Universitat de Girona
Escola Politècnica Superior

Grau en Enginyeria Informàtica

PROJECTE FINAL DE GRAU

Rush Hour

Autor:
Carla Davesa Sureda

Tutor:
Dr. Mateu Villaret Ausellé

MEMÒRIA

Convocatòria:
Setembre 2023

Departament :
Informàtica, matemàtica aplicada i estadística

Projecte: Projecte Final de Grau
Document: Memòria
Títol: Rush Hour
Autor: Carla Davesa Sureda
Data: Setembre 2023

Estudi:
Grau en Enginyeria Informàtica
Universitat de Girona

Supervisor:
Dr. Mateu Villaret Ausellé
Universitat de Girona
Email: mateu.villaret@udg.edu

Índex

Índex de figures

M’agradaria expressar el meu agraïment a totes les persones que m’han acompanyat i recolzat al llarg del desenvolupament d’aquest projecte.

Per començar, vull expressar el meu agraïment al meu tutor, el Dr. Mateu Villaret Ausellé, per la seva dedicació, predisposició, orientació constant i suport durant tot el treball.

També vull agrair al Dr. Joan Espasa Arxer per la seva ajuda en moments de dubte. La seva disposició a donar un cop de mà i compartir el seu coneixement han estat molt enriquidors i m’han ajudat a tirar endavant oferint un altre punt de vista.

També vull expressar la meva gratitud al Grup de Recerca en Lògica i Intel·ligència Artificial (LAI) de la Universitat de Girona. Per oferir-me accés als seus recursos com les eines del clúster que han estat fonamentals i han facilitat molt el desenvolupament del projecte. A més, agraeixo al Dr. Miquel Bofill Arasa per ajudar-me i ensenyar-me a utilitzar eficaçment el clúster.

Finalment, vull donar les gràcies a totes les persones que han format part d’alguna manera d’aquest projecte.

Estic molt agraïda per tot el que he après i aconseguit i sé que ha estat gràcies a totes aquestes contribucions.

Moltes gràcies a tots.

Capítol 1

Introducció, motivacions, propòsit i objectius del projecte

El joc *Rush Hour* és un trencaclosques de desplaçament de blocs creat per l'enginyer i dissenyador japonès Nob Yoshigahara l'any 1978. A la Figura 1.1 en podem veure una imatge de com es comercialitza. La seva idea original era proporcionar un enigma de lògica que fos accessible per a tothom i alhora fos un repte intrigant.

Inspirat en el trànsit urbà congestionat, el joc va ser concebut com una representació en miniatura d'una situació de trànsit on caldria moure vehicles per tal d'obrir pas a un cotxe vermell que ha de sortir del tauler.
A primera vista pot semblar una tasca senzilla però el Rush Hour és un problema difícil ja que no només requereix de moviments físics, sinó que també cal una capacitat d'anticipar moviments i calcular estratègies.

Inicialment, el Rush Hour va ser introduït al mercat japonès com a joc de tauler. No obstant això, va guanyar popularitat de manera ràpida i es va estendre a nivell mundial a través de diferents edicions i adaptacions.

FIGURA 1.1: Imatge d'una comercialització del joc.

El seu èxit va anar més enllà dels límits del món dels jocs i cridà l'atenció d'entusiastes de la lògica i experts en resolució de problemes. Com veurem, hi ha diversos treballs que es centren en estudiar-ne la complexitat computacional i a cercar mètodes eficients de resolució, ja sigui mitjançant la planificació d'IA o bé la resolució de problemes combinatoris.

La visió del joc com a problema de planificació és molt natural ja que clarament podem veure una noció d'estat del joc (la disposició dels cotxes), un estat inicial (la disposició inicial del nivell del joc que volem resoldre), un estat final (aquell on el cotxe vermell pot sortir) i unes accions que ens permeten evolucionar dins l'espai d'estats (els moviments dels cotxes). La resolució de restriccions es pot fer servir per a resoldre problemmes de planificació, i, de fet, serà on ens centrarem especialment.

El nostre treball pretén explorar com la utilització d'eines informàtiques pot oferir noves perspectives als desafiaments que aquest joc planteja. Amb això, buscarem generar nous avenços dins d'àmbits relacionats amb la resolució de problemes combinatoris i la planificació en IA.

Des de sempre, he experimentat una gran atracció pels problemes complexos. Em motiva afrontar-me a qüestions difícils per intentar trobar mètodes eficients que les resolguin de la millor manera possible.

Aquest interès fa que els jocs d'estratègia siguin una part essencial de la meva vida. Concretament, el joc Rush Hour ocupa un lloc molt especial des de la meva infància. Recordo les hores que passava movent els vehicles en el tauler i intentant trobar la manera de fer sortir el cotxe vermell.

Afrontar aquest repte amb una visió més madura i una formació acadèmica més sòlida, m'ha permès canviar la perspectiva del que pensava que era un simple joc entretingut, per veure'l com el que realment és, un autèntic repte computacional.

Els objectius específics que volíem assolir en aquest projecte eren:

1. Aprofundir en l'estudi del joc Rush Hour, i comprendre els detalls que el fan tan fascinant.

2. Explorar i aprendre diferents tècniques útils per resoldre el joc Rush Hour. Es preveu implementar models de programació en diferents llenguatges, com ara MiniZinc, Essence Prime, SAT, SMT i PDDL.

3. Augmentar la mida del tauler original del joc (6x6) fins a tamanys més grans com ara 7x7, 8x8, 9x9, etc. amb l'esperança de trobar configuracions més complexes que posin a prova els models creats, i així comparar la seva eficàcia en un nivell superior de dificultat.

4. Construir un recull de configuracions complexes per analitzar els diferents models i comparar l'eficàcia d'aquests.

5. Analitzar el concepte de dificultat en el joc Rush Hour i identificar els factors que influeixen en aquesta.

6. Explorar l'expressió del problema com una fórmula QBF i avaluar la seva utilitat per a la resolució de problemes més grans i complexos.

Com veurem, no hem assolit tots els objectius (possiblement massa ambiciosos de bon principi per un TFG) però sí el principal, poder analitzar el comportament de diferents aproximacions a la resolució de problemes combinatoris, per a la resolució del Rush Hour, així com explorar la noció de dificultat dels escenaris. Hem pogut

comprovar que PDDL domina les altres aproximacions existents per aquest problema i que la segona aproximació més robusta és la basada en SAT amb OptiLog. També hem pogut observar que als resoledors amb Minizinc i Essence Prime hi ha algun tipus d'optimització que els acaba resultant molt dures fent fins i tot timeout de (600s) mentre que PDDL és capaç de resoldre en pocs segons.

Creiem que val la pena comentar que en part aquesta desviació d'objectius que ens ha portat a aprofundir una mica més en els diferents marcs de resolució del problema (SAT, PDDL i Essence Prime) han sigut deguts a una situació personal en la que se m'ha obert la possibilitat d'anar a fer una tesi doctoral a la Universitat de St. Andrews i la temàtica de l'estudi requereix un bon coneixement d'aquestes eines.

Capítol 2

Marc de treball i conceptes previs

En aquest capítol, s'estableixen les bases dels conceptes que considerem essencials per a comprendre el contingut del nostre projecte.

2.1 Problemes de planning

Un problema de planificació (planning) en IA, és una tasca computacional que implica trobar una seqüència d'accions per aconseguir un conjunt d'objectius en una situació donada. Aquesta seqüència d'accions ha de satisfer un conjunt de restriccions i tenir en compte les interaccions i les conseqüències de cada acció presa. En altres paraules, es tracta de determinar quines accions cal dur a terme i en quin ordre per arribar a un resultat desitjat.

El joc Rush Hour es considera un problema de planificació, ja que implica trobar una seqüència d'accions per moure els vehicles en un tauler i desbloquejar el cotxe vermell. Les restriccions venen donades per la configuració inicial del tauler i les regles de moviment dels vehicles.

2.2 Complexitat computacional

Aquest apartat té com a objectiu establir les bases per comprendre la complexitat dels problemes en el context de la teoria de la computació. Definim alguns conceptes basats en el llibre de Michael Sipser, Introduction To The Theory Of Computation [1]. Per a una comprensió més profunda es recomana consultar el llibre.

La teoria de la complexitat computacional és una investigació del temps, la memòria o altres recursos necessaris per resoldre problemes computacionals.

2.2.1 Màquina de Turing

La Màquina de Turing és un model abstracte de càlcul proposat per Alan Turing el 1936. És una màquina d'estats finits amb memòria il·limitada (una o més cintes) i sense restriccions. Pot fer tot el que un ordinador real pot fer, tot i que hi ha problemes que no pot resoldre. Això ens permet definir els límits de la computació, tant pel que fa a la decidibilitat com pel que fa a la complexitat.
Diem que una màquina de Turing és determinista si les instruccions que ha d'executar no tenen grau de llibertat i només depenen de l'estat actual i el contigut de la

cinta i la posició del capçal. Si per una configuració concreta la màquina te diverses instruccions possibles a executar, pot triar, diem que és no-determinista.

2.2.2 Classes PSPACE i PSPACE-complet

Aquestes classes es basen en l'espai computacional, és a dir, quanta memòria necessita una màquina de Turing per a resoldre un problema en funció de la mida d'aquest.

- PSPACE és la classe de problemes que es poden decidir en l'espai polinomial en una màquina de Turing determinista.
- PSPACE-complet és la classe de problemes almenys tan difícils com els problemes més difícils en PSPACE i qualsevol problema en PSPACE pot ser reduït de manera eficient a ell mateix.

2.2.3 Classes P, NP i NP-complet

Aquestes classes es basen en la noció del temps.

- P (Polinòmica) és la classe de problemes que poden ser decidits en temps polinòmic per una màquina de Turing determinista d'una sola cinta.
- NP (Polinòmica no Determinista) és la classe de problemes que es poden decidir en temps polinòmic per una màquina no determinista.
- NP-complet (NP-Complete), direm que un problema és NP-complet si és NP i qualsevol problema NP es pot reduir eficientment a ell.

Avui en dia no s'ha trobat cap problema NP-complet que s'hagi pogut resoldre en temps polinòmic. De fet, si es trobés voldria dir que les classes P i NP són iguals. Aquesta és una pregunta fonamental en la teoria de la complexitat computacional i encara no ha estat resolta.

Aquests problemes NP-complets habitualment es resolen mitjançant algorismes deterministes de complexitat exponencial. És per això que són problemes difícils o durs computacionalment tot i aparèixer quotidianament: problemes d'horaris, de programació de tasques, trencaclosques, etc.

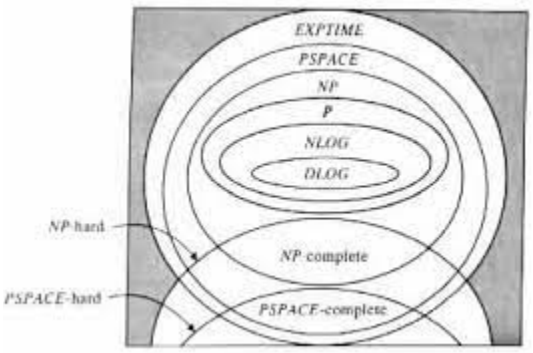


FIGURA 2.1: Classes de complexitat i relacions entre classes.

La classe P està continguda dins PSPACE (P $\subseteq$ PSPACE), ja que qualsevol problema que pot ser resolt en temps polinomial també pot ser resolt utilitzant una quantitat de memòria polinomial. NP també està continguda dins PSPACE (NP $\subseteq$ PSPACE). Mostrem aquestes relacions entre classes de complexitat a la Figura 2.1.

2.3 Fórmula de lògica proposicional

Una fórmula de lògica proposicional, també anomenada expressió booleana, es construeix amb variables Booleanes (que poden prendre el valor cert o fals), operadors AND (conjunció, símbol $\wedge$), OR (disjunció, símbol $\vee$), NOT (negació, $\neg$) i parèntesis.

S'anomena literals a una variable o la seva negació. Una clàusula és una disjunció de literals. Una fórmula està en la seva Forma Normal Conjuntiva (CNF) si és una conjunció de clàusules.

2.4 Resolució de problemes SAT

El problema de Satisfacibilitat Booleana (SAT) consiteix en, donada una fórmula de lògica proposicional (típicament en CNF), determinar si existeix una assignació de valors de veritat (verdader o fals) a aquest conjunt de variables que faci que la fórmula donada sigui certa.

A continuació, mostrem un exemple d'una fórmula en CNF.

$$(A \vee B) \wedge (\neg B \vee C)$$

Una interpretació que satisfà aquesta fórumla seria: {A=cert, B=fals, C=cert}

$$\begin{gathered}(\text{cert} \vee \text{fals}) \wedge (\neg\text{fals} \vee \text{cert}) \\ \text{cert} \wedge (\text{cert} \vee \text{cert}) \\ \text{cert} \wedge \text{cert} \\ \mathbf{cert}\end{gathered}$$

I una interpretació que no satisfà aquesta fórmula seria: {A=fals, B=cert, C=fals}

$$\begin{gathered}(\text{fals} \vee \text{cert}) \wedge (\neg\text{cert} \vee \text{fals}) \\ \text{cert} \wedge (\text{fals} \vee \text{fals}) \\ \text{cert} \wedge \text{fals} \\ \mathbf{fals}\end{gathered}$$

Una fórmula insatisfactible es aquella que no pot ser satisfeta per cap assignació de valors a les variables. Un exemple seria:

$$A \wedge \neg A$$

Aquesta fórmula diu que A ha de ser cert i fals al mateix moment, el que és impossible en la lógica proposicional. No hi ha cap combinació de valors per A que faci que sigui verdadera.

El problema SAT és NP-complet[1].

2.5 Problemes de satisfacció de restriccions (Constraint Satisfaction Problem)

El problema de satisfacció de restriccions (CSP) implica un conjunt de variables cadascuna de les quals té un domini de valors possibles i un conjunt de restriccions que defineixen les combinacions de valors permeses per a les variables. El seu objectiu és trobar solucions, és a dir, assignacions de valors a les variables, que compleixin totes les restriccions.

El joc de decidir si Rush Hour es pot resoldre en un nombre de moviments determinat, es pot tractar com un problema de satisfacció de restriccions, pel que, en aquest treball, farem ús de diversos llenguatges per modelar-lo i trobar solucions.

2.6 Descripció del joc Rush Hour

El Rush Hour és un joc clàssic tipus trencaclosques que planteja un repte aparentment senzill, però que ràpidament es converteix en un problema complex que requereix habilitats de resolució, estratègia i paciència.

El joc es juga en un tauler quadrat de dimensions 6×6 que pot representar una situació de trànsit real tipus embús, o un pàrquing del que s'ha de treure un cotxe i n'hi ha d'altres de creuats. L'objectiu principal del joc és alliberar un cotxe vermell atrapat entre altres vehicles mitjançant el moviment dels vehicles dins del tauler. Cada vehicle es pot moure en la seva direcció horitzontal o vertical en les caselles buides, així doncs els seus moviments estan limitats pels altres vehicles i per les parets del tauler.

2.6.1 Conceptes

Definim alguns conceptes per proporcionar una base sòlida per a l'enteniment d'aquest projecte.

- **Estat**: Es refereix a una situació específica i concreta de les posicions dels vehicles al tauler en un moment determinat de la resolució del joc.
- **Configuració/instància**: La configuració (inicial) o instància és l'estat inicial del problema que es busca resoldre.
- **Desplaçament/Moviment**: Aquest terme fa referència al procés de moure un vehicle d'una casella a una altra de consecutiva. És una acció individual en què el vehicle canvia de posició i per tant ocupa unes caselles diferents de les que ocupava abans de moure'l.
- **Pas**: Aquest terme fa referència a l'acció de moure un vehicle una o més caselles consecutives. És a dir, realitzar una sèrie de moviments consecutius amb un mateix vehicle sense interrupció.
- **Estat Objectiu/Final**: És qualsevol estat on el cotxe vermell es troba a la posició objectiu, és a dir a la dreta del tot de la fila on està. Cal destacar que no cal modelar que el cotxe es trobi del tot fora ja que si està a la dreta de tot és obvi que pot sortir.

- **Estats accessibles**: Fa referència als diferents estats que es poden assolir mitjançant accions per resoldre un problema determinat, és a dir, els conjunt d'estats que són accessibles entre ells amb moviments dels vehicles.

- **Complexitat**: Es refereix a la mesura de quanta dificultat o recursos es requereixen per resoldre un problema. En aquest context, la complexitat pot indicar quan de temps, memòria o recursos computacionals es necessiten per trobar una solució.

- **Dificultat**: La dificultat és una mesura subjectiva de com de complicat és resoldre un problema. Aquest concepte pot variar segons l'experiència, habilitats i coneixements de la persona que intenta resoldre el problema. En aquest context intentarem definir els factors que al nostre entendre determinaran la dificultat de les configuracions.

- **Optimització**: Aquest concepte fa referència al procés de trobar la millor solució o el millor resultat possible dins d'un conjunt de possibilitats. En el present treball podríem parlar d'optimització quan intentem trobar la solució amb el mínim nombre de passos o moviments.

2.6.2 Regles bàsiques

Vehicles

El joc convencional està format per tres tipus de vehicles.

- **Cotxe vermell:** és el vehicle que hem d'aconseguir treure del tauler. Té mida 2 (és a dir ocupa dues caselles) amb orientació horitzontal i fixat a la fila de sortida (que en el problema original és la tercera començant per dalt).

- **Cotxes:** són la resta de vehicles de mida 2 i que poden tenir orientació horitzontal o vertical.

- **Camions:** són la resta de vehicles de mida 3 i que poden tenir orientació horitzontal o vertical.

Restriccions

Les regles que regeixen el funcionament (bàsic) del joc són les següents:

1. Els vehicles no poden ser girats, només es poden moure endavant o enrere en la seva direcció, horitzontal o vertical.

2. Cap vehicle pot saltar els altres vehicles.

3. Cap vehicle pot sortir del tauler.

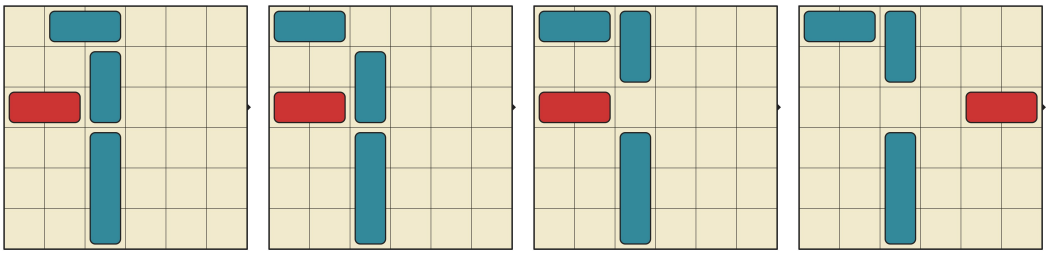
4. Mai hi haurà més d'un vehicle ocupant una mateixa posició o casella.

5. El cotxe vermell ha d'estar col·locat en les posicions de sortida del tauler en l'últim pas.

Es podrien considerar generalitzacions del joc on hi ha caselles ocupades sempre per una paret, o l'escenari és més gran de 6×6 o hi ha vehicles més llargs o més curts, etc.

2.6.3 Exemple de resolució d'una instància

A continuació, presentem un exemple detallat de com es pot resoldre una instància concreta del problema Rush Hour. Aquest exemple (vegeu Figura 8.5) té com a objectiu il·lustrar els passos que es fan servir en el procés de resolució.



(A) Estat inicial (B) Movem cotxe superior (C) Decantem cotxe vertical (D) Movem vermell i arriben a l'estat final

FIGURA 2.2: Passos per a la Resolució d'una instància senzilla.

Aquesta configuració està formada per quatre vehicles: el cotxe vermell, dos cotxes i un camió.

En la Figura 2.2a es mostra l'estat inicial d'aquesta configuració, que representa la disposició dels vehicles al tauler al començament del joc.

En la Figura 2.2b es mostra el primer pas, on el cotxe amb orientació horitzontal es mou un desplaçament cap a l'esquerra.

En la Figura 2.2c es mostra el segon pas. En aquest pas, el cotxe amb orientació vertical es mou un desplaçament cap amunt.

En la Figura 2.2d es mostra el tercer i últim pas, on el cotxe vermell es mou quatre desplaçaments a la dreta i s'aconsegueix l'objectiu del joc.

La solució que acabem de mostrar és la més òptima en quant a passos i desplaçaments. El nombre de passos que es necessiten per resoldre-la són tres i el nombre total de desplaçaments realitzats són sis.

2.6.4 Complexitat computacional del joc

En aquesta secció, s'exposen els principal resultats sobre la complexitat computacional del Rush Hour que podem trobar a la literatura.

En primer lloc, Flake et al.[2], van demostrar que, per a una versió generalitzada del Rush Hour, decidir si una configuració inicial és resoluble o no és un problema PSPACE-complet. Una versió generalitzada és aquella en la que la dimensió del tauler no està limitada a una mida específica i la ubicació de sortida no està fixada a un lloc específic.

Aquesta demostració indica que, si més no de moment, no existeix cap algorisme de temps polinòmic capaç de trobar una solució per a qualsevol configuració del Rush Hour. Aquest resultat contrasta amb el resultat obtingut a [3] on es demostra que decidir si una instància del *15-puzzle*, on l'objectiu del joc és aconseguir ordenar les caselles d'esquerra a dreta i de dalt a baix (vegeu Figura 2.3), te solució és P mentre que trobar-ne la solució òptima pel que fa al número de passos és (només) NP-complet.

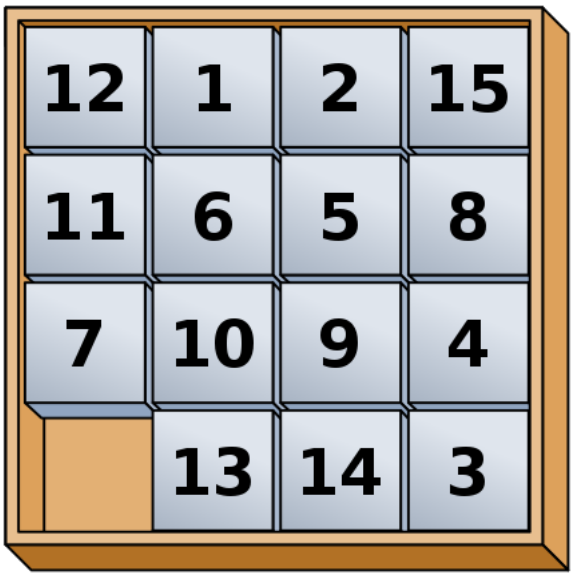


FIGURA 2.3: Exemple d'una instància de 15-puzzle.

Malgrat aquesta indicació, Fernau et al. [4] van aconseguir demostrar que, sempre que el nombre total de moviments o el nombre total de vehicles estigui limitat per una constant, es poden trobar solucions en temps polinomial. Aquest resultat fa referència a la complexitat paramètrica.

Capítol 3

Infraestructura i Recursos

En aquest apartat, es descriu la infraestructura i els recursos que s'han utilitzat per dur a terme la investigació i el desenvolupament d'aquest projecte.

3.1 Característiques de l'ordinador personal

L'ordinador que s'ha utilitzat està equipat amb un processador Intel Core i7-8550U amb quatre nuclis i vuit threads; té una velocitat base de 1.80 GHz i una velocitat màxima de 1.99 GHz. També disposa de 16,0 GB de memòria RAM, amb 15,9 GB d'ús disponible. El sistema operatiu és Windows 11 Home de 64 bits amb processador basat en x64.

3.2 Integració de subsistema de Windows per a Linux (WSL)

La instal·lació del Subsistema de Windows per a Linux (WSL) ha estat un pas fonamental en el desenvolupament del projecte, ja que diversos programes i eines que volíem utilitzar no es trobaven disponibles en el format de Windows.

3.3 Clúster

A més de l'ordinador personal, el Grup de Recerca en Lògica i Intel·ligència Artificial (LAI) de la Universitat de Girona ens ha posat en disposició el seu clúster.

Un clúster és una agrupació de diverses màquines o servidors connectats en xarxa que treballen col·lectivament per aconseguir un objectiu comú.

Aquesta infraestructura permet executar programes que requereixen un gran poder de processament o una gran quantitat de memòria de manera més eficient i ràpida. Però de fet pels nostres interessos el que ha sigut de gran utilitat és el fet de disposar de diverses màquines alhora ja que un dels avantatges més rellevants del clúster és la seva capacitat per a llançar diversos processos simultàniament i mantenir-los executant-se sense limitacions de temps. Això ha permès iniciar tasques i deixar-les en funcionament sense necessitat de mantenir l'ordinador obert i sense preocupar-se per interrupcions. Aquesta característica és fonamental per aquest projecte, ja que molts dels programes requereixen temps significatius per a la seva execució i a més s'han de llençar molts processos.

```
Arquitectura:                         x86_64
modo(s) de operación de las CPUs:     32-bit, 64-bit
Orden de los bytes:                   Little Endian
CPU(s):                               8
Lista de la(s) CPU(s) en línea:       0-7
Hilo(s) de procesamiento por núcleo:  2
Núcleo(s) por «socket»:               4
«Socket(s)»                           1
Modo(s) NUMA:                         1
ID de fabricante:                     GenuineIntel
Familia de CPU:                       6
Modelo:                               158
Nombre del modelo:                    Intel(R) Xeon(R) E-2234 CPU @ 3.60GHz
Revisión:                             10
CPU MHz:                              3600.000
CPU MHz máx.:                         4800,0000
CPU MHz mín.:                         800,0000
BogoMIPS:                             7200.00
Virtualización:                       VT-x
Caché L1d:                            32K
Caché L1i:                            32K
Caché L2:                             256K
Caché L3:                             8192K
CPU(s) del nodo NUMA 0:               0-7
```

FIGURA 3.1: Informació de les CPUs del Clúster.

Aquest clúster disposa de 20 nodes, i cada node té 8 CPUs (vegeu informació de la CPU a la Figura 3.1), 4 CPUs fisiques i 4 de virtuals (hyperthreading). Cada una d'aquestes CPUs té 4 nuclis (o cores), que permeten realitzar múltiples tasques simultàniament mitjançant l'execució paral·lela d'aquests nuclis.

Capítol 4

Estudis i decisions

En aquest apartat examinem de manera exhaustiva les decisions clau que hem adoptat durant el desenvolupament d'aquest projecte, a més de descriure el programari que hem decidit utilitzar i justificar-ne les eleccions.

4.1 Selecció de llenguatges

Per a resoldre els desafiaments que planteja el joc Rush Hour, hem optat per utilitzar una sèrie de llenguatges i eines especialitzades que s'expliquen a continuació.

4.1.1 MiniZinc

MiniZinc és un llenguatge de modelatge de problemes de satisfacció de restriccions i d'optimització. És un llenguatge declaratiu d'alt nivell que és independent del solucionador.

Utilitzar les restriccions per descriure les regles del joc i les variables amb dominis ben definits per descriure els vehicles i els valors que poden adquirir, fa que aquest llenguatge sigui ideal per expressar les configuracions del joc.

Aquesta elecció ha estat reforçada pel fet que ja havíem tingut l'oportunitat d'utilitzar MiniZinc anteriorment, en concret a l'assignatura *Programació Declarativa. Aplicacions* de 4t de GEINF, el que ens va proporcionar una familiaritat prèvia amb aquest llenguatge i les seves característiques.

Ha estat de molta ajuda el Manual de MiniZinc [5].

4.1.2 OptiLog per SAT

També s'ha optat per codificar el problema en SAT. SAT permet una representació precisa de les restriccions, és eficient en la resolució de problemes complexos, ofereix flexibilitat per adaptar-se a diferents dimensions i variants, i utilitza eines establertes per optimitzar el procés i avaluació de solucions.

OptiLog és un marc de treball/llibreria de Python per a la creació ràpida de prototips de sistemes basats en SAT (satisfactibilitat booleana). Inclou funcionalitats per a la càrrega i creació de fórmules, així com l'ús de solvers SAT que són estat de l'art, modelització de fórmules d'alt nivell, i codificacions de restriccions pseudo-booleanes

i de cardinalitat. A més, ofereix eines per a la configuració automàtica i l'execució i anàlisi d'experiments, fet que facilita la investigació i l'optimització dels models.

S'ha optat per utilitzar OptiLog principalment pel seu enfocament en la simplicitat i eficiència. La seva interfície basada en Python, que és un llenguatge de programació amb el qual hi tenim familiaritat, ha facilitat l'ús de les seves funcions i ens ha permès aprendre ràpidament com utilitzar OptiLog per resoldre els reptes del joc. D'altra banda, ofereix totes les eines necessàries per desenvolupar sistemes basats en SAT, el que ha proporcionat la capacitat de modelar situacions complexes d'una manera efectiva. H hagut de dedicar temps a aprendre'l Per a l'aprenentatge d'aquesta eina, es va utilitzar el Manual d'OptiLog [6] com a referència fonamental.

4.1.3 Essence Prime

Essence Prime, com MiniZinc, també és un llenguatge de modelatge de problemes de satisfacció de restriccions i d'optimització. També és un llenguatge declaratiu d'alt nivell que és independent del solucionador.

Savile Row és una eina de reformulació que pren aquesta especificació i la tradueix automàticament al llenguatge d'entrada d'un solucionador de restriccions de més baix nivell. En el procés de traducció, Savile Row aplica reformulacions per millorar el model. Això pot incloure l'eliminació d'expressions comunes entre parts diferents del model, millorant la propagació de restriccions i la eficiència del solucionador.

S'ha optat per Essence Prime com a llenguatge de modelització perquè permet descriure el problema de manera més intuïtiva, facilitant la comprensió del problema i la seva traducció a restriccions. Alhora, comporta no haver-se de preocupar pels detalls del solucionador específic. Savile Row li proporciona aquesta flexibilitat per ajustar-se a una gran varietat de solucionadors, incloent SAT, MaxSAT i SMT, el que permet treballar amb un enfocament potent.
A més, ens resulta útil en el nostre treball per a dur a terme comparacions amb el nostre model codificat en SAT, ja que podem aprofundir en les diferències i avantatges entre les dues aproximacions.

S'ha realitzat prèviament un estudi del llenguatge Essence Prime basat en el Manual de Savile Row [7].

4.1.4 PDDL

La utilització del llenguatge estàndard PDDL (Planning Domain Definition Language) es fonamenta en el seu reconeixement en el camp de la planificació. Ha estat de gran ajuda per al nostre aprenentatge el Manual PDDL - The Planning Domain Definition Language [8].

És un llenguatge que té una estructura clara dividint el problema a resoldre en dos arxius: un que defineix el domini del problema i un altre que descriu una instància específica del problema.

- Fitxer del Domini: s'estableixen els predicats i les accions que defineixen les

regles i les operacions possibles en el domini del problema. Els predicats representen propietats o relacions entre objectes, mentre que les accions es defineixen amb les condicions que s'han de complir per dur-se a terme (precondicions) i amb com els predicats canvien en executar-se l'acció (efectes).

- Fitxer del Problema: es defineixen els objectes específics de la instància del problema, l'estat inicial de l'escenari i els objectius que s'han de complir per considerar el problema resolt.

Aquesta separació facilita flexibilitat en l'especificació. El Fitxer del Domini pot ser reutilitzat per diferents problemes que comparteixen regles similars, mentre que el Fitxer del Problema es pot adaptar fàcilment per diferents instàncies sense necessitat de reescriure les regles del domini.

4.1.5 Scala

Scala és un llenguatge de programació multi-paradigma modern dissenyat per expressar patrons de programació comuns d'una manera concisa, elegant i segura. Integra perfectament característiques dels llenguatges funcionals i orientats a objectes.

La decisió d'optar pel llenguatge de programació Scala per al desenvolupament dels programes traductors de representació d'instàncies es basa principalment en la nostra familiaritat prèvia amb aquest llenguatge degut a l'assignatura *Programació Declarativa. Aplicacions* de 4t de GEINF.

A més d'això, Scala ofereix una gran versatilitat en la implementació de funcions i una destacada eficàcia en la manipulació de cadenes de caràcters. Unes capacitats crucials per dur a terme les operacions de traducció de representacions.

4.2 Selecció del viewpoint, representació i optimització

Amb l'objectiu d'explorar diverses aproximacions i comparar-ne els resultats, s'han seleccionat quatre enfocaments diferents: MiniZinc, Essence Prime, SAT i PDDL amb diversos viewpoints i representacions, i dos criteris d'optimització principals diferents.

4.2.1 Viewpoints i representació

- **MiniZinc i Essence Prime** comparteixen el mateix viewpoint en la representació del tauler. Aquest va ser inspirat pel treball de Cian et al.[9].

  Per a representar quina és la situació i orientació de cada cotxe o camió es consideren 3 coordenades que es distribueixen de la següent manera tal com es feu a la Figura 4.1: hi ha una coordenada superior, una inferior i una vertical a l'esquerra.
  Les coordenades superior i inferior prenen valors de l'1 al 6, en sentit d'esquerra a dreta, representant les columnes del tauler. La coordenada superior es representa en forma negativa, mentre que la inferior es representa en forma positiva.
  La coordenada vertical es representa a l'esquerra i també pren valors de 1 a 6, en sentit de baix a dalt, representant les files del tauler.

D'aquesta manera per indicar on s'ubica un vehicle farem servir una coordenada, *initial*, i una altra *versus*:

- Si *versus* és positiu vol dir que estem fent referència a un vehicle que esta orientat horitzontalment. A més, *versus* ens indicarà en aquest cas la fila i *initial* la columna de la casella més a l'esquerra que ocupa el vehicle.

- Si *versus* és negatiu vol dir que estem fent referència a un vehicle que està orientat verticalment. A més, $|versus|$ ens indicarà la columna i *initial* la fila de la casella de més envall del vehicle.

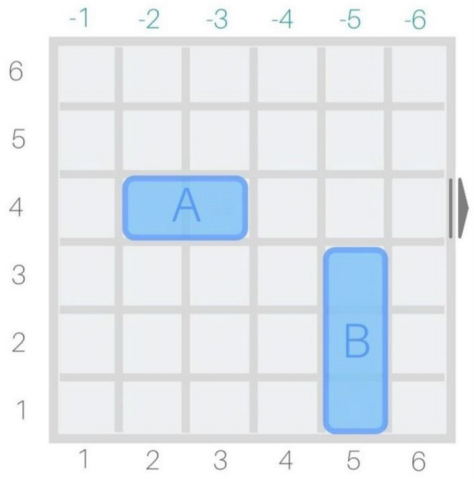


FIGURA 4.1: Exemple de representació amb viewpoint genèric.

Amb aquesta representació, la qual anomenarem "genèrica", serà possible determinar, amb tan sols dues coordenades, la posició exacta d'un vehicle i la seva direcció, amb l'ajuda del signe corresponent. Juntament amb un altre paràmetre que indicarà la mida del vehicle, aquest model compacte permetrà emmagatzemar una gran quantitat d'informació amb un ús eficient d'espai. Així per exemple el cotxe A i el camió B de la Figura 4.1 quedarien determinats respectivament amb:
sizes[A]=2, initial[A]=2, versus[A]=4 i *sizes[B]=3, initial[B]=1, versus[B]=-5*.

- **SAT** fa servir com a dades d'entrada el mateix viewpoint que els anteriors, però el viewpoint es modifica per adaptar-se a la codificació de les restriccions i les variables del problema de forma Booleana. En deixem la descripció per més endavant quan en donguem el model concret.

- **PDDL** implica un viewpoint més tradicional, amb un format compatible amb PDDL, centrat en una descripció molt completa de l'estat inicial, l'objectiu i les accions possibles dels vehicles.
  Les posicions del tauler es representen mitjançant un sistema de coordenades bidimensional, on s'utilitzen dos eixos per identificar cada casella del tauler. L'eix "X" s'utilitza per numerar les files del tauler i l'eix "Y" per les columnes.

Representació lexicogràfica (o de Fogleman)

I, per últim, una representació molt important en el nostre projecte ha estat la que va desenvolupar Michael Fogleman en el seu treball "Solving Rush Hour, the Puzzle" [10]. Fogleman va realitzar un treball essencial en aquest àmbit, el qual ens ha servit com a inspiració durant tot aquest projecte.
Ell, en tot el seu treball, utilitza un viewpoint de representació lexicogràfic on cada caràcter representa una casella del tauler. Les lletres simbolitzen els vehicles, i els punts representen els espais en blanc. El cotxe vermell sempre és representat amb la lletra A, mentre que la resta dels vehicles es representen de manera consecutiva seguint l'ordre alfabètic. Això significa que el primer vehicle diferent al cotxe vermell es representa amb la lletra B, el segon amb la C, i així successivament.
En mostrem un exemple:

GBB.L.GHI.LMGHIAAMCCCK.M..JKDDEEJFF.

Aquest exemple està representat al tauler de la Figura 4.2.

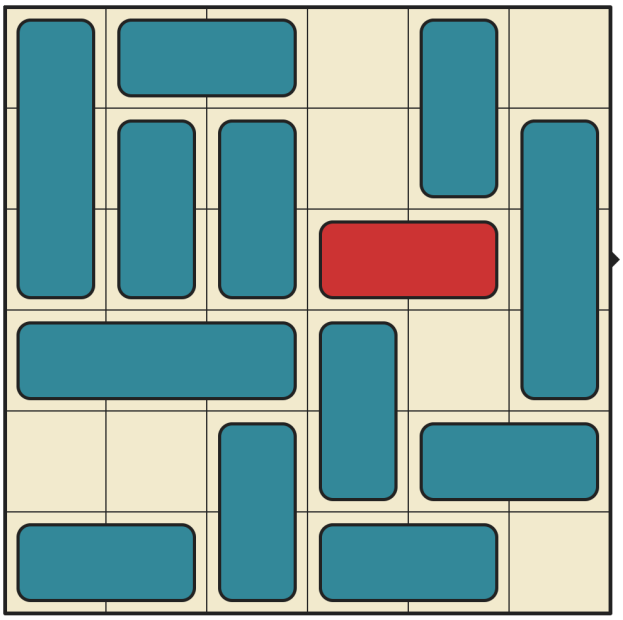


FIGURA 4.2: Tauler corresponent a la representació: "GBB.L.GHI.LMGHIAAMCCCK.M..JKDDEEJFF." de Fogleman.

Com que en l'experimentació treballem amb les instàncies que ell va aconseguir generar, hem creat un traductor per passar d'aquesta expressió a la forma de representació genèrica. Aquest traductor el detallarem també en el capítol 7.

La selecció de diverses aproximacions té com a objectiu proporcionar una comparació exhaustiva dels resultats. Utilitzar MiniZinc i Essence Prime amb el mateix viewpoint ens permetrà aprofundir en les diferències entre aquests dos llenguatges i avaluar-ne l'eficàcia en la modelització del problema. L'enfocament SAT permetrà explorar la utilitat d'un enfocament basat en restriccions booleanes, i l'ús de PDDL ens donarà una visió general del problema des d'una perspectiva de planificació.

4.2.2 Optimització

De cada aproximació hem considerat dos criteris d'optimització diferents utilitzant però models pràcticament idèntics. La diferència es centra amb diferents restriccions en els moviments dels vehicles, cada un amb un propòsit específic.

El primer model permet la movilitat completa dels vehicles, possibilitant-los moure's una o més caselles d'acord amb les seves restriccions. Aquest model ens servirà per a la recerca de solucions òptimes en termes de mínim de passos necessaris per resoldre el problema.

D'altra banda, el segon model restringeix els moviments dels vehicles a només un desplaçament per pas. Això significa que cada vehicle només pot moure's una casella en cada moviment. Aquest model ens ajudarà a trobar l'optimalitat en termes de desplaçaments mínims requerits per arribar a una solució, és a dir, la distància total recorreguda de tots els vehicles. És important mencionar que, en aquest segon model, quan parlem de "desplaçaments", significarà el mateix que dir "passos" ja que aquests només podran ser d'un desplaçament.

La implementació d'aquests dos models diferents ens permetrà examinar els resultats i les solucions generades per cadascun d'ells, ajudant-nos a comprendre millor les implicacions de les restriccions de moviment en la resolució del joc Rush Hour.

A més, pels llenguatges de MiniZinc, Essence Prime i PDDL hem creat un tercer model que aplica conjuntament les dues optimitzacions. És a dir, que, per la solució amb desplaçaments lliures resolta amb el mínim nombre de passos, també es minimitzin el nombre total de desplaçaments.

Més endavant, en el Capítol 7, es detallaran els models implementats i com s'han adaptat a cada aproximació específica, incloent la parametrització de les instàncies i les variables corresponents.

4.3 Instàncies interessants

Aquest projecte s'ha centrat en les instàncies resolubles que compleixen amb una sèrie de regles específiques, assegurant que siguin autènticament singulars i presents en la nostra base de dades. Aquest concepte el va definir Fogleman en el seu treball [10].

Anomenem instàncies "interessants" a les que presenten les següents característiques:

- Cap fila pot estar totalment omplerta amb peces horitzontals i cap columna pot estar totalment omplerta amb peces verticals. Aquestes només formen parets de peces que mai es podran moure. No són gaire interessants.
- Cap peça horitzontal pot aparèixer a la fila principal excepte la peça principal en si. Això permetria que altres cotxes també sortissin del tauler.
- Les instàncies haurien de ser totalment "sense resoldre" - no es poden fer més difícils movent les peces al voltant.

- Les instàncies haurien de ser "mínimes" - cada peça en el puzle és important. Treure qualsevol peça alteraria la solució.

Des dels inicis hem considerat aquest enfocament molt apropiat, ja que ens permet limitar el nombre d'instàncies amb les quals treballem, evitar repeticions innecessàries i instàncies redundants.

4.4 Selecció d'instàncies per a l'experimentació

Després d'haver realitzat l'experimentació i analitzat un total de 24,483 configuracions en el Capítol 8, s'ha decidit seguir un criteri específic per seleccionar les configuracions que seran utilitzades per a l'anàlisi i la comparació dels diferents models que hem creat. Aquest procés de selecció ens permetrà disposar d'una mostra representativa de configuracions que aborden diferents aspectes rellevants de la nostra investigació.

Concretament, hem decidit seleccionar les vint configuracions amb els valors més alts en quatre factors diferents. Aquesta selecció s'ha efectuat en l'ordre descrit a continuació i, un cop escollides les configuracions d'un grup, les eliminem abans de seleccionar les del següent. Aquesta metodologia assegura que no es repeteixen configuracions en més d'un grup.

Els quatre factors que hem considerat per a la selecció són els següents:

1. **Temps d'execució en PDDL**
2. **Passos mínims**
3. **Relació cost total (desplaçament) / passos**
4. **Estats accessibles**

Aquest recull de 80 configuracions seleccionades seran utilitzades per analitzar i comparar els diferents models que hem creat, amb l'objectiu de determinar la seva eficàcia en relació amb els factors mencionats. A més, ens permetrà identificar quines característiques influeixen en la capacitat dels models per resoldre els problemes.

Els experiments amb aquestes instàncies ens han suggerit considerar algun altre factor que influeix notablement en la dificultat. Aquests factors són els de les instàncies que tenen un nombre passos mínims més alt, i les que tenen uns desplaçaments mínims més alts. En concret, vàrem considerar instàncies amb un nombre de passos superior a 40 i amb un desplaçaments superiors a 70. D'aquestes ens hem quedat amb les que tenen un conjunt d'estats accessibles superiors a 10000. En total hem aconseguin un conjut de 43 instàncies que són més difícils, no pel PDDL però si per la resta d'aproximacions tal com veurem al capítol d'experimentació.

Capítol 5

Metodologia i enfocament

Al tractar-se d'un treball més encarat a la recerca, hem seguit una metodologia similar a la que el meu tutor fa servir per a alguns dels treballs que realitza. Així doncs, no hem optat per seguir un marc metodològic específic, com Scrum o altres, ja que aquests són més adients per a projectes de desenvolupament de productes, mentre que el nostre treball se centra en la investigació i la resolució de problemes. La metodologia ha consistit en definir uns objectius inicials amb unes fases o etapes successives per a assolir-los. Per garantir l'evolució adequada del projecte és ideal fer reunions de seguiment.

Des de l'inici, hem mantingut una comunicació constant amb el meu tutor a través de reunions setmanals o quinzenals en funció dels objectius concrets que ens plantejàvem per a cada etapa del treball. Com hem dit, aquestes etapes les vam definir al principi i les hem seguit en funció del temps real requerit per anar-les superant. Les sessions de treball conjuntes ens han permès compartir i revisar els avenços realitzats, i analitzar i definir les direccions futures de manera conjunta. Les fases del projecte s'expliquen amb detall al següent apartat.

Aquesta forma de treballar ens ha permès mantenir-nos enfocats i garantir que cada pas que preníem estava alineat amb els objectius finals. El fet de definir objectius clars a curt termini, ens ha proporcionat una estructura organitzada per a cada fase del projecte.

És veritat que en alguns moments hem hagut de fer canvis en la nostra planificació inicial degut a la necessitat d'aprofundir més en algun tema o altre. En aquests casos, vam adaptar-nos i centrar-nos en les àrees que consideràvem més importants o viables.

En part aquestes necessitats de canvi han sigut propiciats per un fet personal que ha esdevingut durant el desenvolupament del projecte, i és el meu acceptament condicionat a una beca de doctorat de la University of St. Andrews. Això ha suscitat la necessitat d'aprofundir més en alguns temes o altres d'acord amb les necessitats de formació per a poder treballar amb el grup de St. Andrews. En particular, calia que em centrés més en Essence Prime, SAT i PDDL i, com es veurà, és el que hem fet.

Capítol 6

Planificació i desenvolupament del projecte

En aquest apartat explicarem quina va ser la nostra planificació inicial i mostrarem amb detall quin ha sigut el desenvolupament final del projecte.

6.1 Planificació inicial

Partint d'una reunió inicial amb el tutor ens vàrem plantejar que caldrien les següents fases i etapes successives per a portar a terme el projecte:

- Investigació preliminar i definició d'objectius
- Preparació del clúster
- Generació d'instàncies
- Desenvolupament de models: MiniZinc, SAT, PDDL
- Analisi de dificulatat de les instàncies
- Avaluació i comparació de models
- Generació d'instàncies amb QBF
- Documentació de la memòria

Aquestes etapes no es van dimensionar inicialment amb temps sinó que s'entenia que amb el seguiment setmanal s'aniria ajustant la dedicació i abast de cadascuna.

Tot seguit mostrarem quin ha sigut el desenvolupament real, en què ha consistit cada fase amb més detall. També indicarem quines tasques s'han hagut de portar a terme que no s'havien previst, per exemple el desenvolupament de traductors de format de les instàncies.

Remarquem aquí que no hem pogut desenvolupar la part de la generació d'instàncies amb QBF. Malaruadament, la feina amb els altres models, la gestió del clúster i la generació d'instàncies han portat massa temps com per poder abordar adequadament aquesta fase. Vist amb perspectiva era un objectiu condicional que es va

proposar inicialment al full del TFG però que, segons el tutor, ha quedat fora de l'abast del present TFG, i de fet es planteja com a treball futur ja que la modelització que s'ha fet amb SAT ens deixa a les portes d'aquesta tasca.

6.2 Desenvolupament final del projecte

6.2.1 Investigació preliminar i definició d'objectius

Abans de començar qualsevol aspecte pràctic del projecte, és important adquirir una base sòlida de coneixement. En aquesta etapa hem dedicat temps a investigar en profunditat el joc Rush Hour. Això inclou la comprensió detallada de les regles del joc, els estudis que s'han realitzat i les estratègies que han estat utilitzades per resoldre'l.

Els primers estudis que vam trobar ens van proporcionar una base sólida per entendre els aspectes fonamentals del joc. En particular hem consultat [2, 9 - 11].

Aquesta avaluació de les publicacions existents així com les de les diverses aproximacions i llenguatges disponibles per a resoldre problemes de Rush Hour, ens ha ajudat a identificar les oportunitats de millora que el nostre projecte podria abordar, i així, a definir els objectius generals del projecte. Per exemple vàrem identificar un model MiniZinc i un en PDDL però no hi era en Essence Prime ni un ad hoc en SAT. També vàrem observar que les nocions d'optimalitat considerades eren limitades.

6.2.2 Funcionament del clúster i execucions

En aquesta etapa, s'explora el funcionament del clúster com a eina per a l'execució de programes que requereixen molta memòria i temps de càlcul, o per a experiments que requereixen de molts còmputs diferents.

Estudi del clúster

Primerament, s'ha dedicat temps a comprendre el funcionament del Clúster i com pot ser utilitzat per a executar els programes que caldria per les necessitats del nostre projecte. Aquesta tasca ha ocupat les primeres setmanes de la fase de generació de configuracions.

Instal·lació requerida

Per dur a terme l'execució dels diferents recursos en el Clúster, ha estat fonamental dur a terme les instal·lacions i configuracions de diverses eines i llibreries necessàries pels programes que hem desenvolupat.

Aquesta tasca ens ha donat l'oportunitat d'explorar l'entorn del clúster i d'aprendre a realitzar instal·lacions en aquest tipus de sistema, una àrea en què no teníem experiència prèvia. Tot i això, en alguns casos, hem experimentat dificultats relacionades amb la compatibilitat de versions. Aquests obstacles ens han requerit més temps del que inicialment havíem previst, ja que hem hagut de solucionar problemes específics de compatibilitat.

Execució de programari

Al llarg de tot el desenvolupament d'aquest projecte s'aprofiten les capacitats del clúster amb diferents objectius. S'executa el generador d'instàncies per obtenir-ne amb major tamany de tauler; es realitzen scripts per executar en cadena varis programes; s'executen els diversos models en les configuracions triades, entre d'altres.

6.2.3 Generació de configuracions

Un cop adquirida la capacitat d'utilitzar correctament el Clúster, ens hem disposat a llançar el codi Generador de les Configuracions Inicials, que es detalla en el següent capítol, per a diferents tamanys de tauler.

Preparació i configuració del generador en el clúster

La tasca principal ha estat instal·lar el codi del generador en l'entorn del Clúster per poder executar-lo. En aquest cas, no s'ha requerit la creació d'un nou codi, ja que, com explicarem més endavant, hem utilitzat una eina existent. Per tant, ens hem assegurat de que tot el programari necessari estigués correctament configurat al clúster per poder transferir el generador i executar-lo adequadament.

Proves

Els tamanys fins a 6×6 no han requerit una complexitat molt gran, s'han generat de manera ràpida i eficient. En canvi, per als tamanys de 7×7 i superiors, hem hagut d'afrontar-nos a reptes significatius relacionats amb la memòria.

Érem conscients que aquests taulers més grans podrien ser més complicats de gestionar, però no teníem una estimació del temps que requeririen. Com que era la primera vegada que féiem ús del clúster i no sabíem la seva capacitat, i el codi que utlitzàvem no era desenvolupat per nosaltres, no teníem expectatives.

Els programes per als taulers amb mides superiors a 6×6 trigaven dies o fins i tot setmanes per executar-se. Si més no, quan més gran la mida del tauler, menys tarda a aturar-se, amb errors relacionats amb la memòria.

Tot això ha complicat significativament l'obtenció de resultats i va requerir una inversió molt gran de temps. No obstant això, vam voler centrar-nos en el tauler de mida 7×7, realitzant nombroses proves fins al final d'aquest projecte, intentant generar la màxima quantitat de configuracions possible. Malauradament el conjunt generat no ha resultat suficientment significatiu. Escenaris més grans com 8×8 han sigut del tot descartats.

6.2.4 Desenvolupament del model MiniZinc

En aquesta fase del projecte, s'opta per utilitzar el llenguatge MiniZinc per a la creació del primer model del joc Rush Hour. El procés d'investigació no ha requerit una quantitat significativa de temps ja que teníem familiaritat amb aquest llenguatge. A més, com s'explicarà més endavant, es va trobar un model preexistent en MiniZinc que s'ajustava, en primera instància, a les necessitats del nostre projecte . Això ens ha permès estalviar temps i centrar-nos en optimitzar i millorar aquest model existent en lloc de crear-ne un completament nou.

6.2.5 Desenvolupament dels traductors en Scala

Durant aquesta fase es desenvolupen dos traductors amb l'objectiu d'automatitzar el procés de transformació d'una representació a una altra de les configuracions del tauler.
En primer lloc, s'ha creat un traductor per passar de la Representació Lexicogràfica a la Genèrica, la qual s'utilitza com a entrada per la majoria dels models utilitzats.
I, més endavant, s'ha desenvolupat un segon traductor que transforma la representació genèrica a un arxiu amb format específic per a PDDL, el qual s'utilitza com a entrada pel model i descriu la instància que es tractarà.

Amb aquesta automatització, s'ha buscat estalviar temps i minimitzar els errors que podrien sorgir durant el procés de conversió manual.

6.2.6 Desenvolupament del model Essence Prime

En aquesta etapa, s'ha realitzat prèviament un estudi del llenguatge Essence Prime. Aquesta investigació juntament amb el model previ creat en MiniZinc com a referència, han proporcionar una base per a la construcció d'aquest model, iniciat des de zero.

6.2.7 Desenvolupament del model SAT amb Optilog

En aquesta fase, s'ha desenvolupat el model SAT utilitzant OptiLog, un marc de treball amb el qual no hi tenia familiaritat.

El procés de creació d'aquest model ha estat més complicat i ha requerit més temps en comparació amb els altres models. A més, també ha estat el més gran en quant a codificació, i les variables que s'han utilitzat han estat diferents, adaptant-se a la manera en què OptiLog funciona.

6.2.8 Desenvolupament del model PDDL

Aquesta fase també ha partit d'un model existent, que s'ha utilitzat com al model de restricció de moviments. Tot i això, ha calgut aprendre també a modelitzar amb PDDL ja que calia entendre bé el model. A més, a partir d'aquest, s'han aplicat modificacions per aconseguir l'altre model, permetent la movilitat total dels vehicles.

6.2.9 Anàlisi de complexitat i dificultat

En aquesta etapa del projecte es realitza un anàlisi exhaustiu de la complexitat i la dificultat de les instàncies. Aquest estudi s'ha fet a partir dels estudis prèviament realitzats, juntament amb les conclusions extretes de les dades que hem tractat en l'experimentació.

6.2.10 Avaluació i comparació de models

En aquesta secció s'avaluen els diferents models creats amb les instàncies escollides per veure les que costen més al solver.

6.2.11 Documentació de la memòria

Finalment s'ha completat la memòria amb un esforç explicatiu i gràfic a la part d'experimentació per a explicar adequadament els resultats.

La Figura 6.1 mostra el nostre Diagrama de Gantt, que il·lustra la seqüència temporal del desenvolupament final del nostre projecte. Aquesta representació gràfica ens ajuda a visualitzar les diferents fases planificades i les dependències entre elles. Cada barra horitzontal representa una tasca o activitat específica, amb les seves durades indicades en setmanes.

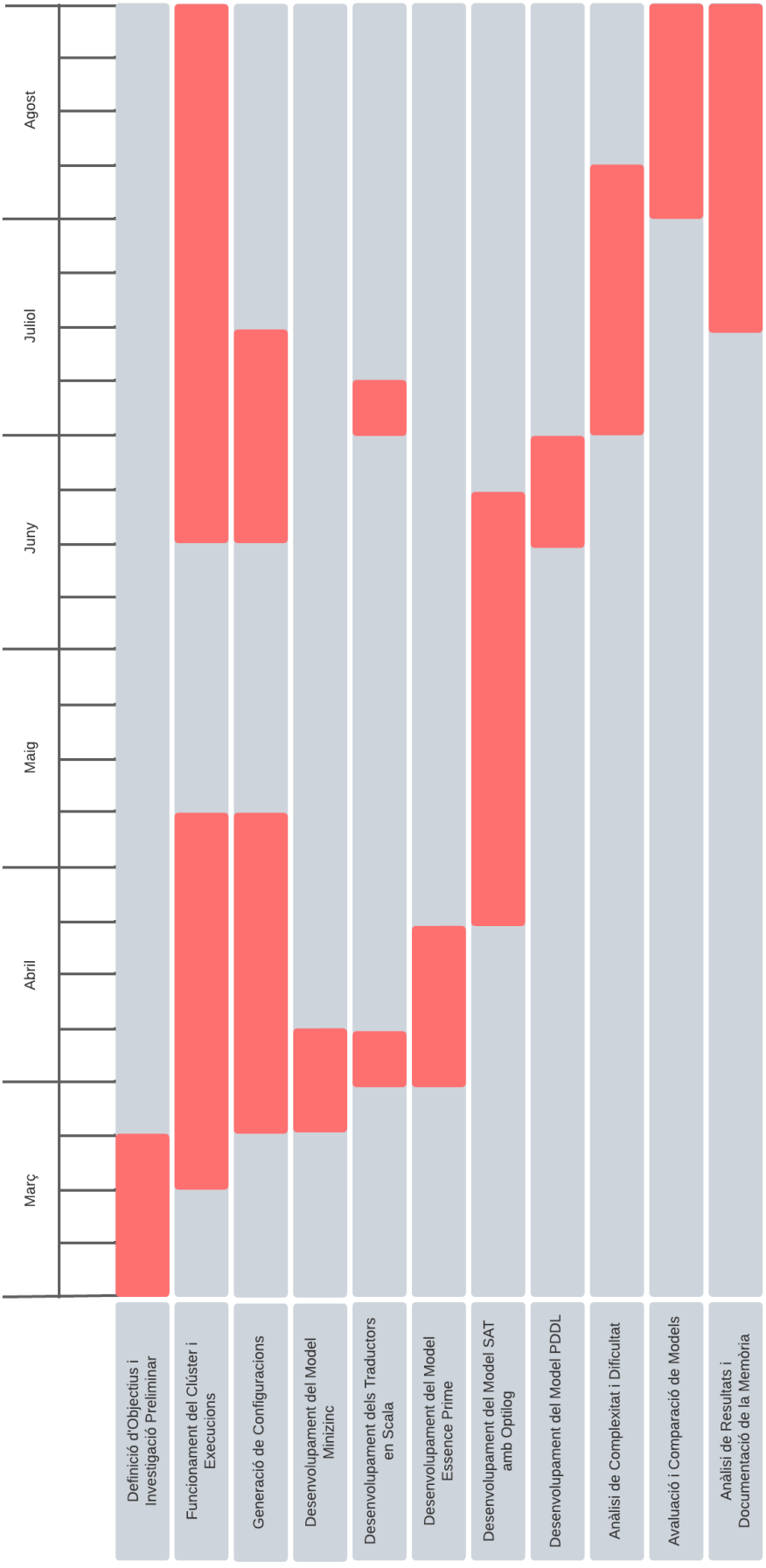


FIGURA 6.1: Diagrama de Gantt

Capítol 7

Implementació de models i programes

En aquesta secció descriurem les aportacions principals del treball que consisteixen amb les diferents modelitzacions i implementacions que hem dut a terme per a poder fer l'estudi comparatiu prèviament esmentat.

Primer de tot ens referirem al generador d'instàncies que hem fet servir per a poder tenir una base sobre la que comparar els nostres models i implementacions.

7.1 Generador de configuracions inicials

Per a aquest propòsit, hem pres un codi creat per Fogleman [10] com a punt de partida per a la nostra implementació.

Ell va desenvolupar un codi en C++ que és capaç de generar totes les configuracions inicials "interessants" de taulers fins a 6×6. En el seu treball, explica que no va poder resoldre el cas de 7×7 ja que requereix recursos que no es pot permetre, probablement un temps exccessiu de CPU.

Aquest generador en C++ crea una sortida d'un fitxer on cada línia inclou una representació lexicogràfica de la configuració, juntament amb el nombre mínim de passos per resoldre-la i el nombre d'estats accessibles (vegeu Figura 7.1).

Com podem observar, ell ha generat també les configuracions amb fins a 2 parets, representades amb la lletra "x". Aquestes parets de dimensió 1×1 actuen com a blocs fixos que limiten el moviment dels vehicles. Cal destacar que en el nostre projecte no hem tingut en compre les configuracions que contenen parets.

El seu codi també inclou diverses utilitats programades en el llenguatge Go, per a tasques com resoldre o representar visualment les configuracions. Aquestes eines les hem utilitzat durant el desenvolupament del nostre projecte.

7.2 Models

En aquest apartat, explorem en profunditat els diferents models de resolució que hem desenvolupat per abordar el desafiament del joc Rush Hour.

```
60 IBBxooIooLDDJAALooJoKEEMFFKooMGGHHHM 2332
58 BBoKMxDDDKMoIAALooIoJLEEooJFFNoGGoxN 9192
55 ooBBMxDDDKMoAAJKoNooJEENIFFLooIGGLox 9712
55 ooBBMxDDDKMoAAJKoNooJEENIFFLooIGGLHH 8257
54 oxCCMoDDDKMoAAJKooooJLEEIFFLoNIGGoxN 6355
54 oooJLxCCCJLoHAAKooHoIKDDooIEEMoFFoxM 3461
54 oooJxoCCCJLoHAAKLoHoIKDDooIEEMoFFoxM 2948
54 BBBKCCDDoKoLIAAKoLIoJEEMFFJooMooxoHH 1845
53 BBBCCNDDoxMNJAAoMOJoKFFOGGKLooxIILoo 4705
53 ooBBoxDDDKooAAJKoMooJEEMIFFLooIGGLHH 4228
53 ooBBoxDDDKooAAJKoMooJEEMIFFLooIGGLox 4102
52 oxCCMNDDDKMNAAJKooooJEEoIFFLooIGGLox 7178
52 oxCCMNDDDKMNAAJKooooJLEEIFFLooIGGHHo 5717
51 xCCoLMooJoLMAAJoLNHIDDDNHIoKEExooKGG 9360
51 GBBoLoGHIoLMGHIAAMCCCKoMooJKDDEEJFFo 4780
```

FIGURA 7.1: Exemple de les instàncies generades amb el Generador de Fogleman.

Com s'ha explicat en el Capítol 4, hem desenvolupat tres models pràcticament idèntics per aquest llenguatge, per abordar optimitzacions diferents. En els apartats següents, ens endinsarem a descriure i mostrar el model que permet una mobilitat total dels vehicles, i sols mencionarem els canvis que hem implementat en els altres models.

7.2.1 Model MiniZinc

Per al desenvolupament d'aquest model, ens hem basat també en el treball de Cian et al. [9].
A partir del seu model de resolució en MiniZinc, s'han realitzat adaptacions per aconseguir-ne un de més genèric, per abordar el joc en taulers de diverses mides. Això s'ha aconseguit mitjançant la parametrització de la mida del tauler amb la variable `boardSize` i la substitució en els dominis i restriccions on s'utilitzava un enter per representar-la, limitant el model a un context genèric.

En aquest model, és necessari passar per paràmetre el nombre de passos necessaris per resoldre el joc, el que fa la tasca del solucionador molt més senzilla. Ens interessa que sigui el nostre model el que ens digui quin és el nombre mínim de passos que es necessiten per a resoldre una instància concreta. Per a això, s'ha creat un programa en Python que comença amb l'hipòtesi que el joc es pot resoldre en només 1 pas i prova a executar el model de MiniZinc. Si el model no es pot resoldre, el programa incrementa el nombre de passos i prova novament, i així segueix fins a trobar la quantitat mínima de passos necessaris per resoldre el joc.
Aquest script en Python que executa el nostre model el mostrem a l'Annex A. Cal dir que aquesta és l'aproximació més natural per a resoldre problemes de planificació en els que es vol minimitzar el nombre d'accions de les solucions.

Per modelar les instàncies, definim una sèrie de paràmetres i variables que ens ajuden a representar la disposició dels vehicles en el tauler i les posicions que ocupen. És una representació simple i eficient que facilita la definició de les restriccions i els moviments dels vehicles que s'utilitzaran per resoldre el joc.

```
int: steps;
int: vehicles;
int: boardSize;

array[1..vehicles] of 2..3: sizes;
array[1..vehicles] of -boardSize..boardSize: versus;
array[1..vehicles] of int: initial;

array[1..vehicles, 1..steps] of var 1..boardSize: pos;
array[1..vehicles, 1..steps-1] of var -(boardSize-2)..(boardSize-2): move;
```

Declaració de paràmetres i variables

Els paràmetres són valors que es proporcionen a l'hora d'executar el model, és a dir, són declarats i el seu valor no canvia. Això permet adaptar el model a diferents instàncies del joc sense necessitat de modificar el codi.

Comencem definint tres paràmetres de tipus enter:

- `boardSize`: defineix la mida del tauler. Això ens permetrà que el model sigui genèric i accepti diferents dimensions de tauler.
- `vehicles`: defineix la quantitat de vehicles.
- `steps`: defineix la quantitat de passos amb la que es provarà de resoldre el model.

Llavors definim tres vectors de longitud igual a la quantitat de vehicles que hi ha en el tauler inicialment:

- `sizes`: emmagatzema la mida de cada vehicle, que pot ser 2 o 3. Aquesta mida representa la quantitat de posicions o caselles que un vehicle ocupa en el tauler.
- `versus`: determina la direcció de cada vehicle. Si el valor és més gran que 0, el vehicle és horitzontal i representa la fila on es troba; i si el valor és menor que 0, el vehicle és vertical i el valor absolut en representa la columna. El valor que pren s'estén des de `-boardSize` fins a `boardSize`.
- `initial`: guarda la fila o columna contrària a la direcció del vehicle, i representa la posició més baixa i a l'esquerra on està situat. El valor que pren s'estén des de 1 fins a `boardSize`-1, ja que el vehicle més petit, de dimensió 2, mai podrà començar en la posició 6. Vegeu Figura 4.1 i descripció i exemple corresponent.

Les variables són valors que el model intenta determinar durant la resolució del problema, en definim dos:

- `pos`: matriu bidimensional que representa la posició de cada vehicle en cada pas. Aquesta posició en el primer pas equivaldrà valor inicial del vehicle, el que guarda `initial`. Podrà prendre valors de 1 a `boardSize`-1 si el vehicle és un cotxe, i de 1 a `boardSize`-2 si el vehicle és un camió. Fixem-nos que la fila (si te orientació horitzontal) o columna (si la te vertical) a la que es troba el vehicle no canviarà i per tant es pot saber consultant versus i initial.

- `move`: matriu bidimensional que guarda el moviment de cada vehicle en cada pas, menys en l'últim, que ja és l'estat final. Si el valor és 0, el vehicle no es mou en aquest pas de temps; en cas contrari, el valor representa la quantitat de posicions que es mou; positiu si és en sentit dreta o amunt, negatiu si és en sentit esquerra o avall. Els moviments estan restringits a un rang de -(`boardSize`-2) a `boardSize`-2.

Restriccions i objectiu

- Les posicions inicials dels vehicles s'estableixen mitjançant una restricció que assegura que en el primer pas cada vehicle ha d'estar en la seva posició inicial.

```
constraint forall(v in 1..vehicles)(pos[v,1]=initial[v]);
```

- L'objectiu és aconseguir que el cotxe vermell, el primer vehicle declarat, arribi a la sortida en l'últim pas. La sortida és la posició `boardSize`-1.

```
constraint pos[1,steps]=boardSize-1;
```

- S'evita que els vehicles surtin del tauler mitjançant la restricció que garanteix que totes les posicions estan dins dels límits.

```
constraint forall(v in 1..vehicles, s in 1..steps)
    (pos[v,s]+sizes[v]-1 <= boardSize);
```

- S'assegura que no se superposin els vehicles amb la mateixa direcció, ni tampoc els vehicles verticals amb els horitzontals, i al contrari.

```
constraint forall(v1,v2 in 1..vehicles, s in 1..steps
  where (v1 < v2 /\ versus[v1] = versus[v2]))
    (pos[v1,s]+sizes[v1]-1 < pos[v2,s] \/
    pos[v2,s]+sizes[v2]-1 < pos[v1,s]);

constraint forall(v1,v2 in 1..vehicles, s in 1..steps
  where (versus[v1] > 0 /\ versus[v2] < 0))
    (not (pos[v1,s] <= -versus[v2] /\
    -versus[v2] <= pos[v1,s]+sizes[v1]-1 /\
    pos[v2,s] <= versus[v1] /\ versus[v1] <= pos[v2,s]+sizes[v2]-1));
```

- Es garanteix que, en cada pas, exactament un vehicle realitzi un moviment. Això evita que en un instant de temps es mogui més d'un vehicle, cosa que podria produir inconsistències.

```
constraint forall(s in 1..steps-1)
     (sum(v in 1..vehicles)(move[v,s]!=0) = 1);
```

- S'assegura que les posicions futures siguin consistents amb els moviments actuals.

```
constraint forall(v in 1..vehicles, s in 1..steps-1)
    (pos[v,s+1] = pos[v,s] + move[v,s]);
```

- S'implementen restriccions per evitar salts de vehicles a la mateixa fila o columna i per evitar salts en vehicles ortogonals.

```
constraint forall(s in 1..steps-1, v1,v2 in 1..vehicles
  where ((v1<v2) /\ versus[v1]=versus[v2]))(
    not (pos[v1,s]<=pos[v2,s] /\ pos[v1,s+1] > pos[v2,s+1]) /\
    not (pos[v2,s]<=pos[v1,s] /\ pos[v2,s+1] > pos[v1,s+1]));

constraint forall(s in 1..steps-1, v1,v2 in 1..vehicles
  where (versus[v1] < 0 /\ versus[v2] > 0))(
    (pos[v2,s] <= -versus[v1] /\
    -versus[v1] <= pos[v2,s]+sizes[v2]-1) ->
        (pos[v1,s] < versus[v2] -> pos[v1,s+1] < versus[v2]) /\
        (pos[v1,s] > versus[v2] -> pos[v1,s+1] > versus[v2]));

constraint forall(s in 1..steps-1, v1,v2 in 1..vehicles
  where (versus[v1] > 0 /\ versus[v2] < 0))(
    (pos[v2,s] <= versus[v1] /\
    versus[v1] <= pos[v2,s]+sizes[v2]-1) ->
        (pos[v1,s] < -versus[v2] -> pos[v1,s+1] < -versus[v2]) /\
        (pos[v1,s] > -versus[v2] -> pos[v1,s+1] > -versus[v2]));
```

- S'assegura que un vehicle no realitzi dos moviments consecutius, el que suposaria fer passos innecessaris[1]

```
constraint forall(v in 1..vehicles,s in 1..steps-2)(
    move[v,s] * move[v,s+1]=0);
```

- L'ordre `solve satisfy` busca trobar una solució que satisfaci totes les restriccions.

```
solve satisfy;
```

A continuació, mencionem les modificacions que hem exercit per obtenir el segon i tercer model.

Pel segon model, que busca la optimització de desplaçaments, en la definició de variables hem modificat els valors que pot prendre la variable `move`. En aquest model aniran des de -1 a 1, per establir que els passos seran sols d'1 desplaçament.
En les restriccions, hem eliminat la que impedeix que un vehicle realitzi dos moviments consecutius, ara sí que ens interessa permetre-ho per trobar els desplaçaments totals mínims.

Pel tercer model, definim la variable `total_moves`, que guarda la suma del nombre total de desplaçaments de tots els vehicles.

```
var int: total_moves = sum([abs(move[v,s]) | v in 1..vehicles,
  s in 1..steps-1 where move[v,s]!=0]);
```

I busquem trobar una solució que minimitzi aquest nombre.

```
solve minimize total_moves;
```

[1] Compte, aquesta restricció només te sentit en un tipus d'optimització, la de passos, com direm més endavant, per l'altra versió s'eliminarà.

7.2.2 Model Essence Prime

Aquest model ha estat creat des de zero, tot i que s'ha inspirat en el model de MiniZinc. Donat que aquests dos llenguatges tenen similituds considerables, no ha estat una tasca complexa adaptar-lo al context de l'Essence Prime.

En aquest model també s'ha d'entrar per paràmetre el nombre de passos necessaris per resoldre el joc. Hem generat un script en Bash que realitza l'execució del model amb el menor nombre de passos possible i va incrementant aquest valor fins a trobar una solució. Aquest script el mostrem a l'Annex B.

Tant les declaracions de paràmetres i variables com les restriccions i objectius en aquests dos models d'Essence Prime són equivalents als del model de MiniZinc, amb l'adaptació necessària al llenguatge. Aquesta similitud ens permetrà evitar la repetició de l'explicació detallada del codi, que el mostrem continuació.

```
given steps, vehicles, boardSize : int

given sizes: matrix indexed by [int(1..vehicles)] of int(2..3)
given versus: matrix indexed by [int(1..vehicles)] of
    int(-boardSize..boardSize)
given initial: matrix indexed by [int(1..vehicles)] of int(1..boardSize)

find pos: matrix indexed by [int(1..vehicles), int(1..steps)] of
    int(1..boardSize)
find move: matrix indexed by [int(1..vehicles), int(1..steps-1)] of
    int(-(boardSize-2)..boardSize-2)

such that
    forAll v : int(1..vehicles) . pos[v,1] = initial[v],

    pos[1,steps] = boardSize-1,

    forAll v : int(1..vehicles) .
        forAll s : int(1..steps) .
            (pos[v,s]+sizes[v]-1) <= boardSize,

    forAll v1 : int(1..vehicles) .
        forAll v2 : int(1..vehicles) .
            forAll s : int(1..steps) .
                v1 < v2 /\ versus[v1] = versus[v2] ->
                    pos[v1,s]+sizes[v1]-1 < pos[v2,s] \/
                    pos[v2,s]+sizes[v2]-1 < pos[v1,s],

    forAll v1 : int(1..vehicles) .
        forAll v2 : int(1..vehicles) .
            forAll s : int(1..steps) .
                versus[v1] > 0 /\ versus[v2] < 0 ->
                    ! (pos[v1,s]   <= -versus[v2] /\
                    -versus[v2] <= pos[v1,s]+sizes[v1]-1  /\
                    pos[v2,s] <= versus[v1]  /\
                    versus[v1] <= pos[v2,s]+sizes[v2]-1),

    forAll s : int(1..steps-1) .
        (sum v : int(1..vehicles) . move[v,s] != 0) = 1,

    forAll v : int(1..vehicles) .
        forAll s : int(1..steps-1) .
            pos[v,s+1] = (pos[v,s] + move[v,s]),
```

```
    forAll v1 : int(1..vehicles) .
        forAll v2 : int(1..vehicles) .
            forAll s : int(1..steps-1) .
                v1<v2 /\ versus[v1]=versus[v2] ->
                    !(pos[v1,s]<=pos[v2,s] /\ pos[v1,s+1] > pos[v2,s+1])
                    /\
                    !(pos[v2,s]<=pos[v1,s] /\ pos[v2,s+1] > pos[v1,s+1]),

    forAll v1 : int(1..vehicles) .
        forAll v2 : int(1..vehicles) .
            forAll s : int(1..steps-1) .
                (versus[v1] < 0 /\ versus[v2] > 0 /\
                pos[v2,s] <= -versus[v1] /\
                -versus[v1] <= pos[v2,s]+sizes[v2]-1) ->
                    (pos[v1,s] < versus[v2] -> pos[v1,s+1] < versus[v2])
                    /\
                    (pos[v1,s] > versus[v2] -> pos[v1,s+1] > versus[v2]),

    forAll v1 : int(1..vehicles) .
        forAll v2 : int(1..vehicles) .
            forAll s : int(1..steps-1) .
                (versus[v1] > 0 /\ versus[v2] < 0 /\
                pos[v2,s] <= versus[v1] /\
                versus[v1] <= pos[v2,s]+sizes[v2]-1) ->
                    (pos[v1,s] < -versus[v2] -> pos[v1,s+1] < -versus[v2])
                    /\
                    (pos[v1,s] > -versus[v2] -> pos[v1,s+1] > -versus[v2])

    forAll v : int(1..vehicles) .
            forAll s : int(1..steps-2) .
                (move[v,s] * move[v,s+1] = 0)
```

Pel segon model es fan els mateixos canvis que en MiniZinc.
I pel tercer model s'afegeix la variable `total_moves` i es restringeix que tingui per valor la suma dels desplaçaments totals. I també s'afegeix `minimising total_moves`, que buscarà trobar una solució que minimitzi aquest valor.

```
find total_moves: int(steps..10000)

minimising total_moves

such that
    total_moves = sum v : int(1..vehicles) .
        sum s : int(1..steps-1) .
            |move[v, s]|,
```

7.2.3 Model SAT-Optilog

La implementació del model SAT que utilitza el marc OptiLog l'hem desenvolupat des de zero, sense utilitzar cap base preexistent.

Aquest model rep els mateixos paràmetres que els models en MiniZinc i Essence Prime, i, a partir de la informació d'aquests, es creen les variables booleanes necessàries per a representar les posicions i moviments dels vehicles en cada instant de temps. Per aplicar les restriccions i assegurar que es compleixen les regles específiques del joc, s'utilitzen clàusules CNF.

Abans de tot, ens centrarem a detallar el significat d'aquestes variables booleanes.

Posicions
Amb els paràmetres sabem on es troba cada vehicle i la seva direcció, per tant, aquestes variables representen les posicions de la columna o fila, contrària al `versus`, en les que el vehicle es pot moure.

Les representem amb una p, seguida del número del vehicle, de l'instant de temps i de la posició de la fila/columna fixa on es troba, el que anomenem versus. Aquesta variable és a cert si el vehicle v en el temps t es troba en la posició k.

$$p_{vtk}$$

Moviments
Aquestes variables es representen amb una m seguida del número del vehicle, de l'instant de temps i de la posició inicial i la posició final del del vehicle en el moviment. Aquesta variable és a cert si el vehicle v en el temps t es desplaça de la posició i a la posició f.

$$m_{vtif}$$

Avaluació de moviment
Es representen amb una v seguida del número del vehicle i de l'instant de temps. Si la variable és a cert significa que el vehicle k en el temps t s'està movent.

$$v_{kt}$$

A continuació, mostrem i expliquem detalladament el programa optilog i el corresponent model que permet la mobilitat total dels vehicles de forma coherent i respectant les regles del joc.

Primer de tot, s'importen els mòduls requerits i s'obtenen els paràmetres que es passen com a cadena única a través d'un argument. Aquestes dades es descomponen i es guarden els valors en les variables pertinents.

També es realitza la inicialització de les variables necessàries per al funcionament del programa. El nombre de passos (`steps`) l'inicialitzem a 1 per a que el solucionador provi de resoldre el programa amb aquesta quantitat, i si no pot, es va augmentant fins que troba una solució, tal com hem explicat que fem amb MiniZinc i amb Essence Prime.

```
from optilog.modelling import *
from optilog.encoders.pb import Encoder
from optilog.formulas import VarManager, CNF
from optilog.solvers.sat import Glucose41

arguments = sys.argv[1]
parts = arguments.split()
try:
    arg1 = int(parts[0])
    arg2 = int(parts[1])
    arg3 = list(map(int, parts[2][1:-1].split(',')))
    arg4 = list(map(int, parts[3][1:-1].split(',')))
    arg5 = list(map(int, parts[4][1:-1].split(',')))
except (ValueError, IndexError):
    sys.exit(1)
```

```
boardSize=arg1
vehicles=arg2
sizes=arg3
versus=arg4
initial=arg5

interpretation = []
steps = 0
while interpretation == []:
    positions, movements_names, vehicles_names, conflicts = [], [], [], []
    n = 1
    variables = VarManager()
    solver = Glucose41()
    cnf = CNF(var_manager=variables)
    steps += 1
```

Per cada vehicle i temps, s'avaluen les posicions que el vehicle pot ocupar. Aquesta primera aproximació del rang de posicions que pot ocupar es defineix en funció de si hi ha més vehicles en el mateix `versus`.
Amb les posicions definides, es creen variables booleanes per a cada vehicle, temps i posició accessible, les quals s'agreguen a la llista global de variables. El mateix es fa amb les variables de moviments, entre les diferents posicions accessibles per cada vehicle i temps, i amb les variables que representen si hi ha moviment.

Per al temps inicial, es creen només les variables de moviment en què la posició inicial es correspon amb la posició del vehicle especificada en els paràmetres, ja que el primer pas no pot ser des de cap altra.

S'afegeix la primera restricció, que és posar a cert les variables de posicions dels vehicles en el temps inicial.

Per totes les variables que s'agreguen a la llista global, s'emmagatzemen les parelles de noms de les variables i els seus índexs a la llista per a, posteriorment, poder accedir a elles.

```
    for v in range(vehicles):
        p1, m1, v1 = [], [], []
        for s in range(steps+1):
            p2, m2 = [], []
            i = 1
            f = boardSize-sizes[v]+1

            if versus.count(versus[v]) > 1:
                actual_index = -1
                for o in range(versus.count(versus[v])):
                    actual_index = versus.index(versus[v], actual_index+1)
                    if initial[actual_index] < initial[v]:
                        i = i + sizes[actual_index]
                    elif initial[actual_index] > initial[v]:
                        f = f - sizes[actual_index]

            for p in range(i,f+1):
                var_pos = Bool('p'+str(v)+str(s)+str(p))
                variables.add_var(var_pos)
                p2.append((var_pos, n))

                for next_p in range(i,f+1):
```

```
                    if p != next_p and ((s==0 and p==initial[v])
                    or (s!=0)):
                        m2.append(Bool('m'+str(v)+str(s)+str(p)
                        +str(next_p)))

                if s == 0 and p == initial[v]:
                    cnf.add_clause([n])

                n = n+1

            if s < steps:
                m1.append(m2)
                v1.append(Bool('v'+str(v)+str(s)))

            p1.append(p2)

        positions.append(p1)
        movements_names.append(m1)
        vehicles_names.append(v1)
```

A continuació, es busquen els conflictes entre els diferents vehicles. Per cada vehicle, cada temps i cada posició, es guarden en un array les posicions dels altres vehicles en el mateix temps que no són compatibles. Un vehicle només pot tenir conflicte amb altres que tinguin el mateix `versus` o amb els vehicles ortogonals a ell.

En tenir les variables ordenades, primer es guarden les que representen les posicions, seguides de les que representen els moviments i de les que representen si hi ha moviment.

```
    for v in range(vehicles):
        c1 = []
        actual_versus = versus[v]
        for s in range(steps+1):
            c2 = []
            for p in positions[v][s]:
                c3 = []
                for other_v in range(vehicles):
                    if other_v != v and versus[other_v] == actual_versus:
                        for pos in positions[other_v][s]:
                            pos_other = str(pos[0])[-1]
                            pos_actual = str(p[0])[-1]
                            if initial[other_v] > initial[v]:
                                if int(pos_actual) + sizes[v] - 1
                                >= int(pos_other):
                                    c3.append(pos)
                            else:
                                if int(pos_other) + sizes[other_v] -1
                                >= int(pos_actual):
                                    c3.append(pos)
                    if other_v != v and
                    ((actual_versus > 0 and versus[other_v] < 0) or
                    (actual_versus < 0 and versus[other_v] > 0)):
                        for pos in positions[other_v][s]:
                            pos_other = str(pos[0])[-1]
                            pos_actual = str(p[0])[-1]
                            if int(pos_other) <= abs(actual_versus) and
                            int(pos_actual) <= abs(versus[other_v]):
                                if (int(pos_other) + sizes[other_v] - 1 >=
                                abs(actual_versus)) and (int(pos_actual) +
                                sizes[v] - 1 >= abs(versus[other_v])):
                                    c3.append(pos)
```

```
                c2.append(c3)
            c1.append(c2)
        conflicts.append(c1)

    begin_movements = variables.max_var()+1
    movements = []
    for m1 in movements_names:
        m_1 = []
        for m2 in m1:
            m_2 = []
            for m3 in m2:
                variables.add_var(m3)
                m_2.append((m3, variables.max_var()))

            m_1.append(m_2)
        movements.append(m_1)
    end_movements = variables.max_var()

    vehicles_mov = []
    for v1 in vehicles_names:
        v_1 = []
        for v2 in v1:
            variables.add_var(v2)
            v_1.append((v2, variables.max_var()))
        vehicles_mov.append(v_1)
```

Un cop hem creat totes les variables necessàries, el següent pas és definir les restriccions que han de ser satisfetes per trobar una solució al problema. Aquestes restriccions es tradueixen en clàusules que s'afegiran al CNF i que serà resolt pel solucionador SAT.

Primerament, declarem les restriccions per controlar els conflictes entre els vehicles. Posem un exemple per entendre aquesta restricció.

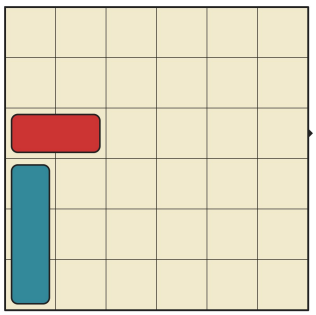


FIGURA 7.2: Exemple d'una instància per mostrar les posicions de conflicte entre dos

Com podem veure en la Figura 7.2, la posició 1 del vehicle 1 (vermell) té conflicte amb les posicions 2, 3 i 4 del vehicle 2 en un temps concret, la fórmula és la següent:

$$\neg((p_{101} \wedge p_{502}) \vee (p_{101} \wedge p_{503}) \vee (p_{101} \wedge p_{504}))$$

que passat a CNF seria:

$$(\neg p_{101} \vee \neg p_{502}) \wedge (\neg p_{101} \vee \neg p_{503}) \wedge (\neg p_{101} \vee \neg p_{504}))$$

impedint que dos posicions incompatibles dels vehicles 1 i 2 es donguessin a l'instant 0.

Seguidament, utilitzem el mètode `exactly_one` per definir la restricció de que, per cada temps, un vehicle ha d'estar exactament en una posició. Així doncs, si una posició d'un vehicle i en un temps determinats és certa, la resta de posicions d'aquest vehicle en aquest temps seran falses. Amb aquest mètode també imposem que, per cada temps, s'ha de moure exactament un vehicle.

Amb el mètode `at_most_one` establim que per cada temps (menys l'últim) si un vehicle es mou, només es pot moure d'una posició a una altra. És a dir, màxim una variable que representa un moviment d'un vehicle en un temps determinat pot estar a cert.

```
    for s in range(steps+1):
        vehicles_time = []
        for v in range(0,vehicles):
            pos = positions[v][s]
            con = conflicts[v][s]

            for i in range(len(pos)):
                confs = [conflict[1] for conflict in con[i]]
                for c in confs:
                    cnf.add_clause([-pos[i][1], -c])

            pos_vehicle = [position[1] for position in positions[v][s]]
            cl1 = Encoder.exactly_one(pos_vehicle, encoding='pairwise')
            cnf.add_clauses(cl1[1])

            if s < steps:
                mv_vehicle=[movement[1] for movement in movements[v][s]]
                cl2 = Encoder.at_most_one(mv_vehicle, encoding='pairwise')
                cnf.add_clauses(cl2[1])

                time_v = [time[1] for time in [vehicles_mov[v][s]]][0]
                vehicles_time.append(time_v)

        if s < steps:
            cl3 = Encoder.exactly_one(vehicles_time, encoding='pairwise')
            cnf.add_clauses(clause3[1])
```

En el següent tall de codi controlem les posicions de moviment.
En posem un exemple: Si existeix una variable que representa el moviment del vehicle 1 en el temps 0 de la posició 4 a la 5 (m_{1045}), si aquesta variable es troba a cert, implica que la posició 4 del cotxe 1 en el temps 0 també ha de ser certa i que la posició 5 del mateix cotxe en un temps més també.

$$m_{1045} \longrightarrow (p_{104} \wedge p_{115} \wedge v_{10})$$

Aquesta expressió convertida en fórmula CNF és de la forma:

$$(\neg m_{1045} \vee p_{104}) \wedge (\neg m_{1045} \vee p_{115}) \wedge (\neg m_{1045} \vee v_{10})$$

A més, si el moviment és de més de 1 desplaçament, s'ha d'implicar que les posicions que es troben entre mig de la inicial i la final d'aquest moviment estiguin a fals en el temps indicat.

```
    for s in range(steps):
        for v in range(vehicles):
            movs = movements[v][s]
            for i in range(len(movs)):
                mov_var = movs[i][1]

                pre_pos = int(str(movs[i][0])[-2])
                post_pos = int(str(movs[i][0])[-1])
                i = 0
                pre_var = positions[v][s][i]
                while int(str(pre_var[0])[-1]) != pre_pos:
                    i = i+1
                    pre_var = positions[v][s][i]
                i = 0
                post_var = positions[v][s+1][i]
                while int(str(post_var[0])[-1]) != post_pos:
                    i = i+1
                    post_var = positions[v][s+1][i]

                if pre_pos<post_pos and pre_pos+1<post_pos:
                    first=post_var[1]-(post_pos-pre_pos)
                    last=post_var[1]
                elif pre_pos>post_pos and pre_pos>post_pos+1:
                    first=post_var[1]
                    last=post_var[1]+(pre_pos-post_pos)
                else:
                    first, last = 0, 1
                for i in range(first+1,last):
                    index = [pos[1] for pos in positions[v][s+1]].index(i)
                    if conflicts[v][s+1][index] != []:
                        for c in conflicts[v][s+1][index]:
                            cnf.add_clause([-mov_var,-c[1]])

                cnf.add_clause([-mov_var,pre_var[1]])
                cnf.add_clause([-mov_var,post_var[1]])
                cnf.add_clause([-mov_var,vehicles_mov[v][s][1]])
```

A continuació, establim la relació entre les variables que representen si hi ha moviment per un vehicle i temps determinats.

En mostrem un exemple. Suposem que el vehicle 1 només es pot trobar en les posicions 3, 4 i 5; llavors la variable que indica que aquest vehicle en el temps 0 s'està movent (v_{10}) implica que un dels moviments possibles del vehicle en aquest temps ha d'estar a cert.

$$v_{10} \longrightarrow m_{1034} \vee m_{1035} \vee m_{1043} \vee m_{1045} \vee m_{1053} \vee m_{1054}$$

Aquesta expressió convertida en fórmula CNF és de la forma:

$$\neg v_{10} \vee m_{1034} \vee m_{1035} \vee m_{1043} \vee m_{1045} \vee m_{1053} \vee m_{1054}$$

També controlem els no-moviments dels vehicles. Quan un vehicle no està en moviment en un temps, la seva posició en el temps superior no varia.
Si el vehicle 1 en el temps 0 no es mou i es troba a la posició 4, en el temps 1 es manté a la posició 4.

$$(\neg v_{10} \wedge p_{104}) \longrightarrow p_{114}$$

Aquesta expressió convertida en fórmula CNF és de la forma:

$$v_{10} \vee \neg p_{104} \vee p_{114}$$

Finalment, declarem la restricció objectiu implicant a cert la posició de sortida del primer vehicle, que és el vermell.

```
            possible_moves = [move[1] for move in movements[v][s]]

            v_mov = vehicles_mov[v][s][1]
            clause = []
            clause.append(-v_mov)
            for pm in possible_moves: clause.append(pm)
            cnf.add_clause(clause)

            pos_nums = [pos[1] for pos in positions[v][s]]
            for p in positions[v][s]: cnf.add_clause([v_mov, -p[1],
    positions[v][s+1][pos_nums.index(p[1])][1]])

    cnf.add_clause([positions[0][-1][-1][1]])

    solver.add_clauses(cnf.clauses)
    solver.solve()
    cnf.write_dimacs_file('example')
    interpretation = variables.decode_dimacs(solver.model())
```

Després d'afegir totes les clàusules al solucionador, es realitza la decodificació de les variables per trobar la interpretació de la solució.

7.2.4 Model PDDL

Per a la generació d'aquest model, es va utilitzar un codi disponible al repositori GitHub de l'usuari ehajdin [12]. Aquest model està dissenyat per restringir els moviments dels vehicles a només un desplaçament per pas, de forma que ens ha servit per al model que busca l'optimització de desplaçaments.
També l'hem utilitzat com a base per generar el segon model, buscant l'optimització de passos. Se li han realitzat modificacions significatives per permetre la mobilitat completa dels vehicles, permetent-los realitzar de 1 a 4 passos de desplaçament.

Tal com hem explicat abans, en PDDL els models es componen per dos fitxers, un que defineix el problema i és específic per cada instància concreta del joc, i l'altre que estableix el domini del problema.

Seguidament, mostrem i expliquem detalladament el model que hem extès per permetre la mobilitat completa dels vehicles.

Problema

Primerament s'estableix el nom del problema, per identificar l'arxiu, i s'especifica el nom del domini que utilitzarà, que estarà definit en l'altre fitxer.

A continuació, s'especifiquen els objectes que seran utilitzats en aquest problema particular. Això inclou tant les diferents posicions possibles `position` del tauler com els vehicles. Cada posició es descriu com `locX_Y`, i cada cotxe es representa amb `carX`, menys el vermell que s'anomena `red`.

```
(define (problem rush-hour-pp)
    (:domain rush-hour-dd)
    (:objects
        loc1_1 loc1_2 loc1_3 loc1_4 loc1_5 loc1_6 loc2_1 loc2_2 loc2_3
    loc2_4 loc2_5 loc2_6 loc3_1 loc3_2 loc3_3 loc3_4 loc3_5 loc3_6 loc4_1
    loc4_2 loc4_3 loc4_4 loc4_5 loc4_6 loc5_1 loc5_2 loc5_3 loc5_4 loc5_5
    loc5_6 loc6_1 loc6_2 loc6_3 loc6_4 loc6_5 loc6_6  - position
        red car1 car2 car3 car4 car5 car6 car7 car8 car9 car10 car11 car12
     - car
    )
```

La següent secció estableix les condicions inicials del problema. Amb el predicat `adjacent` es descriuen les relacions d'adjacència entre les diferents posicions del tauler. Aquesta relació és ternària ja que relaciona tres posicions seguides ja siguin vertical o horitzontalment. Per definir les mides de cada vehicle s'utilitzen els predicats `small`, si és un cotxe, i `large` si és un camió. També s'identifiquen les posicions que estan ocupades mitjançant `isOccupied`, i es determinen les posicions que ocupa cada vehicle amb `containsCar`.

```
    (:init
        (ADJACENT loc1_1 loc2_1 loc3_1)
        (ADJACENT loc1_1 loc1_2 loc1_3)
        (ADJACENT loc3_1 loc2_1 loc1_1)
        (ADJACENT loc1_3 loc1_2 loc1_1)
        (ADJACENT loc1_2 loc2_2 loc3_2)
        (ADJACENT loc2_1 loc2_2 loc2_3)
        (ADJACENT loc3_2 loc2_2 loc1_2)
        (ADJACENT loc2_3 loc2_2 loc2_1)
        (ADJACENT loc1_3 loc2_3 loc3_3)
        (ADJACENT loc3_1 loc3_2 loc3_3)
        (ADJACENT loc3_3 loc2_3 loc1_3)
        (ADJACENT loc3_3 loc3_2 loc3_1)
        (ADJACENT loc1_4 loc2_4 loc3_4)
        (ADJACENT loc4_1 loc4_2 loc4_3)
        (ADJACENT loc3_4 loc2_4 loc1_4)
        (ADJACENT loc4_3 loc4_2 loc4_1)
        (ADJACENT loc1_5 loc2_5 loc3_5)
        (ADJACENT loc5_1 loc5_2 loc5_3)
        (ADJACENT loc3_5 loc2_5 loc1_5)
        (ADJACENT loc5_3 loc5_2 loc5_1)
        (ADJACENT loc1_6 loc2_6 loc3_6)
        (ADJACENT loc6_1 loc6_2 loc6_3)
        (ADJACENT loc3_6 loc2_6 loc1_6)
        (ADJACENT loc6_3 loc6_2 loc6_1)
        (ADJACENT loc2_1 loc3_1 loc4_1)
        (ADJACENT loc1_2 loc1_3 loc1_4)
        (ADJACENT loc4_1 loc3_1 loc2_1)
        (ADJACENT loc1_4 loc1_3 loc1_2)
        (ADJACENT loc2_2 loc3_2 loc4_2)
        (ADJACENT loc2_2 loc2_3 loc2_4)
```

```
        (ADJACENT loc4_2 loc3_2 loc2_2)
        (ADJACENT loc2_4 loc2_3 loc2_2)
        (ADJACENT loc2_3 loc3_3 loc4_3)
        (ADJACENT loc3_2 loc3_3 loc3_4)
        (ADJACENT loc4_3 loc3_3 loc2_3)
        (ADJACENT loc3_4 loc3_3 loc3_2)
        (ADJACENT loc2_4 loc3_4 loc4_4)
        (ADJACENT loc4_2 loc4_3 loc4_4)
        (ADJACENT loc4_4 loc3_4 loc2_4)
        (ADJACENT loc4_4 loc4_3 loc4_2)
        (ADJACENT loc2_5 loc3_5 loc4_5)
        (ADJACENT loc5_2 loc5_3 loc5_4)
        (ADJACENT loc4_5 loc3_5 loc2_5)
        (ADJACENT loc5_4 loc5_3 loc5_2)
        (ADJACENT loc2_6 loc3_6 loc4_6)
        (ADJACENT loc6_2 loc6_3 loc6_4)
        (ADJACENT loc4_6 loc3_6 loc2_6)
        (ADJACENT loc6_4 loc6_3 loc6_2)
        (ADJACENT loc3_1 loc4_1 loc5_1)
        (ADJACENT loc1_3 loc1_4 loc1_5)
        (ADJACENT loc5_1 loc4_1 loc3_1)
        (ADJACENT loc1_5 loc1_4 loc1_3)
        (ADJACENT loc3_2 loc4_2 loc5_2)
        (ADJACENT loc2_3 loc2_4 loc2_5)
        (ADJACENT loc5_2 loc4_2 loc3_2)
        (ADJACENT loc2_5 loc2_4 loc2_3)
        (ADJACENT loc3_3 loc4_3 loc5_3)
        (ADJACENT loc3_3 loc3_4 loc3_5)
        (ADJACENT loc5_3 loc4_3 loc3_3)
        (ADJACENT loc3_5 loc3_4 loc3_3)
        (ADJACENT loc3_4 loc4_4 loc5_4)
        (ADJACENT loc4_3 loc4_4 loc4_5)
        (ADJACENT loc5_4 loc4_4 loc3_4)
        (ADJACENT loc4_5 loc4_4 loc4_3)
        (ADJACENT loc3_5 loc4_5 loc5_5)
        (ADJACENT loc5_3 loc5_4 loc5_5)
        (ADJACENT loc5_5 loc4_5 loc3_5)
        (ADJACENT loc5_5 loc5_4 loc5_3)
        (ADJACENT loc3_6 loc4_6 loc5_6)
        (ADJACENT loc6_3 loc6_4 loc6_5)
        (ADJACENT loc5_6 loc4_6 loc3_6)
        (ADJACENT loc6_5 loc6_4 loc6_3)
        (ADJACENT loc4_1 loc5_1 loc6_1)
        (ADJACENT loc1_4 loc1_5 loc1_6)
        (ADJACENT loc6_1 loc5_1 loc4_1)
        (ADJACENT loc1_6 loc1_5 loc1_4)
        (ADJACENT loc4_2 loc5_2 loc6_2)
        (ADJACENT loc2_4 loc2_5 loc2_6)
        (ADJACENT loc6_2 loc5_2 loc4_2)
        (ADJACENT loc2_6 loc2_5 loc2_4)
        (ADJACENT loc4_3 loc5_3 loc6_3)
        (ADJACENT loc3_4 loc3_5 loc3_6)
        (ADJACENT loc6_3 loc5_3 loc4_3)
        (ADJACENT loc3_6 loc3_5 loc3_4)
        (ADJACENT loc4_4 loc5_4 loc6_4)
        (ADJACENT loc4_4 loc4_5 loc4_6)
        (ADJACENT loc6_4 loc5_4 loc4_4)
        (ADJACENT loc4_6 loc4_5 loc4_4)
        (ADJACENT loc4_5 loc5_5 loc6_5)
        (ADJACENT loc5_4 loc5_5 loc5_6)
        (ADJACENT loc6_5 loc5_5 loc4_5)
        (ADJACENT loc5_6 loc5_5 loc5_4)
        (ADJACENT loc4_6 loc5_6 loc6_6)
```

```
        (ADJACENT loc6_4 loc6_5 loc6_6)
        (ADJACENT loc6_6 loc5_6 loc4_6)
        (ADJACENT loc6_6 loc6_5 loc6_4)

        (SMALL red)
        (SMALL car1)
        (SMALL car2)
        (SMALL car3)
        (SMALL car4)
        (LARGE car5)
        (SMALL car6)
        (SMALL car7)
        (SMALL car8)
        (SMALL car9)
        (SMALL car10)
        (SMALL car11)
        (LARGE car12)

        (containsCar loc4_1 red)
        (containsCar loc4_2 red)
        (containsCar loc6_3 car1)
        (containsCar loc6_4 car1)
        (containsCar loc6_5 car2)
        (containsCar loc6_6 car2)
        (containsCar loc3_2 car3)
        (containsCar loc3_3 car3)
        (containsCar loc3_4 car4)
        (containsCar loc3_5 car4)
        (containsCar loc1_1 car5)
        (containsCar loc1_2 car5)
        (containsCar loc1_3 car5)
        (containsCar loc1_4 car6)
        (containsCar loc1_5 car6)
        (containsCar loc5_1 car7)
        (containsCar loc6_1 car7)
        (containsCar loc2_1 car8)
        (containsCar loc3_1 car8)
        (containsCar loc4_3 car9)
        (containsCar loc5_3 car9)
        (containsCar loc4_5 car10)
        (containsCar loc5_5 car10)
        (containsCar loc4_6 car11)
        (containsCar loc5_6 car11)
        (containsCar loc1_6 car12)
        (containsCar loc2_6 car12)
        (containsCar loc3_6 car12)

        (isOccupied loc4_1)
        (isOccupied loc4_2)
        (isOccupied loc6_3)
        (isOccupied loc6_4)
        (isOccupied loc6_5)
        (isOccupied loc6_6)
        (isOccupied loc3_2)
        (isOccupied loc3_3)
        (isOccupied loc3_4)
        (isOccupied loc3_5)
        (isOccupied loc1_1)
        (isOccupied loc1_2)
        (isOccupied loc1_3)
        (isOccupied loc1_4)
        (isOccupied loc1_5)
        (isOccupied loc5_1)
```

```
        (isOccupied loc6_1)
        (isOccupied loc2_1)
        (isOccupied loc3_1)
        (isOccupied loc4_3)
        (isOccupied loc5_3)
        (isOccupied loc4_5)
        (isOccupied loc5_5)
        (isOccupied loc4_6)
        (isOccupied loc5_6)
        (isOccupied loc1_6)
        (isOccupied loc2_6)
        (isOccupied loc3_6)
    )
```

Finalment, s'estableix l'objectiu que s'ha de complir per considerar el problema resolt. Aquest és que el cotxe vermell ocupi les posicions de sortida del tauler, que són `loc4_5` i `loc4_6`.

```
    (:goal (and
        (containsCar loc4_5 red)
        (containsCar loc4_6 red)
    ))
)
```

Domini

En el fitxer del domini es descriuen les diferents accions i predicats que s'utilitzaran per resoldre les instàncies del problema.

Primerament, es defineix el domini amb un nom identificador i es defineixen els requisits necessaris pel planificador.

S'estableixen els tipus utilitzats, `position` i `car`; i també es defineixen els predicats utilitzats en les condicions i efectes de les accions, que són `adjacent`, `small`, `large`, `isOccupied` i `containsCar`.

```
(define (domain rush-hour-dd)
    (:requirements :typing :strips :action-costs)
    (:types position car - object)

    (:predicates
        (ADJACENT ?p1 - position ?p2 - position ?p3 - position)
        (SMALL ?c - car)
        (LARGE ?c - car)
        (isOccupied ?p - position)
        (containsCar ?p - position ?c - car)
    )
```

Les accions representen les diferents operacions que es poden realitzar en aquest domini. Per als cotxes s'han definit les accions `move-small1`, `move-small2`, `move-small3` i `move-small4`, per moure el cotxe 1, 2, 3 o 4 desplaçaments, respectivament. Val a dir que el model inicial només permetia moviments d'una casella, l'extensió natural, que seria passar com a paràmetre el nombre de caselles del desplaçament, no es pot implementar de forma natural amb PDLL degut al seu escas suport d'enters i d'estructures de dades sofisticades com podrien ser matrius. És per això que s'ha hagut de "replicar" i adaptar el codi. D'aquesta manera podem considerar adequadament la noció d'optimització de passos.

Aquestes accions descriuen les condicions (precondicions) que han de ser certes abans que l'acció pugui ser executada i els efectes (postcondicions) que descriuen els canvis que es produeixen després d'executar l'acció.

Similarment, per als camions s'han definit les accions `move-large1`, `move-large2` i `move-large3` amb les seves respectives precondicions i efectes.

```
    (:action move-small1
        :parameters (?c-car ?p1-position ?p2-position ?p3-position)
        :precondition (and
            (SMALL ?c)
            (ADJACENT ?p1 ?p2 ?p3)
            (containsCar ?p1 ?c)
            (containsCar ?p2 ?c)
            (not (isOccupied ?p3))
        )
        :effect (and
            (not (isOccupied ?p1))
            (not (containsCar ?p1 ?c))
            (containsCar ?p3 ?c)
            (isOccupied ?p3)
        )
    )

    (:action move-small2
        :parameters (?c-car ?p1-position ?p2-position ?p3-position
            ?p4-position)
        :precondition (and
            (SMALL ?c)
            (ADJACENT ?p1 ?p2 ?p3) (ADJACENT ?p2 ?p3 ?p4)
            (containsCar ?p1 ?c)
            (containsCar ?p2 ?c)
            (not (isOccupied ?p3))
            (not (isOccupied ?p4))
        )
        :effect (and
            (not (isOccupied ?p1))
            (not (isOccupied ?p2))
            (not (containsCar ?p1 ?c))
            (not (containsCar ?p2 ?c))
            (containsCar ?p3 ?c)
            (containsCar ?p4 ?c)
            (isOccupied ?p3)
            (isOccupied ?p4)
        )
    )

    (:action move-small3
        :parameters (?c-car ?p1-position ?p2-position ?p3-position
            ?p4-position ?p5-position)
        :precondition (and
            (SMALL ?c)
            (ADJACENT ?p1 ?p2 ?p3) (ADJACENT ?p2 ?p3 ?p4)
            (ADJACENT ?p3 ?p4 ?p5)
            (containsCar ?p1 ?c)
            (containsCar ?p2 ?c)
            (not (isOccupied ?p3))
            (not (isOccupied ?p4))
            (not (isOccupied ?p5))
        )
        :effect (and
            (not (isOccupied ?p1))
```

```
            (not (isOccupied ?p2))
            (not (isOccupied ?p3))
            (not (containsCar ?p1 ?c))
            (not (containsCar ?p2 ?c))
            (not (containsCar ?p3 ?c))
            (containsCar ?p4 ?c)
            (containsCar ?p5 ?c)
            (isOccupied ?p4)
            (isOccupied ?p5)
        )
    )

    (:action move-small4
        :parameters (?c-car ?p1-position ?p2-position ?p3-position
            ?p4-position ?p5-position ?p6-position)
        :precondition (and
            (SMALL ?c)
            (ADJACENT ?p1 ?p2 ?p3) (ADJACENT ?p2 ?p3 ?p4)
            (ADJACENT ?p3 ?p4 ?p5) (ADJACENT ?p4 ?p5 ?p6)
            (containsCar ?p1 ?c)
            (containsCar ?p2 ?c)
            (not (isOccupied ?p3))
            (not (isOccupied ?p4))
            (not (isOccupied ?p5))
            (not (isOccupied ?p6))
        )
        :effect (and
            (not (isOccupied ?p1))
            (not (isOccupied ?p2))
            (not (isOccupied ?p3))
            (not (isOccupied ?p4))
            (not (containsCar ?p1 ?c))
            (not (containsCar ?p2 ?c))
            (not (containsCar ?p3 ?c))
            (not (containsCar ?p4 ?c))
            (containsCar ?p5 ?c)
            (containsCar ?p6 ?c)
            (isOccupied ?p5)
            (isOccupied ?p6)
        )
    )

    (:action move-large1
        :parameters (?c-car ?p1-position ?p2-position ?p3-position
            ?p4-position)
        :precondition (and
            (LARGE ?c)
            (ADJACENT ?p1 ?p2 ?p3) (ADJACENT ?p2 ?p3 ?p4)
            (containsCar ?p1 ?c)
            (containsCar ?p2 ?c)
            (containsCar ?p3 ?c)
            (not (isOccupied ?p4))
        )
        :effect (and
            (not (isOccupied ?p1))
            (not (containsCar ?p1 ?c))
            (containsCar ?p4 ?c)
            (isOccupied ?p4)
        )
    )

    (:action move-large2
        :parameters (?c-car ?p1-position ?p2-position ?p3-position
```

```
            ?p4-position ?p5-position)
        :precondition (and
            (LARGE ?c)
            (ADJACENT ?p1 ?p2 ?p3) (ADJACENT ?p2 ?p3 ?p4)
            (ADJACENT ?p3 ?p4 ?p5)
            (containsCar ?p1 ?c)
            (containsCar ?p2 ?c)
            (containsCar ?p3 ?c)
            (not (isOccupied ?p4))
            (not (isOccupied ?p5))
        )
        :effect (and
            (not (isOccupied ?p1))
            (not (isOccupied ?p2))
            (not (containsCar ?p1 ?c))
            (not (containsCar ?p2 ?c))
            (containsCar ?p4 ?c)
            (containsCar ?p5 ?c)
            (isOccupied ?p4)
            (isOccupied ?p5)
        )
    )

    (:action move-large3
        :parameters (?c-car ?p1-position ?p2-position ?p3-position
            ?p4-position ?p5-position ?p6-position)
        :precondition (and
            (LARGE ?c)
            (ADJACENT ?p1 ?p2 ?p3) (ADJACENT ?p2 ?p3 ?p4)
            (ADJACENT ?p3 ?p4 ?p5) (ADJACENT ?p4 ?p5 ?p6)
            (containsCar ?p1 ?c)
            (containsCar ?p2 ?c)
            (containsCar ?p3 ?c)
            (not (isOccupied ?p4))
            (not (isOccupied ?p5))
            (not (isOccupied ?p6))
        )
        :effect (and
            (not (isOccupied ?p1))
            (not (isOccupied ?p2))
            (not (isOccupied ?p3))
            (not (containsCar ?p1 ?c))
            (not (containsCar ?p2 ?c))
            (not (containsCar ?p3 ?c))
            (containsCar ?p4 ?c)
            (containsCar ?p5 ?c)
            (containsCar ?p6 ?c)
            (isOccupied ?p4)
            (isOccupied ?p5)
            (isOccupied ?p6)
        )
    )
)
```

Per al desenvolupament dels altres dos models hem realitzat algunes modificacions que detallem a continuació.

El segon model, que restringeix el desplaçament a un per pas, utilitza el mateix fitxer problema. En canvi, el fitxer domini és igual al que acabem de mostrar però reduït, ja que només té dues accions possibles, `move-small1` i `move-large1`.

Per últim, en el tercer model que hem desenvolupat, que se centra en l'optimització simultània dels passos i els desplaçaments, hem realitzat algunes addicions als fitxers de problema i domini.

En primer lloc, en el domini, hem introduït l'etiqueta `:action-costs` per indicar que les accions porten associat un cost numèric. Per aconseguir-ho, hem afegit l'expressió `(increase (total-cost) x)` als efectes de cada acció. El valor de `x` està vinculat a l'acció que es duu a terme.

Inicialment, vam intentar assignar un valor de x basat directament en el nombre de desplaçaments d'una acció. Per exemple, assignar un valor de 1 si l'acció implica un sol desplaçament, 2 si s'implica dos, i així successivament. No obstant això, el planificador interpretava una acció de 2 desplaçaments com més costosa que dues accions amb un sol desplaçament. Això feia que es realitzessin passos innecessaris i per tant no minimitzés aquest nombre, i no es complia l'objectiu que buscàvem.

Finalment, vam trobar una solució: en lloc d'incrementar només el valor del desplaçament, l'incrementem en 5 més. El motiu del valor 5 és que el màxim cost que pot tenir una acció és 4. Amb això, el planificador interpreta com a més costós realitzar una acció de més que no moure's més desplaçaments en una acció, i aconseguim que agrupi els desplaçaments en menys passos.

Una vegada obtenim una solució, podem calcular fàcilment el nombre total de desplaçaments totals restant cinc vegades la longitud total del pla al valor del cost total.

I en el problema hem afegit la mètrica `(:metric minimize (total-cost))` per minimitzar el nombre total del cost del pla.

7.3 Traductors d'instàncies

En aquest apartat, s'explica com s'han desenvolupat els traductors d'instàncies per passar d'una representació del tauler a una altra. Aquesta capacitat és fonamental per a la nostra recerca, ja que ens permet treballar amb diverses representacions sense pèrdua d'informació. A continuació, detallem els dos traductors que hem creat:

7.3.1 Traductor de representació lexicogràfica a genèrica

El primer traductor que hem desenvolupat serverix per convertir la representació lexicogràfica d'una configuració en la que nosaltres anomenem genèrica.
Aquest traductor és crucial perquè en la base de dades de la qual disposem, on hi ha totes les configuracions interessants, estan representades lexicogràficament. En canvi, els models que hem desenvolupat accepten entrades amb representació genèrica.

Aquest codi rep per paràmetre la cadena lexicogràfica i comença a buscar si existeix la lletra 'A', la tracta, i així progressivament fins que no troba més lletres.

En aquest tractament, primer es comprova quantes vegades apareix la lletra, per esbrinar la mida del vehicle. Llavors es mira, segons la col·locació d'aquestes en la cadena, si el vehicle és vertical o horitzontal. Com que sabem el tamany total de la cadena, i per tant, del tauler, aconseguim saber la fila i columna on es troba cada vehicle.

```
import scala.math.sqrt
import scala.sys.exit

object Translate {
  def main(args: Array[String]): Unit = {
    if (args.length > 0) {
      val board: String = args(0)
      val sizeBoard: Int = sqrt(board.length).toInt
      var letter: Char = 'A'
      var sizes: List[Int] = List()
      var versus: List[Int] = List()
      var initial: List[Int] = List()

      while (board.exists(l => l == letter)) {
        var posCar: List[Int] = List()
        sizes = sizes :+ board.count(_ == letter)
        var index = 0

        for (x <- 1 to board.count(_ == letter)) {
          index = board.indexOf(letter, index) + 1
          posCar = posCar :+ index
        }

        var column: Int = posCar.head
        var row: Int = sizeBoard
        val horizontal: Boolean = column + 1 == posCar.last |
            column + 2 == posCar.last | column + 3 == posCar.last

        if (!horizontal) column = posCar.last
        while (column > sizeBoard) {
          row = row - 1
          column = column - sizeBoard
        }

        if (horizontal) {
          initial = initial :+ column
          versus = versus :+ row
        }
        else {
          initial = initial :+ row
          versus = versus :+ -column
        }
        letter = (letter + 1).toChar
      }

      print(sizeBoard.toString + " " + sizes.length + " [" + sizes.head)
      for (s <- sizes.tail) print("," + s)
      print("] [" + versus.head)
      for (v <- versus.tail) print("," + v)
      print("] [" + initial.head)
      for (v <- initial.tail) print("," + v)
      println("]")
    } else {
      println("No s'han proporcionat arguments.")
    }
  }
}
```

Un exemple de sortida:

7.3.2 Traductor de representació genèrica a arxiu PDDL

El segon traductor converteix la representació genèrica d'una configuració a un fitxer problema del llenguatge PDDL.

Aquest traductor ha estat molt útil ja que la representació en PDDL és específica i costosa de representar manualment.

```
import java.io._
object ToPDDL {
  def convertStringToList(input: String): List[Int] = {
    input.stripPrefix("[").stripSuffix("]").split(",").map(_.toInt).toList
  }
  def main(args: Array[String]): Unit = {
    if (args.length > 0) {
      val boardSize: Int = args(0).toInt
      val vehicles: Int = args(1).toInt
      val sizes: List[Int] = convertStringToList(args(2))
      val versus: List[Int] = convertStringToList(args(3))
      val initial: List[Int] = convertStringToList(args(4))

      val file = new FileWriter()

      file.write("(define (problem rush-hour-pp)\n    (:domain rush-hour-
    dd)\n    (:objects\n        ")

      for (i <- 1 to boardSize) {
        for (j <- 1 to boardSize) {
          file.write("loc" + i + "_" + j + " ")
        }
      }
      file.write(" - position\n        ")

      for (v <- 0 until vehicles) {
        if (v == 0) file.write("red")
        else file.write(" car" + v)
      }
      file.write(" - car\n    )\n    (:init\n        ")

      for (a <- 1 to boardSize - 2) {
        val b = a + 1
        val c = a + 2
        for (i <- 1 to boardSize) {
          file.write("(ADJACENT loc" + a + "_" + i + " loc" + b + "_" + i
    + " loc" + c + "_" + i + ")\n        ")
          file.write("(ADJACENT loc" + i + "_" + a + " loc" + i + "_" + b
    + " loc" + i + "_" + c + ")\n        ")
          file.write("(ADJACENT loc" + c + "_" + i + " loc" + b + "_" + i
    + " loc" + a + "_" + i + ")\n        ")
          file.write("(ADJACENT loc" + i + "_" + c + " loc" + i + "_" + b
    + " loc" + i + "_" + a + ")\n        ")
        }
      }

      for (v <- 0 until vehicles) {
        if (v == 0) file.write("(SMALL red)\n")
        else {
          if (sizes(v) == 2) file.write("(SMALL car" + v + ")\n")
          else file.write("(LARGE car" + v + ")\n")
        }
        file.write("        ")
      }
```

```
      var occupied: List[String] = List()
      for (v <- 0 until vehicles) {
        for (s <- 0 until sizes(v)) {
          var col, row: Int = 0
          if (versus(v) > 0) {
            col = initial(v) + s
            row = versus(v).abs
          }
          else {
            row = initial(v) + s
            col = versus(v).abs
          }
          occupied = occupied :+ (row.toString + "_" + col.toString)
          file.write("(containsCar loc" + row + "_" + col)
          if (v == 0) file.write(" red)\n        ")
          else file.write(" car" + v + ")\n        ")
        }
      }

      for (o <- occupied) {
        file.write("\n        (isOccupied loc" + o + ")")
      }

      file.write("\n        (:goal (and\n        (containsCar loc4_5 red)\
    n")
        + ("(containsCar loc4_6 red)\n    ))\n    (:metric minimize (total-
    cost))\n)")
      file.close()
    } else {
      println("No s'han proporcionat arguments")
    }
  }
}
```

Capítol 8

Experimentació

En aquest capítol, posem a prova tots els programes i models que hem recopilat i desenvolupat. Ens centrem en tres grans objectius: intentar generar configuracions amb una mida de tauler major, recollir les configuracions més complexes per posar a prova els models i realitzar comparacions entre aquests, destacant les seves diferències i similituds en eficiència.

A través d'aquesta anàlisi comparativa, busquem obtenir una comprensió més profunda de quins models presenten millors resultats en termes de resolució del problema, i quines particularitats contribueixen a aquestes diferències.

En els següents experiments s'ha fet servir un timeout de 600 segons per resolució d'instància i s'hi assignat un node de clúster exclusiu. Això vol dir 16Gb de memòria i una CPU Intel(R) Xeon(R) E-2234 a 3.6GHz.

Les aproximacions de resolució han sigut :

- PDDL amb el planner Fast Downward amb l'heurística seq-opt-lmcut,
- MiniZinc amb el LazyFD solver Chuffed 0.12.1,
- Essence Prime amb els SAT solver Cadical, i
- OptiLog amb el SAT solver Glucose41.

Primer de tot parlarem de la generació d'instàncies, després de les propietats d'aquestes i finalment analitzarem com es comporten les diferents aproximacions (models PDDL, Minizinc, Essence Prime i OptiLog) amb les instàncies generades segons les diferents característiques considerades: la dificultat que ha tingut PDDL en resoldre-les, la quantitat de passos mínims requerits per trobar una solució, la relació cost total i la quantitat d'estats accessibles.

8.1 Generació d'instàncies

Observant el punt en que s'ha quedat l'experimentació de Fogleman i tenint a la nostra disposició el clúster, un dels nostres objectius principals se centra en arribar a obtenir totes les configuracions interessants del 7x7.

El primer pas és transferir els fitxers al clúster i compilar el codi per obtenir l'executable. Per executar el programa en el clúster, dissenyem un arxiu slurm que estableix

els paràmetres necessaris, el mostrem en l'Annex C.

Durant tot el desenvolupament d'aquest projecte, hem fet ús del clúster per intentar assolir aquest objectiu, però ens hem anat afrontant amb problemes relacionats amb la gestió de la memòria. El fet d'augmentar el tamany del tauler fa incrementar significativament els requisits de memòria.

Aquest impediment ens ha portat a analitzar a fons el codi i experimentar amb diferents estratègies a través del fitxer slurm. Cada intent ha requerit temps i anàlisi, i ha fet que aquesta tasca hagi ocupat una part molt gran del nostre esforç al llarg del projecte.

Després de molts intents, la quantitat màxima de configuracions interessants que hem aconseguit generar en un tauler 7×7, abans que el generador tornés a interrompre's per falta de memòria, ha estat de 28.826. Malauradament aquestes instàncies no aporten increment de dificultat i per tant no han sigut considerades.

8.2 Anàlisi d'instàncies

En aquest apartat, es descriu com s'ha utilitzat una combinació de models i programes desenvolupats durant el projecte per analitzar la complexitat de les diferents configuracions del joc Rush Hour.

Per a dur a terme aquest anàlisi, hem disposat de la base de dades de Fogleman [10] de mida de tauler 6×6, la qual consta de 476.118 configuracions.

Ja que aquest nombre és força elevat i hagués exigit un temps d'execució molt gran, hem decidit descartar vàries configuracions.

Primer de tot, hem eliminat les quals el mínim nombre de passos per arribar a obtenir una solució és menor a 20, ja que es consideren fàcils de resoldre i no tenen un pes significatiu. A més, és el rang on es concentren la majoria de les instàncies, com va demostrar Fogleman [10]. Fet que es pot apreciar a la Figura 8.1, on es mostra el nombre d'instàncies pel mínim nombre de passos necessaris per a resoldre-les.

Amb aquesta eliminació ens queden un total de 24.483 configuracions que utilitzem per realitzar l'estudi. Totes aquestes configuracions les hem registrat en un fitxer de text (observeu Figura 8.2).

El primer número per l'esquerra representa el nombre de passos mínims per resoldre-la, la cadena lexicogràfica representa el tauler i l'últim número representa el nombre d'estats accessibles.

Aquest fitxer servirà com a entrada per al programa que hem creat, per realitzar l'execució seguida de diversos programes, entre ells els dos traductors i el model d'optimització de passos amb Pddl, i l'extracció de les característiques de cada configuració a Excel.

Hem escollit Pddl com a model d'aquesta primera experimentació base perquè és l'aproximació que, en principi, sembla més eficient i no ens podíem permetre una aproximació molt costosa en termes de recursos computacionals i temps.

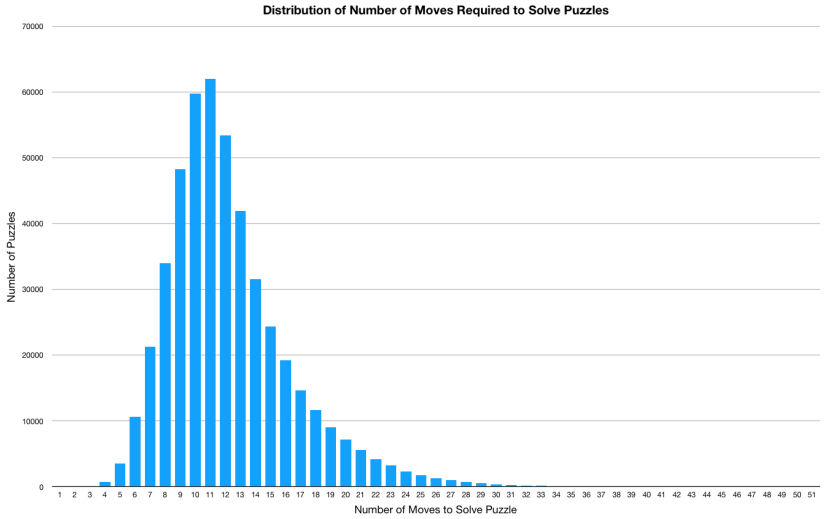


FIGURA 8.1: Distribució d'instàncies pel nombre de passos de resolució.

```
51 GBB.L.GHI.LMGHIAAMCCCK.M..JKDDEEJFF. 4780
50 ..IBBB..IK..AAJK.LCCJDDLGHEE.LGHFF.. 4643
49 BBBKLMHCCKLMH.AALMDDJ....IJEE..IFFGG 24132
49 ...JBB.CCJL.AAIKL.DDIKEEHFFF.MH.GG.M 4192
49 ...JBB.CCJL.AAIKL.DDIKEEHFFF.MHGGG.M 2616
49 ..HBBB..HJ..AAIJ.KCCIDDK.GEE.K.GFF.. 2295
48 BBBKLMHCCKLMH.AAL.DDJ....IJEE..IFFGG 37740
48 G..BBBGCCJ.KAAHJ.K..HDDK..IEE...IFFF 5082
```

FIGURA 8.2: Fitxer d'entrada 6×6 per a l'anàlisi

A l'Annex D, es mostra el codi d'aquest programa desenvolupat en Python.

El programa tracta cada línia del fitxer d'entrada, que representa una configuració específica, i la tracta individualment. Aquesta línia amb forma lexicogràfica, que representa el tauler, la passa pels dos traductors que hem generat anteriorment, per convertir-la en la seva forma de representació genèrica i llavors en un fitxer problema de PDDL. Seguidament, executa el model PDDL amb els corresponents fitxers de domini i problema, i registra la informació de sortida i les característiques següents de cada configuració:

- **Total time**: Temps total que ha trigat el planificador en resoldre el problema. Inclou tot el procés, des de la càrrega del problema fins a la generació i validació de la solució final.
- **Board size**: La mida del tauler
- **Board**: La representació lexicogràfica del tauler
- **Reachable states**: Nombre d'estats accessibles
- **Steps**: Passos mínims per resoldre-la
- **Vehicles**: Nombre de vehicles
- **Trucks**: Nombre de camions

- **Plan cost**: Desplaçaments totals mínims per resoldre-la
- **Ratio plan cost / steps**: Relació entre desplaçaments i passos

Així, obtenim els valors de totes les configuracions, els guardem en una taula de dades i els recopilem en un Excel (vegeu Figura 8.3) amb les respectives columnes.

A partir d'aquest Excel, ens disposem a analitzar les característiques i factors de les configuracions que influeixen en la complexitat a l'hora de resoldre-les.

Board size	Board	Reachable States	Steps	Vehicles	Trucks	Total time	Plan cost	Cars	Ratio plan cost / step
6	G.BBBLGCCJ.LAAIJ...HI.DD.HEEK.FF..K.	293622	20	12	1	0,67	38	11	1,900
6	BBI....HICCL.HAAKLDD.JK.G..JEEGFFF..	293622	20	12	1	0,06	32	11	1,600
6	HBBCCMH.JDDMAAJK...I.KEE.IFFL.GG..L.	278666	26	13	0	0,48	46	13	1,769
6	G..JBBGCCJ.L.H.AAL.HDDKMEEI.KM..IFF.	278666	24	13	0	0,33	42	13	1,750
6	BBI..L.HICCL.HAAKLDD.JK.G..JEEGFFF..	154363	20	12	2	0,07	34	10	1,700
6	GBBB..G..JCCAA.JKL.HDDKL.HIEELFFI...	154363	20	12	2	0,05	34	10	1,700
6	GH.BBBGHCCK.GAAJK....JDD..IEELFFI..L	154363	20	12	2	0,02	32	10	1,600
6	..JBBBG.JCCMGAAK.MHI.KDDHIEEL.FF..L.	142135	26	13	1	0,54	47	12	1,808
6	BBCC..H..KDDHAAKL.HIEELM.IJFFMGGJ...	142135	24	13	1	0,34	40	12	1,667
6	.IBBCC.IDD.MAAJ..MH.JEEMHFFKL.GG.KL.	142135	22	13	1	0,30	40	12	1,818
6	..JBBMCCJ.LM.IAALMHI.KDDHEEK...FFGG.	142135	23	13	1	0,18	36	12	1,565
6	HBBK..HI.KCCHIAAL.DDJ.LM..JEEMFFGG..	142135	20	13	1	0,14	34	12	1,700
6	BB.K..GI.KCCGIAALMHDD.LMH.JEEMFFJ...	141944	28	13	1	0,64	51	12	1,821
6	..IBBLCCI..LGHAAKMGHDDKMG..JEEFF.J..	141944	30	13	1	0,51	52	12	1,733
6	HBBB.MHCCK.MAAJK...IJDD..IEEL.FFGGL.	141944	20	13	1	0,30	40	12	2,000
6	HBBCCMH.JDDMAAJK...I.KEE.IFFL.GGG.L.	141944	27	13	1	0,28	46	12	1,704

FIGURA 8.3: Excel amb les característiques de les instàncies.

8.2.1 Factors que afecten a la complexitat

A partir de les dades recopilades anteriorment, volem generar representacions gràfiques amb l'objectiu d'analitzar amb detall els factors que poden influir en la complexitat de les configuracions. Aquesta anàlisi ens permetrà obtenir una millor comprensió dels factors i, en conseqüència, prendre decisions més fonamentades en el nostre estudi.

Nombre d'estats accessibles

Com hem mencionat anteriorment, un factor ja conegut que influeix directament en la complexitat és el nombre d'estats accessibles que una instància té.

En la Figura 8.4, mostrem el temps mitjà de resolució en PDDL pel nombre d'estats accessibles de les 24.483 configuracions.

Com podem apreciar, quants més estats accessibles té una configuració, més complexe és trobar una solució.

Nombre mínim de passos i desplaçaments

El següent factor que ens interessa analitzar és el mínim nombre de passos necessaris per resoldre una configuració determinada. Volem comprovar si aquest número influeix directament en la complexitat.
En la Figura 8.5, podem observar la variació en el temps mitjà que el solucionador PDDL necessita per trobar una solució mentre optimitzem el nombre de passos.

A continuació, realitzem un anàlisi similar pel que fa al mínim nombre de desplaçaments necessaris. En la Figura 8.6, es mostra com canvia el temps mitjà requerit pel

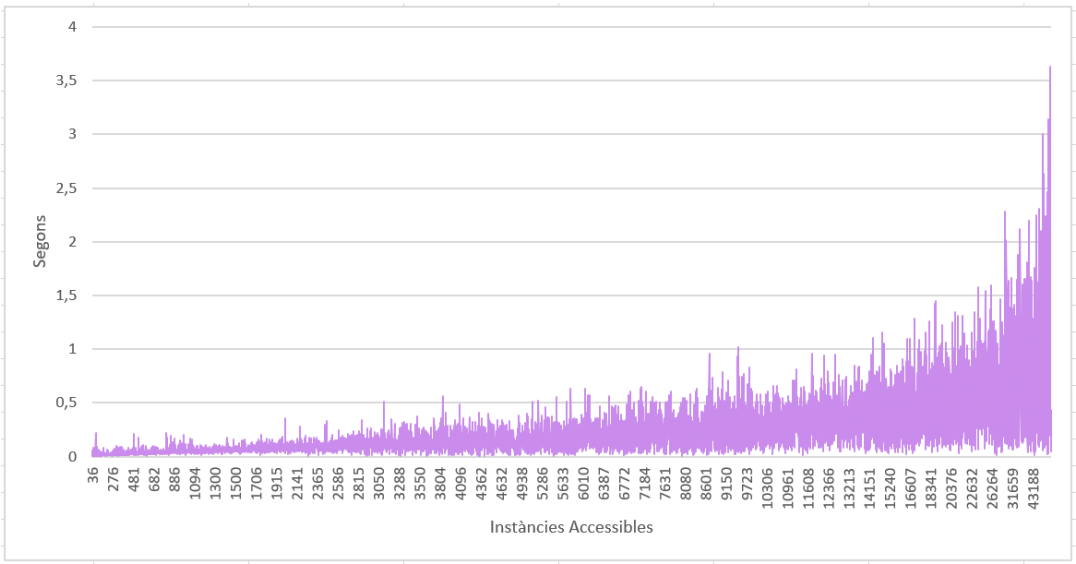


FIGURA 8.4: Temps mitjà de resolució de les instàncies amb PDDL segons la quantitat d'estats accessibles.

solucionador Pddl per trobar una solució mentre optimitzem el nombre de desplaçaments.

Les dues gràfiques mostren una tendència semblant i podem extreure observacions comunes.

En general, a mesura que augmenta el nombre mínim de passos o desplaçaments, el temps mitjà de resolució també augmenta. Això significa que les configuracions amb més passos o desplaçaments mínims tendeixen a ser més complexes i requereixen més temps per ser resoltes.

No obstant això, en les dues gràfiques, observem variacions en els valors més alts. Algunes configuracions amb nombres de passos o desplaçaments específics tenen temps mitjans que s'allunyen de la tendència general, ja sigui augmentant o disminuint. Aquestes diferències podrien indicar que altres factors estan influïnt en la complexitat. Com ja hem vist, un d'aquests factors és el nombre d'estats accessibles de les configuracions.

Per aprofundir en aquesta qüestió i comprendre millor les oscil·lacions que hem detectat en els valors més alts de passos i desplaçaments, hem realitzat un estudi específic sobre com el nombre d'estats accessibles influeixen en aquests.
Hem seleccionat el subconjunt de configuracions que tenen valors màxims significativament alts en termes de passos (més de 40) i desplaçaments (més de 70). S'il·lustren en la Figura 8.7 i en la Figura 8.8, respectivament.

Com esperàvem, podem dir que el nombre d'estats accessibles en les configuracions té un impacte força gran en la seva complexitat. De fet, aquest factor pot ser tan o fins i tot més influent que el nombre de passos o desplaçaments a l'hora de determinar quan difícil és resoldre una configuració, si més no per PDDL.

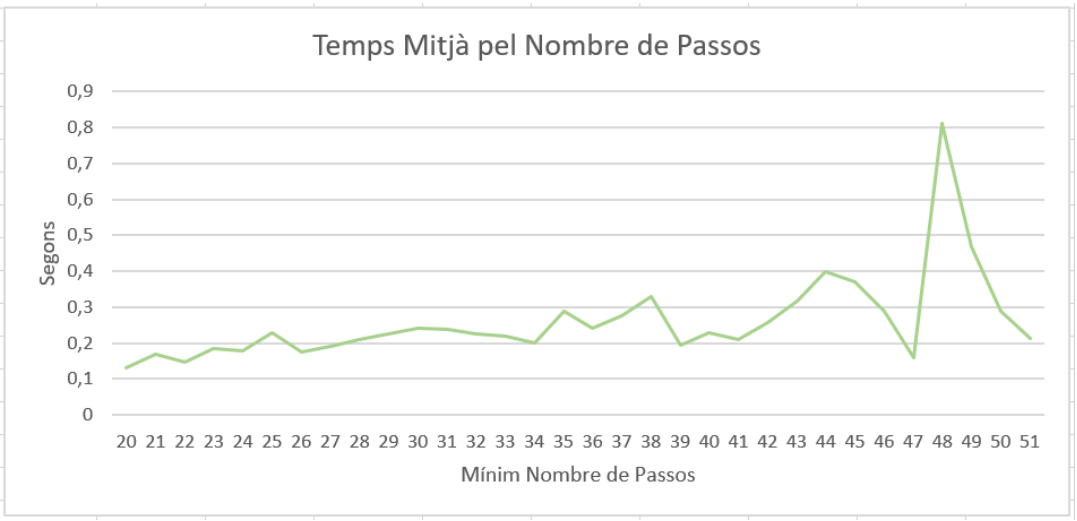


FIGURA 8.5: Temps mitjà de resolució de les instàncies amb PDDL segons nombre de passos.

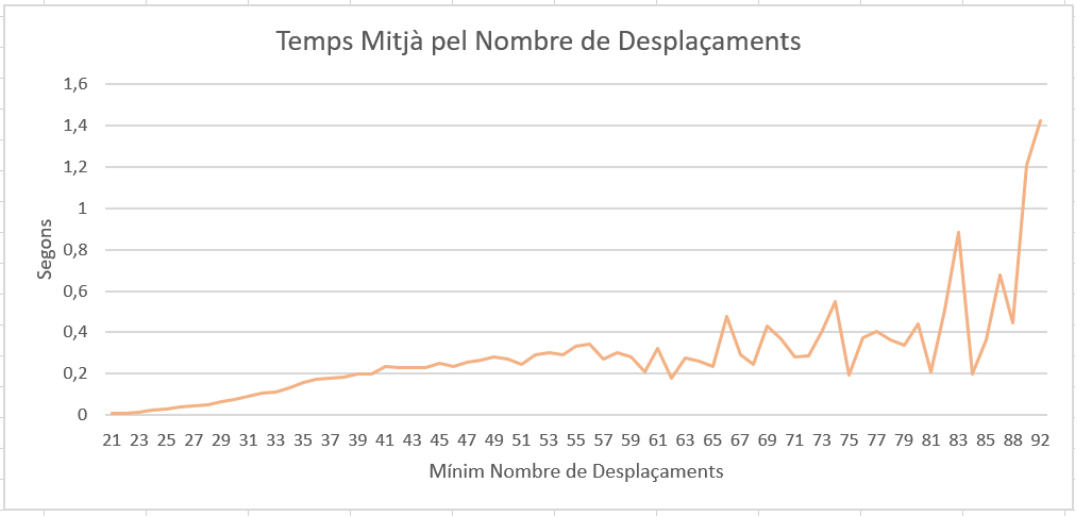


FIGURA 8.6: Temps mitjà de resolució de les instàncies amb PDDL segons nombre de desplaçaments.

8.3 Avaluació i comparació dels models

Basant-nos en les dades recopilades a l'Excel que hem explicat en l'apartat anterior, procedim a seleccionar les configuracions que considerem més significatives per a dur a terme la comparativa dels models que hem creat. Aquesta selecció es detalla a la Secció 4.4.

En aquest apartat, utilitzem els models que hem desenvolupat per resoldre aquestes configuracions seleccionades, amb l'objectiu d'analitzar els resultats i poder realitzar una bona comparativa.

Hem decidit establir un límit de temps (timeout) de 600 segons per a l'execució dels models en aquesta anàlisi. La raó principal és que no tenim interès en les comparacions quan els temps de resolució s'allunyen tan considerablement, ja que això dificulta la creació de les gràfiques i la seva interpretació.

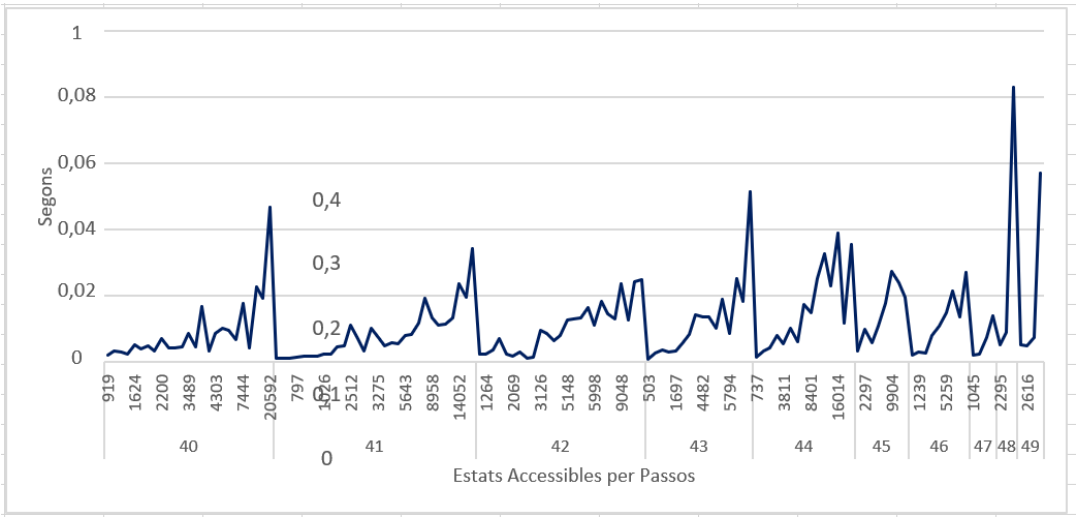


FIGURA 8.7: Temps mitjà de resolució amb PDDL segons nombre de passos i nombre d'estats accessibles.

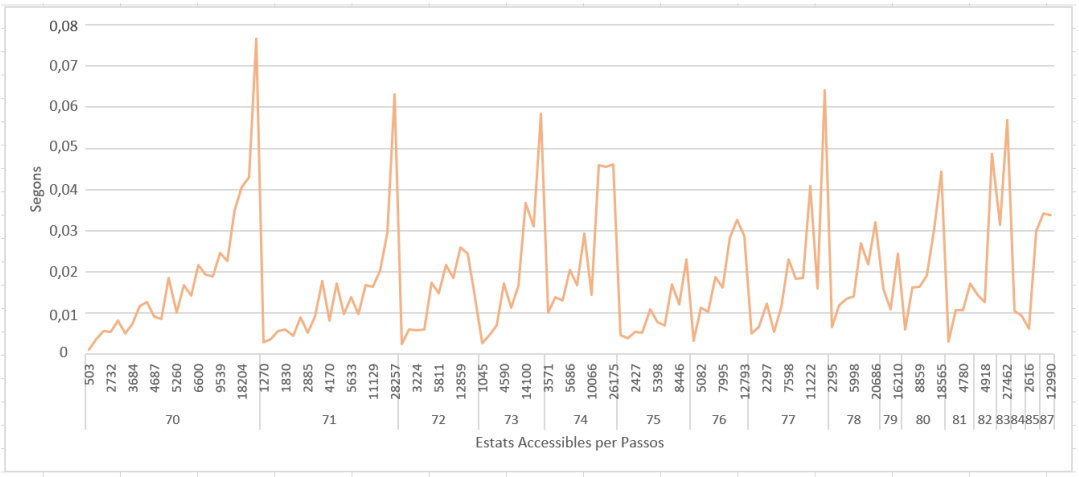


FIGURA 8.8: Temps mitjà de resolució amb PDDL segons desplaçaments i nombre d'estats accessibles.

8.3.1 Temps d'execució en PDDL

En el procés de selecció de les configuracions, inicialment consideràvem que potser hi hauria una correlació entre les instàncies més exigents pel PDDL i les altres aproximacions. No obstant això, en arribar als resultats, hem pogut concloure que aquesta correlació no existeix. El PDDL demostra una eficiència sorprenent en la resolució de les instàncies, la qual cosa no està relacionada amb els resultats observats en les altres aproximacions.

Mostrem només un gràfic (vegeu Figura 8.9), el de l'optimització de desplaçaments, per demostrar aquesta no correlació. Els dos gràfics restants, d'optimització de passos i d'optimització de passos i desplaçaments, són semblants i no ens aporten res interessant, per això no els comentem.

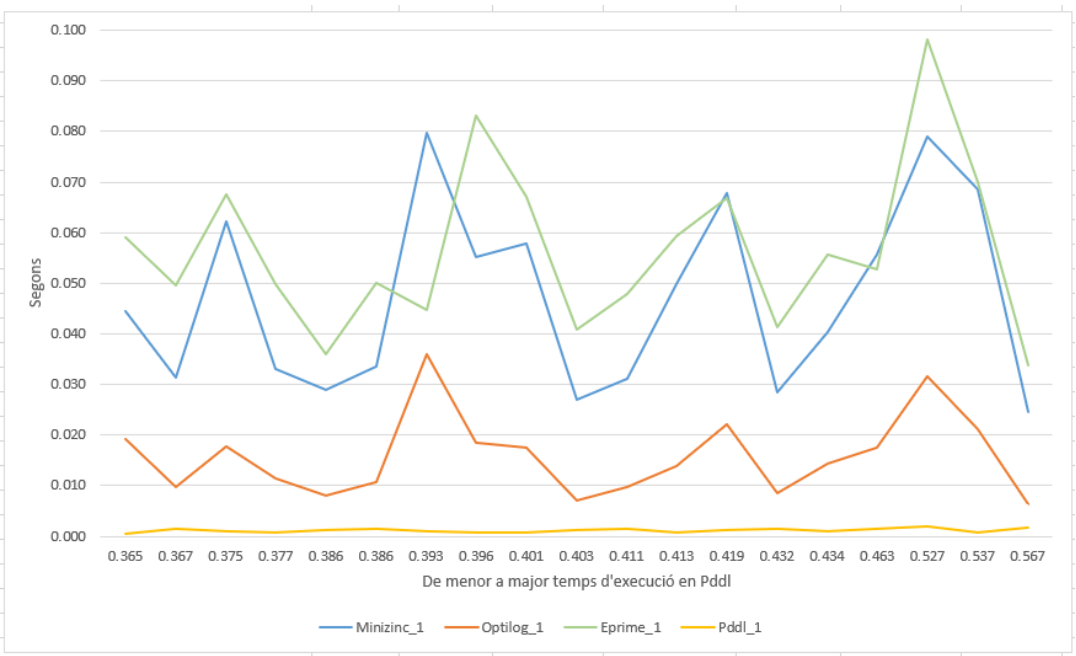


FIGURA 8.9: Temps de resolució de les instàncies difícils PDDL per les diferents aproximacions optimitzant desplaçaments.

8.3.2 Passos mínims per a trobar una solució

El nombre de passos mínims crèiem que podia ser un valor que afectés significativament a la dificultat de resolució, en partiular a les aproximacions no PDDL, ja que a cada pas s'ha de regenerar el problema. Parlem de les aproximacions en MiniZinc, Essence Prime o OptiLog.

A continuació, presentem els temps mitjans de resolució per a diferents optimitzacions i models. Aquests temps mitjans de resolució ens ofereixen una visió més detallada de com les optimitzacions i els models influeixen en la capacitat de resoldre problemes amb diferents nombres mínims de passos.

En la Figura 8.10, observem la resolució pels diferents models amb optimització de passos de cada aproximació de resolució.

Com podem veure, MiniZinc presenta resultats inestables, amb temps de resolució que oscil·len significativament i que inclouen pics inusuals en algunes configuracions. Això indica que MiniZinc no és consistent i pot tenir dificultats amb certes instàncies.

D'altra banda, Essence Prime, OptiLog i PDDL mostren una variació similar en els seus temps de resolució, segueixen una tendència relativament constant en què els temps augmenten lleugerament de manera progressiva amb els casos amb més passos i desplaçaments mínims.

En la Figura 8.11, es mostren els temps de resolució per a les mateixes configuracions, però amb optimització de desplaçaments. Aquí podem observar que, a diferència de l'òptima de passos, tots els models presenten variacions notables.

MiniZinc també mostra una variació significativa en els seus temps de resolució, la

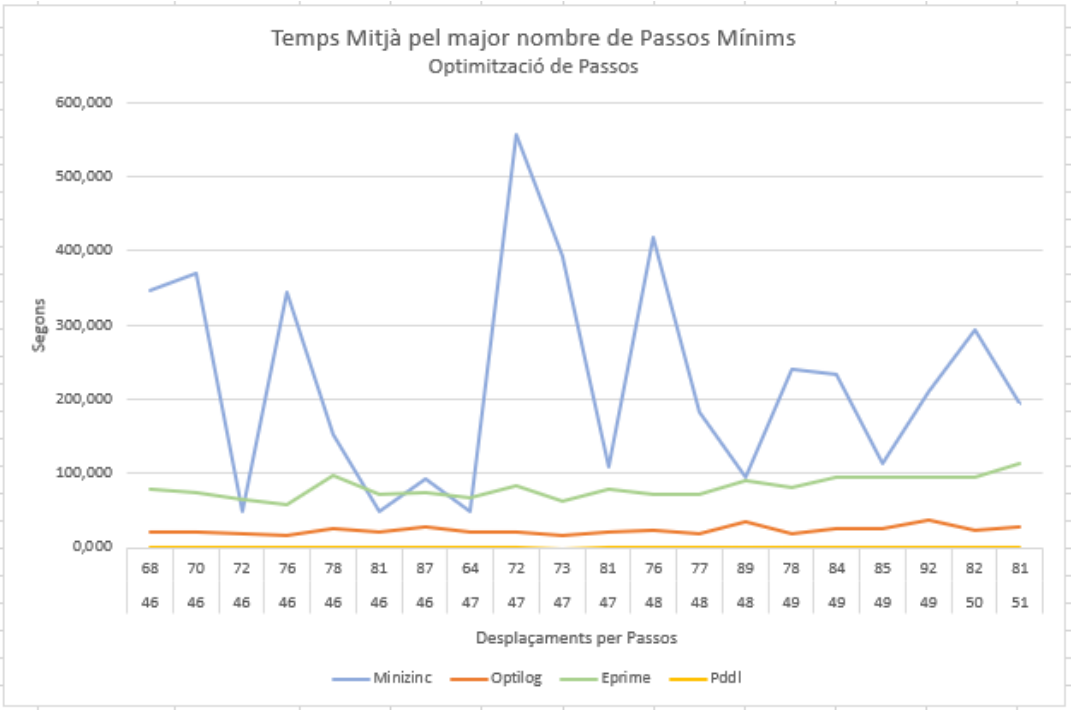


FIGURA 8.10: Temps de resolució de les instàncies amb més passos mínims per les diferents aproximacions optimitzant passos.

qual es veu reflectida en els pics que arriben al valor de temps límit (600 segons), indicant que no s'ha trobat una solució. És interessant veure que, malgrat aquesta inestabilitat, en molts casos supera a Essence Prime en quant a eficiència.

Els pics que observem en totes les aproximacions coincideixen amb situacions on hi ha més desplaçament en un pas concret, el que provoca que els models tinguin més problemes per trobar una solució. Ja que quan es passa al següent pas i la quantitat de desplaçaments es redueix, els temps de resolució descrementen i resolen les configuracions de manera molt més ràpida. Això ens suggereix que la quantitat de desplaçaments en un pas específic afecta la complexitat de resolució dels models.

En l'últim gràfic d'aquestes configuracions, on es busca l'optimització de passos i desplaçaments, PDDL segueis essent l'aproximació més eficient ja que la majoria de les instàncies es resolen en menys d'1 segon. MiniZinc, pel contrari, presenta temps de resolució molt elevats, arribant la majoria dels cops al límit de temps i, per tant, a no trobar una solució. Essence Prime mostra, en bona part del gràfic, temps de resolució semblants als de MiniZinc, encara que força més eficients i amb força menys timouts.

En els tres gràfics, es pot notar una lleugera tendència a l'alça a mesura que augmenten els passos i desplaçaments. No és tot lo visible que esperàvem, però s'hi acosta una mica.
Concluïm que PDDL es destaca com la solució més eficient. Optilog també demostra ser una opció bastant eficaç, amb temps de resolució generalment raonables. D'altra banda, Essence Prime mostra eficiència fins que ha d'optimitzar tenint en compte desplaçaments, moment en què els temps augmenten significativament. I finalment, MiniZinc es caracteritza per ser inconsistent, amb temps de resolució que varien de manera considerable.

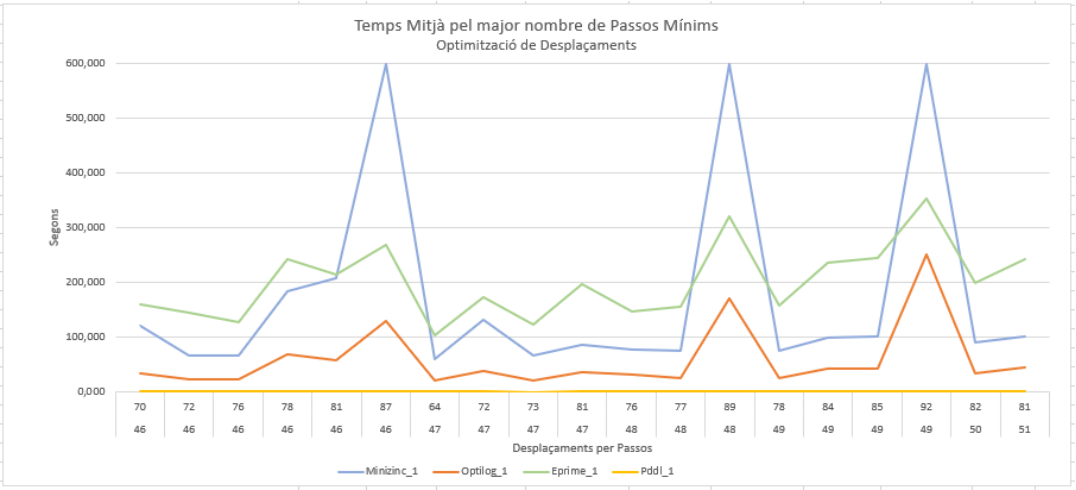


FIGURA 8.11: Temps de resolució de les instàncies amb més passos mínims per les diferents aproximacions optimitzant desplaçaments.

8.3.3 Relació cost total (desplaçament) / passos

Com hem observat en la secció anterior, s'aprecia una relació destacable entre el nombre de passos i el nombre de desplaçaments necessaris en la resolució de les instàncies. Per tant, vam decidir centrar-nos en les configuracions que presenten una relació més estreta entre aquests dos factors. Aquesta relació es calcula dividint el nombre mínim de desplaçaments pel nombre mínim de passos requerits per resoldre una instància.

Lamentablement, després de vàries observacions, no hem establert cap relació significativa amb aquest factor. La manca d'observacions rellevants ens ha portat a la conclusió que no és pertinent incloure els gràfics relacionats amb aquesta anàlisi, ja que no aporten cap informació interessant o rellevant per als nostres objectius.

8.3.4 Estats accessibles

A l'inici d'aquest capítol, observeu la Figura 8.4, hem identificat els estats accessibles com un factor crucial que influeix en la complexitat de la resolució d'instàncies en PDDL. En analitzar els gràfics (Figures 8.13 i 8.14) que mostren les instàncies amb el major nombre d'estats accessibles sota les dues optimitzacions, pasos i desplaçaments, no s'observa cap correlació significativa entre nombre d'estats i temps de resolució.

Cal tenir en compte que aquestes instàncies presenten un elevat nombre d'estats accessibles, però també són semblants en aquest aspecte. Això podria fer que fos difícil avaluar les variacions dels estats accessibles en aquest apartat, a més, perquè també hi afecten altres factors.

Llavors, hem decidit aprofitar aquestes dades per demostrar un altre factor que influeix en la complexitat de les instàncies. Com que els valors dels estats accessibles són molt similars en aquest conjunt d'instàncies, podem concloure que aquest factor específic no està influint de manera significativa en els resultats que estem observant. Per tant, podem evaluar el factor del nombre de desplaçaments.

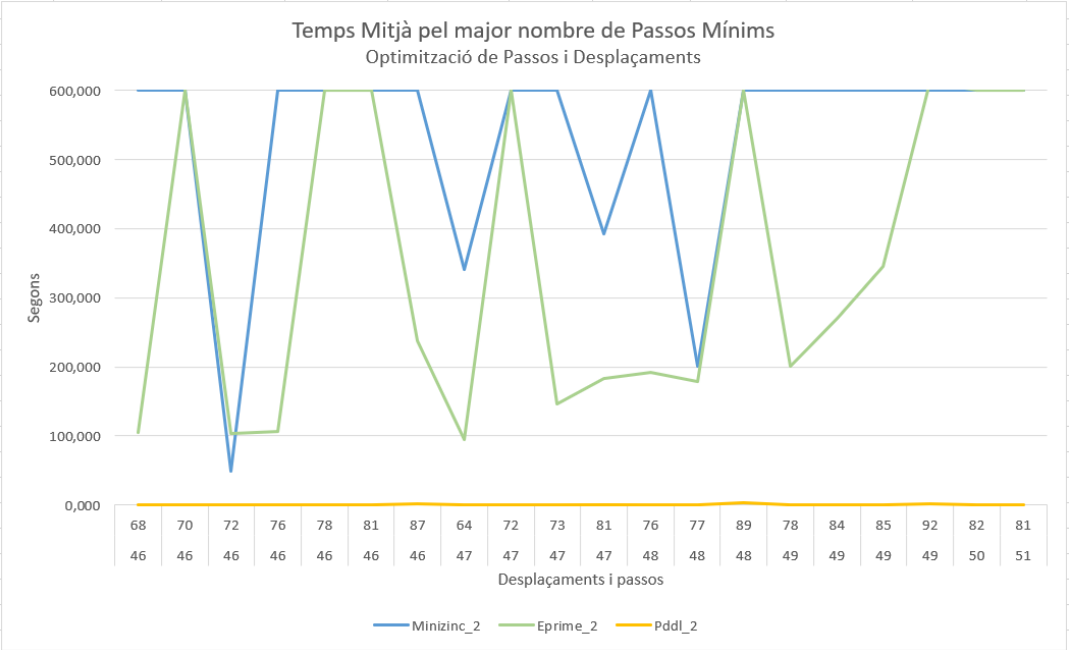


FIGURA 8.12: Temps de resolució de les instàncies amb més passos mínims per les diferents aproximacions optimitzant passos i desplaçaments.

Si ara en comptes d'ordenar-les de menor a major nombre d'estats accessibles, ho fem de menor a major distància recorreguda, obtenim les Figures 8.15 i 8.16.

Per tant, podem observar que amb un nombre d'estats elevat, a més desplaçament més compleixitat de resolució per les aproximacions MiniZinc, Essence Prime i OptiLog.

Aquesta nova perspectiva ens revela que la distància recorreguda és un factor determinant en la complexitat per les instàncies amb similar nombre d'estats accessibles.

8.3.5 Major nombre de passos i desplaçaments mínims amb més estats accessibles

Després de realitzar els anàlisis en els experiments anteriors i identificar els factors crítics que afecten a la complexitat dels models, s'ha decidit sotmetre els models a una última prova utilitzant les instàncies que es consideren més dures de resoldre.

Com hem comentat a la Secció 4.4, on detallem com s'ha efectuat l'elecció, disposem de 43 instàncies amb els majors nombres de passos i desplaçaments, i amb el nombre d'estats accessibles major a 10.000.

La nostra pregunta inicial es centrava en determinar quin factor d'ordenació de les dades seria més apropiat per a la identificació d'estudis interessants, tenint en compte els tres factors disponibles que són el nombre de passos, el nombre de desplaçaments mínims i el nombre d'estats accessibles.

Hem realitzat diversos estudis.

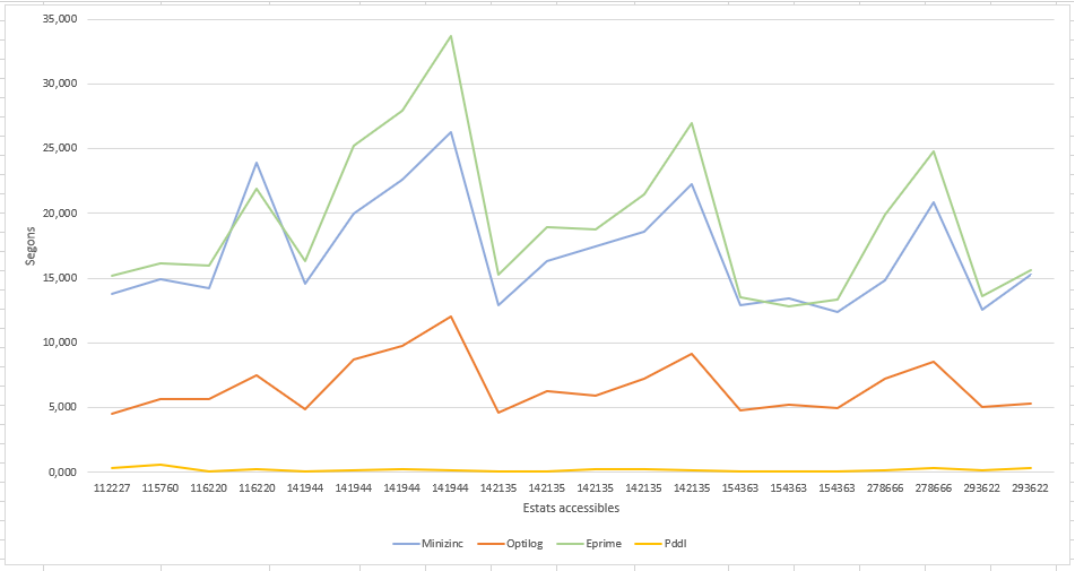


FIGURA 8.13: Temps de resolució de les instàncies amb més estats accessibles per les diferents aproximacions optimitzant passos.

En la Figura 8.17 mostrem la mitjana de temps de resolució pel nombre de desplaçaments de les instàncies. Com podem veure, la dificultat pels solvers augmenta pronuncialment a mesura que augmenten els desplaçaments.

De manera semblant, la Figura 8.18 mostra el mateix pero optimitzant el nombre de passos i de desplaçaments. És interessant veure com els solvers tenen dificultats addicionals quan han de realitzar optimització doble, troben aquesta tasca particularment complicada i necessiten més temps de càlcul.

En resum, aquests gràfics revelen que les instàncies amb un nombre més gran de desplaçaments o amb optimització doble presenten un augment substancial de la complexitat, i els solvers tenen dificultats addicionals en aquests escenaris, cosa que es tradueix en temps de resolució més llargs.

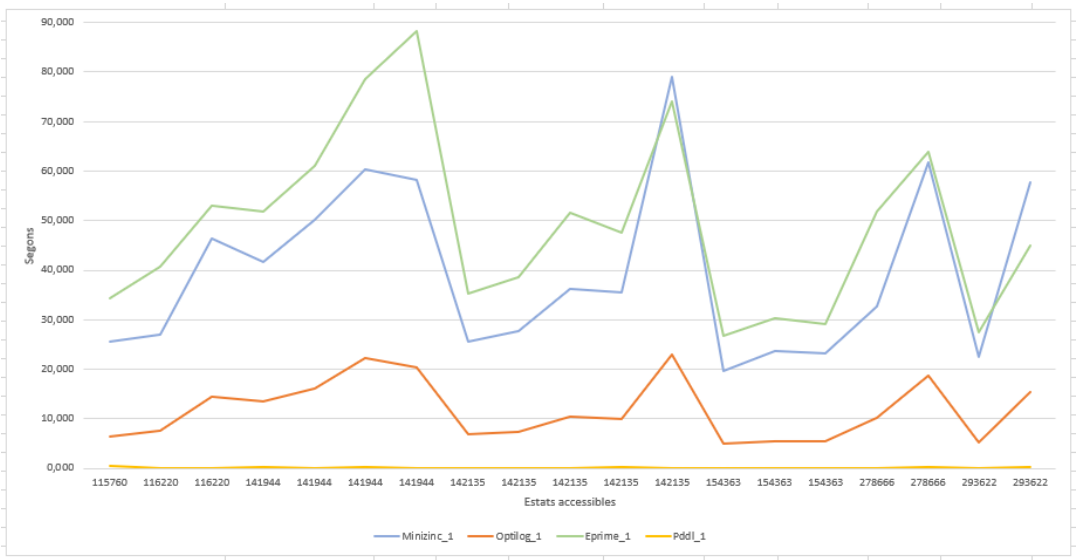


FIGURA 8.14: Temps de resolució de les instàncies amb més estats accessibles per les diferents aproximacions optimitzant desplaçaments.

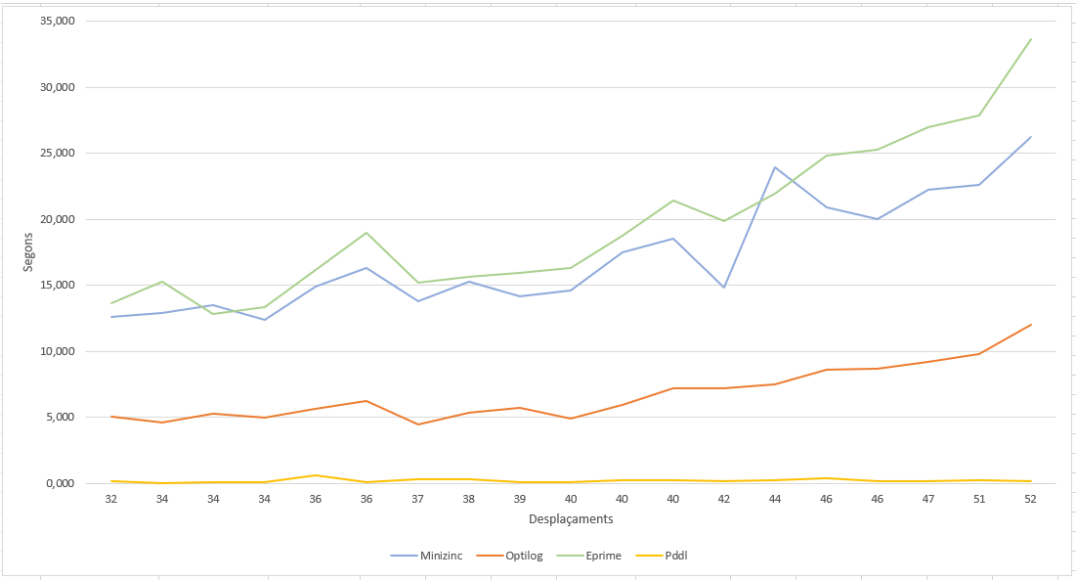


FIGURA 8.15: Temps de resolució per desplaçaments de les instàncies amb més estats accessibles per les diferents aproximacions optimitzant passos.

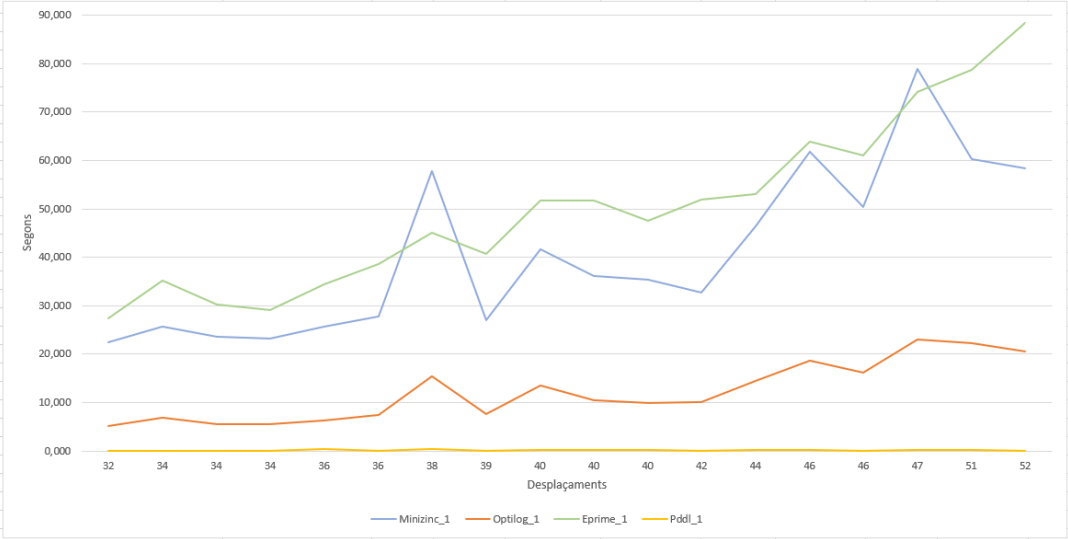


FIGURA 8.16: Temps de resolució per desplaçaments de les instàncies amb més estats accessibles per les diferents aproximacions optimitzant desplaçaments.

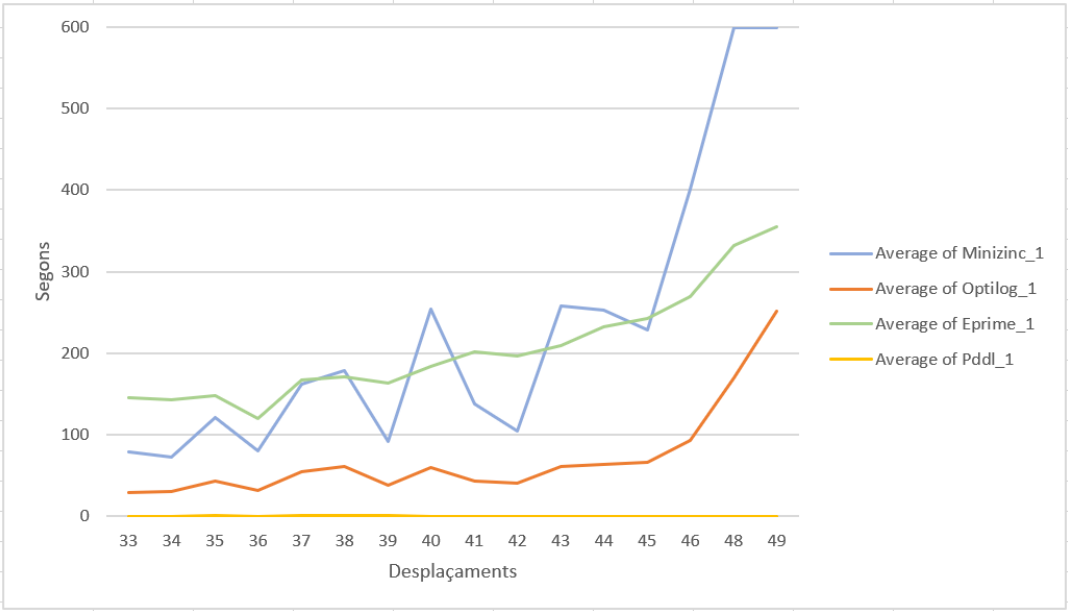


FIGURA 8.17: Temps mitjà de resolució per desplaçaments de les instàncies més dures per les diferents aproximacions optimitzant passos.

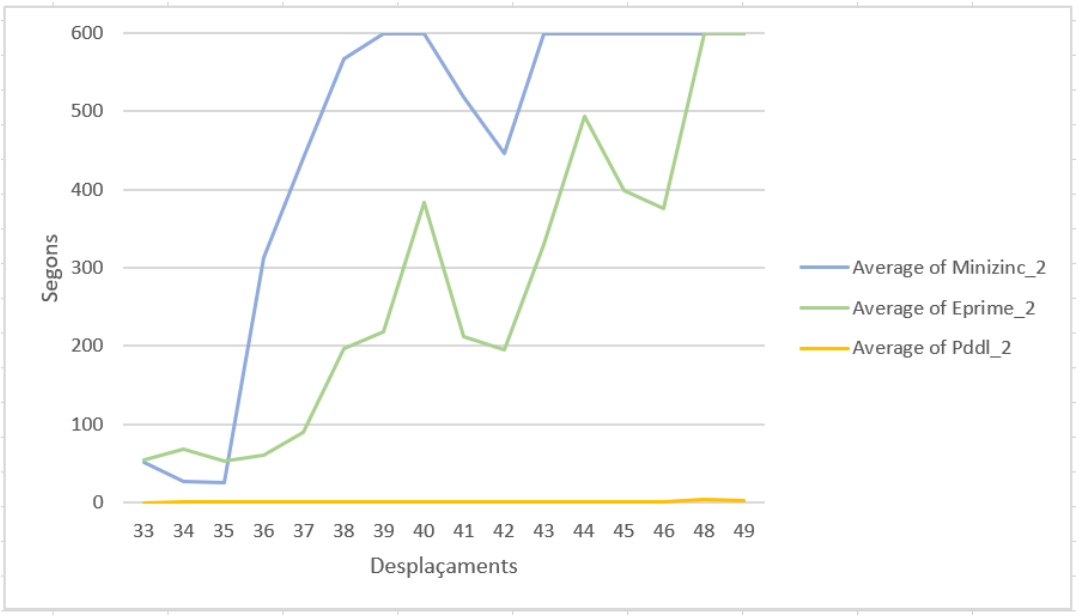


FIGURA 8.18: Temps mitjà de resolució per desplaçaments de les instàncies més dures per les diferents aproximacions optimitzant passos i desplaçaments.

Capítol 9

Conclusions

El joc de Rush Hour és un problema complex i presenta reptes de dificultat quan es tracta de buscar solucions òptimes, en concret, i sobretot, a les aproximacions basades en programació de restriccions. Personalment ens ha ofert l'oportunitat d'estudiar nombroses tècniques i marcs de treball per a la resolució de problemes complexes i en particular de plannificació: PDDL-planning, MiniZinc-Programació de restriccions, i Essence Prime i Optilog - SAT.

Les conclusions més tècniques les obtenim en el marc de la dificultat de les instàncies i de l'eficiència dels solucionadors:

- L'estudi i classificació de la dificultat de les instàncies és un aspecte complex i que requereix encara de més experiments a poder ser amb instàncies que suposin un repte més significatiu a aproximacions de resolució com la del PDDL o la de SAT. No obstant això hem pogut relacionar un augment de temps de resolució amb un increment en el número de passos. Aquest fenòmen s'observa sobretot quan fem l'anàlisi en instàncies que tenen un gran nombre d'estats accessibles i per tant, on l'espai de cerca serà més gran.

- Pel que fa a les aproximacions de solucionadors considerades, clarament el conjunt d'instàncies disponibles no són cap mena de repte per a PDDL. L'aproximació amb Optilog i SAT podríem dir que és la segona més robusta, no obstant, quan estem fent optimització de desplaçaments, comencen a ser instàncies més difícils per al SAT solver. L'aproximació amb SAT solving d'Essence Prime també presenta dificultats en resoldre algunes instàncies fent fins i tot fa algun timeout. Finalment l'aproximació que mostra un comportament menys robust i un tant erràtic és la de resolució de restriccions amb MiniZinc i el LazyFD solver Chuffed.

Per dificultat de problema, on més han patit les aproximacions Essence Prime i MiniZinc ha sigut en la optimització de passos i desplaçaments on han fet nombrosos timeouts mentre que PDDL resolia en segons (la versió SAT no s'ha implementat per aquest tipus d'optimització).

Pel que fa a les conclusions personals, com he dit, l'aprenentatge de diferens tècniques per a resoldre problemes combinatoris ha sigut un enriquiment molt important. Així com la gestió de clúster per llençar experiments i recuperar resultats, la confecció de gràfiques per a mostrar els resultats, o l'escriputra de programes per a transformació de formats. Per no parlar dels problemes tècnics en instal·lació de softwares diversos.

Espero, i de fet estic convençuda, que aquests aprenentatges em seran molt útils i han sentat la base per a poder dur a terme uns estudis de doctorat en el grup d'IA de la Universitat de St. Andrews.

Per últim, he tingut un primer tast en el desengany d'algunes hipotesis inicials de treball però també la satisfacció de fer troballes de vegades inesperades. Entenc això és quelcom habitual en el món de la recerca.

Capítol 10

Treball futur

Com a treball futur ens plantegem encara la implementació d'un model SMT (SAT mòdul teories) i una versió en programació lineal entera per a tenir una visió completa en la comparació de les tècniques de resolució. Com hem dit per tal de tenir una comparació més potent ens convidria instàncies més dures.

El tema de generar instàncies més dures segueix siguent el principal repte que ens queda pel futur. Així doncs una de les primeres tasques que em proposo dur a terme és la d'explorar amb més deteniment aquest aspecte. Ens plantegem tres opcions principals:

- Crafted: la idea seria mirar d'encaçular instàncies difícils en mapes més grans que estiguessin més o menys encapçulats en "sarcòfags" de manera que per exemple es poguessin haver de resoldre dues instàncies de 6×6 de forma encadenada en un mapa de 6×13 que tindria una paret amb un forat pel cotxe vermell a la fila de sortida del cotxer vermell. Evidentment aquest patró es podria repetir amb diversos encadenaments.

- Explorar Fogleman: pel que hem vist hi ha un problema de memòria que fa que el 7×7 no es pugui resoldre. Caldria una anàlisi a fons a veure si hi ha alguna "fuga de memòria" reparable o veure si realment hi ha una explosió exponencial que faci que no hi hagi prou recursos per a l'execució.

- QBF: buscar un encoding amb QBF (Quantified Boolean Formulas) que ens permeti expressar la construcció d'escenaris amb un nombre de passos més grans en taulers més grans.

Annex A

Script en Python pel model Minizinc

```
from minizinc import Instance, Model, Solver
import time
import sys
import pandas as pd
import openpyxl

# S'inicialitzen les columnes que es volen extreure a l'Excel i es crea la
    taula de dades buida amb les columnes respectives
columns = ['Board','Steps','Moves','Execution time']
df = pd.DataFrame(columns=columns)

# S'obtenen els parametres
if len(sys.argv) < 2:
    print("Es requereixen arguments")
    sys.exit(1)
arguments = sys.argv[1]
parts = arguments.split()
try:
    arg1 = int(parts[0])
    arg2 = int(parts[1])
    arg3 = list(map(int, parts[2][1:-1].split(',')))
    arg4 = list(map(int, parts[3][1:-1].split(',')))
    arg5 = list(map(int, parts[4][1:-1].split(',')))
except (ValueError, IndexError):
    print("Els valors no son valids.")
    sys.exit(1)

# Obre i tanca l'arxiu test.dzn en mode d'escriptura per esborrar el
    contingut
with open("minizinc/test.dzn", "w") as file:
    pass

# Escriu en l'arxiu els parametres que llegira el model
with open("minizinc/test.dzn", "w") as file:
    file.write(f"boardSize={arg1};\n")
    file.write(f"vehicles={arg2};\n")
    file.write(f"sizes={arg3};\n")
    file.write(f"versus={arg4};\n")
    file.write(f"initial={arg5};\n")

# Es puja el model
rush_hour = Model("minizinc/rush_hour_all.mzn")
# Es passa l'arxiu, que conte els parametres de la configuracio, al model
rush_hour.add_file("minizinc/test.dzn")
```

```
# Inicialitza el solver Chuffed i es crea una instancia pel model
chuffed = Solver.lookup("chuffed")
instance = Instance(chuffed, rush_hour)
# Es registra el temps d'inici
start_time = time.time()

steps = 1
instance["steps"] = steps

# Prova de resoldre la instancia
result = instance.solve()

# Fins que no troba una solucio, s'incrementa en una unitat el nombre de
    passos i ho torna a provar
while result.solution is None:
    steps += 1
    instance = Instance(chuffed, rush_hour)
    instance["steps"] = steps
    result = instance.solve()

end_time = time.time()

if result.solution is None:
    print("Unsolvable")
else:
    # Escriu la solucio i el temps d'execucio en un fitxer de text
    with open("result.txt", "w") as file:
        file.write(str(result.solution),"\n")
        file.write("Time: ", str(end_time - start_time), " seconds\n")
```

Annex B

Script en Bash pel model Essence Prime

```
#!/bin/bash
start_time=$(date +%s.%N)

arguments="$1"

steps=1
boardSize=$(echo "$arguments" | cut -d ' ' -f 1)
vehicles=$(echo "$arguments" | cut -d ' ' -f 2)
sizes=$(echo "$arguments" | cut -d ' ' -f 3)
versus=$(echo "$arguments" | cut -d ' ' -f 4)
initial=$(echo "$arguments" | cut -d ' ' -f 5)

output=$(savilerow savilerow/rush_hour_all.eprime -sat -run-solver -params
     "letting boardSize be $boardSize
    letting steps be $steps
    letting vehicles be $vehicles
    letting sizes be $sizes
    letting versus be $versus
    letting initial be $initial" 2>&1)

while ! (echo "$output" | grep -q "Created solution")
do
    ((steps++))
    output=$(savilerow savilerow/rush_hour_all.eprime -sat -run-solver -
    params "letting boardSize be $boardSize
        letting steps be $steps
        letting vehicles be $vehicles
        letting sizes be $sizes
        letting versus be $versus
        letting initial be $initial" 2>&1)
done
```

Annex C

Arxiu slurm per executar el Generador d'Instàncies

```
#!/bin/bash
#SBATCH --partition=sense
#SBATCH --job-name=rushhour                    # Job name
#SBATCH --mail-type=END,FAIL                   # Mail events (NONE, BEGIN, END
    , FAIL, ALL)
#SBATCH --mail-user=carladavesa4@gmail.com # Where to send mail
#SBATCH --cpus-per-task=4                      # Take all cores
#SBATCH --mem=0
#SBATCH -n 1
#SBATCH --output=rushhour%j.log    # Standard output and error log

date
./cpp/src/rush.exe
date
```

Annex D

Extractor de les característiques de les configuracions a Excel

```
import subprocess
import pandas as pd
import openpyxl

# S'inicialitzen les columnes que es volen extreure a l'excel
columns = ['Board size','Board','Search space','Steps','Vehicles','Trucks'
    ,'Total time', 'Search time', 'Planner time', 'Plan cost']
# Es crea el DataFrame buit amb les columnes respectives
df = pd.DataFrame(columns=columns)

# Es defineix l'execucio del model PDDL
def run_fast_downward():
    command = ["fast-downward.py", "--alias", "seq-opt-lmcut", "pddl/
    domain_all.pddl", "pddl/problem.pddl"]
    try:
        result = subprocess.run(command, stdout=subprocess.PIPE, text=True
    , check=True)
        return result.stdout
    except subprocess.CalledProcessError as e:
        print("Error:", e)
        return None

# S'extreuen els valors i s'afegeixen a un nou DataFrame que es concatena
    amb el general
def extract_and_save_values(output, searchSpace, steps, board, boardSize,
    vehicles, trucks):
    extracted_values = {}
    extracted_values['Board size'] = boardSize
    extracted_values['Board'] = board
    extracted_values['Search space'] = searchSpace
    extracted_values['Steps'] = steps
    extracted_values['Vehicles'] = vehicles
    extracted_values['Trucks'] = trucks

    for line in output.splitlines():
        for value in columns:
            if value in line:
                this_value = line[line.find(value) + len(value)+2:]
                extracted_values[value] = this_value

    new_row = pd.DataFrame([extracted_values], columns=columns)
    global df
    df = pd.concat([df, new_row], ignore_index=True)
```

```
# Es llegeix l'arxiu linea per linea
with open('6x6_to_analyse.txt', 'r') as file:
    lines = file.readlines()
    # Compila el primer programa
    compile_process1 = subprocess.run(['scalac', 'configurations/src/main/
    scala/Translate.scala'], capture_output=True, text=True)
    compile_output1 = compile_process1.stdout

for line in lines:
    parts = line.split()
    steps = parts[0]
    board = parts[1]
    searchSpace = parts[2]

    # S'executa el primer programa (Traductor de Lexicografic a Generic)
    run_process1 = subprocess.run(['scala', '-cp', '.', 'Translate', board
    ], capture_output=True, text=True, input=board)
    run_output1 = run_process1.stdout
    parts1 = run_output1.split()
    boardSize = parts1[0]
    vehicles = parts1[1]
    trucks = parts1[2].count('3')

    # Es compila i executa el segon programa (Traductor de Generic a arxiu
     problema PDDL)
    compile_process2 = subprocess.run(['scalac', 'create_configs_pddl/src/
    main/scala/ToPDDL.scala'], capture_output=True, text=True)
    compile_output2 = compile_process2.stdout

    run_process2 = subprocess.run(['scala', '-cp', '.', 'ToPDDL',
    run_output1], capture_output=True, text=True, input=run_output1)
    run_output2 = run_process2.stdout
    run_error2 = run_process2.stderr

    # S'executa el model PDDL
    output = run_fast_downward()
    if output:
        extract_and_save_values(output, searchSpace, steps, board,
    boardSize, vehicles, trucks)

# Es guarda el DataFrame en un arxiu xlsx
excel_filename = "extracted_values_6x6.xlsx"
df.to_excel(excel_filename, index=False)
```

Bibliografia

[1] Michael Sipser. *Introduction To The Theory Of Computation*. 2012. URL: https://mog.dog/files/SP2019/Sipser_Introduction.to.the.Theory.of.Computation.3E.pdf.

[2] Gary William Flake i Eric B. Baum. "Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants"". A: *Theoretical Computer Science* 270.1 (2002), pàg. 895 - 911. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(01)00173-6. URL: https://www.sciencedirect.com/science/article/pii/S0304397501001736.

[3] Daniel Ratner i Manfred Warmuth. "Finding a Shortest Solution for the NxN Extension of the 15-Puzzle is Intractable". A: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*. AAAI'86. Philadelphia, Pennsylvania: AAAI Press, 1986, pàg. 168 - 172.

[4] Henning Fernau et al. *On the parameterized complexity of the generalized rush hour puzzle.* Gen. de 2003, pàg. 6 - 9.

[5] Guido Tack Peter J. Stuckey Kim Marriott. *The MiniZinc Handbook*. URL: https://www.minizinc.org/doc-2.7.6/en/index.html.

[6] Josep Alòs et al. "OptiLog V2: Model, Solve, Tune and Run". A: *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*. Ed. de Kuldeep S. Meel i Ofer Strichman. Vol. 236. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 25:1 - 25:16. ISBN: 978-3-95977-242-6. DOI: 10.4230/LIPIcs.SAT.2022.25. URL: https://drops.dagstuhl.de/opus/volltexte/2022/16699.

[7] Peter Nightingale. "Savile Row Manual". A: (nov. de 2021). URL: https://arxiv.org/pdf/2201.03472.pdf.

[8] Malik Ghallab et al. *PDDL - The Planning Domain Definition Language*. URL: https://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf.

[9] Lorenzo Cian, Talissa Dreossi i Agostino Dovier. "Modeling and Solving the Rush Hour Puzzle". A: *University of Udine, DMIF, Via delle Scienze 206, 33100 Udine, Italy* (2022), pàg. 4 - 7. URL: https://ceur-ws.org/Vol-3204/paper_29.pdf.

[10] Michael Fogleman. "Solving Rush Hour, the Puzzle". A: (jul. de 2018). URL: https://www.michaelfogleman.com/rush/.

[11] Jelle van Assema. "On the Hardness of 6x6 Rush Hour". A: *Afstudeerproject Thesis Bachelor Kunstmatige Intelligentie Faculteit der Natuurwetenschappen, Wiskunde en Informatica Universiteit van Amsterdam* (juny de 2014). URL: https://staff.fnwi.uva.nl/b.bredeweg/pdf/BSc/20132014/VanAssema.pdf.

[12] ehajdini. "Rush Hour - PDDL". A: (març de 2019). URL: https://github.com/ehajdini/AI/blob/master/RushHour_PDDL/domain1.pddl.