Inteligencia Artificial: Pràctica de Cerca Local

Ferriol Falip, Martí Puerta, Xavi Medina

Curs 2023-24 Q2

Índex

1 Descripció del problema

Estem davant d'un problema complex de distribució de fitxers en un sistema de servidors repartits geogràficament. El nostre objectiu és gestionar les sol·licituds dels usuaris per accedir a fitxers específics. Cada usuari té una identificació única, així com cada fitxer en el sistema. Els fitxers es troben distribuïts en diversos servidors, formant un sistema de fitxers distribuït. Cada servidor té assignat un conjunt de fitxers, cada un d'ells amb certa replicació mínima per garantir la disponibilitat.

La dificultat principal cau en el fet que els servidors i els usuaris es troben en ubicacions geogràfiques diferents, influint en els temps de transmissió dels fitxers. Això implica que la latència en la transferència de fitxers pot variar segons la ubicació del servidor i l'usuari. El servidor encarregat de gestionar les sol·licituds disposa d'informació sobre quins servidors contenen còpies dels fitxers i els temps de transmissió associats a cada un. El repte doncs consisteix a gestionar de manera eficient les sol·licituds dels usuaris, assegurant la disponibilitat dels fitxers i minimitzant el temps de transmissió.

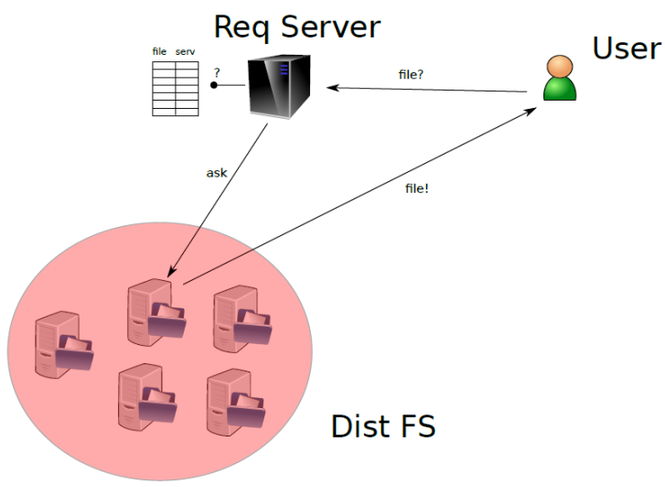

Figura 1: Sistema de fitxers distribuït

1.1 Elements del problema

Per resoldre el problema tenim 4 paràmetres que cal tenir en compte i que variaran segons l'experiment:

- Nombre d'usuaris
- Nombre de servidors
- Nombre màxim de peticions que pot fer un usuari
- Nombre mínim de replicacions d'un fitxer a diferents servidors

D'aquests paràmetres en podrem extreure els dos principals elements del problema, les peticions i els servidors.

- **Peticions:** una petició ve donada per un identificador d'usuari i un identificador de fitxer representant doncs que l'usuari *userId* demana el fitxer *fileId*.
- **Servidors:** per cada servidor S_i sabem els fitxers que conte i el temps de transmissió a cada usuari.

1.2 Solució i criteris per avaluar-la

Una solució del problema doncs haurà de determinar per cada servidor quins fitxers haurà de transmetre i a quins usuaris per cobrir totes les peticions.

Par la seva banda, la qualitat de la solució ha d'estar vinculada al temps total de transmissió dels fitxers. No obstant això, si només considerem aquesta mesura, podríem obtenir solucions on alguns servidors hauran d'enviar molts fitxers mentre que d'altres no en enviaran cap. Ens interessa més equilibrar la càrrega dels servidors, de manera que les transmissions estiguin més o menys repartides entre ells.

Per obtenir i avaluar les possibles solucions del problema tindrem en compte els següents dos criteris:

- Minimitzar el temps de transmissió dels fitxers per el servidor que triga més temps a transmetre totes les seves peticions.
- Minimitzar el temps total de transmissió dels fitxers tenint en compte que volem que els temps totals de transmissió de cada servidor siguin el màxim de similars entre ells.

1.3 Cerca local

Un cop analitzat el problema és fàcil arribar a la conclusió que aquest es pot resoldre utilitzant algorismes de cerca local. És així ja que és un problema d'optimització on no estem buscant un òptim global i ens interessa aconseguir el millor estat on puguem arribar, a partir de diferents criteris per avaluar-lo i sense que ens importi el camí per arribar-hi.

2 Implementació

2.1 Implementació de l'estat

Per representar l'estat tenim dues parts principals, una part estàtica que no varia en tot el problema i una part dinàmica que va variant al llarg de l'execució.

La idea de la part estàtica és estalviar espai i temps, calculant només un cop la informació que no variarà al llarg del problema. En el nostre cas tenim la següent informació emmagatzemada de forma estàtica.

- ***fileLoc:*** per cada fitxer tenim un *array* amb tots els servidors que contenen aquest fitxer.
- ***transTime:*** per cada parell (*userId*, *serverId*) conté el temps de transmissió entre aquests dos.

La part dinàmica de l'estat correspon a la solució que actualment estem considerant, és a dir, per cada usuari quin servidor li passa els fitxers que aquest ha demanat.

- ***actualBoard:*** per cada usuari tenim una llista de parells (*fileId*, *serverId*) que representen per cada fitxer que ha demanat l'usuari quin servidor està transmetent el fitxer.

2.2 Operadors

Per modificar l'estat hem implementat els següents operadors. Pel càlcul del factor de ramificació assumim que tenim p peticions i s servidors.

Moure

- *Paràmetres:* l'identificador de l'usuari *userId*, l'identeifcador del fitxer *fileId* i l'identificador del nou servidor *serverId*.
- *Precondició:* *serverId* conté el fitxer *fileId* i *serverId* és diferent al server assignat actualment.
- *Postcondició:* la petició de l'usuari *userId* del fitxer *fileId* ara està servida per el servidor *serverId*.
- *Factor de ramificació:* el factor de ramificació serà el nombre de peticions per el nombre de replicacions de cada fitxer, per tant de l'ordre de $\mathcal{O}(p * s)$

Intercanvi

- *Paràmetres:* els identificadors dels usuaris $userId_1$ i $userId_2$ i els identificadors dels fitxers $fileId_1$ i $fileId_2$.
- *Precondició:* els fitxers $fileId_1$ i $fileId_2$ estan presents als servidors que s'intercanviaran i els servidors que transmeten els dos fitxers dels dos fitxers són diferents.
- *Postcondició:* la petició $(userId_1, fileId_1)$ ara està servida per $serverId_2$ i la petició $(userId_2, fileId_2)$ està servida per $serverId_1$
- *Factor de ramificació:* El factor de ramificació serà el nombre de peticions pel nombre de peticions amb fitxers que tinguin servidors intercanviables, és a dir serà de l'ordre de $\mathcal{O}(p^2 * s)$.

2.3 Solucions inicials

Per generar les solucions inicials del problema hem definint dos estratègies.

Random

La primera busca un enfocament aleatori assignant a cada petició un server aleatori dels que tenen el fitxer demanat. L'objectiu d'aquest enfocament és que en executar diversos cops el mateix problema podrem obtenir mínims locals diferents i d'aquesta forma poder explorar un tros de l'espai de solucions diferent a cada execució i, potencialment, obtenir solucions més properes al límit global.

Greedy

La segona estratègia per generar solucions inicials és basa en un enfoc *greedy* on assignem a cada petició el server que té un temps de transmissió menor, és a dir, el servidor més proper a l'usuari és el que transmet el fitxer. Aquesta estratègia al no tenir cap grau d'aleatorietat provocarà que diferents execucions del mateix problema arribin al mateix mínim local i per tant explorarem sempre la mateixa part de l'espai de solucions. L'objectiu però d'aquest mètode és començar amb una solució inicial bona i per tant propera a una possible solució que no difereixi gaire del mínim global.

2.4 Funció heurística

Per resoldre el problema tenim dues heurístiques que deriven dels dos criteris per avaluar la solució mencionats anteriorment.

- El primer heurístic calcula el valor del servidor que té la suma de transmissions màxima.

$$H_1 = max_{s_i \in S}(total\_time(s_i))$$

- El segon heurístic calcula la suma dels quadrats del temps de transmissió de cada servidor.

$$H_2 = \sum_{s_i \in S} total\_time(s_i)^2$$

Tenint en compte que $total\_time(s_i)$ és el temps que tarda el servidor s_i a transmetre tots el fitxers que li pertoquen. En el segon heurístic elevem al quadrat per tal de que les solucions més distribuïdes siguin molt millors que les que no ho són.

3 Experiments

3.1 Experiment 1

El primer experiment consisteix en avaluar i comparar els operadors que hem creat per solucionar el problema.

- **Observació:** El conjunt d'operadors pot influir en les solucions obtingudes per *Hill Climbing*.
- **Plantejament**: Utilitzarem 3 conjunts d'operadors, només l'operador *Move*, només l'operador *Swap* i els dos operadors combinats.
- **Hipòtesis:** Utilitzar més operadors ens permetrà expandir més nodes i per tant obtenir una solució millor.
- **Metodologia:**
  - Utilitzarem les mateixes 10 llavors generades aleatòriament per tots els conjunts d'operadors.
  - Treballarem amb 200 usuaris amb màxim 5 peticions per usuari i 50 servidors amb un mínim de replicació de 5.
  - Utilitzarem l'estratègia de generació *Random* i avaluarem les solucions només amb el primer heurístic.

Un cop fets els experiments i recollides les dades dels costos i nodes expandits de cada solució obtinguda amb els tres conjunts d'operadors, obtenim els següents resultats que es poden observar a la Taula 1.

	MOVE		**SWAP**		**MOVE+SWAP**	
SEED	VALOR	NODES	VALOR	NODES	VALOR	NODES
1569741360	39021	33	30648	134	34614	58
1785505948	29360	213	35749	66	31495	92
516548029	31339	210	33485	134	37594	59
1302116447	32927	84	28631	151	29604	148
1368843515	32865	79	26324	229	31293	91
663681053	32778	105	26819	202	27698	204
1182054491	29898	178	26358	205	30480	128
251269761	39801	28	30649	131	25973	257
1283218719	33302	124	31689	153	27011	265
1678332854	38862	59	39603	69	32665	147

Taula 1: Taula de resultats de l'experiment 1

Per comparar els resultats obtinguts de forma visual fem *boxplots* de les dades i obtenim els gràfics de la figura 2.

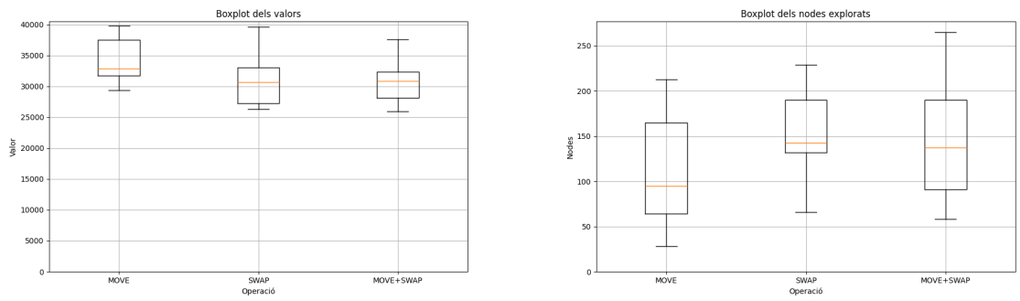

Figura 2: *Boxplots* dels costos i els nodes explorats dels resultats obtinguts

Respecte el cost heurístic podem observar que operar només amb l'operador de moure obtenim costos pitjors respecte la resta de conjunts d'operadors. També podem observar que entre el conjunt d'operadors *Swap* i *Move+Swap* a simple vista no hi ha una gran diferència. Respecte el nombre de nodes expandits veiem que només utilitzant l'operador de moure expandim generalment menys nodes que en els altres conjunts d'operadors. Un cop més a simple vista no hi ha una gran diferència entre el conjunt d'operadors *Swap* i *Move+Swap*.

Podem concloure doncs que utilitzar només l'operador d'intercanvi o ambdós operadors obtindrem resultats similars. Tot i això nosaltres hem optat per adoptar el conjunt d'operadors moure i intercanvi ja que els costos són lleugerament més estables i utilitzar més operadors ens permet explorar més nodes en alguns casos tot i que això augmenti el factor de ramificació i, per conseqüència, el temps d'execució i una pitjor eficiència espacial.

3.2 Experiment 2

El segon experiment consisteix en comprovar quina de les dues estratègies per generar les solucions inicials dona millors resultats.

- **Observació:** L'estratègia per generar solucions pot influir en les solucions a les quals arriba *Hill Climbing*.
- **Plantejament**: Utilitzarem les dues estratègies de generació inicial *Random* i *Greedy*, explicades anteriorment.
- **Hipòtesis:** L'estratègia de generació influeix en el cost final obtingut
- **Metodologia:**
  - Generarem 10 llavors aleatòries i les utilitzarem per les dues estratègies de generació
  - Treballarem amb 200 usuaris amb màxim 5 peticions per usuari i 50 servidors amb un mínim de replicació de 5.
  - Avaluarem les solucions només amb el primer heurístic.
  - També compararem 10 execucions del mateix problema utilitzant l'estratègia *Random* i la *Greedy*

Un cop fets els experiments i recopilades totes les dades obtenim els resultats de la Taula 2.

	RANDOM		GREEDY	
SEED	VALOR	NODES	VALOR	NODES
1569741360	34614	58	13463	34
1785505948	31495	92	13626	32
516548029	37594	59	12446	86
1302116447	29604	148	11123	84
1368843515	31293	91	10768	59
663681053	27698	204	13992	26
1182054491	30480	128	12973	65
251269761	25973	257	12117	57
1283218719	27011	265	13643	57
1678332854	32665	147	12724	70

Taula 2: Taula de resultats de l'experiment 2

Per poder observar visualment els resultats fem *boxplots* de les dades i obtenim els gràfics de la figura 3.

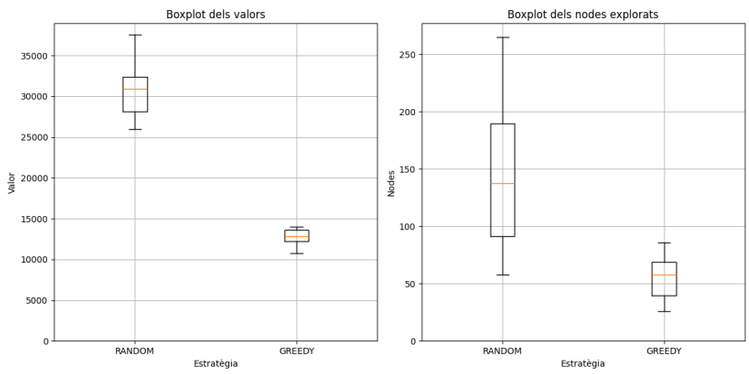

Figura 3: *Boxplots* de les dades obtingudes a l'experiment 2

Podem observar que clarament l'estratègia *Greedy* dona resultats molt més bons que l'estratègia *Random*. Això podria ser perquè l'heurístic actual busca minimitzar el cost màxim i l'estratègia *Greedy* aconsegueix resultats molt bons des del principi i, per altra banda, l'estratègia *Random* es queda estancada en mínims locals molt abans de poder apropar-se a les solucions del mètode *Greedy*. Tot i això hem decidit seguir utilitzant l'estratègia *Random* en els següents experiments perquè ens permet explorar una extensió més gran de l'espai de solucions.

A la Taula 3 podem observar el resultat d'executar 10 vegades el mateix problema amb l'estratègia *Random* i l'estratègia *Greedy*. Podem observar com igualment el cost utilitzant el mètode *Greedy* és més petit que el cost mínim de les execucions amb el mètode *Random*.

	RANDOM		GREEDY	
SEED	VALOR	NODES	VALOR	NODES
1569741360	32251	114	13463	34
	31671	132		
	36281	58		
	34201	93		
	34018	85		
	25907	292		
	35391	60		
	38638	41		
	28496	167		
	33561	81		

Taula 3: Resultats de 10 execucions amb *Random* comparat amb *Greedy*

3.3 Experiment 3

El tercer experiment consisteix en trobar els paràmetres que donen millors resultats per *Simulated Annealing*.

- **Observació:** El valor de k i λ influirà en el resultat que obtindrà *Simulated Annealing*
- **Plantejament**: Per diferents combinacions de k i λ observem el cost de la solució final.
- **Hipòtesis:** Per valors alts de k i petits de λ obtindrem pitjors resultats.
- **Metodologia:**
  - Treballarem amb un problema aleatori.
  - Farem una execució per totes les combinacions de $k = 1, 5, 25, 250, 1000$ i $\lambda = 0.1, 0.01, 0.001$. No considerem λ més petites ja que al fer-ho el temps d'execució creix considerablement.
  - Avaluarem les solucions només amb el primer heurístic.

Un cop fets els experiments i recopilades totes les dades obtenim els resultats de la Taula 4.

		K				
		1	5	25	250	1000
Lambda	0.1	27840	28082	29053	35135	40644
	0.01	17554	17146	17797	17584	18157
	0.001	11418	11183	11449	11272	11264

Taula 4: Resultats de l'experiment 3

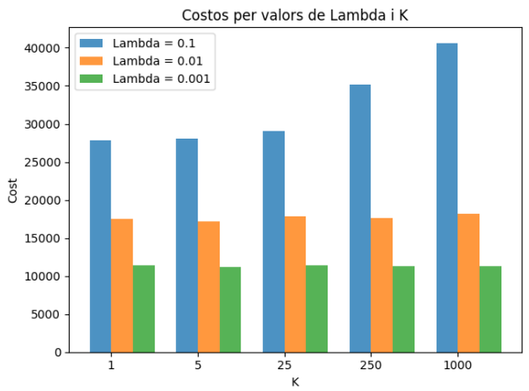

Figura 4: Gràfica dels costos segons k i λ

A la figura 4 veiem gràficament les dades obtingudes i podem concloure fàcilment que per $\lambda = 0.001$ s'obtenen millors resultats. També podem observar que el valor de la k per λ petita no és gaire rellevant ja que per tots els valors de k el cost final és similar. Veient que per $k = 5$ obtenim el millor cost hem decidit adoptar aquests valors de k i λ per la resta d'execucions amb *Simulated Annealing*.

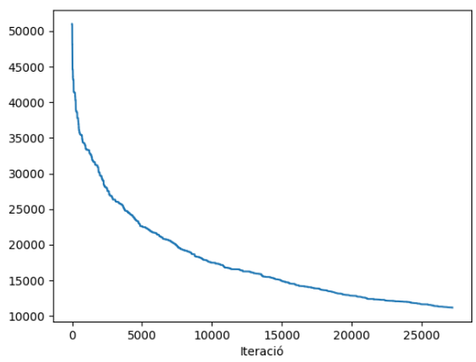

Figura 5: Evolució del cost de la solució per $k = 5$ i $\lambda = 0.001$

3.4 Experiment 4

Al quart experiment volem observar com varia el temps d'execució respecte al nombre d'usuaris i servidors.

- **Observació:** El nombre d'usuaris i servidors pot afectar al temps d'execució.
- **Plantejament**: Anem augmentant el nombre d'usuaris o servidors per veure com evoluciona.
- **Hipòtesis:** més usuaris o servidors resultarà en un increment del temps d'execució.
- **Metodologia:**
  - Per un problema aleatori
  - Amb 50 servidors comencem amb 100 usuaris i els anem incrementant fins veure la tendència
  - Amb 200 usuaris comencem amb 50 servidors i els anem incrementant fins a veure la tendència.

Un cop fets els experiments i recopilades totes les dades obtenim els resultats de les Taules 5 i 6.

SEED	NUM USERS	TEMPS (ms)	NODES	ms/node
1569741360	100	10156	105	96.723
	200	92131	103	894.475
	300	486485	180	2702.694
	400	780559	133	5868.864

Taula 5: Resultats de l'experiment 4, evolució del temps d'execució respecte el nombre d'usuaris.

SEED	NUM SERVERS	TEMPS (ms)	NODES	ms/node
1569741360	50	63181	146	432.746
	100	57703	122	472.975
	150	36573	92	397.532
	200	23679	57	415.421
	250	45948	105	437.6
	300	16484	41	402.048
	350	66259	149	444.691
	400	96869	203	477.187
	450	34531	71	486.352
	500	32928	68	484.235
	550	60657	106	572.235
	600	35969	62	580.145
	650	22297	37	602.621
	700	34882	56	622.892
	750	60009	98	612.336

Taula 6: Resultats de l'experiment 4, evolució del temps d'execució segons el nombre de servidors.

Observant les gràfiques de la figura 6 es veu clarament que la tendència al incrementar els usuaris és créixer de forma cada vegada més gran, similar a una funció exponencial. Per altre banda podem veure que incrementant els servidors el temps d'execució també tendeix a créixer però de forma més suau, similar a una funció lineal.

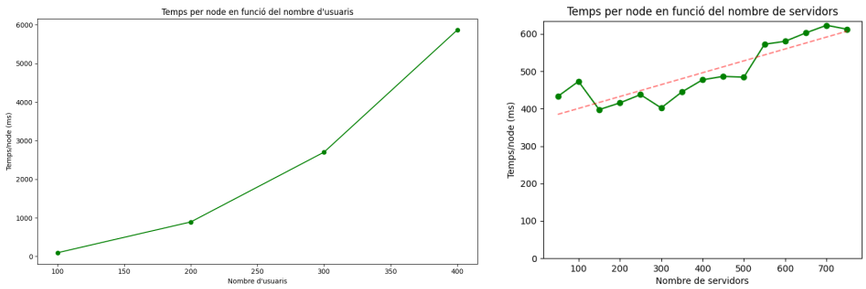

Figura 6: Gràfics de les dades obtingudes a l'experiment 4

3.5 Experiment 5

En aquest experiment busquem comparar els temps tant d'execució com del cost total de les solucions trobades de fitxers, entre els dos criteris que ens planteja el problema utilitzant l'algoritme de *Hill Climbing*.

- **Observació:** L'ús d'un heurístic o l'ús de l'altre proporciona solucions diferents que pot portar a temps d'execució o cost total diferents.
- **Plantejament**: Executem els mateixos problemes emprant l'heurística que busca minimitzar el temps màxim de transmissió d'un servidor i després emprant l'heurística que busca minimitzar la suma de quadrats de temps de transmissió de cada servidor.
- **Hipòtesis:** Al intentar equilibrar la càrrega en el segon heurístic, el temps de transmissió total quan l'utilitzem serà més petit que quan només intentem minimitzar el màxim.
- **Metodologia:**
  - Generem 6 problemes aleatoris
  - Executem els 6 problemes amb un heurístic
  - Executem els 6 problemes amb l'altre heurístic

Un cop fets els experiments i recopilades totes les dades obtenim els resultats de les Taules 7 i 8.

HEURISTIC MINIMITZAR MAX TRANSMITTING TIME		
SEED	TOTAL TRANSMITTING TIME	PROCESSING TIME (ms)
1569741360	1389788	52120
1785505948	1408964	40588
516548029	1508598	21293
1302116447	1340974	28944
1368843515	1525593	6970
663681053	1465568	14544

Taula 7: Resultats de l'experiment 5 per el primer heurístic

HEURISTIC MINIMITZAR MAX TRANSMITTING TIME		
SEED	TOTAL TRANSMITTING TIME	PROCESSING TIME (ms)
1569741360	474667	119240
1785505948	487664	104583
516548029	496743	120656
1302116447	465329	97954
1368843515	459845	112004
663681053	461563	120078

Taula 8: Resultats de l'experiment 5 per el segon heurístic

	H1		**H2**	
	TRANS TIME	PROCES TIME (ms)	TRANS TIME	PROCES TIME (ms)
MITJA	1439914.2	27409.8	474301.8	112419.2

Taula 9: Mitjanes dels resultats obtinguts amb els dos heurísitcs

Com és pot veure a la Taula 9 el temps total de transmissió millora significativament amb el segon heurístic. En canvi, el temps d'execució és considerablement major comparat amb el primer heurístic.

3.6 Experiment 6

En aquest experiment busquem comparar els temps tant d'execució com de transmissió total de fitxers, entre els dos criteris que ens planteja el problema utilitzant l'algorisme de *Simulated Annealing*.

- **Observació:** L'ús d'un heurístic o l'ús de l'altre proporciona solucions diferents que pot comportar a temps d'execució o de transmissió total diferents.
- **Plantejament**: Executem els mateixos problemes emprant l'heurística que busca minimitzar el temps màxim de transmissió d'un servidor i després emprant l'heurística que busca minimitzar la suma de quadrats de temps de transmissió de cada servidor.
- **Hipòtesis:** Al intentar equilibrar la càrrega en el segon heurístic, el temps de transmissió total quan l'utilitzem serà més petit que quan només intentem minimitzar el màxim.
- **Metodologia:**
  - Generem 6 problemes aleatoris
  - Executem els 6 problemes amb un heurístic
  - Executem els 6 problemes amb l'altre heurístic

Un cop fets els experiments i recopilades totes les dades obtenim els resultats de les Taules 10 i 11.

HEURISTIC MINIMITZAR MAX TRANSMITTING TIME		
SEED	TOTAL TRANSMITTING TIME	PROCESSING TIME (ms)
1569741360	545140	76002
1785505948	562065	73021
516548029	588677	74688
1302116447	538590	73047
1368843515	542776	74124
663681053	546341	74576

Taula 10: Resultats de l'experiment 6 amb el primer heurístic

HEURISTIC MINIMITZAR MAX TRANSMITTING TIME		
SEED	TOTAL TRANSMITTING TIME	PROCESSING TIME (ms)
1569741360	472592	56146
1785505948	486038	53299
516548029	497954	53640
1302116447	466278	53420
1368843515	458410	55054
663681053	463217	54811

Taula 11: Resultats de l'experiment 6 amb el sogon heurístic

	H1		**H2**	
	TRANS TIME	PROCES TIME (ms)	TRANS TIME	PROCES TIME (ms)
MITJA	553931.5	74243	474081.5	54395

Taula 12: Mitjanes dels resultats obtinguts amb els dos heurístics

Com és pot veure a la Taula 12 i, igual que a l'experiment 5, el temps total de transmissió millora significativament amb el segon heurístic. En canvi, a diferència de l'experiment 5, el temps per trobar la solució és lleugerament menor al utilitzar el segon heurístic quan utilitzem *Simulated Annealing*.

3.7 Experiment 7

En aquest experiment busquem observar com evolucionen tant el temps per trobar una solució com el temps total de transmissió quan augmentem de 5 en 5 el numero mínim de replicacions.

- **Observació:** Augmentar el mínim número de replicacions fa variar el numero de fitxers que pot tenir un servidor, per tant els usuaris tindran més servidors als que accedir. El que pot variar els resultats obtinguts.
- **Plantejament**: Executem els mateixos problemes emprant les dos heurístiques i veurem com afecta anar augmentant el mínim número de replicacions.
- **Hipòtesis:** Al poder accedir a més servidors des de els usuaris ja que els fitxers poden estar més vegades repetits i estar a més servidors, veurem una disminució en el temps total d'enviaments i un augment en el temps d'execució.

- **Metodologia:**
  - Generem 5 problemes aleatoris. Cada problema augmenta en 5 el mínim de replicacions.
  - Executem els 5 problemes amb un heurístic i *Hill Climbing*.
  - Executem els 5 problemes amb l'altre heurístic i *Hill Climbing*.

Un cop fets els experiments i recopilades totes les dades obtenim els resultats de la Taula 13.

HEURISTIC MINIMITZAR MAX TRANSMITTING TIME			
SEED	TOTAL TRANSMITTING TIME	PROCESSING TIME (ms)	NUM REPLICACIONES
1569741360	1600486	19967	5
1785505948	1499789	49426	10
516548029	1499786	59798	15
1302116447	1396327	169480	20
1368843515	1461502	299790	25
HEURISTIC MINIMITZAR SUMA DE QUADRATS			
SEED	TOTAL TRANSMITTING TIME	PROCESSING TIME (ms)	NUM REPLICACIONES
1569741360	473853	122370	5
1785505948	282073	299827	10
516548029	235174	941977	15
1302116447	174552	975223	20
1368843515	164137	1306752	25

Taula 13: Resultats de l'experiment 7

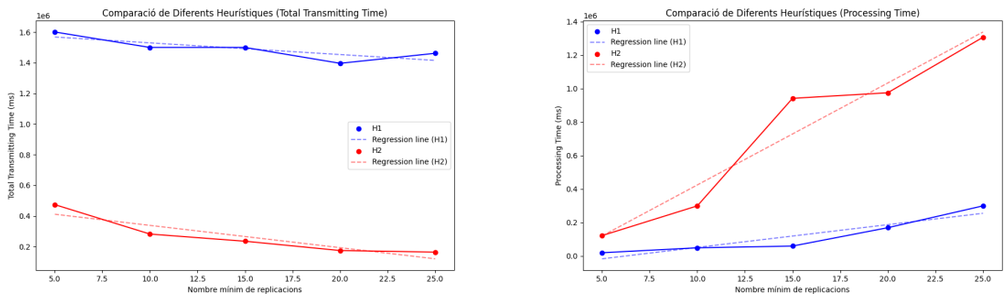

Figura 7: Gràfiques de l'evolució del temps total de transmissió i el temps d'execució

Com havíem previst a la hipòtesi i es pot veure clarament a la figura 7, sense importar l'heurístic emprat, el temps d'execució augmenta considerablement, tot i que de forma més significativa amb el segon heurístic. També es pot veure com, al incrementar el mínim de replicació, el temps total de transmissió es fa més petit.

4 Conclusions

4.1 Comparació dels heurístics (experiment 8)

Minimitzar el temps màxim de transmissió d'un servidor

- Avantatges:

  Temps d'execució relativament menor en comparació amb l'estratègia de minimitzar la suma de quadrats dels temps de transmissió. Pot conduir a solucions on s'optimitza el temps de transmissió del servidor amb màxim temps de transmissió, la qual cosa pot ser beneficiós en escenaris on es prioritza l'equitat en el temps de resposta per a tots els usuaris.

- Desavantatges:

  Pot no tenir en compte de manera òptima la càrrega de treball de tots els servidors, la qual cosa podria resultar en un desequilibri en la distribució de la càrrega i, per tant, en una eficiència global menor en termes de temps total de transmissió dels fitxers. Sembla tenir tendència a generar més mínims locals degut a que si no es pot rebaixar la carrega del servidor amb cost màxim, no és possible generar una millor solució.

Minimitzar la suma de quadrats dels temps de transmissió de cada servidor:

- Avantatges:

  Permet equilibrar de manera més efectiva la càrrega entre els servidors, la qual cosa condueix a una millor utilització dels recursos i a una distribució més equitativa del temps de transmissió entre els servidors.

  Proporciona solucions on el temps total de transmissió és significativament menor, la qual cosa és crucial per a optimitzar l'eficiència global del sistema.

- Desavantatges:

  Temps d'execució considerablement major en comparació amb l'estratègia de minimitzar el temps màxim de transmissió d'un servidor, la qual cosa podria ser problemàtica en aplicacions on es requereix una resposta ràpida.

Atès que el segon heurístic resulta en un temps total de transmissió fins a tres vegades menor, el seu avantatge respecte al primer heurístic és clar. No obstant això, continua sent important considerar el temps d'execució i la distribució de la càrrega entre els servidors en triar l'heurístic més adequat per a un problema específic. En general, si el temps total de transmissió és crític i es pot tolerar un temps d'execució més llarg, llavors minimitzar la suma de quadrats dels temps de transmissió de cada servidor pot ser la millor opció. D'altra banda, si l'eficiència computacional és prioritària i es pot acceptar un cert desequilibri en la càrrega entre els servidors, llavors minimitzar el temps màxim de transmissió d'un servidor podria ser més apropiat, a més a més pel primer heurístic com hem vist a l'experiment 2 utilitzar l'estratègia *Greedy* per generar les solucions inicials també pot ser una manera per obtenir solucions més bones.

4.2 Comparació dels algorismes

En primer lloc, després de realitzar una anàlisi dels resultats obtinguts en aplicar els algorismes de *Hill Climbing* i *Simulated Annealing* en el nostre problema, s'evidencia clarament que *Simulated Annealing* supera significativament a *Hill Climbing* en termes d'eficiència i qualitat de la solució. Encara que tots dos algorismes són tècniques de cerca local, *Simulated Annealing* mostra una capacitat superior per a escapar d'òptims locals i trobar solucions més pròximes a l'òptim global.

Durant les nostres proves, observem que *Simulated Annealing* va aconseguir trobar solucions que minimitzaven tant el temps de transmissió dels fitxers per al servidor amb el màxim temps de transmissió com el temps total de transisió, mentre que *Hill Climbing* tendia a quedar-se atrapat en òptims locals i oferia solucions pitjors en comparació. Això suggereix que, en el nostre context específic, on la qualitat de la solució està lligada al temps de transmissió dels fitxers i al balanç de càrrega entre els servidors, *Simulated Annealing* és l'elecció més adequada.

Per tant, basant-nos en els resultats obtinguts, concloem que *Simulated Annealing* és una opció més efectiva en la resolució del nostre problema. La seva capacitat per explorar l'espai de cerca de manera més completa i trobar solucions de major qualitat el converteix en l'opció preferida per a optimitzar la distribució de fitxers en el nostre sistema de fitxers distribuït.

5 Treball d'innovació

5.1 Descripció del tema

AlphaStar és un sistema d'intel·ligència artificial desenvolupat per DeepMind capaç de jugar a un gran nivell al joc StarCraft II. Utilitza xarxes neuronals profundes i diferents tècniques d'aprenentatge per aprendre estratègies complexes. Ha superat jugadors professionals humans, marcant un avenç significatiu en la IA en jocs d'estratègia en temps real.

5.2 Divisió de la feina

Primer vam realitzar una pluja d'idees per decidir quin tema tractaríem. Vam optar per investigar sobre AlphaStar i vam pensar en possibles temes sobre els que parlar envers aquesta IA.
Aquest es un conjunt de temes sobre els quals podriem parlar:

- Cerca d'informació sobre el desenvolupament de l'algorisme de AlphaStar.
- Recerca sobre els fonaments teòrics de la intel·ligència artificial aplicada als jocs.
- Dades sobre l'entrenament i les tècniques d'aprenentatge utilitzades en el projecte.
- Recerca sobre la implementació tècnica de AlphaStar, incloent-hi detalls sobre l'arquitectura de xarxes neuronals utilitzades.
- Cerca d'informació sobre els entorns de simulació utilitzats per a l'entrenament i avaluació del sistema.
- Recopilació de dades sobre els desafiaments i limitacions trobats durant el desenvolupament de AlphaStar.
- Cerca d'informació sobre els resultats i assoliments aconseguits per AlphaStar en competicions i proves.
- Recerca sobre les implicacions ètiques i socials de l'aplicació d'intel·ligència artificial en jocs competitius.
- Recopilació de dades sobre les reaccions i opinions de la comunitat científica i de jugadors davant el desenvolupament de AlphaStar.

Fins al moment tota la feina l'hem fet en conjunt però a partir d'ara cada integrant del grup s'encarregarà de desenvolupar alguns d'aquests temes per separat.

5.3 Referències

1. **AlphaStar: Mastering the real-time strategy game StarCraft II:**
   https://deepmind.google/discover/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii/
   [Data de consulta: 29-03-2024]
2. **AlphaStar: Inteligencia artificial de DeepMind que ha logrado ganar 10-1 a profesionales en StarCraft II:**
   https://www.xataka.com/robotica-e-ia/alphastar-inteligencia-artificial-deepmind-que-ha-logrado-ganar-10-1-a-profesionales-starcraft-ii
   [Data de consulta: 12-04-2024]
3. **AlphaStar (software):**
   https://en.wikipedia.org/wiki/AlphaStar_(software)
   [Data de consulta: 29-03-2024]
4. **AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning:**
   https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/
   [Data de consulta: 29-03-2024]
5. **Alguns vídeos:**
   - **How AlphaStar Became a StarCraft Grandmaster - AI and Games #48:**
     https://www.youtube.com/watch?v=lPERfjRaZug
     [Data de consulta: 12-03-2024]
   - **Las CRÍTICAS tras la victoria de AlphaStar - Data Coffee #10:**
     https://www.youtube.com/watch?v=M3nn3K7u1R4
     [Data de consulta: 29-03-2024]