10
2.4.1	Proporció del pes transportable	11
2.4.2	Nombre de paquets	11
2.5	Experiment 5	12
2.6	Experiment 6: Influència de la ponderació de la felicitat	12
2.7	Experiment 7	14
2.8	Experiment 8: Com canviarien les solucions si variem el preu d'emmagatzematge a l'alça o a la baixa?	14
2.8.1	Disminució preu emmagatzematge	14
2.9	Experiment 9, especial	14
3	Conclusions	16
4	Treball d'Innovació: DeepVariant	17
4.1	Tema	17
4.2	Responsabilitats d'equip	17
4.3	Fonts	17
4.4	Dificultats	17

1 Part Descriptiva

1.1 Descripció del problema

La companyia fictícia *Ázamon* ha d'optimitzar els seus enviaments diaris de n paquets a una ciutat, considerant diversos factors. Cada paquet té un pes w_i i una prioritat p_i , que defineix el termini màxim per ser entregat. L'empresa rep ofertes diàries de diverses companyies de transport, i el repte és trobar la millor manera de distribuir els paquets entre aquestes ofertes, minimitzant els costos i maximitzant la satisfacció dels clients.

Els costos inclouen tant el transport, amb preus per quilogram que varien segons l'empresa, com l'emmagatzematge dels paquets que no es recullen immediatament. La felicitat dels clients augmenta si els paquets arriben abans de la data límit prevista. Per tant, cal equilibrar l'eficiència en costos amb la rapidesa en el lliurament per maximitzar la satisfacció del client.

1.2 Elements del problema

Cada paquet té un pes $w_i \in \{0.5, 1.0, 1.5, \dots, 10.0\}$ kg i una prioritat $p_i \in \{1, 2, 3\}$, que defineix el termini d'entrega: un dia per $p_i = 1$, entre 2 i 3 dies per $p_i = 2$, i entre 4 i 5 dies per $p_i = 3$. Cadascuna d'aquestes prioritats té un cost diferent, $p_i = 1$ val 5 euros, $p_i = 2$ val 3 euros i $p_i = 3$ val 1.5 euros.

Les empreses de transport ofereixen cada dia m opcions amb una capacitat màxima $C_j \in \{5, 10, 15, \dots, 50\}$ kg, un preu per quilogram transportat c_j , i un temps d'entrega $t_j \in \{1, 2, 3, 4, 5\}$ dies. A més, si els paquets no es recullen immediatament, cal assumir un cost d'emmagatzematge de 0.25 euros per quilogram i dia. Els clients es mostren més satisfets amb entregues anticipades, la qual cosa augmenta proporcionalment als dies d'antelació.

1.3 Definició solució

Una solució segons el problema que se'ns ha descrit és una assignació dels paquets a les ofertes de transport del dia apropiades amb les restriccions que la suma dels pesos dels paquets assignats a una oferta no poden superar la seva capacitat màxima i que tots els paquets arribin dins del termini d'entrega que depèn de la seva prioritat.

Formalment, si tenim n paquets i m ofertes de transport, una solució seria una tupla que associa cada paquet p_i a una oferta o_j , tal que $1 \leq i \leq n$ i $1 \leq j \leq m$, amb les restriccions de capacitat i temps respectades.

Aleshores el nostre objectiu serà, mitjançant algorismes, generar una solució que compleixi els requisits mencionats, que serà l'estat final, partint d'un estat inicial que ens permeti arribar a una solució més òptima, en la qual es té en compte criteris de qualitat. En tot moment, també s'intentarà maximitzar la felicitat dels clients.

1.4 Espai de cerca

L'espai de cerca dels nostres algorismes és l'espai de solucions al nostre problema, és a dir, la totalitat de les solucions que compleixen els requisits del nostre problema que s'han esmentat en l'apartat anterior sense importar la seva qualitat donada per uns criteris en formen part. És important conèixer la mida de l'espai de cerca per justificar la utilització de certs algorismes per aquest problema en comptes d'utilitzar un algorisme de força bruta senzill.

La grandària de l'espai de cerca està determinat pel nombre de combinacions possibles d'assignació de paquets a ofertes de transport. Aleshores, assumim que m és el nombre d'ofertes del dia que hi ha i que tenim n paquets a distribuir amb la seva prioritat. La grandària màxima de l'espai de cerca seria de l'ordre de $O(m^n)$, la qual cosa representa totes les combinacions possibles d'assignació de paquets que correspon al cas pitjor on totes les combinacions són vàlides, ja que cada paquet pot ser assignat a

qualsevol de les m ofertes.

No obstant això, aquest número pot reduir-se significativament a causa de les restriccions que impedeixen unes certes assignacions inviables, com assignar paquets a una oferta que no pot transportar el pes total o assignar paquets d'alta prioritat a una oferta de llarg temps de lliurament. Aquestes restriccions limiten el nombre de combinacions vàlides, però encara així, l'espai de cerca pot ser considerablement gran, especialment quan el nombre de paquets i ofertes augmenta ja que creixerà exponencialment. Aleshores, es complica l'eficiència de l'algorisme de força bruta i, per tant, s'ha d'utilitzar altres algorismes.

1.5 Metodologia de resolució

Per resoldre aquest problema, utilitzarem algorismes de cerca local. En particular, s'han seleccionat els algorismes de *Hill Climbing* i *Simulated Annealing*, que exploraran l'espai de cerca format per totes les assignacions possibles dels paquets a les ofertes de transport.

- L'algorisme de *Hill Climbing* intentarà millorar successivament la solució actual fent petits canvis a l'assignació dels paquets.
- L'algorisme de *Simulated Annealing* permetrà l'acceptació temporal de solucions pitjors, amb l'objectiu d'evitar quedar-se atrapat en òptims locals.

Una de les principals raons per utilitzar algorismes de cerca local és la seva eficàcia per a explorar espais de solucions grans i complexos. Aquests algorismes permeten explorar l'entorn d'una solució inicial i fer petits canvis successius, el que facilita trobar solucions millorades de forma ràpida.

Una de les seves grans avantatges és que no requereixen conèixer tota l'estructura de l'espai de solucions, sinó que operen localment a partir d'una única solució. A més, l'algorisme de *Simulated Annealing* és particularment útil perquè permet escapar dels òptims locals, una característica crucial en problemes amb espais de cerca irregulars o amb molts màxims i mínims locals.

D'aquesta manera, utilitzant la cerca local, podem obtenir solucions raonablement bones en un temps raonable, ajustant paràmetres i comparant diferents configuracions en els experiments.

A través de diversos experiments, es provaran diferents configuracions de paràmetres i es compararan els resultats dels dos algorismes per determinar quin proporciona millors solucions en termes de cost i felicitat.

1.6 Implementació de l'Estat

L'estat del nostre problema s'implementa mitjançant la classe *Estado*, que gestiona l'assignació dels paquets a les diferents ofertes de transport. Aquest estat inclou les següents estructures de dades:

- *Paquetes paquetes*: Conté la llista de paquets que s'han de distribuir, que s'han generat de manera aleatòria per a un problema.
- *Transporte ofertas*: Una llista que guarda les ofertes de transport disponibles, generats a partir dels paquets de manera aleatòria de manera que sempre hi hagi espai per a enviar tots els paquets dins el termini d'entrega.
- *List<Integer> asignaciones*: Una llista que representa les assignacions actuals dels paquets a les ofertes. Cada element de la llista correspon a un paquet. El valor assignat que contenen aquests paquets correspon a l'índex de l'oferta a la qual s'ha assignat (o -1 si no està assignat a cap oferta). És a dir $asignaciones[i] = j$ vol dir que el paquet amb índex i a *paquetes* té assignat la oferta amb índex j a *ofertas*.
- *List<Double> espacioDisponibleOfertas*: Guarda l'espai disponible per a cada oferta de transport, que permet controlar la capacitat restant per a cada una de les ofertes assignades i per saber si un paquet pot ser assignat a una oferta.

- *Integer felicidad*: Aquest valor guarda la felicitat total acumulada, que s'augmenta quan els paquets s'entreguen abans del que estava previst. Obtenim un punt de felicitat per cada dia abans que s'entrega el paquet del dia mínim que s'ha d'entregar segons la prioritat escollida pel client.
- *Double precio*: Aquest valor guarda el preu total que té assignar els paquets a les ofertes segons *asignaciones*, incloent els costos de transport de cada paquet en la seva oferta assignada i els d'emmagatzematge si estan assignats a ofertes que triguen 3 o més dies.

Inicialment, l'estat es configura assignant -1 a cada paquet de la llista *asignaciones*, indicant que no hi ha cap oferta de transport assignada al paquet. A més, *espacioDisponibleOfertas* s'inicialitza amb la capacitat màxima de cada oferta, que posteriorment s'anirà actualitzant quan els paquets es vagin assignant. El preu i la felicitat s'inicialitzen a 0 ja que no hi ha cap assignació fet a cap paquet; al generar la solució inicial i aplicar els operadors ja es van actualitzant conformement.

1.7 Operadors

Per executar els algorismes de cerca local hem definit i implementat un conjunt d'operadors. Per parlar del seu factor de ramificació considereem n el nombre de paquets a assignar i m el nombre de ofertes de transport disponibles per repartir els paquets. Els operadors escollits seran els següents:

- *swapPaquet*($p1$, $p2$): Siguin $p1$ i $p2$ dos paquets diferents que pertanyen a ofertes de transport diferents, mourà $p1$ a la posició de $p2$ i a l'inrevés. La condició d'aplicabilitat d'aquesta operació és que cap de les dues ofertes sobrepassa la seva capacitat màxima i que cap paquet s'entrega més tard que el màxim de dies segons la prioritat després de fer el swap. Aquest operador té un factor de ramificació de $O(n^2)$ ja que per cada paquet el podem intercanviar amb la resta de paquets en el cas pitjor. Per ser més precisos el factor de ramificació és $O(\frac{n(n-1)}{2})$, això surt dels nombres combinatoris, d'agafar els n paquets de 2 en 2 on l'ordre no importa; aleshores obtenim el nombre combinatori $\binom{n}{2}$. Segons la fórmula per trobar el valor d'un nombre combinatori genèric $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ obtenim la següent expressió $\frac{n!}{2!(n-2)!}$, la qual es pot simplificar en l'expressió donada abans $\frac{n(n-1)}{2}$.
- *mourePaquet*(p , o): Sigi p un paquet i o una oferta diferent a la qual ja està assignada, mourà el paquet p a la oferta o . La condició d'aplicabilitat d'aquest operador és que p càpiga dins de o i s'entregui abans del màxim de dies d'entrega associada a la prioritat del paquet. El factor de ramificació d'aquest operador és de l'ordre de $O(n(m-1))$, ja que tenim que per cada paquet el podem assignar a totes les ofertes que hi ha menys a la que ja està assignada.

Amb aquests dos operadors serem capaços de recórrer tot l'espai de solucions.

1.8 Generador de Solucions Inicials

El fet de tenir dues funcions generadores de solucions inicials ens permetrà analitzar el comportament dels dos algorismes de cerca local que utilitzarem. Veurem com la configuració de l'estat inicial ens permetrà, o no, fer una búsqueda eficaç a través de l'espai de cerca i si ens estancarem, o no, en mínims locals. I sobretot, quina de les dues funcions generadores de solucions inicials ens dona un millor rendiment per cadascun dels algorismes.

1.8.1 Entrega just al dia

Aquesta funció no té en compte la maximització de la felicitat dels clients ni la reducció de costos. Simplement, assigna els paquets a una oferta que compleixi els requisits d'entrega. La idea principal és buscar una oferta que compleixi amb aquesta condició i tingui espai suficient per al paquet. Per evitar quedar-nos sense espai a les ofertes de major prioritat, endrecem la llista de paquets segons aquesta prioritat. D'aquesta manera, assignem abans els paquets més urgents. Per tant, es pot resumir així:

Algorisme 1 Ordenar paquets per ordre de prioritats

```
1: Ordenar paquets per ordre de prioritats
2: for  $\forall p_i \in \text{Paquets}$  do
3:    $j \leftarrow 0$ 
4:   while  $j < \text{nombre d'ofertes}$  and  $p_i$  no està assignat do
5:      $C_j \leftarrow \text{capacitat de l'oferta } o_j$ 
6:     if  $p_i \leq C_j$  and  $\text{dia\_entrega}(p_i) \leq \text{dia\_oferta}(o_j)$  then
7:       Assignar  $p_i$  a  $o_j$ 
8:     else
9:        $j \leftarrow j + 1$  ▷ Provar amb la següent oferta  $o_{j+1}$ 
10:    end if
11:  end while
12: end for
```

1.8.2 Maximitzar la felicitat i minimitzar els costos

El segon mètode intenta generar una solució inicial que no només respecti els terminis d'entrega, sinó que també busqui maximitzar la felicitat dels clients i minimitzar els costos.

Per maximitzar la felicitat haurem d'aconseguir assignar els paquets per tal que s'entreguin tots tan aviat com es pugui. Per fer això, ordenarem la llista d'ofertes de menor a major segons el dia d'entrega (*Ofertes*). Farem el mateix per les prioritats del paquet (*Paquets*). I el mateix per minimitzar els costos: endreçarem primer les ofertes més barates. D'aquesta manera, s'enviaran els paquets el més aviat possible i, en cas d'empat, amb les ofertes de menor cost.

Un cop ambdues llistes estan ordenades intentarem assignar els paquets de la següent manera. Vegeu a continuació el pseudocodi:

Algorisme 2 Ordenar paquets per ordre de prioritats i per cost

```
1: Ordenar paquets per ordre de prioritats i cost
2: Ordenar ofertes per dies d'entrega
3: for  $\forall p_i \in \text{Paquets}$  do
4:    $j \leftarrow 0$ 
5:   while  $j < \text{nombre d'ofertes}$  and  $p_i$  no està assignat do
6:      $C_j \leftarrow \text{capacitat de l'oferta } o_j$ 
7:     if  $p_i \leq C_j$  then
8:       Assignar  $p_i$  a  $o_j$ 
9:     else
10:       $j \leftarrow j + 1$  ▷ Provar amb la següent oferta  $o_{j+1}$ 
11:    end if
12:  end while
13: end for
```

1.9 Funció Heurística

Hem creat dues funcions heurístiques per poder comparar els resultats que ens generen. Cada una d'elles tindrà criteris diferents:

- *Heurística 1*: $H_1 = \text{cost}$, aquest heurístic està dissenyat per minimitzar exclusivament els costos, sense tenir en compte ningú altre factor de qualitat. Aquest heurístic ens permetrà minimitzar el cost de la assignació de paquets.
- *Heurística 2*: $H_2 = -10 * \text{felicitat} + \text{cost}$, per altra banda, aquest heurístic ens permetrà tenir en compte la felicitat total de la assignació de paquets. La funció d'evaluació serà igual que la anterior però li restarem la felicitat multiplicada per deu. La constant que multiplica la felicitat és degut a que a l'hora de calcular la felicitat i el preu, aquest últim pren més rellevància en la funció heurística si no multipliquem per cap constant cap terme al tenir un rang més ampli de valors possibles, és a dir, pren valors més grans que la felicitat. Aleshores estariem prioritzant més la reducció de costos,

així que amb la constant multipliquem intentem posar al mateix nivell de importància el reduir els costos i augmentar la felicitat. Així doncs estarem maximitzant felicitat i minimitzant cost a la vegada, ja que volem minimitzar la funció heurística.

1.10 Resultats

2 Part Experimental

2.1 Experiment 1

Un factor clau en els algoritmes de cerca local és el mètode utilitzat per recórrer l'espai de cerca. Per aquest motiu, emprem operadors, que apliquen modificacions a un estat base i ens permeten navegar a través d'aquest espai. En un conjunt d'operadors, l'objectiu és minimitzar el factor de ramificació, amb l'aspiració de millorar principalment l'eficiència temporal, tot i que també es busca una millor eficiència espacial.

Per provar la hipòtesi que certs conjunts d'operadors donen millors resultats en una funció heurística, realitzarem un experiment amb un escenari en què s'han d'enviar 100 paquets, i la proporció del pes transportable per les ofertes és de 1,2. Utilitzarem l'algoritme de *Hill Climbing* per a aquest propòsit. Els diferents conjunts d'operadors seran $C_1 = \{\text{move}\}$, $C_2 = \{\text{swap}\}$, $C_3 = \{\text{move}, \text{swap}\}$.

A més també hem de establir quin algorisme generador de solucions inicial emprarem. En aquest cas establirem que l'algorisme que sempre usarem serà el que genera les solucions més aleatòries.

Observació	Com afecten els diferents conjunts d'operadors per trobar resultats més òptims.
Plantejament	Farem ús de diferents conjunts d'operadors per veure com difereixen les solucions que troben.
Hipòtesi	El conjunt amb els operadors <i>swap</i> i <i>move</i> serà el més optim.
Mètode	<ul style="list-style-type: none">• Agafarem un conjunt de 10 llavors aleatòries.• Establirem els algoritmes <i>Hill Climbing</i>, i generador de solucions aleatori.• Executarem 10 experiment els conjunts d'operadors C_1, C_2 i C_3, un per cada llavor.• Medirem els resultats i n'extreurem conclusions.

Ja amb els experiment executats, hem colocat els resultats al projecte *GitHub* (vegeu l'annex per accedir-hi), en el path `./Latex/spreadsheets/ex1.csv`. A partir d'aquestes dades farem un box plot per tenir una representació més visual d'aquestes. Vegeu el boxplot a continuació.

Boxplot del preu resultant de diferents operator set

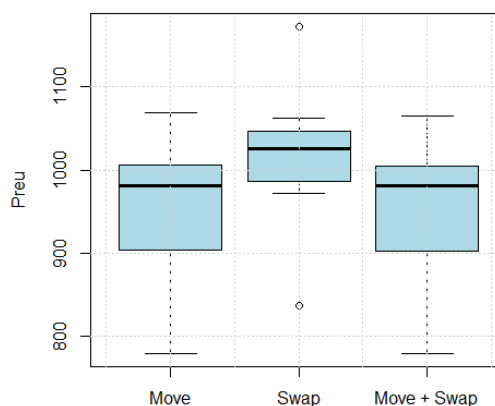


Figura 1: Boxplot del preu resultant de diferents operator sets

Operador	Mediana	Q1	Q3	IQR
Move	981.245	917.275	1003.02	85.745
Swap	1025.675	992.235	1046.285	54.05
Move + Swap	980.87	916.525	1002.11	85.5875

Taula 1: Estadístiques del preu de diferents operator sets

Com podem observar en la *Figura 1*, el resultat obtingut amb l'operador *swap* dista molt dels altres. Així que serà el primer que comentarem. Per començar, cal entendre que amb només un *swap* no recorrem tot l'espai de cerca per un simple motiu: una oferta de transport que inicialment tingui un paquet assignat mai més podrà estar buida. Aquesta limitació es reflecteix en una mediana major. Podem consultar les dades numèriques de l'experiment a la *Taula 1*. A més, veiem que el rang interquantílic (IQR) amb *swap* és el menor, la qual cosa indica que aquest operador ens proporciona resultats consistents, però no els millors.

Per altra banda, observem que els resultats de *move* i *move + swap* són molt similars. Això implica que amb el *move* podem explorar la gran majoria de l'espai de cerca, ja que no té la restricció que presenta l'altre operador. En aquest cas, la única limitació que tindriem es donaria quan volguéssim fer un swap de dos paquets. Amb l'operador de *move*, aquest procés s'ha de realitzar en dues operacions com a norma general; però, si un dels paquets no hi càpiga en l'altra oferta, necessitarem tres operacions, utilitzant una oferta auxiliar per assignar temporalment aquell paquet.

En casos puntuals, seria probable que no poguéssim fer aquest swap només amb el *move*. Aquest escenari s'ha donat durant els experiments, ja que en alguns casos hem observat una diferència molt petita en el cost, ja que el *swap* ha pogut complementar el que el *move* no podia. Per aquesta raó, els dos conjunts d'operadors que estem analitzant presenten resultats similars, però el conjunt que inclou l'operador *swap* obté resultats lleugerament millors en alguns casos.

Respecte a la nostra hipòtesi inicial, observem que s'ha complert, ja que el conjunt d'operadors que inclou tant *move* com *swap* ha mostrat un rendiment lleugerament superior en alguns casos. No obstant això, no esperàvem una similitud tan notable entre els resultats obtinguts amb els conjunts d'operadors C_2 i C_3 . Aquesta similitud indica que l'operador *swap* pot no ser essencial per millorar significativament els resultats en la majoria dels escenaris analitzats.

A partir d'aquesta observació, podríem considerar l'opció de reduir el factor de ramificació eliminant l'operador *swap* del conjunt d'operadors utilitzats. La raó darrere d'aquesta decisió es fonamenta en el principi d'eficiència: simplificar el conjunt d'operadors pot conduir a una reducció del temps de càlcul i a una millora en la velocitat d'exploració de l'espai de cerca.

En resum, els resultats indiquen que, tot i que el *swap* aporta certes millores en situacions particulars, la seva presència no és extremadament rellevant. Això suggereix que una aproximació més eficient podria ser treballar exclusivament amb l'operador *move*, permetent-nos explorar l'espai de cerca de manera més àgil i reduint potencialment el factor de ramificació, facilitant així la millora del rendiment global de l'algoritme.

2.2 Experiment 2

Un altre punt que creiem molt important a l'hora de plantejar aquests problemes és la solució inicial. Una solució inicial de qualitat proporciona un diferent punt de partida per explorar l'espai de solucions, la qual cosa pot conduir a diferents resultats finals, o bé a una diferent velocitat per trobar una solució òptima.

En aquest experiment, compararem dues estratègies de generació de solucions inicials per avaluar si hi ha alguna diferència notable en els resultats finals quan es fa servir l'algoritme *Hill Climbing* i els operadors escollits a l'apartat anterior: *swap* i *move*.

Les dues estratègies que considerarem per la generació de l'estat inicial són:

- *Solució Inicial 1*: aquesta és una estratègia que genera solucions inicials amb una qualitat més alta, és a dir, busquem des d'un principi una solució més propera a la que creiem la solució final, més òptima, ja que intentem fer una assignació de paquets a ofertes amb el menor preu possible.
- *Solució Inicial 2*: aquesta estratègia simplement pretén assignar els paquets a ofertes i en cap moment dona prioritat a les ofertes més barates, és per això que està més lluny de la solució final que busquem, però és més ràpida i simple de generar que la primera.

Plantejament	Observem la influència de dues solucions inicials en la qualitat de la solució final.
Hipòtesi	Una solució inicial menys òptima, la <i>Solució inicial 2</i> , proporcionarà significativament millors resultats.
Mètode	<ul style="list-style-type: none"> • Agafarem un conjunt de 10 llavors aleatòries. • Establirem els algorismes <i>Hill Climbing</i> per un problema amb 100 paquets i una proporció de pes transportable per les ofertes d'1,2. • Executarem els 10 experiments amb la <i>Solució inicial 1</i> i la <i>Solució inicial 2</i>, un per cada llavor. • Medirem els resultats i n'extraurem conclusions.

Els resultats, que es poden trobar al path `./Latex/spreadsheets/exp2.csv` de *GitHub* (vegeu l'annex per accedir-hi), es poden resumir amb el gràfic i la taula que hi ha a continuació:

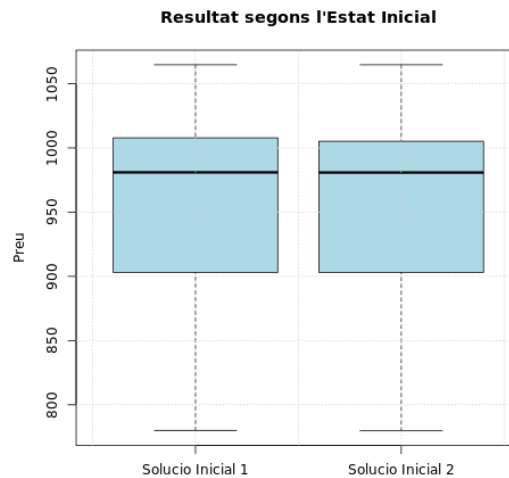


Figura 2: Boxplot del preu resultant segons la solució inicial

Operador	Mediana	Q1	Q3	IQR
Solució 1	980.975	916.58	1004.178	87.5975
Solució 2	980.87	916.525	1002.112	85.5875

Taula 2: Estadístiques del preu segons la solució inicial

Com clarament veiem a la *Figura 2*, les diferències entre els resultats obtinguts segons les solucions inicials donades són mínimes, tant en la mediana com en el rang interquartílic (IQR). La *Solució inicial 1*, és a dir, la més "òptima", presenta una mediana respecte al preu lleugerament superior (980,975 comparada amb 980,87), però la diferència no és significativa. El rang interquartílic és també lleugerament

més ampli per aquesta solució, amb un IQR de 87,60 en comparació amb els 85,59 de l'altra solució inicial.

Aquestes similituds porten a pensar que ambdues estratègies acaben atrapades en mínims locals, independentment de la qualitat de la solució inicial, per això les diferències en els resultats finals són gairebé inexistents en l'algoritme de *Hill Climbing*, és a dir, totes dues estratègies portaran cap a resultats similars.

Conclusió

Després d'analitzar els resultats, podem concloure que no hi ha una diferència significativa entre les dues estratègies de generació de solucions inicials. Això ens dona a pensar que almenys en aquest escenari, l'algoritme *Hill Climbing* tendeix a quedar-se en mínims locals, independentment de la qualitat de la solució inicial. Per tant, pels pròxims experiments, optarem per la generació de solucions inicials menys òptimes, és a dir, per la *Solució inicial 2*, ja que simplifica el procés de generar l'Estat Inicial i no empitjora la solució final.

2.3 Experiment 3

L'algoritme de *Simulated Annealing* tal com està implementada accepta els paràmetres: nombre total d'iteracions, iteracions per cada canvi de temperatura (divisor de l'anterior), paràmetres k i λ de la funció d'acceptació d'estats; els quals s'han de determinar experimentalment per tal d'obtenir el millor rendiment possible d'aquest algoritme.

Observació	
Plantejament	Observem la influència de dues solucions inicials en la qualitat de la solució final.
Hipòtesi	Una solució inicial menys òptima, la <i>Solució inicial 2</i> , proporcionarà significativament millors resultats.
Mètode	<ul style="list-style-type: none">• Agafarem un conjunt de 10 llavors aleatòries.• Establirem els algorismes <i>Hill Climbing</i> per un problema amb 100 paquets i una proporció de pes transportable per les ofertes d'1,2.• Executarem els 10 experiments amb la <i>Solució inicial 1</i> i la <i>Solució inicial 2</i>, un per cada llavor.• Medirem els resultats i n'extraurem conclusions.

2.4 Experiment 4

Un cop hem experimentat amb els diversos factors que influeixen en la búsqueda d'una solució òptima al problema i hem determinat quins funcionen millor, ens interessa ara també veure com evoluciona el temps d'execució per trobar una solució per veure si és factible trobar-ho en un temps raonable.

L'objectiu és analitzar com evoluciona el temps d'execució necessari per trobar una solució utilitzant l'algoritme de *Hill Climbing*, en funció de dos factors clau: el nombre de paquets a enviar i la proporció de pes transportable per les ofertes.

L'escenari és el mateix que en els experiments anteriors: treballem amb la generadora d'una solució inicial aleatòria, la funció heurística que només té en compte els costos i els operadors de *swap* i *moure* que ja han estat seleccionats pels seus resultats en els experiments previs. Aquest cop, el focus es desplaça cap a l'anàlisi de la complexitat temporal del sistema, estudiant com es veu afectat el temps d'execució quan variem els dos paràmetres mencionats.

Com que tenim dos factors a estudiar, dividim aquest experiment en dues parts:

- *Impacte de la proporció del pes transportable*: es fixarà el nombre de paquets a 100 i s'anirà augmentant-la proporció del pes transportable des de 1,2 en increments de 0,2 on volem identificar com

la relació entre la capacitat de les ofertes i el pes dels paquets afecta el rendiment de l'algoritme.

- *Impacte del nombre de paquets*: es fixarà la proporció del pes transportable a 1, 2 i es variarà el nombre de paquets, començant per 100 i augmentant-lo de 50 en 50. S'analitzarà com l'increment en la quantitat de paquets afecta el temps d'execució.

2.4.1 Proporció del pes transportable

Observació	Com afecta la proporció del pes transportable al temps d'execució per trobar solucions òptimes.
Plantejament	Analitzarem l'impacte d'augmentar la proporció del pes transportable sobre el temps d'execució.
Hipòtesi	Es preveu que a mesura que augmenta la proporció del pes transportable, el temps d'execució es redueixi, ja que amb una capacitat més gran de les ofertes, es podrien trobar solucions més ràpidament.
Mètode	<ul style="list-style-type: none">• Agafarem un conjunt de 10 llavors aleatòries.• Establirem els algorismes <i>Hill Climbing</i> per un problema amb 100 paquets, amb la funció heurística que minimitza només costos i amb el generador de solucions aleatori.• Executarem 10 experiments per a cada una de les proporcions de pes transportable {1.2, 1.4, 1.5, 1.6, 1.8, 2.0 }, un per cada llavor.• Mesurarem els resultats i n'extraurem conclusions.

Els resultats dels experiments realitzats, que es poden trobar al path `./Latex/spreadsheets/exp3.csv` de *GitHub* (vegeu l'annex per accedir-hi), els podem veure representats en els gràfics que hi ha a continuació:

2.4.2 Nombre de paquets

Observació	Com afecta el nombre de paquets al temps d'execució per trobar solucions òptimes.
Plantejament	Analitzarem l'impacte d'augmentar el nombre de paquets sobre el temps d'execució.
Hipòtesi	Es preveu que a mesura que augmenta el nombre de paquets, el temps d'execució també augmenti, ja que el problema es torna més complex en gestionar un nombre més gran de paquets.
Mètode	<ul style="list-style-type: none">• Agafarem un conjunt de 10 llavors aleatòries.• Establirem els algorismes <i>Hill Climbing</i> per un problema amb la proporció del pes de transport fixat 1, 2, amb la funció heurística que minimitza només costos i amb el generador de solucions aleatori.• Executarem 10 experiments per a cada nombre de paquets {100, 150, 200, 250, 300, 350, 400 }, un per cada llavor.• Mesurarem els resultats i n'extraurem conclusions.

2.5 Experiment 5

2.6 Experiment 6: Influència de la ponderació de la felicitat

Un altre punt clau a l'hora de crear algoritmes és l'heurística. En aquest cas, una de les heurístiques proposades no només té en compte una optimització del preu total de transport i emmagatzematge, sinó que també té en compte l'optimització de la felicitat dels clients. És important saber conèixer quins són els objectius a l'hora de fer la distribució per veure quanta importància es dona a cadascun d'aquests factors, ja que no podem, alhora, minimitzar al màxim el preu i maximitzar la felicitat dels clients.

En canvi, creiem que el canvi en la ponderació de la felicitat, no hauria d'alterar gaire el temps d'execució de l'algoritme, ja que el recorregut de solucions que hi ha de fer hauria de ser semblant.

Per comprovar tot això, ara avaluarem l'heurística per veure quanta influència té la felicitat en el preu resultant, variant la ponderació de la felicitat en aquesta. Per veure això, farem servir l'algoritme de *Hill Climbing*, amb un enviament de 100 paquets i una proporció de pes transportable per les ofertes d'1,2.

Observació	Creiem que si volem augmentar la felicitat dels nostres clients, també hi haurà un augment del cost total. Però en canvi, això no influirà en el temps d'execució.
Plantejament	Generem diverses solucions variant la ponderació de la felicitat dins l'heurística per veure com influeix en el preu i en el temps d'execució.
Hipòtesi	El preu serà més elevat si augmentem la ponderació de la felicitat dins l'heurística, però el temps emprat per trobar la solució serà el mateix.
Mètode	<ul style="list-style-type: none">• Agafarem un conjunt de 10 llavors aleatòries per cada ponderació de felicitat.• Establirem els algoritmes <i>Hill Climbing</i>, i generador de solucions pseudo-aleatori, <i>Solució Inicial 2</i>.• Executarem l'experiment amb 100 paquets i una proporció d'1,2.• Anirem augmentat de 5 en 5 la felicitat, partint de 0 i arribant fins a 20.• Medirem els resultats (preu i temps) i n'extreurem conclusions.

Ja amb els experiment executats, hem colocat els resultats al projecte *GitHub* (vegeu l'annex per accedir-hi), en el path `./Latex/spreadsheets/ex6.csv` i `./Latex/spreadsheets/ex6b.csv`. A partir d'aquestes dades hem fet dos gràfics representatiu dels resultats.

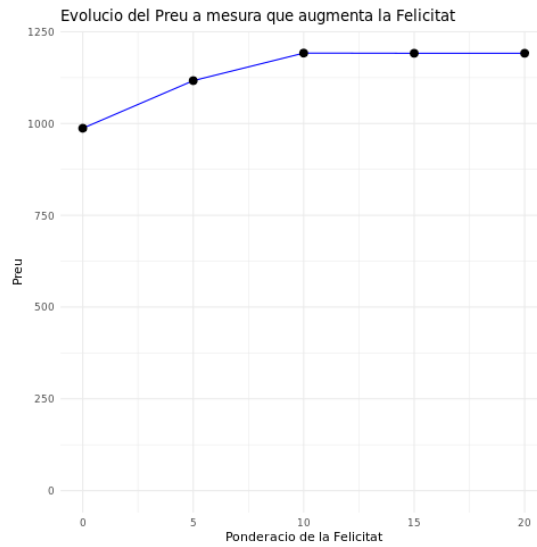


Figura 3: Gràfic del preu resultant al variar la ponderació de la felicitat

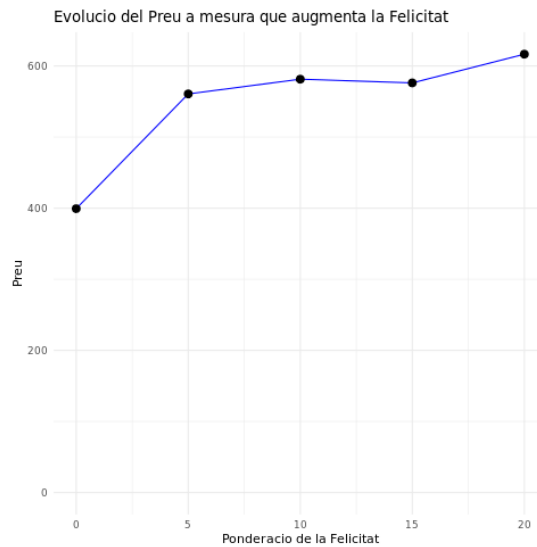


Figura 4: Gràfic del preu resultant al variar la ponderació de la felicitat

Operador	Mediana	MedianaTemps
Felicitat*0	1010.5	395.5
Felicitat*5	1141.105	523.0
Felicitat*10	1243	535.5
Felicitat*15	1243.5	579.5
Felicitat*20	1243.5	600.5

Taula 3: Estadístiques del preu i temps d'execució segons la ponderació de la felicitat

Com podem veure a la *Figura 3*, la felicitat té una petita influència en el cost final, ja que si comparem la mediana entre quan no tenim en compte la felicitat a l'heurística i quan tenim una ponderació de 20 dins d'aquesta, veiem que hi ha una diferència d'una mica més de 100.

També podem veure que a mesura que aquesta augmenta, entre els experiments que multipliquen la felicitat per 10 i els que ho fan per 20, la diferència és gairebé única. Això és possible que sigui degut a que s'apropa a un mínim local i aquesta ponderació de la felicitat no és tant alta en comparació.

Si avaluem ara el temps d'execució en aquests algorismes, vist a la *Figura 4*, sí que ens sorprèn trobar-nos un petit augment en aquest a mesura que augmentem la ponderació de la felicitat. Malgrat que aquest també depèn d'altres factors externs a l'algoritme, és possible que això sigui degut a que el ventall de solucions properes quan tenim en compte la felicitat, pot ser major. És a dir, com estem restant la felicitat al cost, hi ha més factors que influeixen al resultat.

Com a conclusió podríem dir que efectivament el preu augmenta amb la felicitat, però que aquest augment es manté estable arribats a un punt. I, respecte el temps d'execució, aquest sí que pot estar influenciat lleugerament per la ponderació d'aquesta felicitat.

2.7 Experiment 7

2.8 Experiment 8: Com canviarien les solucions si variem el preu d'emmagatzematge a l'alza o a la baixa?

2.8.1 Disminució preu emmagatzematge

Si fem que el cost d'emmagatzematge sigui més baix, això clarament farà que el cost total disminueixi, ja que aquest té una influència directa en el resultat, sent 0,25€/kg/día una gran quantitat. Si ens fixem en la felicitat resultant, aquesta és possible que baixi, perquè quan el preu baixa, el cost de mantenir els paquets al magatzem és menor, per la qual cosa tindrà menys pes dins l'heurística.

En canvi, si augmentem aquest cost, és possible que el resultat sigui el contrari, ja que quan el preu puja, té proporcionalment un pes més gran a l'heurística, cosa que per una banda farà que es doni menys importància a la felicitat però, per l'altra, fa que s'intenti tenir els paquets el mínim temps possible al magatzem, perquè això serà costós, per tant, s'entregaran els paquets el més aviat possible i augmentarà la felicitat. Però, per altra banda, si tenim en compte que està augmentant una part important de l'heurística, també veiem el cost total augmentarà.

2.9 Experiment 9, especial

L'eficàcia temporal és un aspecte clau en els algorismes d'intel·ligència artificial. En aquest experiment, per a una entrada donada, extraurem dades sobre el temps necessari perquè l'algoritme s'executi. Els paràmetres que proporcionarem al programa seran: 100 paquets, una proporció de 1,2 i una llavor de 1234. A més, utilitzarem el mètode *Hill Climbing* amb els operadors més òptims, que vam examinar en el primer experiment. Aquest conjunt d'operadors inclourà les operacions *move* i *swap*. Només ens falta definir un últim paràmetre: el generador de solucions inicials. Aquest també ha estat seleccionat a partir d'experiments anteriors, més concretament del segon experiment, on vam obtenir els millors resultats amb el generador més 'aleatori'.

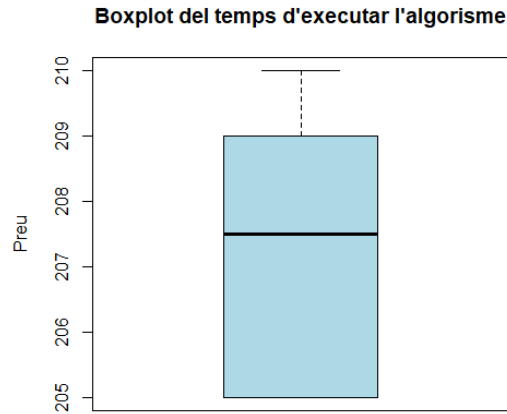


Figura 5: Boxplot del preu resultant segons la solució inicial

A la *Figura 5*, observem la distribució dels temps d'execució obtinguts. La mediana és de 207,5 ms i l'IQR és de 3,5, cosa que indica que el temps emprat pel programa és força estable.

Amb aquests paràmetres, *Hill Climbing* aconsegueix trobar un mínim on l'assignació de paquets té un cost de 794,675 €.

3 Conclusions

4 Treball d'Innovació: DeepVariant

4.1 Tema

Hem escollit DeepVariant, un algorisme de machine learning desenvolupat per Google, que utilitza xarxes neuronals profundes per identificar variants genètiques en dades de seqüenciació de l'ADN. Mitjançant imatges generades a partir de les lectures de seqüència, DeepVariant detecta mutacions petites amb una precisió superior respecte a altres mètodes tradicionals.

4.2 Responsabilitats d'equip

Malgrat que el treball s'està fent col·laborativament i tots els membres del grup participen en totes les tasques, s'ha decidit dividir-ho en tres parts per cercar i aconseguir informació al respecte.

- *Introducció i context:*
Introducció a DeepVariant: què és i per què és rellevant.
Importància de DeepVariant en la detecció de variants genètiques.
Visió general del problema que resol en l'anàlisi de dades de seqüenciació d'ADN.
Per què l'aprenentatge profund és útil en aquest context: beneficis respecte als mètodes tradicionals de detecció de variants (comparació amb altres mètodes).
- *Xarxes Neuronals Convolucionals (CNN):*
Explicació de què són les xarxes neuronals convolucionals (CNNs).
Com s'utilitzen les CNNs en DeepVariant per identificar patrons genètics.
Avantatges de l'ús de CNNs per analitzar dades de seqüenciació d'ADN (perquè reconeixen patrons visuals).
Relació entre les imatges generades per DeepVariant i l'anàlisi de dades genètiques.
- *Funcionament Pipeline:*
Generació d'imatges a partir de dades de seqüenciació d'ADN.
Com les imatges es passen per la xarxa neuronal convolucional per ser classificades.
Processos d'inferència: com es decideix quina variant està present (substitució, inserció, deleció).
Entrenament de la xarxa neuronal amb dades genètiques.
Presa de decisions sobre la presència de variants genètiques i avantatges de precisió i fiabilitat.

4.3 Fonts

- *Creating a universal SNP and small indel variant caller with deep neural networks:* Rellevància: Base teòrica de DeepVariant i explicació de l'ús de xarxes neuronals. Data d'accés: 13 d'octubre de 2024 Enllaç: <https://doi.org/10.1101/092890> *****SEGONA PART AL WHATS
- : Enllaç: <https://github.com/google/deepvariant> També porta a altres documents com: <https://google.github.io/02-20-looking-through-deepvariants-eyes/>

4.4 Dificultats

- *Accés a referències:* Alguns articles rellevants requereixen subscripcions per poder obtenir informació detallada.
- *Informació tècnica:* Hem hagut de fer un sobreesforç perquè es requereix d'un mínim de coneixements sobre l'ADN per entendre la proposta.