Planificador de projectes de programació

Pràctica de planificació

Intel·ligència Artificial

Àlex Domínguez, Manel Murillo, Joan Sales

12 de juny de 2023

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Índex

1 Introduction

En aquesta pràctica hem de resoldre un problema de planificació. Farem servir *Metric Fast Forward*. Aquest planificador ens permet atacar problemes definits en PDDL (*Planning Domain Definition Language*).

Els problemes modelats amb PDDL es divideixen en dues parts:

- **Modelització del domini.** Aquesta part conté el domini del problema, és a dir, definirem els tipus, les accions i els predicats.
- **Modelització del problema.** Aquí hi ha la informació concreta del problema. Aquí hi seran els objectes que participen en el problema concret, l'estat inicial i l'estat objectiu

Una vegada modelat el problema, el planificador buscarà una resposta vàlida, on es veurà el conjunt d'accions que han portat de l'estat inicial a l'estat objectiu.

2 Descripció del problema

Se'ns ha encarregat dissenyar un planificador per organitzar un projecte de programació de gran escala. Haurem de repartir les tasques a realitzar entre els programadors dels quals disposem.

Cada tasca té un grau de dificultat (que va de 1 a 3), i un temps estimat el qual es necessita per completar-la.

Disposem també d'un grup de programadors. Cada un d'ells té un grau d'habilitat (entre 1 i 3). No volem assignar als programadors tasques que superin en més d'1 la seva habilitat. En cas que se li assigni a un programador una tasca 1 nivell per sobre de la seva habilitat, el temps que trigarà a resoldre-la s'incrementarà en dues hores.

Per últim, cada tasca completada haurà de ser revisada per un programador diferent del qual l'ha completat. Aquesta revisió tindrà el mateix grau de dificultat que la tasca original. Cada programador té associada una qualitat (1 ó 2). Si la tasca original l'ha realitzat un programador amb qualitat 1, la nova tasca generada necessitarà 1 hora de revisió, en canvi, si una tasca l'ha realitzat un programador amb qualitat 2, la nova tasca generada necessitarà 2 hores de revisió. Aquestes noves tasques de revisió no generen cap tasca nova i no tenen en compte el nivell d'habilitat del programador que les revisa.

3 Nivell bàsic

Donat el conjunt de tasques i programadors disponibles, obtenir una assignació de totes les tasques a programadors sense tenir en compte la tasca addicional que genera cada assignació a un programador.

Aquest és l'únic cas en què no hem usat fluents per a resoldre el problema, això ha provocat que Habilitat i Dificultat fossin variables del domini. Aquest cas és diferent de la resta pel que fa a com hem desenvolupat el codi. El nostre objectiu era resoldre-ho sense la necessitat d'utilitzar fluents.

3.1 Domini

3.1.1 Variables

- **Programador.** S'utilitzarà per a les variables que corresponen al conjunt de programadors.
- **Tasca.** S'utilitzarà per a les variables que corresponen al conjunt de tasques.
- **Habilitat.** Els tres valors diferents que pot tenir com a grau d'habilitat un programador.
- **Dificultat.** Els tres valors diferents que pot tenir com a grau de dificultat una tasca.

3.1.2 Predicats

- **(te ?p - programador ?h - habilitat)**. Assigna una habilitat a un programador.
- **(es ?t - tasca ?d - dificultat)**. Assigna una dificultat a una tasca.
- **(assignada ?t - tasca)**. Comprova si la tasca està assignada o no.
- **(disponible ?p - programador)**. Comprova si el programador està disponible o no.
- **(realitza ?p - programador ?t - tasca)**. Assigna un programador a una tasca.
- **(assumible ?d - dificultat ?h - habilitat)**. Assigna quins graus d'habilitat dels programadors poden fer les diferents dificultats de les tasques.

3.1.3 Accions

- **assignar**

  Paràmetres

  - ?p - programador

- ?h - dificultat
- ?t - tasca
- ?d - dificultat

Precondició

- La tasca no ha estat assignada.
- El programador està disponible.
- El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

Efecte

- El programador ja no està disponible.
- La tasca està assignada.
- El programador p realitza la tasca t.

3.2 Problema

3.2.1 Objectes

- Un conjunt de programadors.
- Les tres habilitats possibles per a un programador.
- Un conjunt de tasques.
- Les tres dificultats possibles per a una tasca.

3.2.2 Estat inicial

- Per a cada programador li assignem el seu grau d'habilitat.
- Posem com a disponibles a tots els programadors.
- Per a cada tasca li assignem el seu grau de dificultat.
- Posem com a no assignades a totes les tasques.
- Definim la relació entre les dificultats de les tasques i el grau d'habilitat que pot resoldre-les.

3.2.3 Estat final

Com a estat final definirem l'objectiu de què totes les tasques estiguin assignades.

3.3 Jocs de prova per al nivell bàsic

Per comprovar la correctesa del programa al nivell bàsic hem creat dos jocs de proves molt senzills:

3.3.1 Joc de Prova 1: Prova factible

En aquest joc de proves es fa un experiment on segur que existeix una assignació que pugui satisfer totes les tasques:

Input:

Programadors	p1	p2	p3
Habilitat	1	1	2

Tasques	t1	t2	t3
Dificultat	2	2	3

Output:

```
step    0: ASIGNAR P1 H1 T2 D2
        1: ASIGNAR P2 H1 T1 D2
        2: ASIGNAR P3 H3 T3 D3
```

Observacions:
Tal i com era previst, el programa crea una assignació vàlida i trivial dels programadors a les tasques sense cap problema

3.3.2 Joc de Prova 2: Prova no factible

En aquest joc de proves es fa un experiment on segur que NO existeix una assignació que pugui satisfer totes les tasques

Input:

Programadors	p1	p2	p3
Habilitat	1	1	1

Tasques	t1	t2	t3
Dificultat	3	3	3

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
 ... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
```

```
... done.


ff: goal can be simplified to FALSE. No plan will solve it
```

Observacions:
Evidentment, el programa ens avisa que no és possible fer una planificació que assigni els programadors a les tasques satisfactòriament.

4 Extensió 1

Tot el que es demana en el nivell bàsic, però afegint que es tingui en compte la tasca addicional de revisió que genera la qualitat dels programadors.

4.1 Domini

4.1.1 Variables

- **Programador.** S'utilitzarà per a les variables que corresponen al conjunt de programadors.
- **Tasca.** S'utilitzarà per a les variables que corresponen al conjunt de tasques.

4.1.2 Funcions

- **(habilitat ?p - programador)**. Assigna un grau d'habilitat a un programador.
- **(dificultat ?t - tasca)**. Assigna un grau de dificultat a una tasca.

4.1.3 Predicats

- **(assignada ?t - tasca)**. Comprova si una tasca està assignada.
- **(revisada ?t - tasca)**. Comprova si una tasca està revisada.
- **(disponible ?p - programador)**. Comprova si un programador està disponible.
- **(assignada_a ?t - tasca ?p - programador)**. Assigna una tasca a un programador.

4.1.4 Accions

- **assignar**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca no ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador està disponible.
  - El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

Efecte

- El programador ja no està disponible.
- La tasca està assignada.
- El programador p realitza la tasca t.

- **revisar**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador revisor no és el mateix que ha fet la tasca original.
  - El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

  Efecte

  - La tasca ha estat revisada.

4.2 Problema

4.2.1 Objectes

- Un conjunt de programadors.
- Un conjunt de tasques.

4.2.2 Estat inicial

- Posem com a disponibles a tots els programadors.
- Posem com a no assignades a totes les tasques.
- Posem com a no revisades a totes les tasques.
- Assignem el grau d'habilitat de cada programador.
- Assignem el grau de dificultat de cada tasca.

4.2.3 Estat final

Com a estat final definirem l'objectiu de què totes les tasques estiguin revisades.

4.3 Jocs de prova per a l'extensió 1

4.3.1 Joc de Prova 1: Prova factible

En aquest cas veurem un joc de prova que es realitza correctament, complint els objectius de l'extensió.

Input:

Programadors	p1	p2	p3
Habilitat	1	2	3

Tasques	t1	t2	t3
Dificultat	1	2	3

Output:

```
step    0: ASIGNAR P1 T1
        1: REVISAR P3 T1
        2: ASIGNAR P2 T2
        3: REVISAR P3 T2
        4: ASIGNAR P3 T3
        5: REVISAR P2 T3
```

Observacions:
El programa genera una planificació trivial i correcta del problema.

4.3.2 Joc de Prova 2: Prova no factible

Input:

Programadors	p1	p2	p3
Habilitat	1	1	1

Tasques	t1	t2	t3
Dificultat	3	3	3

Output:

```
best first search space empty! problem proven unsolvable.
```

Observacions:
Com era d'esperar, el programa no ha sigut capaç de fer l'assignació ja que no existeixen programadors hàbils per fer cap tasca difícil.

5 Extensió 2

Tot el que es demana en la extensió 1, però ara ens interessa minimitzar el temps total que s'utilitza per a resoldre totes les tasques (suma de les hores).

5.1 Domini

5.1.1 Variables

- **Programador.** S'utilitzarà per a les variables que corresponen al conjunt de programadors.
- **Tasca.** S'utilitzarà per a les variables que corresponen al conjunt de tasques.

5.1.2 Funcions

- **(habilitat ?p - programador)**. Assigna un grau d'habilitat a un programador.
- **(dificultat ?t - tasca)**. Assigna un grau de dificultat a una tasca.
- **(calitat ?p - programador)**. Assigna un grau de qualitat a un programador.
- **(temps ?t - tasca)**. Assigna un temps de duració a una tasca.
- **(tempsTotal)**. És el temps total de compliment de totes les tasques assignades amb les revisions.

5.1.3 Predicats

- **(assignada ?t - tasca)**. Comprova si una tasca ha estat assignada.
- **(revisada ?t - tasca)**. Comprova si una tasca ha estat revisada.
- **(disponible ?p - programador)**. Comprova si el programador està disponible.
- **(assignada_a ?t - tasca ?p - programador)**. Assigna la tasca t al programador p.
- **(revisat_per ?t - tasca ?p - programador)**. Assigna la revisió de la tasca t al programador p.

5.1.4 Accions

- **assignar**

  Paràmetres

  - ?p - programador

- ?t - tasca

Precondició

- La tasca no ha estat assignada.
- La tasca no ha estat revisada.
- El programador té com a mínim un grau d'habilitat igual que el grau de dificultat de la tasca.

Efecte

- La tasca està assignada.
- El programador p realitza la tasca t.
- S'incrementa el temps total amb el temps que ha portat completar la tasca.

- **revisar**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador revisor no és el mateix que ha fet la tasca original.
  - El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

  Efecte

  - La tasca ha estat revisada.
  - La tasca t ha estat revisada pel programador p.
  - S'incrementa el temps total amb el temps que ha portat completar la revisió.

- **assignarDur**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

- La tasca no ha estat assignada.
- La tasca no ha estat revisada.
- El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

Efecte

- La tasca ha estat assignada.
- La tasca t ha estat assignada al programador p.
- S'incrementa el temps total amb el temps que ha portat completar la tasca més dues hores addicionals per no tenir l'habilitat suficient que requeria la tasca.

5.2 Problema

5.2.1 Objectes

- Un conjunt de programadors.
- Un conjunt de tasques.

5.2.2 Estat inicial

- Posem com a disponibles a tots els programadors.
- Posem com a no assignades a totes les tasques.
- Posem com a no revisades a totes les tasques.
- Assignem el grau d'habilitat de cada programador.
- Assignem la qualitat de cada programador.
- Assignem el grau de dificultat de cada tasca.
- Assignem el temps que necessita tasca per a completar-se.
- Inicialitzem el temps total a 0.

5.2.3 Estat final

Com a estat final definirem l'objectiu de què totes les tasques estiguin revisades. En aquest cas afegirem una mètrica, minimitzar el valor de tempsTotal.

5.3 Jocs de prova per a l'extensió 2

5.3.1 Joc de Prova 1: Prova factible

Input:

Programadors	p1	p2	p3
Habilitat	1	2	3
Qualitat	1	1	1
nTasques	0	0	0

Tasques	t1	t2	t3
Dificultat	1	2	3
Temps	3	3	3

Output:

```
step 0: ASIGNAR P3 T3
     1: ASIGNAR P2 T2
     2: ASIGNAR P1 T1
     3: REVISAR P2 T1
     4: REVISAR P1 T2
     5: REVISAR P1 T3
```

Observacions:

5.3.2 Joc de Prova 2: Prova no factible

Input:

Programadors	p1	p2	p3
Habilitat	1	1	1
Qualitat	1	1	1
nTasques	0	0	0

Tasques	t1	t2	t3
Dificultat	3	3	3
Temps	3	3	3

Output:

```
best first search space empty! problem proven unsolvable
```

Observacions:
Per aquest experiment, hem decidit utilitzar un input molt semblat al que hem utilitzat en l

6 Extensió 3

Tot el que es demana en l'extensió 2, però ara ens interessa poder tenir el màxim de persones treballant perquè es puguin fer més coses en paral·lel, de forma que volem limitar el número de tasques que podem assignar a una persona a 2.

6.1 Domini

6.1.1 Variables

- **Programador.** S'utilitzarà per a les variables que corresponen al conjunt de programadors.
- **Tasca.** S'utilitzarà per a les variables que corresponen al conjunt de tasques.

6.1.2 Funcions

- **(habilitat ?p - programador)**. Assigna un grau d'habilitat a un programador.
- **(calitat ?p - programador)**. Assigna un grau de qualitat a un programador.
- **(nTasques ?p - programador)**. Assigna un nombre de tasques a un programador.
- **(dificultat ?t - tasca)**. Assigna un grau de dificultat a una tasca.
- **(temps ?t - tasca)**. Assigna un temps de duració a una tasca.
- **(tempsTotal)**. És el temps total de compliment de totes les tasques assignades amb les revisions.

6.1.3 Predicats

- **(assignada ?t - tasca)**. Comprova si una tasca ha estat assignada.
- **(revisada ?t - tasca)**. Comprova si una tasca ha estat revisada.
- **(disponible ?p - programador)**. Comprova si programador està disponible.
- **(assignada_a ?t - tasca ?p - programador)**. Assigna la tasca t al programador p.
- **(revisat_per ?t - tasca ?p - programador)**. Assigna la revisió de la tasca t al programador p.

6.1.4 Accions

- **assignar**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca no ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador té com a mínim l'habilitat igual que la dificultat de la tasca.

  Efecte

  - La tasca està assignada.
  - El programador p realitza la tasca t.
  - S'incrementa el temps total amb el temps que ha portat completar la tasca.
  - S'incrementa el nombre de tasques d'aquell programador en una unitat.

- **revisar**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador revisor no és el mateix que ha fet la tasca original.
  - El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

  Efecte

  - La tasca ha estat revisada.
  - La tasca t ha estat revisada pel programador p.

- S'incrementa el temps total amb el temps que ha portat completar la revisió.

- **assignarDur**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca no ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

  Efecte

  - La tasca ha estat assignada.
  - La tasca t ha estat assignada al programador p.
  - S'incrementa el temps total amb el temps que ha portat completar la tasca més dues hores addicionals per no tenir l'habilitat suficient que requeria la tasca.
  - S'incrementa el nombre de tasques d'aquell programador en una unitat.

6.2 Problema

6.2.1 Objectes

- Un conjunt de programadors.
- Un conjunt de tasques.

6.2.2 Estat inicial

- Posem com a disponibles a tots els programadors.
- Posem com a no assignades a totes les tasques.
- Posem com a no revisades a totes les tasques.
- Assignem el grau d'habilitat de cada programador.
- Assignem la qualitat de cada programador.
- Inicialitzem el nombre de tasques fetes per cada programador a 0.
- Assignem el grau de dificultat de cada tasca.

- Assignem el temps que necessita tasca per a completar-se.
- Inicialitzem el temps total a 0.

6.2.3 Estat final

Com a estat final definirem l'objectiu de que totes les tasques estiguin revisades i es compleixi que cada programador faci com a màxim 2 tasques. Seguirem fent servir la mètrica per minimitzar el tempsTotal.

6.3 Jocs de prova per a l'extenisió 3

6.3.1 Joc de Prova 1: Prova factible

En aquest cas veurem un joc de prova que es realitza correctament, complint els objectius de l'extensió.

Input:

Programadors	p1	p2	p3	p4	p5
Habilitat	2	2	1	2	1
Qualitat	1	1	1	1	1
nTasques	0	0	0	0	0

Tasques	t1	t2	t3	t4
Dificultat	1	1	1	2
Temps	1	3	2	4

Ouput:

```
step 0: ASIGNAR P1 T4
    1: ASIGNAR P1 T2
    2: ASIGNAR P2 T3
    3: REVISAR P3 T3
    4: REVISAR P2 T2
    5: REVISAR P3 T4
    6: ASIGNAR P4 T1
    7: REVISAR P5 T1
```

Observacions:
En aquest cas hem vist com es compleixen els objectius de l'extensió, que era limitar a cada programador a que realitzés 2 tasques.

6.3.2 Joc de Prova 2: Prova no factible

Input:

Programadors	p1	p2	p3	p4	p5
Habilitat	1	1	1	2	1
Qualitat	1	1	1	1	1
nTasques	0	0	0	0	0

Tasques	t1	t2	t3	t4	t5	t6	t7	t8	t9
Dificultat	2	1	2	2	2	2	2	1	1
Temps	1	1	2	3	4	3	1	4	2

Ouput:

```
advancing to distance:  18
                        17
                        16
                        15
                        14
                        13
                        12
                        11
                        10
                        9
                        8^C
```

Observacions:

Podem observar que la execució es para (o avança molt lentament) quan queden 8 passos per acabar, això és degut a que hi han masses tasques com per a que els programadors només facin com a màxim dues cadascun.

L'execució s'atura exactament quan porta 10 assignacions de tasques, les màximes possibles amb 5 programadors.

7 Extensió 4

Tot el que es demana en l'extensió 3, però ara, encara que volem tenir més persones perquè es facin coses en paral·lel, també volem limitar-ho. A més de mantenir el límit en el nombre de tasques, volem fer una versió en la qual s'optimitzi la suma ponderada entre el nombre de persones que estan fent tasques i el temps total que es tarden a resoldre-les. **Quina és la ponderació més adequada?**

7.1 Domini

7.1.1 Variables

- **Programador.** S'utilitzarà per a les variables que corresponen al conjunt de programadors.
- **Tasca.** S'utilitzarà per a les variables que corresponen al conjunt de tasques.

7.1.2 Funcions

- **(habilitat ?p - programador)**. Assigna un grau d'habilitat a un programador.
- **(calitat ?p - programador)**. Assigna un grau de qualitat a un programador.
- **(nTasques ?p - programador)**. Assigna un número de tasques a un programador.
- **(dificultat ?t - tasca)**. Assigna un grau de dificultat a una tasca.
- **(temps ?t - tasca)**. Assigna un temps de duració a una tasca.
- **(progEnUs)**. Programadors en ús.
- **(tempsTotal)**. És el temps total de compliment de totes les tasques assignades amb les revisions.

7.1.3 Predicats

- **(assignada ?t - tasca)**. Comprova si una tasca ha estat assignada.
- **(revisada ?t - tasca)**. Comprova si una tasca ha estat revisada.
- **(disponible ?p - programador)**. Comprova si programador està disponible.
- **(assignada_a ?t - tasca ?p - programador)**. Assigna la tasca t al programador p.
- **(revisat_per ?t - tasca ?p - programador)**. Assigna la revisió de la tasca t al programador p.

7.1.4 Accions

- **assignar**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca no ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador té com a mínim la habilitat igual que la dificultat de la tasca.

  Efecte

  - La tasca està assignada.
  - El programador p realitza la tasca t.
  - S'incrementa el temps total amb el temps que ha portat completar la tasca.
  - Si es tracta de la primera tasca que fa aquell programador, incrementem en una unitat el nombre de programadors en ús.
  - S'incrementa el nombre de tasques d'aquell programador en una unitat.

- **revisar**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador revisor no és el mateix que ha fet la tasca original.
  - El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

  Efecte

  - La tasca ha estat revisada.

- La tasca t ha estat revisada per el programador p.
- S'incrementa el temps total amb el temps que ha portat completar la revisió.

- **assignarDur**

  Paràmetres

  - ?p - programador
  - ?t - tasca

  Precondició

  - La tasca no ha estat assignada.
  - La tasca no ha estat revisada.
  - El programador té un grau d'habilitat com a mínim una unitat per sota que el grau de dificultat de la tasca.

  Efecte

  - La tasca ha estat assignada.
  - La tasca t ha estat assignada al programador p.
  - S'incrementa el temps total amb el temps que ha portat completar la tasca més dues hores addicionals per no tenir l'habilitat suficient que requeria la tasca.
  - Si es tracta de la primera tasca que fa aquell programador, incrementem en una unitat el nombre de programadors en ús.
  - S'incrementa el nombre de tasques d'aquell programador en una unitat.

7.2 Problema

7.2.1 Objectes

- Un conjunt de programadors.
- Un conjunt de tasques.

7.2.2 Estat inicial

- Posem com a disponibles a tots els programadors.
- Posem com a no assignades a totes les tasques.
- Posem com a no revisades a totes les tasques.
- Assignem el grau d'habilitat de cada programador.

- Assignem la qualitat de cada programador.
- Inicialitzem el nombre de tasques fetes per cada programador a 0.
- Assignem el grau de dificultat de cada tasca.
- Assignem el temps que necessita tasca per a completar-se.
- Inicialitzem el temps total a 0.

7.2.3 Estat final

Com a estat final definirem l'objectiu de què totes les tasques estiguin revisades i es compleixi que cada programador faci com a màxim 2 tasques. Continuarem fent servir la mètrica per minimitzar el tempsTotal.

7.3 Jocs de prova per a l'extensió 4

7.3.1 Joc de Prova 1: Prova factible

Input:

Programadors	p1	p2	p3	p4	p5
Habilitat	2	2	1	2	1
Qualitat	1	1	1	1	1
nTasques	0	0	0	0	0

Tasques	t1	t2	t3	t4
Dificultat	1	1	1	2
Temps	1	3	2	4

Output:

```
step 0: ASIGNAR P1 T4
    1: ASIGNAR P1 T2
    2: ASIGNAR P2 T3
    3: REVISAR P2 T2
    4: REVISAR P5 T4
    5: ASIGNAR P5 T1
    6: REVISAR P3 T1
    7: REVISAR P3 T3
```

Observacions:

7.3.2 Joc de Prova 2: Prova no factible

Input:

Programadors	p1	p2	p3	p4	p5
Habilitat	1	1	1	2	1
Qualitat	1	1	1	1	1
nTasques	0	0	0	0	0

Tasques	t1	t2	t3	t4	t5	t6	t7	t8	t9
Dificultat	2	1	2	2	2	2	2	1	1
Temps	1	1	2	3	4	3	1	4	2

Output:

```
advancing to distance:  18
                        17
                        16
                        15
                        14
                        13
                        12
                        11
                        10
                        9
                        8^C
```

Observacions:
Exactament com a la extensió 3, el programa s'atura quan porta 10 assignacions de tasques ja que són les màximes possibles amb 5 programadors.

8 Conclusió

Realitzant aquesta pràctica hem pogut veure com funciona el llenguatge de programació declaratiu PDDL. Hem vist la seva potència i com no necessitàvem preocupar-nos del funcionament intern que feien els algoritmes per arribar a una solució, ja que això ho feia de forma interna. Només li havíem d'especificar el domini i el problema a resoldre.

Si haguéssim intentat fer aquesta pràctica en qualsevol altre llenguatge, com per exemple un llenguatge imperatiu, hauria sigut una tasca que ens hagués comportat molta més feina i temps invertit.

Hem descobert una nova eina per resoldre problemes de planificació i hem pogut veure aplicat allò que se'ns ha explicat a les classes de teoria.

9 Annex 1: Generador de jocs de prova

Hem creat un generador automàtic de jocs de prova per facilitar l'experimentació. El nostre generador l'hem escrit en C++. Aquest programa genera 30 fitxers d'entrada amb un nombre aleatori de programadors i tasques (dins d'un rang), que va augmentant en blocs de 3 fins a arribar a 18 programadors.

Els jocs de prova es generen amb nom "ProblemX_Y_Z"on X és el número mínim de programadors, Y és el número màxim de programadors (la prova es fa amb un valor aleatori dins d'aquest rang) i Z és el número de prova que es fa amb aquests valors.

El número de tasques hem considerat que també fos aleatori dins d'un rang assequible per fer la prova, concretament com un número entre el mínim de programadors d'aquella execució i el doble del màxim de programadors de l'execució. Després de fer experiments, hem arribat a que aquest és el rang més adient i estable.

9.1 Detalls tècnics

9.1.1 Funció randInt(min, max)

Aquesta funció retorna un valor enter entre el màxim i el mínim (inclosos) dels paràmetres que se li passen, s'utilitza en tots els punts del generador que necessitin ser aleatoris.

Té un cost de O(1).

9.1.2 Funció genera_sets(min, max)

Aquesta funció s'encarrega de generar les estructures de dades, que en aquest cas són dos conjunts d'Strings. Primer de tot es netegen (ja que es generen contínuament durant la execució) i després es van inserint elements fins a arribar a la quantitat aleatòria definida dins del rang.

Té un cost de O(P+T) on P és el número aleatori de programadors i T el número aleatori de tasques.

9.1.3 Funció crea_arxiu(outputFile)

Aquesta funció crea un arxiu i l'omple amb el format correcte per a la entrada del programa "Dominio.pddl".

Escriu la capçalera, les instàncies dels objectes, els predicats adients amb les estructures de dades prèviament creades per "genera_sets(min, max)", el goal i la mètrica (en cas d'haver).

Té un cost asimptòtic de O(2*P + 2*T) on P és el número aleatori de programadors i T el número aleatori de tasques.

9.1.4 Main

El Main del generador és el que determina la llavor aleatòria per a la generació de números i després crea tots els arxius d'entrada del problema, fent 6 execucions per cada grup de 3 programadors (6 execucions amb 3 a 6 programadors, 6 execucions amb 6 a 9 programadors, 6 execucions amb 9 a 12 programadors...) i les tasques que això comporta.

En total es crida a la funció crea_arxiu i genera_sets 30 cops, per tant el cost total del programa és 30 * O(2*P + 2*T), que asimptòticament segueix sent O(P + T) on P és el número aleatori de programadors i T el número aleatori de tasques.

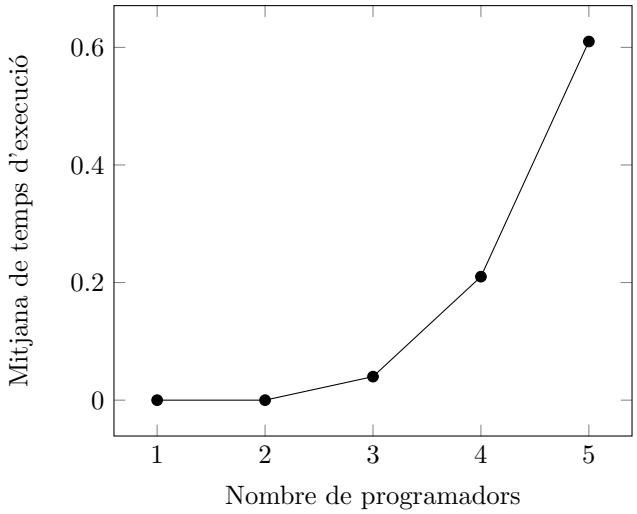
10 Annex 2: Experimentació de l'extensió 2

Hem utilitzat el generador aleatori de jocs de proves per obtenir problemes de tamany creixent i experimentar com evoluciona el temps de resolució en l'extensió 2 a mesura que s'augmenta el nombre de tasques i programadors.

A continuació mostrem una taula i un gràfic amb els valors obtinguts amb diferents nombres de programadors i tasques. En el valor de les mitjanes s'ha de tenir en compte que hem eliminat els valors més alts de cada execució.

Execucions	1	2	3	4	5	6	Average
(1) De 3 - 6 Programadors	0.00	0.00	0.00	0.00	0.00	0.00	0.00
(2) De 6 - 9 Programadors	0.00	0.00	0.00	0.01	0.00	0.00	0.00
(3) De 9 - 12 Programadors	0.03	0.01	0.05	0.05	0.04	0.03	0.04
(4) De 12 - 15 Programadors	0.11	0.20	0.10	0.09	0.60	0.14	0.21
(5) De 15 - 18 Programadors	0.60	0.40	1.18	0.94	0.22	0.34	0.61

Gràfic del temps mitjà de les diferents execucions de l'extensió 2



A continuació es pot veure la taula que representa el nombre de tasques i programadors que hi han en cada execució del generador de jocs de prova

Execucions	Programadors	Tasques	T/P
De 3 - 6 Programadors Execució 1	6	6	1.00
De 3 - 6 Programadors Execució 2	5	7	1.40
De 3 - 6 Programadors Execució 3	3	4	1.33
De 3 - 6 Programadors Execució 4	5	9	1.80
De 3 - 6 Programadors Execució 5	6	8	1.33
De 3 - 6 Programadors Execució 6	4	8	2.00
De 6 - 9 Programadors Execució 1	7	12	1.71
De 6 - 9 Programadors Execució 2	6	11	1.83
De 6 - 9 Programadors Execució 3	6	9	1.50
De 6 - 9 Programadors Execució 4	9	11	1.22
De 6 - 9 Programadors Execució 5	7	10	1.43
De 6 - 9 Programadors Execució 6	8	9	1.13
De 9 - 12 Programadors Execució 1	9	17	1.89
De 9 - 12 Programadors Execució 2	9	13	1.44
De 9 - 12 Programadors Execució 3	12	17	1.42
De 9 - 12 Programadors Execució 4	10	18	1.80
De 9 - 12 Programadors Execució 5	11	15	1.36
De 9 - 12 Programadors Execució 6	10	14	1.40
De 12 - 15 Programadors Execució 1	15	18	1.20
De 12 - 15 Programadors Execució 2	14	24	1.71
De 12 - 15 Programadors Execució 3	15	19	1.27
De 12 - 15 Programadors Execució 4	12	20	1.67
De 12 - 15 Programadors Execució 5	15	30	2.00
De 12 - 15 Programadors Execució 6	15	20	1.33
De 15 - 18 Programadors Execució 1	17	27	1.59
De 15 - 18 Programadors Execució 2	15	27	1.80
De 15 - 18 Programadors Execució 3	18	32	1.78
De 15 - 18 Programadors Execució 4	17	32	1.88
De 15 - 18 Programadors Execució 5	15	22	1.47
De 15 - 18 Programadors Execució 6	15	25	1.67

10.1 Observacions

Podem veure que el creixement del temps d'execució és gairebé exponencial a mesura que el nombre de programadors i tasques augmenta, a més el nombre de tasques promig assignat a programadors no varia massa (sempre està entre 1 i 2), el qual és lògic, ja que hem establert nosaltres aquest rang de tasques.

El creixement exponencial del temps significa que aquest problema és un problema difícil de solucionar per a casos grans i que és necessària la utilització d'una Intel·ligència Artificial per aconseguir trobar una resposta més fàcilment. Un algoritme normal (per exemple de backtracking) podria servir, però seria molt més costós.

11 Annex 3: Inputs i Outputs dels Jocs de Proves

11.1 Jocs de prova per al nivell bàsic

11.1.1 Joc de Prova 1: Prova factible

Input:

```
(define (problem planificator)
(:domain planificator3000)
(:objects
p1 p2 p3 p4 p5 p6 - programador
h1 h2 h3   - habilitat
t1 t2 t3 t4 t5 - tasca
d1 d2 d3 - dificultat
)
(:init
(te p1 h1)
(te p2 h2)
(te p3 h1)
(te p4 h3)
(te p5 h2)
(te p6 h1)

(disponible p1)
(disponible p2)
(disponible p3)
(disponible p4)
(disponible p5)
(disponible p6)

(es t1 d1)
(es t2 d2)
(es t3 d3)
(es t4 d2)
(es t5 d3)
(not (asignada t1))
(not (asignada t2))
(not (asignada t3))
(not (asignada t4))
(not (asignada t5))

(assumible d1 h1)
(assumible d1 h2)
(assumible d1 h3)
```

```
(assumible d2 h1)
(assumible d2 h2)
(assumible d2 h3)
(assumible d3 h2)
(assumible d3 h3)

)
(:goal
(forall (?t - tasca) (asignada ?t))
)
)
```

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
 ... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
 ... done.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

advancing to distance:    3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: ASIGNAR P1 H1 T2 D2
        1: ASIGNAR P2 H1 T1 D2
        2: ASIGNAR P3 H3 T3 D3


time spent: 0.00 seconds instantiating 7 easy, 0 hard action templates
```

```
0.00 seconds reachability analysis, yielding 16 facts and 7 actions
0.00 seconds creating final representation with 16 relevant facts,
0 relevant fluents
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 12 states, to a max depth of 0
0.00 seconds total time
```

11.1.2 Joc de Prova 2: Prova no factible

Input:

```
(define (problem planificator)
(:domain planificator3000)
(:objects
p1 p2 p3 - programador
h1 h2 h3 - habilitat
t1 t2 t3 - tasca
d1 d2 d3 - dificultat
)
(:init
(te p1 h1)
(te p2 h1)
(te p3 h1)

(disponible p1)
(disponible p2)
(disponible p3)

(es t1 d3)
(es t2 d3)
(es t3 d3)
(not (asignada t1))
(not (asignada t2))
(not (asignada t3))

(assumible d1 h1)
(assumible d1 h2)
(assumible d1 h3)
(assumible d2 h1)
(assumible d2 h2)
(assumible d2 h3)
(assumible d3 h2)
(assumible d3 h3)
```

```
)
(:goal
(forall (?t - tasca) (asignada ?t))
)
)
```

Ouput:

11.2 Jocs de prova per a l'extensió 1

11.2.1 Joc de Prova 1: Prova factible

Input:

```
(define (problem planificator)
 (:domain planificator3000)
 (:objects
 p1 p2 p3  - programador
 t1 t2 t3  - tasca
 )
 (:init

 (disponible p1)
 (disponible p2)
 (disponible p3)

 (not (asignada t1))
 (not (asignada t2))
 (not (asignada t3))

 (not (revisada t1))
 (not (revisada t2))
 (not (revisada t3))

 (= (habilitat p1) 1)
     (= (habilitat p2) 2)
     (= (habilitat p3) 3)

     (= (dificultat t1) 1)
     (= (dificultat t2) 2)
     (= (dificultat t3) 3)

 )
 (:goal
 (forall (?t - tasca) (revisada ?t))
 )
)
```

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
 ... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
```

```
 ... done.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

advancing to distance:    6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: ASIGNAR P1 T1
        1: REVISAR P3 T1
        2: ASIGNAR P2 T2
        3: REVISAR P3 T2
        4: ASIGNAR P3 T3
        5: REVISAR P2 T3


time spent: 0.00 seconds instantiating 18 easy, 0 hard action templates
            0.00 seconds reachability analysis, yielding 33 facts and 16 actions
            0.00 seconds creating final representation with 33 relevant facts,
            0 relevant fluents
            0.00 seconds computing LNF
            0.00 seconds building connectivity graph
            0.00 seconds searching, evaluating 22 states, to a max depth of 0
            0.00 seconds total time
```

11.2.2 Joc de Prova 2: Prova no factible

Input:

```
(define (problem planificator)
 (:domain planificator3000)
 (:objects
 p1 p2 p3  - programador
 t1 t2 t3  - tasca
```

```
    )
    (:init

    (disponible p1)
    (disponible p2)
    (disponible p3)

    (not (asignada t1))
    (not (asignada t2))
    (not (asignada t3))

    (not (revisada t1))
    (not (revisada t2))
    (not (revisada t3))

    (= (habilitat p1) 1)
           (= (habilitat p2) 1)
           (= (habilitat p3) 1)

           (= (dificultat t1) 3)
           (= (dificultat t2) 3)
           (= (dificultat t3) 3)

    )
    (:goal
    (forall (?t - tasca) (revisada ?t))
    )
   )
```

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
... done.

no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

best first search space empty! problem proven unsolvable.
```

```
time spent: 0.00 seconds instantiating 18 easy, 0 hard action templates
            0.00 seconds reachability analysis, yielding 33 facts and 0 actions
            0.00 seconds creating final representation with 33 relevant facts, 0
            relevant fluents
            0.00 seconds computing LNF
            0.00 seconds building connectivity graph
            0.00 seconds searching, evaluating 1 states, to a max depth of 0
            0.00 seconds total time
```

11.3 Jocs de prova per a l'extensió 2

11.3.1 Joc de Prova 1: Prova factible

Input:

```
(define (problem planificator)
 (:domain planificator3000)
 (:objects
 p1 p2 p3  - programador
 t1 t2 t3  - tasca
 )
 (:init

 (disponible p1)
 (disponible p2)
 (disponible p3)

        (= (calitat p1) 1)
        (= (calitat p2) 1)
        (= (calitat p3) 1)

 (not (asignada t1))
 (not (asignada t2))
 (not (asignada t3))

 (not (revisada t1))
 (not (revisada t2))
 (not (revisada t3))

 (= (habilitat p1) 1)
     (= (habilitat p2) 2)
     (= (habilitat p3) 3)

     (= (dificultat t1) 1)
     (= (dificultat t2) 2)
     (= (dificultat t3) 3)

     (= (temps t1) 3)
        (= (temps t2) 3)
        (= (temps t3) 3)

        (= (tempsTotal) 0)

 )
 (:goal
```

```
 (forall (?t - tasca) (revisada ?t))
 )
 (:metric minimize (tempsTotal))
)
```

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
 ... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
 ... done.


metric established (normalized to minimize):
    ((1.00*[RF0](TEMPSTOTAL)) - () + 0.00)

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((1.00*[RF0](TEMPSTOTAL)) - () + 0.00)

advancing to distance:    6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: ASIGNAR P3 T3
        1: ASIGNAR P2 T2
        2: ASIGNAR P1 T1
        3: REVISAR P2 T1
        4: REVISAR P1 T2
        5: REVISAR P1 T3


time spent: 0.00 seconds instantiating 27 easy, 0 hard action templates
            0.00 seconds reachability analysis, yielding 39 facts and 15 actions
            0.00 seconds creating final representation with 39 relevant facts,
            1 relevant fluents
```

```
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 33 states, to a max depth of 0
0.00 seconds total time
```

11.3.2 Joc de Prova 2: Prova no factible

Input:

```
(define (problem planificator)
(:domain planificator3000)
(:objects
p1 p2 p3  - programador
t1 t2 t3  - tasca
)
(:init

(disponible p1)
(disponible p2)
(disponible p3)

        (= (calitat p1) 1)
        (= (calitat p2) 1)
        (= (calitat p3) 1)

(not (asignada t1))
(not (asignada t2))
(not (asignada t3))

(not (revisada t1))
(not (revisada t2))
(not (revisada t3))

(= (habilitat p1) 1)
     (= (habilitat p2) 1)
     (= (habilitat p3) 1)

     (= (dificultat t1) 3)
     (= (dificultat t2) 3)
     (= (dificultat t3) 3)

        (= (temps t1) 3)
        (= (temps t2) 3)
        (= (temps t3) 3)
```

```
        (= (tempsTotal) 0)

)
(:goal
(forall (?t - tasca) (revisada ?t))
)
    (:metric minimize (tempsTotal))
)
```

Output:

```
    ff: parsing domain file
    domain 'PLANIFICATOR3000' defined
     ... done.
    ff: parsing problem file
    problem 'PLANIFICATOR' defined
     ... done.


    metric established (normalized to minimize): ((1.00*[RF0](TEMPSTOTAL)) - () + 0.00)

    checking for cyclic := effects --- OK.

    ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
        metric is ((1.00*[RF0](TEMPSTOTAL)) - () + 0.00)

    best first search space empty! problem proven unsolvable.



    time spent: 0.00 seconds instantiating 27 easy, 0 hard action templates
                0.00 seconds reachability analysis, yielding 39 facts and 9 actions
                0.00 seconds creating final representation with 39 relevant facts,
                1 relevant fluents
                0.00 seconds computing LNF
                0.00 seconds building connectivity graph
                0.00 seconds searching, evaluating 1 states, to a max depth of 0
                0.00 seconds total time
```

11.4 Jocs de prova per a l'extensió 3

11.4.1 Joc de Prova 1: Prova factible

Input:

```
(define (problem planificator)
    (:domain planificator3000)

    (:objects
        p1 p2 p3 p4 p5 - programador
        t1 t2 t3 t4 - tasca
    )

    (:init

        (disponible p1)
        (= (habilitat p1) 2)
        (= (calitat p1) 1)
        (= (nTasques p1) 0)

        (disponible p2)
        (= (habilitat p2) 2)
        (= (calitat p2) 1)
        (= (nTasques p2) 0)

        (disponible p3)
        (= (habilitat p3) 1)
        (= (calitat p3) 1)
        (= (nTasques p3) 0)

        (disponible p4)
        (= (habilitat p4) 2)
        (= (calitat p4) 1)
        (= (nTasques p4) 0)

        (disponible p5)
        (= (habilitat p5) 1)
        (= (calitat p5) 1)
        (= (nTasques p5) 0)

        (not (asignada t1))
        (not (revisada t1))
        (= (dificultat t1) 1)
        (= (temps t1) 1)
```

```
        (not (asignada t2))
        (not (revisada t2))
        (= (dificultat t2) 1)
        (= (temps t2) 3)

        (not (asignada t3))
        (not (revisada t3))
        (= (dificultat t3) 1)
        (= (temps t3) 2)

        (not (asignada t4))
        (not (revisada t4))
        (= (dificultat t4) 2)
        (= (temps t4) 4)


        (= (tempsTotal) 0)

    )

    (:goal (and
           (forall(?t - tasca) (revisada ?t))
           (forall (?p - programador) (<= (nTasques ?p) 2))
           )
    )

    (:metric minimize (tempsTotal))

)
```

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
 ... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
 ... done.


metric established (normalized to minimize): ((1.00*[RF0](TEMPSTOTAL)) - () + 0.00)

checking for cyclic := effects --- OK.
```

```
ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((1.00*[RF0](TEMPSTOTAL)) - () + 0.00)

advancing to distance:    8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: ASIGNAR P1 T4
        1: ASIGNAR P1 T2
        2: ASIGNAR P2 T3
        3: REVISAR P3 T3
        4: REVISAR P2 T2
        5: REVISAR P3 T4
        6: ASIGNAR P4 T1
        7: REVISAR P5 T1


time spent: 0.00 seconds instantiating 60 easy, 0 hard action templates
            0.00 seconds reachability analysis, yielding 76 facts and 38 actions
            0.00 seconds creating final representation with 76 relevant facts,
            11 relevant fluents
            0.00 seconds computing LNF
            0.00 seconds building connectivity graph
            0.00 seconds searching, evaluating 104 states, to a max depth of 0
            0.00 seconds total time
```

11.4.2 Joc de Prova 2: Prova no factible

Input:

```
(define (problem planificator)
    (:domain planificator3000)

    (:objects
        p1 p2 p3 p4 p5 - programador
        t1 t2 t3 t4 t5 t6 t7 t8 t9 - tasca
```

```
)

(:init

    (disponible p1)
    (= (habilitat p1) 1)
    (= (calitat p1) 1)
    (= (nTasques p1) 0)

    (disponible p2)
    (= (habilitat p2) 1)
    (= (calitat p2) 1)
    (= (nTasques p2) 0)

    (disponible p3)
    (= (habilitat p3) 1)
    (= (calitat p3) 1)
    (= (nTasques p3) 0)

    (disponible p4)
    (= (habilitat p4) 2)
    (= (calitat p4) 1)
    (= (nTasques p4) 0)

    (disponible p5)
    (= (habilitat p5) 1)
    (= (calitat p5) 1)
    (= (nTasques p5) 0)

    (not (asignada t1))
    (not (revisada t1))
    (= (dificultat t1) 2)
    (= (temps t1) 1)

    (not (asignada t2))
    (not (revisada t2))
    (= (dificultat t2) 1)
    (= (temps t2) 1)

    (not (asignada t3))
    (not (revisada t3))
    (= (dificultat t3) 2)
    (= (temps t3) 2)

    (not (asignada t4))
    (not (revisada t4))
```

```
        (= (dificultat t4) 2)
        (= (temps t4) 3)

        (not (asignada t5))
        (not (revisada t5))
        (= (dificultat t5) 2)
        (= (temps t5) 4)

        (not (asignada t6))
        (not (revisada t6))
        (= (dificultat t6) 2)
        (= (temps t6) 3)

        (not (asignada t7))
        (not (revisada t7))
        (= (dificultat t7) 2)
        (= (temps t7) 1)

        (not (asignada t8))
        (not (revisada t8))
        (= (dificultat t8) 1)
        (= (temps t8) 4)

        (not (asignada t9))
        (not (revisada t9))
        (= (dificultat t9) 1)
        (= (temps t9) 2)


        (= (tempsTotal) 0)

    )

    (:goal (and
           (forall(?t - tasca) (revisada ?t))
           (forall (?p - programador) (<= (nTasques ?p) 2))
           )
    )

    (:metric minimize (tempsTotal))

)
```

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
 ... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
 ... done.


metric established (normalized to minimize): ((1.00*[RF0](TEMPSTOTAL)) - () + 0.00)

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((1.00*[RF0](TEMPSTOTAL)) - () + 0.00)

advancing to distance:   18
                         17
                         16
                         15
                         14
                         13
                         12
                         11
                         10
                          9
                          8^C
```

11.5 Jocs de prova per a l'extensió 4

11.5.1 Joc de Prova 1: Prova factible

Input:

```
(define (problem planificator)
    (:domain planificator3000)

    (:objects
        p1 p2 p3 p4 p5 - programador
        t1 t2 t3 t4 - tasca
    )

    (:init

        (disponible p1)
        (= (habilitat p1) 2)
        (= (calitat p1) 1)
        (= (nTasques p1) 0)

        (disponible p2)
        (= (habilitat p2) 2)
        (= (calitat p2) 1)
        (= (nTasques p2) 0)

        (disponible p3)
        (= (habilitat p3) 1)
        (= (calitat p3) 1)
        (= (nTasques p3) 0)

        (disponible p4)
        (= (habilitat p4) 2)
        (= (calitat p4) 1)
        (= (nTasques p4) 0)

        (disponible p5)
        (= (habilitat p5) 1)
        (= (calitat p5) 1)
        (= (nTasques p5) 0)

        (not (asignada t1))
        (not (revisada t1))
        (= (dificultat t1) 1)
        (= (temps t1) 1)
```

```
        (not (asignada t2))
        (not (revisada t2))
        (= (dificultat t2) 1)
        (= (temps t2) 3)

        (not (asignada t3))
        (not (revisada t3))
        (= (dificultat t3) 1)
        (= (temps t3) 2)

        (not (asignada t4))
        (not (revisada t4))
        (= (dificultat t4) 2)
        (= (temps t4) 4)


        (= (progEnUs)   0)
        (= (tempsTotal) 0)

    )

    (:goal (and
           (forall(?t - tasca) (revisada ?t))
           (forall (?p - programador) (<= (nTasques ?p) 2))
           )
    )

    (:metric minimize (+ (* 0.1 (tempsTotal)) (* 0.9 (progEnUs))))

)
```

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
 ... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
 ... done.


metric established (normalized to minimize):
    ((0.10*[RF0](TEMPSTOTAL)0.90*[RF1](PROGENUS)) - () + 0.00)
```

```
task contains conditional effects. turning off state domination.



checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((0.10*[RF0](TEMPSTOTAL)0.90*[RF1](PROGENUS)) - () + 0.00)

advancing to distance:    8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: ASIGNAR P1 T4
        1: ASIGNAR P1 T2
        2: ASIGNAR P2 T3
        3: REVISAR P2 T2
        4: REVISAR P5 T4
        5: ASIGNAR P5 T1
        6: REVISAR P3 T1
        7: REVISAR P3 T3


time spent: 0.00 seconds instantiating 60 easy, 0 hard action templates
            0.00 seconds reachability analysis, yielding 76 facts and 38 actions
            0.00 seconds creating final representation with 76 relevant facts,
            12 relevant fluents
            0.00 seconds computing LNF
            0.00 seconds building connectivity graph
            0.01 seconds searching, evaluating 2240 states, to a max depth of 0
            0.01 seconds total time
```

11.5.2 Joc de Prova 2: Prova no factible

Input:

```
(define (problem planificator)
    (:domain planificator3000)

    (:objects
        p1 p2 p3 p4 p5 - programador
        t1 t2 t3 t4 t5 t6 t7 t8 t9 - tasca
    )

    (:init

        (disponible p1)
        (= (habilitat p1) 1)
        (= (calitat p1) 1)
        (= (nTasques p1) 0)

        (disponible p2)
        (= (habilitat p2) 1)
        (= (calitat p2) 1)
        (= (nTasques p2) 0)

        (disponible p3)
        (= (habilitat p3) 1)
        (= (calitat p3) 1)
        (= (nTasques p3) 0)

        (disponible p4)
        (= (habilitat p4) 2)
        (= (calitat p4) 1)
        (= (nTasques p4) 0)

        (disponible p5)
        (= (habilitat p5) 1)
        (= (calitat p5) 1)
        (= (nTasques p5) 0)

        (not (asignada t1))
        (not (revisada t1))
        (= (dificultat t1) 2)
        (= (temps t1) 1)

        (not (asignada t2))
        (not (revisada t2))
        (= (dificultat t2) 1)
        (= (temps t2) 1)
```

```
    (not (asignada t3))
    (not (revisada t3))
    (= (dificultat t3) 2)
    (= (temps t3) 2)

    (not (asignada t4))
    (not (revisada t4))
    (= (dificultat t4) 2)
    (= (temps t4) 3)

    (not (asignada t5))
    (not (revisada t5))
    (= (dificultat t5) 2)
    (= (temps t5) 4)

    (not (asignada t6))
    (not (revisada t6))
    (= (dificultat t6) 2)
    (= (temps t6) 3)

    (not (asignada t7))
    (not (revisada t7))
    (= (dificultat t7) 2)
    (= (temps t7) 1)

    (not (asignada t8))
    (not (revisada t8))
    (= (dificultat t8) 1)
    (= (temps t8) 4)

    (not (asignada t9))
    (not (revisada t9))
    (= (dificultat t9) 1)
    (= (temps t9) 2)

    (= (progEnUs)   0)
    (= (tempsTotal) 0)

)

(:goal (and
       (forall(?t - tasca) (revisada ?t))
       (forall (?p - programador) (<= (nTasques ?p) 2))
       )
)
```

```
    (:metric minimize (+ (* 0.1 (tempsTotal)) (* 0.9 (progEnUs))))

)
```

Output:

```
ff: parsing domain file
domain 'PLANIFICATOR3000' defined
 ... done.
ff: parsing problem file
problem 'PLANIFICATOR' defined
 ... done.


metric established (normalized to minimize):
    ((0.10*[RF0](TEMPSTOTAL)0.90*[RF1](PROGENUS)) - () + 0.00)

task contains conditional effects. turning off state domination.



checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((0.10*[RF0](TEMPSTOTAL)0.90*[RF1](PROGENUS)) - () + 0.00)

advancing to distance:   18
                         17
                         16
                         15
                         14
                         13
                         12
                         11
                         10
                          9
                          8^C
```