

Documentació de la pràctica de cerca local

Logística de transports

**Laboratori d'Inteligència Artificial
2n quadrimestre – curs 2010/11**

Jordi Llamas Pons

Felip Moll Marquès



FIB

Facultat d'Informàtica
de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Índex

Definició del problema.....	1
Estructura de dades.....	2
Matriu.....	2
Peticions.....	3
Llista de peticions.....	3
Camions.....	4
Graella HCP.....	4
Llista d'endarrerits.....	5
Classes del programa.....	6
Variables i dades globals :: Global.java.....	6
Elements de AIMA.....	8
Implementació de l'estat.....	8
Representació de l'estat.....	8
Mida de l'espai de cerca.....	9
Algorismes de generació de l'estat inicial.....	10
Funcions Heurístiques.....	12
Beneficis màxims.....	12
Mínima diferència de hores.....	14
Generació de Successors.....	15
Elecció d'operadors.....	15
Algorismes de generació.....	17
Experiments.....	18
Experiment 1.....	18
Resultats.....	18
Conclusions.....	19
Experiment 2.....	20
Resultats.....	20
Conclusions.....	21
Experiment 3.....	22
Resultats.....	22
Experiment 4.....	23
Resultats.....	23
Conclusions.....	23
Experiment 5.....	24
Resultats.....	24
Conclusions.....	24
Experiment 6.....	25
Resultats.....	25
Conclusions.....	26
Experiment 7.....	27
Resultats.....	27
Conclusions.....	27
Experiment 8.....	28
Resultats.....	28
Conclusions.....	28
Funcionalitats extra.....	29
Interfície del programa.....	29
Classe externa.....	32
Codi font.....	32

Definició del problema

Ens demanen realitzar la gestió d'una empresa de transports. Aquesta empresa (nosaltres) rebra al final de cada dia una llista de peticions a entregar al dia següent. Cada petició tindrà un identificador de producte, una quantitat a entregar, una hora límit d'entrega i finalment, un centre de producció. L'objectiu per tant serà entregar aquestes peticions d'acord amb unes restriccions imposades per el problema.

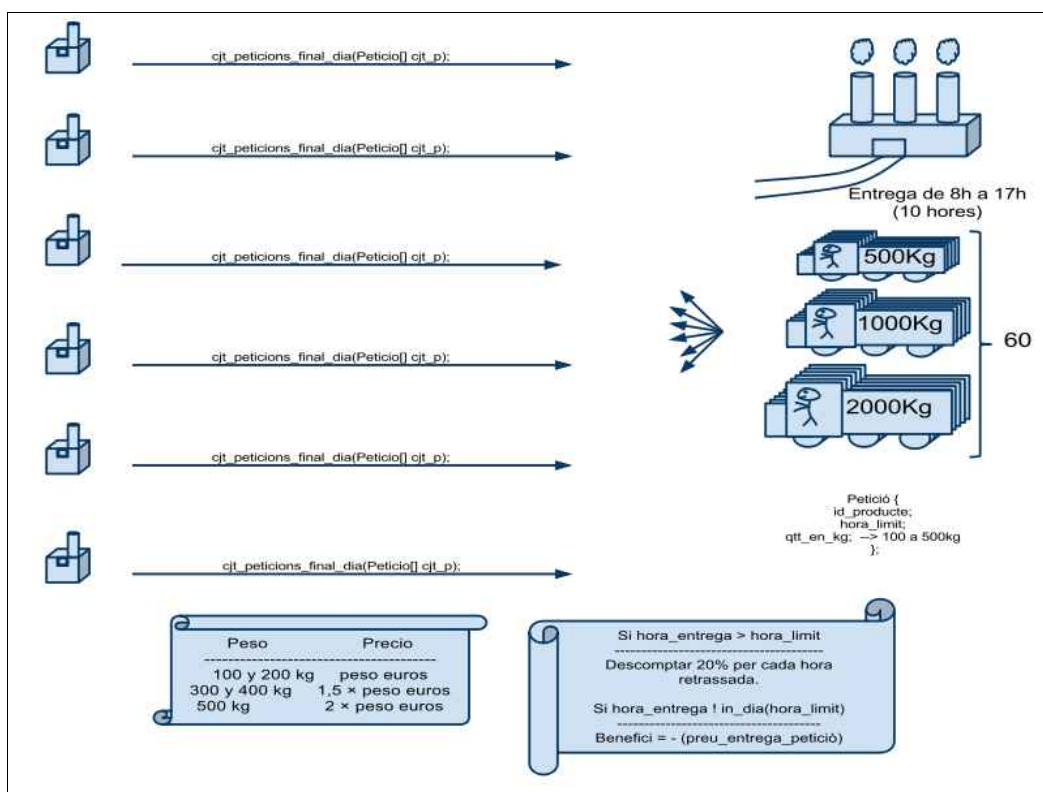
Hi haurà dos objectius en quant a l'entrega de productes:

1. Maximitzar els beneficis
2. Minimitzar la diferència absoluta entre la hora d'entrega límit i hora d'entrega efectiva de cada petició.

Aquests dos objectius ens marcaran dues heurístiques que haurem d'implementar.

A més hi haurà altres restriccions, que enunciem i descrivim gràficament en el següent esquema:

1. Hi ha 6 centres de producció als que enviar les peticions. Cada centre té la seva llista de peticions, que s'especifica al principi del problema.
2. Tenim 60 camions per fer els transports. Cada camió pot ser de tipus 500kg, 1000kg o 2000kg. Al principi del problema s'especifica el nombre de cada tipus.
3. Es realitza un transport cada hora, de 8h a 17h, i s'envia un camió diferent cada vegada.
4. Es defineix un benefici per cada entrega realitzada. Veure detall al gràfic.



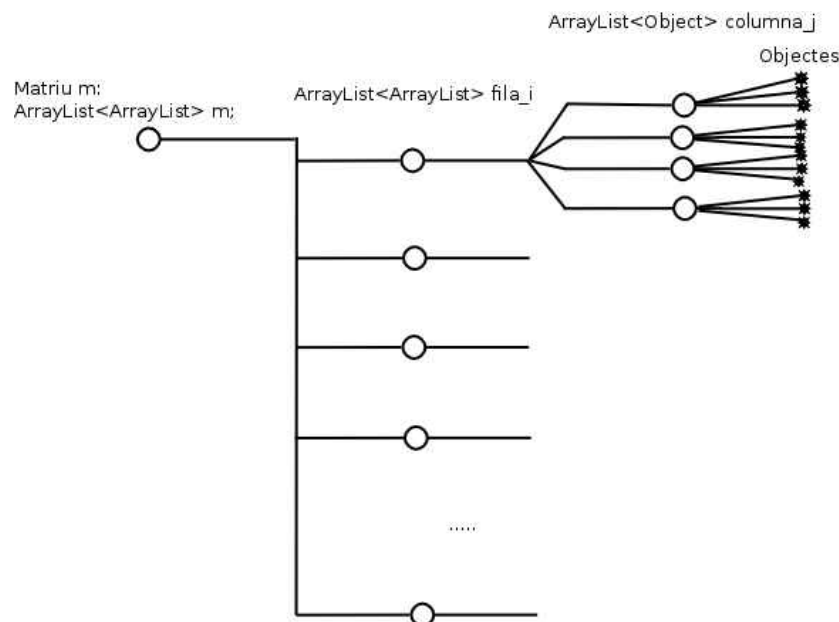
Estructura de dades

L'enunciat del problema ens ha portat a definir les següents estructures de dades.

Matriu

Java 1.6 no implementa actualment cap classe que permeti disposar d'una matriu d'objectes de dades. Degut a les estructures que hem decidit necessitàvem una estructura d'aquest tipus, i per tant la hem implementat.

La forma d'implementació és senzilla. Es tracta d'un ArrayList típic de Java que conté ArrayLists i que representen les files. Cada fila, és ella mateixa un altre ArrayList on hi ha fins a N elements, que representen les columnes. Aquests elements tornen a ser ArrayLists, però aquesta vegada ho són d'objectes. Això ens aporta molta flexibilitat: poder afegir un a classe objecte de Java, és a dir, qualsevol cosa dins cada casella de la matriu! Podem posar-hi altres llistes, objectes sols, etc.



Els beneficis de fer servir ArrayLists són els costos d'accés.

- El cost d'inserir un element dins una ArrayList és, segons les especificacions $O(1)$.
- L'accés a un element concret a la classe ArrayList de Java, per un nombre de fila i columna donat, és també constant $O(1)$.
- Les operacions per comprovar la mida i si l'ArrayList és buit són $O(1)$.
- Totes les altres operacions són $O(N)$.

L'únic inconvenient, punt a tenir en compte o particularitat, és que ens saltem tota comprovació de tipus. Això ho fem perquè permetem afegir Objectes genèrics al final.

L'estructura presentada per tant és tan insegura com els punters a C o C++, i s'ha d'anar molt en compte quan es programa. A més ens saltem conscientment el paradigma de l'orientació a objectes.

No obstant aquest perill, creiem una bona idea fer servir la matriu, sobretot pels costos d'accés.

A més, haver implementat aquesta matriu ens fa pensar que potser Java no és el millor llenguatge per programar programes d'intel·ligència artificial, o en general programes que cerquin l'eficiència, perquè protegir molt a l'usuari, en el seu cas, significa no tenir punters (almenys de forma directe), i al no tenir punters estas impeding la millora en quant a l'eficiència.

Finalment volem dir que hem hagut d'afegir a cada classe que fa servir la matriu, la línia `@SuppressWarnings ("unchecked")`. Això és perquè Java ens avisa de que no es fan comprovacions de tipus.

Referències:

A programmer's guide to Java certification: a comprehensive primer, Volum 1 – Cap 11.4, pàg. 443,

<http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html>

Peticions

Les peticions tenen bàsicament tres atributs. Aquests són: identificador de producte, quantitat de producte i hora límit d'entrega.

A efectes pràctics, després d'estudiar l'enunciat i abstreure'ns de l'àmbit dels transports, centres i camions, ens hem adonat que l'identificador del producte no el necessitàvem. Podíem tractar cada petició tinguéssim o no el tipus de producte que anava a la petició, ja que en el fons aquest paràmetre no té cap influència a les restriccions del problema. Per això hem decidit eliminar l'atribut, i per tant, si tenim N Peticions, estalviar N enters (ja siguin de 32 o 64 bits) i en conseqüència estalviar una mica de memòria RAM.

Llista de peticions

La llista de peticions que ens donen tots els 6 centres de producció a final de dia han d'estar emmagatzemades en algun lloc. Primer de tot havíem pensat en crear una simple llista de peticions, amb potser algun ordre, però en el que al cap i a la fi l'hauríem de recórrer alguna vegada en temps $O(N)$.

Per això hem decidit crear una matriu de peticions, d'ara endavant “matriu de peticions” o “graella de peticions”. Aquesta matriu té:

- A les files, les hores límit per les peticions, que van de 08h a 17h. Per tant té 10 files.
- A les columnes hi ha cadascun dels sis centres de producció, de CP1 a CP6. Té 6 columnes.
- A cada casella, una llista de peticions per aquell CP i hora límit.

Aquesta matriu és una mica especial, ja que dins cada casella no hi ha una petició, sinó que hi ha N peticions, per tant és una Matriu de llistes de peticions de 10×6 . Cadascuna d'aquestes llistes és una llista enllaçada simple, realitzada amb la classe “ArrayList” de Java 1.6 que permet accés amb cost $O(1)$. Hem de fer notar que en aquest cas tenim informació duplicada i que és la hora límit de les peticions. Això passa perquè dins cada petició s'emmagatzema la hora límit de la petició, però a la matriu aquesta petició és emmagatzemada a una casella que ja ens proporciona informació sobre la hora límit. Ho feim així per tenir accés directe a les peticions que ens interressi i per no haver de fer cerques que costarien almenys $O(\log)$. En aquest cas ens costa $O(1)$.

Al codi, la matriu de Peticions s'anomena “PETICIONS” i s'ha implementat amb la Matriu descrita anteriorment en aquest capítol.

PETICIONS forma part de l'enunciat del problema com hem dit al primer apartat, juntament amb totes les altres restriccions.

Graella pedidos (hora límit, centre de producció i petició)	CP1	CP2	CP3	CP4	CP5	CP6
08h	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 300kg, Prod2:}	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	
09h		{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 300kg, Prod2:}	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 300kg, Prod2:}
10h	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 300kg, Prod2:}	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}	{Prod1: 100kg, Prod2: 200kg, Prod4: 400kg}	{Prod1: 100kg, Prod2: 200kg, Prod4: 500kg}
11h						
12h						
13h		---		---		
14h			---			
15h		---		---		
16h		---	---			
17h						

Camions

Un camió és un objecte que representa el que el seu nom indica. Té una capacitat de càrrega màxima que anomenarem “tipus de camió”, i que ve definida per l'enunciat. També transporta peticions, i en el nostre cas, manté una llista (ArrayList) de peticions. Finalment d'aquesta llista es deriva un valor de càrrega que transporta actualment, valor que lògicament guardem en un atribut de tipus enter per no haver de fer un recorregut $O(N)$ per la llista de peticions que transporta, cada vegada que es demani la càrrega del camió.

Graella HCP

Aquesta graella és una matriu que ens serveix per emmagatzemar Camions destinats a un centre en particular i a una hora també determinada. Podríem haver guardat una llista de Camions i dins cada camió la hora límit i el centre al que anava destinat, però com en el cas de la matriu de peticions no disposaríem d'accés directe. Això ens dificultaria la tasca de programar els operadors i els faria més ineficients.

La graella HCP és una Matriu definida com segueix:

- A les files, les hores de servei, que van de 08h a 17h. Per tant té 10 files.
- A les columnes hi ha cadascun dels sis centres de producció, de CP1 a CP6. Té 6 columnes.
- A cada casella, un i només un camió.

El control de que dins cada casella de la matriu només hi hagi un sol camió, és tasca nostra d'assegurar-ho. Recordem que la matriu ens permet llibertat, i la llibertat a banda dels beneficis que ens aporta requereix anar amb compte.

Graella H-CP	CP1	CP2	CP3	CP4	CP5	CP6
08h	T1	T2	T1	T1	T1	T3
09h						
10h						
11h						
12h						
13h			...			
14h						
15h						
16h						
17h						T2

Llista d'endarrerits

Aquesta llista és una matriu com les explicades anteriorment, però que només té una fila. Manté el nombre de columnes igual al nombre de centres que tenim. Emmagatzema dins cada casella una llista de Peticions que no han pogut ser entregades al dia actual, és a dir, que no hi caben dins la Graella HCP i que hauran de ser entregades a les 08h del dia següent.

Haguéssim pogut optar per posar una fila més a la graella HCP però ho hem volgut separar perquè en el cas de HCP s'hi guarden Camions, i en aquest cas directament hi guardem Peticions. Si ho haguéssim mesclat hagués complicat el còdi i hagués fet difícil la comprensió i programació.

Amb aquestes poques estructures de dades ja estem preparats per començar a programar les classes necessàries per representar l'estat, les heurístiques, els operadors, i per fer la connexió amb AIMA.

Classes del programa

En aquest apartat descriurem les classes i les funcions que creuem que siguin necessàries comentar.

Variables i dades globals :: Global.java

Aquesta classe proporciona variables globals i estructures de dades que no són canviants. L'objectiu és centralitzar tota la informació que l'usuari pot voler modificar i a més escriure algunes funcions com la generació de les peticions inicials que no formen part d'una classe concreta.

Hem volgut fer el nostre programa el màxim de modificable possible. Això significa que en tots els llocs on hàviem de posar una constant, per exemple el nombre de centres de l'enunciat del problema, ho hem fet posant el nom d'una variable continguda dins Global.java, de la forma Global.NOM_VARIABLE. Així si volem modificar les condicions de l'enunciat només hem de modificar un sol nombre en un únic fitxer. Igual ho hem fet amb totes les condicions de l'enunciat que poden variar en funció de l'entrada de dades inicial de l'usuari, per exemple nombre de camions de cada tipus.

Principalment les variables que tenim són:

Variable	Descripció
T1, T2, T3	Enters que indiquen la càrrega de cada un dels tres tipus de camions.
nT1, nT2, nT3	Enters que indiquen quants camions hi ha de cada tipus.
H_INI, H_FI, HORES_SERVEI	Constants definides segons l'enunciat. Són les hores en que s'inicia el servei, en que s'acaba, i el total d'hores que disposem al dia per fer transports.
N_CENTRES	Nombre de centres segons l'enunciat. Per defecte li donem valor 6.
preus_transport[5]	És un vector on hi ha les constants definides per l'enunciat en quant als preus de cada petició. L'accés es directe, associant l'index 0 al valor de les peticions de 100kg, 1 al de 200kg, 2 al de 300kg, i successivament fins a l'últim.
pesos_peticions[5]	Són els possibles pesos de peticions que es defineixen a l'enunciat.
probabilitatsHores[HORES_SERVEI], probabilitatPesos[5]	Són tres vectors de probabilitats. El sumatori dels valors que contenen és 100. Hi ha els valors que es necessiten per crear les peticions

probabilitatsCamions[3]	per els experiments. L'índex del vector probabilitatsHores + 8 representa la hora de servei i el valor la probabilitat que té una petició de tenir aquesta hora límit. Cada posició del vector de prob. de pesos coincideix amb l'índex de pesos_peticions. Igual amb les probs. dels camions.
Int Steps, stiter, k, lamb	Valors paramètrics variables segons l'input d'entrada per l'algorisme de simulated annealing.
LINEAL, MAX_COMPACT	Valors d'una enumeració que representen estratègies heurístiques.
PETICIONS	Matriu de peticions explicada a l'apartat d'estructura de dades

Funcions rellevants:

IniciaProblemaDefault(int numPeticions, boolean aleatori, int probsH[], int probsP[])

Si (aleatori == true)

Genera numPeticions i les associa a cada centre amb probabilitat uniforme.

Hi assigna un pes entre 100 i 500kg, i una hora límit entre 8 i 17h. Escollir el pes i hora límit es realitza mitjançant un algorisme probabilístic creat per nosaltres, que com a suport fa servir els dos vectors de probabilitats. La idea de l'algorisme és omplir un sac d'hores i agafar-ne una: per cada hora possible de 08h a 17h, mirar quina probabilitat té (de 0 a 100) d'aparèixer, i posar-ni tantes com siguin al sac. Llavors amb generar un índex amb la funció random de probabilitats uniformes, indexar el sac i agafar una hora. Per els pesos es segueix el mateix procediment.

Llavors insereix la petició a la casella corresponent de PETICIONS.

Aquest algorisme és costós en temps i en espai, però és la forma més intuïtiva de realitzar-lo, i a més, només es fa una vegada al principi de l'execució del problema. Per tant no influeix en els resultats de temps.

Si (aleatori == false)

Omple PETICIONS amb una llista de peticions definida a mà.

Elements de AIMA

Implementació de l'estat

Representació de l'estat

El problema de l'enunciat i l'abstracció d'aquest ens dona lloc a representar l'estat mitjançant les estructures de dades que hem explicat abans i a pensar una combinació d'elements tals que puguin definir adequadament tot l'entramat de la pràctica.

Els elements d'aquest estat es combinen tots dintre de la classe Estat.java, i són els següents:

Matriu de Peticions

Si llegim l'enunciat ens adonem que hem d'entregar una sèrie de peticions que ens donen els 6 centres de producció a final del dia. Aquestes peticions es troben dins una llista de peticions que representem amb la matriu PETICIONS.

Podríem dir intuïtivament que PETICIONS forma part de l'estat, ja que una vegada anem assignant peticions als respectius camions sembla que han de desaparèixer d'aquesta matriu i posar-se dins els camions. En aquest cas la matriu de peticions està duplicada en tots els estats diferents, i això implica l'ús d'una bona quantitat de memòria RAM.

Una implementació més eficient seria la que fa servir els “pseudo-punters” de Java. Ens basem en que cada objecte té un identificador únic, i per tant, al moure una petició de la llista de peticions inicials a un camió, el que realment feim és copiar l'identificador. D'aquesta manera no perdem la forma inicial de la matriu PETICIONS i tenim que a dins dels camions només hi ha punters a peticions reals. D'aquesta forma, per molts estats que tinguem, l'únic que tenim duplicat són simples punters a peticions reals i la matriu PETICIONS resta intacte.

Conclusió: la llista inicial de peticions que ens donen els centres de producció, no es copia a cada estat diferent.

Matriu HCP

Necessitem guardar tots els camions que omplim de peticions. Podríem tenir una llista de camions, però llavors hauríem de guardar dins cada camió la hora en que fa l'entrega i el centre de producció al que es dirigeix. Per millorar això podríem definir tantes llistes de camions com centres de producció, i llavors només hauríem de guardar dins cada camió la hora en que fa l'entrega. El problema és que si volem afegir noves peticions hauríem de recórrer aquesta llista de camions i el cost seria en cada cas lineal, $O(N)$. Per això la deducció instantània que fem, és que necessitem una matriu on guardar els camions.

L'estructura matriu HCP, explicada a l'apartat anterior, forma part de l'estat. Conté la informació més important de l'estat i representa la informació final que necessitem per resoldre el problema.

Cada casella de la matriu pot contenir el valor null, que vol dir que no hi ha cap camió assignat a aquella hora i centre de producció, o d'altra banda pot contenir un Camió.

L'accés directe a aquesta matriu sempre serà $O(1)$, cosa que ens facilita molt totes les tasques de lectura que requerim, així com les d'escriptura. Seria costós per exemple afegir un camió a mitat del dia dins una llista enllaçada.

Llista d'endarrerits

A banda de tots els camions que contenen peticions i que podem assignar a totes les hores del dia en que es dona servei, per cada un dels centres de producció, és possible que ens quedin peticions que no puguem assignar a camions degut a que estan plens o que ens empitjoren la solució final.

Totes aquestes peticions que no es poden assignar, han de ser emmagatzemades. Aquesta és la funció de la llista d'endarrerits. Seguint amb la filosofia de la matriu i l'accés constant, aquesta llista és una matriu de 1 fila i nombre de centres de producció columnes. A cada casella hi ha una llista de peticions que no cal tenir ordenades i que són aquestes peticions sense assignar.

Camions

A banda de tot el comentat anteriorment, l'enunciat ens posa la restricció d'haver de tenir com a màxim un nombre de camions de cada tipus determinat. Hem sentit gent que havia decidit tenir una llista de camions pre-creats i anar-lo's utilitzant a mesura que els necessites. Aquest seria el cas en que la matriu HCP estès sempre plena de camions, i on s'haguessin de fer intercanvis per poder moure camions de banda a banda.

Nosaltres hem reduït el problema a “tres enters”. Suposem que tenim un nombre inicial de camions. en un moment determinat podem decidir afegir una petició a un camió d'un tipus. Llavors el que farem serà mirar si l'enter corresponent al tipus és major que 0, i si ho és llavors podrem crear el camió i restarem a aquest enter una unitat. La gran avantatge és que no tenim sempre els 60 camions creats (tal com diu l'enunciat) i guardats dins el garatge, sinó que els anem creant sota demanda. De la forma que es creen, s'eliminen. Els tres enters s'inicialitzen al principi, quan no hi ha cap camió a la matriu, amb el valor Global.nTi per $i:\{1,2,3\}$. Els noms de les variables són: numCamionsTipusi també per $i:\{1,2,3\}$.

Mida de l'espai de cerca

Gràcies a la eficient implementació d'aquest estat, hem pogut determinar de forma intuïtiva l'espai de cerca del nostre problema.

En aquest cas podem dir informalment que l'espai serà aproximadament el següent:

(permutacions de 60 elements de tres tipus, sense repetició) * 6*(permutacions sense repetició de N elements agafats en 10 posicions).

Més descriptivament, la mida de l'espai resulta de combinar tots els possibles camions que tenim dins la matriu HCP, i, per cada combinació, per cada centre, agafar totes les peticions del centre i col·locar-les de totes les formes possibles dins els camions del centre. Com podem veure l'espai és molt gran, i depèn de les dades d'entrada del problema.

Algorismes de generació de l'estat inicial

Per generar l'estat inicial, hem estudiat i implementat dues tècniques diferents. Aquestes dues tècniques es poden escollir en temps d'invocació del programa mitjançant la variable:

```
-g <g>          Estratègia generació estat inicial <lineal,maxcompact>
```

Aquest paràmetre modifica la variable que se li passa a la funció `TransportsHillClimbingSearch`. Internament el paràmetre pot valer `Global.LINEAL` o `Global.MAX_COMPACT`.

La primera tècnica, que hem anomenat “LINEAL”, ha consistit en el següent:

- Per cada centre de producció
 - Per cada petició a assignar a aquell centre (matriu PETICIONS)
 - Assignar la petició al primer camió lliure, començant a mirar a les 08h i acabant a les 17h. Si no es pot assignar, es col·loca a la matriu d'endarrerits.

La bondat d'aquesta generació és pot considerar elevada per els requisits de maximitzar beneficis, però molt dolenta per el requisit de minimitzar diferència en hora d'entrega efectiva i hora límit. Això és perquè en gairebé tots els casos, la funció posa les peticions a un camió amb entrega abans de la hora límit d'entrega.

Vegem un exemple de fragment d'execució de l'estratègia LINEAL:

Llista de peticions inicials	Resultat de la generació de l'estat
<p>CENTRE: 1</p> <p>-----</p> <p>->Hora:8</p> <p> ->@Pet. id: 1 , Qtt: 500 , Hl: 8</p> <p> ->@Pet. id: 2 , Qtt: 300 , Hl: 8</p> <p>->Hora:9</p> <p>->Hora:10</p> <p> ->@Pet. id: 3 , Qtt: 100 , Hl: 10</p> <p>->Hora:11</p> <p> ->@Pet. id: 4 , Qtt: 400 , Hl: 11</p> <p>->Hora:12</p> <p>->Hora:13</p> <p> ->@Pet. id: 5 , Qtt: 100 , Hl: 13</p> <p>->Hora:14</p> <p> ->@Pet. id: 6 , Qtt: 200 , Hl: 14</p> <p> ->@Pet. id: 7 , Qtt: 500 , Hl: 14</p> <p> ->@Pet. id: 8 , Qtt: 200 , Hl: 14</p> <p>.....</p>	<p>CENTRE: 1</p> <p>-----</p> <p>->Hora:8</p> <p> ->Camió: 2000kg , carregat amb: 2000kg</p> <p> ->@Pet. id: 1 , Qtt: 500 , Hl: 8</p> <p> ->@Pet. id: 2 , Qtt: 300 , Hl: 8</p> <p> ->@Pet. id: 3 , Qtt: 100 , Hl: 10</p> <p> ->@Pet. id: 4 , Qtt: 400 , Hl: 11</p> <p> ->@Pet. id: 5 , Qtt: 100 , Hl: 13</p> <p> ->@Pet. id: 6 , Qtt: 200 , Hl: 14</p> <p> ->@Pet. id: 8 , Qtt: 200 , Hl: 14</p> <p> ->@Pet. id: 11 , Qtt: 200 , Hl: 15</p> <p>->Hora:9</p> <p> ->Camió: 2000kg , carregat amb: 2000kg</p> <p> ->@Pet. id: 7 , Qtt: 500 , Hl: 14</p> <p> ->@Pet. id: 9 , Qtt: 500 , Hl: 15</p> <p> ->@Pet. id: 10 , Qtt: 300 , Hl: 15</p> <p> ->@Pet. id: 12 , Qtt: 500 , Hl: 16</p> <p> ->@Pet. id: 13 , Qtt: 200 , Hl: 16</p> <p>->Hora:10</p> <p> ->Camió: 500kg , carregat amb: 400kg</p> <p> ->@Pet. id: 14 , Qtt: 400 , Hl: 17</p> <p>->Hora:11</p> <p>->Hora:12</p> <p>->Hora:13</p> <p>->Hora:14</p> <p>.....</p>

Com podem observar, aquest algorisme col·loca al principi tot el que pot, sense importar-li el criteri de mínima diferència horària.

Finalment hem de dir que el cost de generació és de l'ordre de $O(N)$, ja que recorre un sol cop la llista de peticions.

La segona tècnica la hem anomenat "MAX_COMPACT" i ha consistit en el següent:

- Per cada centre de producció
 - Per cada petició a assignar a aquell centre (matriu PETICIONS)
 - Assignar la petició al primer camió lliure però només si es troba a la hora límit de la petició.

La bondat d'aquesta solució no és elevada en el cas que vulguem maximitzar beneficis, ja que hi ha una elevada probabilitat de que no es puguin col·locar totes les peticions a la seva hora i per tant en quedin algunes sense col·locar. D'altra banda, aquesta solució és molt bona si el que es vol és minimitzar la diferència absoluta entre la hora límit d'una petició, i la hora d'entrega efectiva.

En problemes que no tenen un nombre desorbitat de peticions i on és possible que hi hagi una solució perfecta, aquesta estratègia de generació de l'estat inicial segur que donarà la millor solució.

A continuació mostrem l'exemple de fragment d'execució de l'estratègia MAX_COMPACT, amb el mateix input de dades que l'anterior exemple:

Llista de peticions inicials	Resultat de la generació de l'estat
CENTRE: 1 ----- ->Hora:8 ->@Pet. id: 1 , Qtt: 500 , Hl: 8 ->@Pet. id: 2 , Qtt: 300 , Hl: 8 ->Hora:9 ->Hora:10 ->@Pet. id: 3 , Qtt: 100 , Hl: 10 ->Hora:11 ->@Pet. id: 4 , Qtt: 400 , Hl: 11 ->Hora:12 ->Hora:13 ->@Pet. id: 5 , Qtt: 100 , Hl: 13 ->Hora:14 ->@Pet. id: 6 , Qtt: 200 , Hl: 14 ->@Pet. id: 7 , Qtt: 500 , Hl: 14 ->@Pet. id: 8 , Qtt: 200 , Hl: 14	CENTRE: 1 ----- ->Hora:8 ->Camio: 1000kg , carregat amb: 800kg ->@Pet. id: 1 , Qtt: 500 , Hl: 8 ->@Pet. id: 2 , Qtt: 300 , Hl: 8 ->Hora:9 ->Hora:10 ->Camio: 500kg , carregat amb: 100kg ->@Pet. id: 3 , Qtt: 100 , Hl: 10 ->Hora:11 ->Camio: 500kg , carregat amb: 400kg ->@Pet. id: 4 , Qtt: 400 , Hl: 11 ->Hora:12 ->Hora:13 ->Camio: 500kg , carregat amb: 100kg ->@Pet. id: 5 , Qtt: 100 , Hl: 13 ->Hora:14 ->Camio: 1000kg , carregat amb: 900kg ->@Pet. id: 6 , Qtt: 200 , Hl: 14 ->@Pet. id: 7 , Qtt: 500 , Hl: 14 ->@Pet. id: 8 , Qtt: 200 , Hl: 14

Finalment hem de dir que el cost de generar aquesta solució és idèntic que en el cas anterior, $O(N)$, ja que recorre una sola vegada la llista de peticions.

Funcions Heurístiques

A l'apartat anterior, on hem comentat els elements de l'enunciat, hem identificat dos objectius que es volien aconseguir:

Hi haurà dos objectius en quant a l'entrega de productes:

1. Maximitzar els beneficis
2. Minimitzar el desfase entre hora d'entrega límit i hora d'entrega efectiva de cada petició.

Entenem que aquests objectius s'han d'aconseguir en situacions diferents, ja que sinó es podrien combinar amb un sol objectiu que fos “maximitzar els beneficis minimitzant la diferència absoluta entre hora d'entrega límit i efectiva”.

Per aconseguir els dos objectius per separat necessitarem implementar dues heurístiques.

Permetrem fer l'activació de l'heurística en temps d'invocació del programa, amb la opció:

-hbenef	Activar heurístiques de maximització de beneficis
-hhores	Activar heurístiques de minimitzar diferència engre hores limit i d'entrega

Si no s'especifica, per defecte s'activa l'heurístic -hbenef. Per cada heurístic activat, es mostren els resultats per separat.

Beneficis màxims

Aquesta funció heurística es basa en maximitzar els beneficis obtinguts a l'estat final. Per fer-ho hem de tenir en consideració els següents factors:

- Pes de cada petició
- Hora en que s'entrega

Mitjançant solament aquests dos valors aconseguirem calcular l'heurístic.

Com hem mostrat a l'apartat “Definició del problema”, el valor que pot arribar a aportar una petició determinada va en funció del seu pes. Podem trobar la definició d'aquests valors a la constant “preus_transport” que tenim a la classe Global:

```
/*Preus calculats per cada pes possible de peticions*/
public static double preus_transport[] = { 100, 200, 300*1.5, 400*1.5, 500*2 };
```

Aquest vector constant serà indexat segons el pes de la petició per obtenir-ne el valor màxim.

El valor que aporta segons el pes, ha de ser ponderat segons la hora en que s'entrega la petició. Com diu l'enunciat, per cada hora de retràs es descompta un 20% del valor total, i si al final la petició no es pot entregar, el valor perdut és el valor màxim de la petició + el valor perdut per nombre de hores de retràs.

L'algorisme que hem escollit és molt senzill.

Recorre iterativament totes les caselles de la matriu HCP i n'obté el camió associat. Llavors demana al camió quin valor porta. El camió ho pot calcular perquè es troba dins la casella de la matriu HCP, que li proporciona la informació de la hora d'entrega, i perquè té la llista de peticions carregades.

Llavors el camió només ha de recórrer iterativament la llista de peticions i fer el càlcul abans esmentat.

Finalment quan l'algorisme ha acabat amb les caselles de la matriu HCP, executa el càlcul per les peticions que es troben a la matriu d'endarrerits.

La suma d'ambdós procediments és el benefici que s'obté de tot l'estat. El temps de càlcul d'aquest valor és $O(n^3)$ ja que hem de recórrer, per cada casella de la matriu, una llista més de peticions.

Perquè aquest algorisme?

Hem escollit aquest algorisme perquè la senzillesa que ens aporta és proporcional a la qualitat de l'heurística. Es clar que si es tenen 2 estats e_1 i e_2 , i es vol calcular quin és millor respecte els beneficis, només hi ha que calcular els beneficis generats a cada estat i quedar-nos amb el major.

Els efectes de l'heurística per tant no poden ser més clars, es retorna el valor major en tot moment i per tant s'aconsegueix l'efecte desitjat.

No obstant això hem pensat posteriorment a la finalització de la pràctica, si seria possible que existissin heurístiques que no donessin el millor benefici en el moment, però que tinguessin en compte alguns passos futurs per tal de que es maximitzés l'efecte posterior. Com diu la teoria una heurística tant “perfecte” com la nostra pot tenir molts de màxims locals, i per tant podria ser convenient aplicar algun factor de correcció, per exemple no calcular el que es perd amb els endarrerits, o mirar la possibilitat de millora si hi ha molts d'endarrerits. Aquests factors poden eliminar alguns màxims locals. D'altra banda hauríem de passar a solucions on es modifiqués l'algorisme, tals com backtracking. Sembla que s'hauria d'estudiar més profundament el cas, cosa que en s'escapa a l'abast d'aquesta pràctica.

Altres millores que hem pensat és la de reduir el cost quadràtic de l'algorisme. Es podria fer molt senzillament implementant a la classe Camió un double que mantingués actualitzat en tot moment el benefici aportat per aquell camió. Tot i així ho hem plantejat com a millora futura ja que no creiem que actualment influeixi molt en els resultats.

Mínima diferència de hores

Aquesta funció heurística ens ajudarà a trobar el valor de la bondat de la solució respecte a l'objectiu que tenim imposat de minimitzar la diferència entre la hora límit de cada petició respecte a la seva hora efectiva d'entrega.

Els factors que influeixen sobre aquesta heurística són clars. Si associem peticions a hores d'entrega posteriors o anteriors a les de la hora límit, el resultat de l'heurística serà molt negatiu. D'aquí podem extreure la conclusió de que si associem peticions a hores d'entrega iguals a les hores límit de les peticions obtindrem un bon valor, i per tant això ens acostarà a l'objectiu que tenim definit.

Els efectes d'aquesta heurística a la cerca clarament afecten sobre l'objectiu a aconseguir. Tot i així és possible, com en el cas anterior, que l'heurístic sigui molt dràstic i faixi eliminar molts d'estats que podrien ser candidats a ser solució millor. Això és degut com hem comentat anteriorment al problema que té Hill Climbing amb els màxims locals.

Una possibilitat de millora per aquest cas seria per exemple el de modificar l'heurística perquè tingués en compte la possibilitat de millorar. Si per exemple tenim molts de forats a hores on no s'hi ha assignat petició, podríem utilitzar aquest fet com a factor de modificació.

L'algorisme és molt senzill. Realitza el següent procediment:

- Per tots els centres de producció
 - Per cada hora
 - N'obté el camió que hi ha associat. Llavors demana al camió quantes de les peticions que porta no coincideixen amb la hora a la que el camió hi està assignat:
 - `c.getHoresPerdudes(h + Global.H_INI)`

Per totes les peticions endarrerides resta `Global.H_FI` a la hora límit de la petició, i hi suma 14 hores (de les 17h a les 08h del següent dia).

La suma de totes aquestes hores és el resultat de l'heurística.

Com podem veure el cost és $O(n^3) + O(n) = O(n^3)$. Podríem millorar aquest cost com hem comentat en apartats anteriors: afegint un atribut a Camió que mantingués en tot moment el nombre de peticions que no es troben a l'hora a la que el camió està assignat. No obstant mantenir aquest enter seria pitjor que fer la cerca que feim actualment. El motiu és que un camió no pot portar més de 20 peticions, i realitzar, per tant, si suposem que tenim el màxim de camions i peticions a la graella HCP, tindrem que es realitza un màxim de: 6 centres x 10 hores x 20 peticions = 1200 iteracions. Un processador realitza aquesta activitat en un temps ínfim. En canvi, si mantenim l'enter, cada vegada que afegim una petició, que la eliminem, o que canviem un camió haurem d'actualitzar el valor. A més és en la generació d'estats successors on més moviment hi ha, i precisament el que més es mou són peticions.

Generació de Successors

La generació de successors és un element important del nostre problema. Sense generar successors de qualitat mai no podrem tenir bones solucions. La objectiu dels successors és, per tant, generar tants successors vàlids com sigui possible. Un cop s'han generat tots aquests successors es passen a l'heurística que selecciona els millors.

Per trobar successors vàlids hem d'implementar operadors que faixin modificacions a l'estat i que mantinguin la validesa de la solució.

Hem provat dos operadors, i un ens ha donat millors resultats que altre com podrem veure detalladament a l'apartat dels experiments.

Els operadors que hem implementat són un operador anomenat swap, i un anomenat addremove.

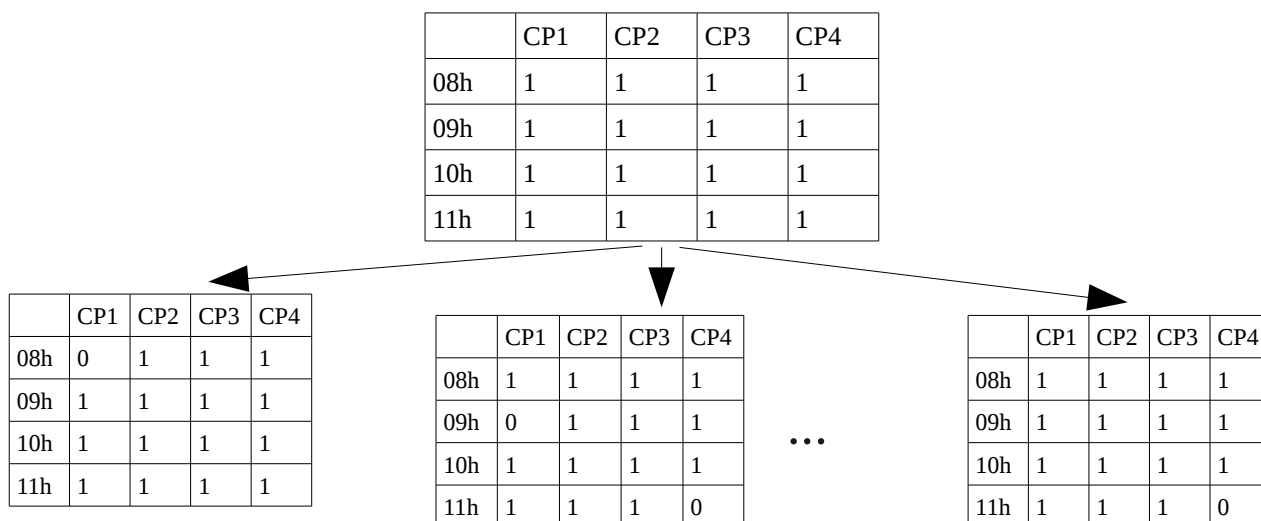
Es poden activar en temps d'invocació del nostre programa, i només es pot activar un o altre:

```
-s <s>                                Tipus d'estratègia de successors
                                     <swap,addrem>
```

Elecció d'operadors

AddRemove: Al principi del projecte vam pensar quins operadors podríem desitjar. Se'ns va ocórrer la possibilitat més senzilla que consistia en generar estats en dos passos. En el primer pas eliminaríem de un en una les peticions que teníem dins l'estat original, i per cada petició eliminada es generava un estat diferent. Així, esquemàticament sortiria una generació similar a això:

Estat inicial:



En el segon pas es seguiria el mateix procediment però s'intentaria posar, a cada casella, una nova petició. En cas de que no es pogués perquè el camió fos ple, s'intentaria canviar el camió per un de nou i de més capacitat, si és que encara n'hi quedaven. Per cada petició afegida, es generava un estat nou. Les peticions s'agafaven i es col·locaven a la llista d'endarrerits seqüencialment.

És en aquest punt on es comproven les condicions d'aplicabilitat. Com hem descrit, a cada inserció o eliminació s'actualitzava la càrrega del camió, i si no hi cabia o el camió quedava buit s'eliminava un camió o s'afegia a l'estat, a la variable *numCamionsTipusi*, per $i:\{1,2,3\}$.

El factor de ramificació podem deduir que coincideix com a molt amb $N + N$ peticions, que és exactament el nombre de passos que fa l'algorisme.

Swap: Més endavant vam decidir perfeccionar els operadors i vam pensar en la possibilitat d'ajuntar els estats que generava add i remove i fer-ho tot en un sol pas. Això ens va donar la idea de generar l'operador d'intercanvi.

Aquest operador també té una lògica senzilla. Recorre tota la matriu HCP i, per cada petició, realitza un intercanvi amb totes les altres peticions del centre de producció que són posteriors, ja siguin a HCP o a Endarrerits. Quan diem que una petició p_i sigui posterior a una altra p_j ens referim a que la petició p_i es troba en una hora anterior a p_j .

Al dissenyar l'estratègia havíem pensat en que podríem fer els intercanvis de totes les peticions amb totes les altres, independentment de la seva posició, però això hagués generat successors repetits, efecte gens desitjat. D'altra banda la linealitat de l'algorisme, que s'executa en ordre, ens permet utilitzar el concepte de anterioritat o posterioritat de peticions.

Vam tenir també el dubte de si intercanviar les peticions de HCP amb les d'endarrerits, i vam concloure que ho havíem de fer ja que sinó les peticions d'endarrerits mai es mourien del seu lloc i, a més, generariem molts pocs estats successors.

Una altra reflexió inicial que vam fer sobre aquest algorisme és que si l'estat inicial que es genera no és bo, i no hi ha cap petició a endarrerits, hi ha la possibilitat de que no es pugui treure la petició que influeix malament en l'heurística de la matriu HCP. Tot i així, això no és desitjable! Sempre serà pitjor tenir una petició al vector d'endarrerits que a la matriu HCP, la penalització és molt més forta en el primer cas.

El factor de ramificació és més elevat que en el primer cas. Aquí es fan com a molt tants intercanvis com combinacions possibles hi ha entre peticions de la matriu i d'endarrerits, és a dir:

$$\text{Peticions_HCP!} + (\text{Peticions_HCP} * \text{Peticions_Endarrerits})$$

Algorismes de generació

Passem a continuació a comentar els algorismes.

Algorisme d'intercanvi

L'algorisme és com segueix:

Per cada centre de producció CP

1. Per cada Petició p_i de CP, que es troba a HCP

Per totes les demés peticions p_j tals que $j > i$ (en ordre a la matriu HCP)

swap (p_i, p_j);

2. Per cada Petició p_i de CP, que es troba a HCP

Per totes les peticions p_j de la matriu endarrerits (en ordre a la matriu)

swap (p_i, p_j);

La condició de que $j > i$ en (1), implica que no es repeteixen estats.

L'ordre de swap() que hem cridat aquí, està definida al propi estat. Aquesta funció intenta fer l'intercanvi de la petició comprovant les restriccions. Intenta també canviar el tipus de camió si és necessari.

El cost d'aquest algorisme és $O(n^3) + O(n) = O(n^3)$, essent la part més costosa la que realitza el swap a la matriu, per haver de recórrer-la tota.

Algorisme d'afegir-treure

L'algorisme és com segueix:

Per cada centre de producció CP

1. Per cada Petició p_i de CP,

Moure la petició a endarrerits, actualitzant camió si escau.

2. Per cada Petició p_i de CP

Agafar la primera petició d'endarrerits i intentar col·locar-la al primer lloc buit de l'estat.

Ambdós passos actualitzen l'estat i comproven les restriccions.

El cost d'aquest algorisme és $O(n^3) + O(n) = O(n^3)$, essent la part més costosa la que realitza el recorregut per la matriu.

Experiments

Experiment 1

Observació	El conjunt d'operadors d'intercanvi, o swap, dona millors resultats que el d'afegir i treure, o addremove per al HillClimbing
Plantejament	Provem els dos conjunts d'operadors per a HillClimbing.
Hipòtesis	El conjunt d'operadors swap proporciona millors resultats que addremove, o viceversa.
Mètode	<ul style="list-style-type: none"> • Executem 10 cops per cada un dels dos tipus d'operadors que volem provar • Cada execució serà amb les condicions donades a l'enunciat: 250 peticions i distribució de capacitats (o tipus) dels camions, pesos i hores de les peticions equiprobable. • Fem servir Hill Climbing • La estratègia de generació de l'estat inicial serà la de maxcompact, que esperem doni millors resultats que la lineal

Resultats

Execució amb operador de swap i generació d'estat inicial maxcompact, amb comanda:

```
java Transports.Main -hbenef -g maxcompact -html -numpet 250 -random -s swap > /tmp/index.html &&
firefox /tmp/index.html
```

Execució	Temps(ms)	Beneficis	Nodes Exp.
1	21232	103270	60
2	15331	89560	48
3	14744	99160	43
4	14463	99920	46
5	16256	72100	54
6	15915	90980	53
7	13020	76910	43
8	16961	102650	48
9	17126	66090	50
10	12439	87820	38
Mitja	15748,7	88846	48,3
St Dev	2471,05583	13226,5005	6,34

Execució amb operador addrem i generació d'estat inicial maxcompact, amb comanda:

```
java Transports.Main -hbenef -g maxcompact -html -numpet 250 -random -s addrem > /tmp/index.html &&
firefox /tmp/index.html
```

Execució	Temps(ms)	Beneficis	Nodes Exp.
1	889	28060	42
2	825	49140	41
3	705	30520	37
4	955	38540	39
5	833	65200	34
6	1039	58470	57
7	942	54100	46
8	1584	82890	75
9	1156	61300	50
10	1112	62490	44
Mitja	1004	53071	46,5
St Dev	245,52574	16943,1333	11,9930535

Conclusions

L'operador swap, o d'intercanvi de peticions, dona millors resultats que addrem, o de treure i posar peticions, per al primer criteri de qualitat, el que busca maximitzar els beneficis. A canvi observem que per aconseguir aquest majors beneficis requereix de més temps de processament.

	swap	addrem
Temps(ms)	15748,7	1004
Beneficis	88846	53071

Experiment 2

Observació	L'estratègia de generació de l'estat inicial lineal dóna resultats lleugeraments millors que l'estratègia maxcompact amb la heurística de maximitzar els beneficis
Plantejament	Provem les dues estratègies de generació d'estat inicial per HillClimbing.
Hipòtesis	L'estratègia de generació maxcompact dóna millors resultats que l'estratègia lineal donada la heurística de maximitzar els beneficis obtinguts, o en canvi es dóna la situació inversa.
Mètode	<ul style="list-style-type: none"> • Executem 10 cops per cada un dels dos tipus d'operadors que volem provar • Cada execució serà partint de les condicions donades a l'enunciat per l'experiment 1: 250 peticions i distribució de capacitats (o tipus) dels camions, pesos i hores de les peticions equiprobable. • Fem servir Hill Climbing

Resultats

Execució amb generació d'estat inicial maxcompact amb la següent comanda:

```
java Transports.Main -hbenef -g maxcompact -html -numpet 250 -random -s swap > /tmp/index.html &&
firefox /tmp/index.html
```

Execució	Temps	Beneficis	Nodes Exp.
1	22138	102390	55
2	21067	95390	63
3	17374	87670	52
4	17887	118680	51
5	12469	85060	39
6	15276	78460	49
7	12896	87740	39
8	15765	77680	50
9	17459	86320	55
10	13945	95600	46
Mitja	16627,6	91499	49,9
St Dev	3229,95088	12235,7994	7,32499526

Execució amb generació d'estat inicial lineal amb la següent comanda:

```
java Transports.Main -hbenef -g lineal -html -numpet 250 -random -s swap > /tmp/index.html && firefox /tmp/index.html
```

Execució	Temps	Beneficis	Nodes Exp.
1	1238	92820	5
2	1130	115100	6
3	2249	94140	9
4	1984	104340	9
5	1395	108720	6
6	5516	70260	15
7	1179	114360	5
8	1569	103420	6
9	1193	94100	4
10	2869	90700	11
Mitja	2032,2	98796	7,6
St Dev	1349,2354	13407,9307	3,40587727

Conclusions

Amb l'estratègia de generació de l'estat inicial maxcompact obtenim resultats lleugerament pitjors que amb l'estratègia lineal, a més que la primera estratègia requereix que l'algorisme HillClimbing expandeixi molts més nodes i per tant el temps a donar una solució és aproximadament unes 8 vegades més lent.

Per tant, per la resta d'experiments utilitzarem la generació d'estat inicial lineal.

	lineal	maxcompact
Temps(ms)	2032,2	16627,6
Beneficis	98796	91499

Experiment 3

Observació	Com més numero d'iteracions li donem a l'algorisme de Simulated Annealing millors resultats obtenim.
Plantejament	Provem Simulated Annealing amb diferents paràmetres i observem els resultats
Hipòtesis	Alguns conjunts de valors per l'algorisme de Simulated Annealing dona millors resultats que altres.
Mètode	<ul style="list-style-type: none"> Fem 5 execucions per cada conjunt de paràmetres que provem per Simulated Annealing, i ens quedem amb la mitja.

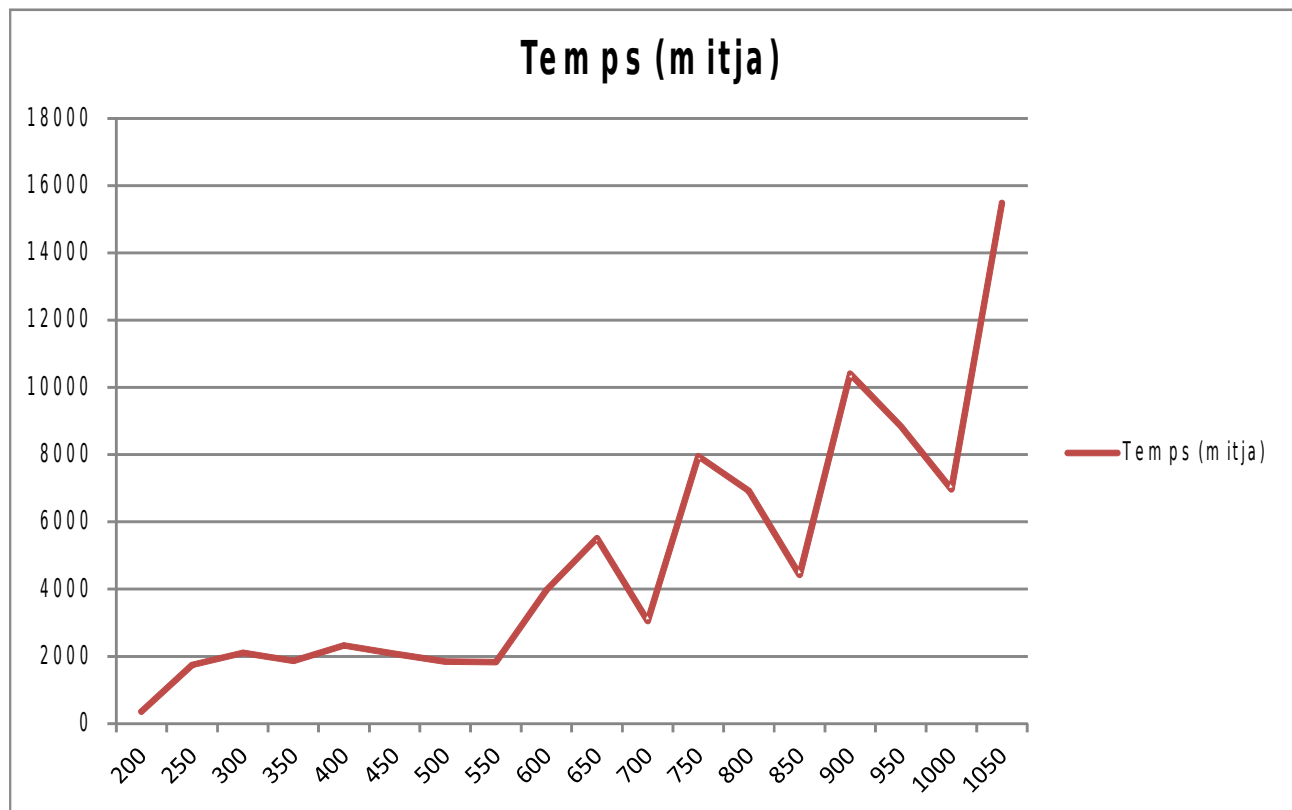
Resultats

Num It	It / pas de t	k	lambda	Beneficis (mitja)
1	1	1	1	53252
100	100	100	100	50274
50	50	50	50	36170
1	1	100	100	31380
1	1	100	1	49712
1	1	1	100	47978
100	100	1	1	50990
100	1	1	1	42804
1	100	1	1	36846
100	1	100	1	39268
100	1	1	1	40996
1	1	100	1	58382
1	100	1	100	36098

Experiment 4

Observació	El creixement del temps de càlcul respecte el nombre de peticions és exponencial
Plantejament	Provem amb valors creixents de nombre de peticions i observem els resultats
Hipòtesis	El creixement del temps de càlcul respecte el nombre de peticions serà lineal com a mínim.
Mètode	<ul style="list-style-type: none"> Fem 5 execucions per cada valor de nombre de peticions: 200, 250, etc Calculem la mitjana i l'anotem Generem el gràfic i observem els resultat

Resultats



Conclusions

A mida que s'augmenta el nombre de peticions per al HillClimbing amb les condicions dels experiments anteriors observem un augment exponencial del temps.

Experiment 5

Observació	La heurística del segon criteri de qualitat (mínima diferència entre hora límit i hora entregada de les peticions) dona menys guanys que la heurística del primer criteri de qualitat (màxims beneficis). A més, aquesta segons heurística fa que el temps de càlcul sigui uns 3 o 4 cops més lent que amb la primera
Plantejament	Provem les dues heurístiques i mirem els guanys i el temps de càlcul
Hipòtesis	Una de les dues heurístiques donarà millors guanys
Mètode	<ul style="list-style-type: none"> Fem 10 execucions per cada heurística i per cada nombre de peticions: 200, 250 i 300 Calculem la mitjana d'aquestes 10 execucions

Resultats

Temps	hbenef	hhores
200	448,2	6895,9
250	1889,7	8658,4
300	1873,6	11233,9

Guanys	hbenef	hhores
200	92517	89922
250	96078	91160
300	60286	57036

Conclusions

La heurística que busca minimitzar la diferència d'hores entre la hora límit d'entrega d'una petició i la hora en que realment s'entrega fa que el temps de càlcul sigui uns 3 o 4 cops més lent que amb la heurística que busca maximitzar els beneficis, sense donar més guanys. Al contrari, els resultats demostren que obtenim menys guanys amb aquesta heurística.

Experiment 6

Observació	
Plantejament	Provem Simulated Annealing i HillClimbing
Hipòtesis	Simulated Annealing hauria de donar millors resultats que HillClimbing
Mètode	<ul style="list-style-type: none"> • Executem Simulated Annealing amb 2 conjunts de paràmetres, els que millors resultats ens han donat a l'experiment número 3 • Fem 10 execucions per cada conjunt de paràmetres, per cada una de les 2 heurístiques i per cada nombre de peticions: 200, 250, i 300

Resultats

Num iteracions = 1

Iteracions per cada pas de $t = 1$

$k = 1$

$\lambda = 1$

Temps	hbenef	hhores
200	12,8	12,3
250	12,6	12,3
300	13,2	13,1

Guanys	hbenef	hhores
200	91710	92631
250	89031	88817
300	64065	62718

Num iteracions = 1

Iteracions per cada pas de $t = 1$

$k = 100$

$\lambda = 1$

Temps	hbenef	hhores
200	13,7	12,3
250	12,5	12
300	13,4	15,5

Guanys	hbenef	hhores
200	94206	88784
250	95132	85574
300	50812	72359

Conclusions

Amb el primer conjunt de paràmetres obtenim un resultat semblant als de l'experiment número 6 en termes de guanys. En canvi, parlant del temps d'execució obtenim que amb Simulated Annealing el temps de càlcul és moltes vegades més curt. Respecte al segon conjunt de paràmetres obtenim el mateix temps que amb el primer conjunt de paràmetres i uns guanys una mica inferiors.

Experiment 7

Observació	Com més camions de més capacitat tinguem millors beneficis retornarà HillClimbing i en menys temps.
Plantejament	Provem els 3 casos que es demanen a l'enunciat
Hipòtesis	Si tenim diferents distribucions de camions tindrem diferents resultats. Potser amb més camions de més capacitat obtindrem millors resultats.
Mètode	<ul style="list-style-type: none"> Fem 10 execucions per cada distribució de camions demanada i amb un nombre de peticions de 250 Calculem la mitjana del temps, beneficis i nodes expandits. Fem servir Hill Climbing

Resultats

	50 25 25	25 50 25	25 25 50
Temps	2046,9	1785,8	990,8
Nodes	7,9	7,4	4,3
Beneficis	60869	84421	115833

Conclusions

Com es pot observar en la taula de resultats, a mida que tenim més camions de la capacitat més gran, millors beneficis obtindrem. Això és així ja que d'aquesta manera l'algorisme disposa de més espai on col·locar peticions, és a dir, la capacitat de la flota de camions és més gran. Per altra banda, observem també que el temps de càlcul i el nombre de nodes que s'expandeixen es redueix a mida que tenim més camions de més capacitat. Això és degut a que amb més espai on posar peticions, i amb un nombre relativament baix de peticions (250) el HillClimbing no ha de cercar masses estats per a trobar una solució bona.

Experiment 8

Observació	Que algunes hores tinguin més probabilitat de tenir peticions no afecta als resultats en gran mesura
Plantejament	Provem amb hores amb més peticions per veure si afecta
Hipòtesis	Si algunes hores tenen més peticions els resultats obtingut variaran
Mètode	<ul style="list-style-type: none"> Reproduïm les condicions de l'experiment número 1

Resultats

Execució	Temps	Nodes Exp.	Beneficis
1	14789	43	76960
2	13780	45	87080
3	15676	49	80270
4	15001	49	94290
5	18203	54	103220
6	19570	58	93090
7	12626	42	66470
8	14561	48	78470
9	15784	50	81290
10	16174	48	91350
Mitja	15616,4	48,6	85249
S Dev	2031,85866	4,81202198	10617,23
Resultats Experiment 1			
Mitja	15748,7	48,3	88846
S Dev	2471,05583	6,34	13226,5

Conclusions

Observant els resultats veiem que el fet de tenir algunes hores amb més probabilitat de tenir peticions que unes altres, en concret 6 hores amb un 7.15% i les altres 4 hores amb un 14,275% aproximadament, no afecta de manera significativa als resultats obtinguts respecte als obtingut a l'experiment número 1.

Funcionalitats extra

Interfície del programa

El nostre programa proporciona una interfície de línia de comandes simplificada. Es pot executar qualsevol tipus de problema mitjançant una sola ordre.

Per executar-lo primer s'ha de compilar i llavors el podem executar amb el paràmetre -h:

```
user@host:~$ cd iafib/
user@host:~/iafib/$ javac Transports/Main.java
user@host:~/iafib/$ java Transports.Main -h
usage: java Transports.main
  -a <steps> <stiter> <k> <lamb>      Executa l'algorisme del Simulated
                                       Annealing amb els paràmetres passats.
  -c <nt1> <nt2> <nt3>                Nombre de camions de tipus1, tipus2 i
                                       tipus3. Han de sumar 60
  -g <g>                              Estratègia generació estat inicial
                                       [lineal,maxcompact]
  -hbenef                             Activar heurístiques de maximització de
                                       beneficis
  -hhores                             Activar heurístiques de minimitzar
                                       diferència engre hores limit i d'entrega
  -html                               Imprimeix l'output en format html
  -numpet <numpet>                   Nombre de peticions màx. aleatòries a
                                       generar. Ha de ser major que 0.
  -probs                              Entrar a l'editor de probabilitats de
                                       les hores, els camions i del pes de les peticions.
  -random                             Executar una mostra no pre-definida.
                                       S'ha de definir nombre màx. de
                                       peticions.
  -s <s>                              Tipus d'estratègia de successors
                                       [swap,addrem]
```

Com podem veure les ordres estan ben explicades.

Una línia d'execució habitual seria:

```
user@host:~/iafib/$ java Transports.Main -hbenef -hhores -g maxcompact -a 10 10 10 10 -html -numpet 1000 -random -s addrem -c 20 10 30 > /tmp/index.html
```

En aquest cas hem escollit generar una solució per les dues heurístiques (-hbenef, -hhores), hem escollit l'estratègia de generació de l'estat inicial MAX_COMPACT (-g maxcompact), i a banda del Hill Climbing que sempre s'executa hem triat fer simulated annealing amb paràmetres 10 10 10 10 (-a 10 10 10 10). Hem escollit que ens mostri l'output en format html, per redirigir-lo a un fitxer anomenat index.html que veurem amb un navegador (-html, > /tmp/index.html). També hem generat el problema de forma aleatòria amb 1000 peticions (-random, -numpet 1000), i hem decidit triar l'estratègia de generació de successors de Addremove (-s addrem). Finalment hem decidit tenir 20 camions de tipus 1, 10 de tipus 2 i 30 de tipus 3 (-c 20 10 30).

Un altre cas seria si volguéssim definir probabilitats per les hores i els pesos. Llavors podríem afegir l'opció "-probs", i entràriem en un editor interactiu de probabilitats:

```
Entra 10 probabilitats (enters) per la distribució peticions entre les hores de 8 a 17h. Valors entre 0 i 100 i pitja ENTER. (Han de sumar 100 en total!)
...input de 10 valors...

Entra 5 probabilitats (enters) per la distribució dels pesos a les peticions de 100 a 500. Valors entre 0 i 100 i pitja ENTER. (Han de sumar 100 en total!)
...input de 5 valors...

Entra 3 probabilitats (enters) per la distribució dels tipus de camions de tipus 500, 1000 i 2000. Valors entre 0 i 100 i pitja ENTER. (Han de sumar 100 en total!)
... input de 3 valors...
```

Finalment, recomanem fer l'output sempre per HTML, i executar una ordre que passi les probabilitats al nostre programa mitjançant una pipe (|), i una redirecció a un fitxer index.html que després obrirem amb un navegador. L'ordre següent ho fa tot, i a més ja obre el navegador.

Es suposa que /tmp/in existeix i conté dues línies correctes, per exemple:

```
user@host:~/iafib$ cat /tmp/in
10 2 8 20 15 29 12 1 2 1
20 7 0 23 50
user@host:~/iafib/$ cat /tmp/in | java Transports.Main -hbenef -hhores -g maxcompact -a 10 10 10 10 -html -numpet 1000 -probs -random -s addrem -c 20 10 30 > /tmp/index.html && firefox /tmp/index.html
```


Un fragment de sortida HTML del nostre programa.

Params. execució:

Ex. Heurística Beneficis: true
 Ex. Heurística Min. Hores: true
 Estratègia successors (swap:true, addrem: false): false
 Estratègia estat inicial (Lineal: 0, Max Compact: 1) : 1
 Generació aleatòria?: true
 Simulated Annealing?: true
 Num peticions a generar: 1000
 Num camions T1, T2, T3: 20,10,30
 Probabilitats horàries de 08 a 17h: 10,2,8,20,15,29,12,1,2,1,
 Probabilitats de pesos de 100 a 500kg: 20,7,0,23,50,

Transports HillClimbing - Maximitzar Beneficis

Search Outcome: SOLUTION_FOUND

Final State: Transports.Estat@2bf14ceb

nodesExpanded: 17

Temps d'execució: 547 milisegons

Heurístic 1 - Beneficis (com major millor, pot haver-hi perdues): -966100.0

Heurístic 2 - Hores desfassades (com menor millor): 14529.0

Graella HCP

HCP	Centre 1	Centre 2	Centre 3	Centre 4	Centre 5	Centre 6
8h	T: 2000Kg Carrega: 2000Kg Pets: @Petid:108,Qtt:500 ,HI:8 @Petid:175,Qtt:100 ,HI:8 @Petid:182,Qtt:100 ,HI:8 @Petid:298,Qtt:500 ,HI:8 @Petid:306,Qtt:500 ,HI:8 @Petid:857,Qtt:100 ,HI:8 @Petid:688,Qtt:200 ,HI:11	T: 2000Kg Carrega: 2000Kg Pets: @Petid:77,Qtt:100 ,HI:8 @Petid:85,Qtt:100 ,HI:8 @Petid:177,Qtt:500 ,HI:8 @Petid:344,Qtt:500 ,HI:8 @Petid:347,Qtt:400 ,HI:8 @Petid:651,Qtt:400 ,HI:8	T: 2000Kg Carrega: 2000Kg Pets: @Petid:38,Qtt:200 ,HI:8 @Petid:146,Qtt:500 ,HI:8 @Petid:234,Qtt:100 ,HI:8 @Petid:281,Qtt:200 ,HI:8 @Petid:356,Qtt:500 ,HI:8 @Petid:402,Qtt:200 ,HI:8 @Petid:443,Qtt:200 ,HI:8 @Petid:633,Qtt:100 ,HI:8	T: 2000Kg Carrega: 2000Kg Pets: @Petid:24,Qtt:400 ,HI:8 @Petid:30,Qtt:100 ,HI:8 @Petid:31,Qtt:400 ,HI:8 @Petid:88,Qtt:400 ,HI:8 @Petid:94,Qtt:500 ,HI:8 @Petid:150,Qtt:100 ,HI:8 @Petid:233,Qtt:100 ,HI:8	T: 2000Kg Carrega: 2000Kg Pets: @Petid:69,Qtt:200 ,HI:8 @Petid:72,Qtt:100 ,HI:8 @Petid:133,Qtt:500 ,HI:8 @Petid:166,Qtt:100 ,HI:8 @Petid:184,Qtt:400 ,HI:8 @Petid:187,Qtt:500 ,HI:8 @Petid:251,Qtt:200 ,HI:8	T: 2000Kg Carrega: 2000Kg Pets: @Petid:11,Qtt:500 ,HI:8 @Petid:79,Qtt:200 ,HI:8 @Petid:125,Qtt:500 ,HI:8 @Petid:185,Qtt:400 ,HI:8 @Petid:241,Qtt:200 ,HI:8 @Petid:277,Qtt:100 ,HI:8 @Petid:539,Qtt:100 ,HI:8
9h	T: 2000Kg Carrega: 2000Kg Pets: @Petid:6,Qtt:200 ,HI:9 @Petid:56,Qtt:500 ,HI:9 @Petid:565,Qtt:500 ,HI:9 @Petid:825,Qtt:500 ,HI:9 @Petid:688,Qtt:200 ,HI:11 @Petid:645,Qtt:100 ,HI:10	-	T: 2000Kg Carrega: 2000Kg Pets: @Petid:183,Qtt:500 ,HI:9 @Petid:601,Qtt:100 ,HI:9 @Petid:856,Qtt:500 ,HI:9 @Petid:870,Qtt:500 ,HI:9 @Petid:387,Qtt:400 ,HI:10	T: 1000Kg Carrega: 1000Kg Pets: @Petid:22,Qtt:400 ,HI:9 @Petid:988,Qtt:200 ,HI:9 @Petid:754,Qtt:400 ,HI:10	T: 1000Kg Carrega: 1000Kg Pets: @Petid:852,Qtt:500 ,HI:9 @Petid:946,Qtt:100 ,HI:9 @Petid:859,Qtt:400 ,HI:10	T: 2000Kg Carrega: 2000Kg Pets: @Petid:18,Qtt:500 ,HI:9 @Petid:270,Qtt:200 ,HI:9 @Petid:710,Qtt:200 ,HI:9 @Petid:937,Qtt:500 ,HI:9 @Petid:986,Qtt:100 ,HI:9 @Petid:581,Qtt:500 ,HI:10
10h	T: 2000Kg Carrega: 2000Kg Pets: @Petid:129,Qtt:400 ,HI:10 @Petid:342,Qtt:100 ,HI:10 @Petid:367,Qtt:500 ,HI:10 @Petid:462,Qtt:500 ,HI:10 @Petid:517,Qtt:500 ,HI:10	T: 2000Kg Carrega: 2000Kg Pets: @Petid:28,Qtt:500 ,HI:10 @Petid:83,Qtt:400 ,HI:10 @Petid:90,Qtt:100 ,HI:10 @Petid:103,Qtt:200 ,HI:10 @Petid:105,Qtt:500 ,HI:10 @Petid:716,Qtt:100 ,HI:10 @Petid:901,Qtt:100 ,HI:10	T: 2000Kg Carrega: 2000Kg Pets: @Petid:5,Qtt:200 ,HI:10 @Petid:52,Qtt:500 ,HI:10 @Petid:118,Qtt:400 ,HI:10 @Petid:186,Qtt:100 ,HI:10 @Petid:361,Qtt:500 ,HI:10 @Petid:824,Qtt:100 ,HI:10 @Petid:142,Qtt:200 ,HI:11	T: 2000Kg Carrega: 2000Kg Pets: @Petid:101,Qtt:100 ,HI:10 @Petid:433,Qtt:500 ,HI:10 @Petid:474,Qtt:100 ,HI:10 @Petid:585,Qtt:500 ,HI:10 @Petid:663,Qtt:500 ,HI:10 @Petid:981,Qtt:100 ,HI:10 @Petid:608,Qtt:200 ,HI:11	T: 2000Kg Carrega: 2000Kg Pets: @Petid:57,Qtt:500 ,HI:10 @Petid:70,Qtt:100 ,HI:10 @Petid:253,Qtt:100 ,HI:10 @Petid:273,Qtt:500 ,HI:10 @Petid:295,Qtt:500 ,HI:10 @Petid:365,Qtt:100 ,HI:10 @Petid:814,Qtt:200 ,HI:10	T: 2000Kg Carrega: 2000Kg Pets: @Petid:63,Qtt:500 ,HI:10 @Petid:327,Qtt:400 ,HI:10 @Petid:333,Qtt:100 ,HI:10 @Petid:357,Qtt:100 ,HI:10 @Petid:454,Qtt:500 ,HI:10 @Petid:564,Qtt:400 ,HI:10
Fet						

Classe externa

Per poder fer la implementació eficient del nostre programa en quant a llegir els paràmetres per línia de comandes, hem hagut d'investigar solucions de qualitat.

La solució que hem trobat és la de fer servir la llibreria d'Apache "Common CLI". Aquesta llibreria és molt dinàmica i extensament provada i suportada per la comunitat Open Source.

Els fitxers es poden trobar al package "*org.apache.commons.cli*" que hem inclòs a la pràctica. Per defecte també es pot trobar a molts entorns.


Codi font

Per treballar hem decidit utilitzar l'eina Subversion. Aquesta eina de control de versions ens permet flexibilitat de moviment, rapidesa de canvis, documentació, disponibilitat del codi, i beneficis de coordinació.

A més es pot veure el progrés de la pràctica ja que a cada modificació s'ha afegit un comentari.

El repositori escollit és OpenSource i és el de Google Code. El podeu trobar a la web:

<http://code.google.com/p/iafib/>



iafib

Práctica 1 de IA, Q2 2010-2011

[Project Home](#)
[Downloads](#)
[Wiki](#)
[Issues](#)

[Source](#)
[Administer](#)

[Checkout](#)
[Browse](#)
[Changes](#)

[Request code review](#)

Committed Changes

81 - 57 of 81 [Older >](#)

Rev	Scores	Commit log message	Date	Author
★ r81		+ Indentació	Today (2 hours ago)	
★ r80		+ Afegida opció -a per el simulated annealing, amb els 4 paràmetres demanats. Ja es crida fins i tot a	Today (2 hours ago)	
★ r79		+ Reimplementada funció de probabilitat uniforme amb ponderacions per generar les peticions inicials.	Today (2 hours ago)	
★ r78		+ preparació per generació de successors de SA	Today (5 hours ago)	
★ r77		+ comentada generació aleatoria de maxpeticions	Today (8 hours ago)	
★ r76		+ save CSS!!	Yesterday (21 hours ago)	
★ r75		+ Afegit possibilitat de modificar probabilitats des de la interfície	Yesterday (22 hours ago)	
★ r74		+ Afegida funcionalitat per generar llista de peticions inicial en base a probabilitats uniformes tal com c	Yesterday (23 hours ago)	
★ r73		+ Afegit càlcul del temps a html output	Yesterday (25 hours ago)	
★ r72		+ Afegit classes per poder passejar bé els paràmetres d'entrada. + Millorat enormement la interacció a	Yesterday (25 hours ago)	
★ r71		+ Afegit documentacio de Heurística max guany	Mar 29 (44 hours ago)	
★ r70		+ preparant s.a., feta funcio per imprimir estat final i descongestionar el main.java	Mar 29 (44 hours ago)	
★ r69		+ Afegit Elements de Aima -> Implementació de l'estat	Mar 29 (2 days ago)	
★ r68		+ Afegit gairebé tots els punts a documentar al .odt	Mar 29 (2 days ago)	
★ r67		+ CSS: canvis a les taules	Mar 29 (2 days ago)	
★ r66		+ afegit una mica de css: de moment només font-family:arial per fer més llegibles els numeros petits	Mar 29 (2 days ago)	
★ r65		+ Eliminada variable estúpida:	Mar 29 (2 days ago)	
★ r64		+ astyle * --style=ansi --indent=tab	Mar 29 (2 days ago)	

Facultat d'Informàtica de Barcelona - UPC

Intel·ligència Artificial - Q2 2010/2011

-- IAFIB --

code.google.com/p/iafib

Autors: Felip Moll Marquès

Jordi Llamas Pons