PRÀCTICA 3: PLANIFICACIÓ

David Pujol Perich, Oriol Aranda Llorens i Pol Garcia Recasens

Quadrimestre 2, 2019/2020

Índex

1.Introducció

Aquesta pràctica ha consistit en resoldre un problema de planificació usant les tècniques adquirides en aquesta assignatura de Intel·ligència Artificial. Aquest problema, doncs, consistirà en determinar quines tasques han de fer els diversos programadors amb els que comptem, tot tenint en compte una sèrie de restriccions que comentarem a l'apartat posterior dedicat al Domini.

2. El problema

El problema al que se'ns introdueix en aquesta pràctica compte amb una base que mantindrem, i tota una sèrie d'extensions del problema bàsic. Inicialment, es planteja un problema que consisteix en l'assignació de programadors a totes les tasques inicialitzades. Cada programador estarà definit per una qualitat i per una habilitat (que indicarà com de bon programador és). Aleshores, cadascuna de les tasques introduïdes inicialment també se'ls hi assignarà una complexitat de la tasca. Dit això, restringuim que un programador es pot assignar a una tasca sempre i quan la seva habilitat sigui com a molt inferior en 1 respecte la complexitat de la tasca.

La resta de versions tenen en compte una extensió del problema que consisteix en afegir les tasques de revisió. Aquestes tasques de revisió vindran associades a cadascuna de les tasques normals de les que acabem de parlar. Una tasca de revisió, tindrà les mateixes limitacions (l'habilitat del programador ha de ser com a molt inferior en 1 respecte la complexitat de la tasca), amb la restricció afegida que un programador no pot revisar una tasca que ha estat realitzada per ell mateix.

A partir d'aquí, cada extensió es centra sobretot en trobar solucions vàlides tot intentant minimitzar o maximitzar diferents paràmetres del problema per a evaluar el seu funcionament.

3.Domini

3.1 Introducció

Tal i com hem comentat anteriorment, aquest és clarament un problema de planificació, pel que farem servir el llenguatge Planning Domain Definition Language (pddl). Per a usar aquest llenguatge, ens caldrà dur a terme diverses tasques. Fonamentalment, haurem de implementar el domini a través de predicats, funcions i finalment les accions.

Abans de començar a procedir a explicar amb més detall cadascun d'aquests apartats que hem implementat del domini, voldríem destacar una decisió de disseny que hem pres. A l'hora d'implementar ens vam plantejar la necessitat o no de crear un objecte per a cadascuna de les tasques normals i crear també un objecte que representés les tasques de revisió. Tot i això, ens va semblar que conceptualment tenia molt més sentit crear un sol objecte tasca, i que a aquest objecte se li associés (a través de predicats) un programador per a la tasca normal, i un altre programador per a realitzar la revisió d'aquesta, però sempre relacionant-ho amb la mateixa tasca.

3.2 Funcions creades

A continuació introduirem molt breument quines són les funcions que hem utilitzat per a implementar la nostre pràctica. Tot i així, voldriem matisar que no tots ells es fan servir en totes les extensions. Tot i així en aquesta secció les introduirem totes.

Habilitat del programador (ability ?p - programmer)

Aquesta funció retorna la habilitat que s'ha definit per a un programador. Aquest valor no canviarà al llarg de tota la execució.

Qualitat d'un programador (quality ?p - programmer)

Aquesta funció retorna la qualitat d'un programador. Aquesta funció també mantindrà el mateix valor amb el que s'inicialitza inicialment en el programa.

Complexitat d'una tasca (complexity ?t - task)

Aquesta funció ens retorna la complexitat que hem definit per a una tasca concreta. Aquesta funció, com les anteriors, no variarà al llarg de la execució.

Duració de la tasca (duration ?t - task)

Aquesta funció ens retorna el valor de la duració d'una tasca. Aquest valor canviarà ja que en funció de la habilitat del programador a la que se li assigna, aquesta duració es veurà incrementada en 2 hores.

Duració de la revisió d'una tasca (reviewDuration ?t - task)

Aquesta funció mantindrà el valor de la duració que implica la revisió d'una tasca. Aquest valor no serà inicialitzat inicialment, sinó que al assignar un programador a la tasca normal, es calcularà la duració que implica que tingui la revisió d'aquesta.

Nombre de tasques assignades a un programador (numTask ?p - programmer)

Aquesta funció ens servirà de contador per tal de poder mantenir constància de quantes tasques ja se li han assignat a un programador (molt útil per a l'extensió que limita aquest valor a 2 com a molt).

Nombre total d'hores dedicades (totalHours)

Funció en el que anirem fent el comptatge global de quantes hores s'ha trigat en realitzar totes les tasques (normals i de revisió) que se'ns planteja en el problema. Molt últil per a l'extensió que intenta minimitzar aquest valor precisament.

Quants programadors estan treballant (workingProgrammers)

Contador que ens indicarà quin és el nombre de programadors que estan treballant a partir de les nostres assignacions. Que un programador treballi vé donat perquè se li hagi assignat com a mínim una tasca. Aquest comptador és necessari per a implementar la última de les extensions que implica una funció de maximització on un dels paràmetres és precisament aquests valors.

3.3 Justificació dels predicats

Ara anem a començar a explicar els diferents predicat que hem definit per al nostre problema, tot i que cal destacar que aquests predicats depenen fortament de l'extensió del programa del que estiguem parlant, ja que, per exemple, en la versió bàsica no hem hagut de tenir en compte cap tasca de revisió. Tot i així, en aquest apartat, els explicarem tots i els justificarem breument.

Assignació d'una tasca (taskAssigned ?t - task)

Aquest prodicat ens indica si una tasca ja se li ha assignat un programador o no.Per tant, si per exemple volguéssim comprovar en una precondició que una tasca no ha estat assignada ja, hauriem de fer el següent:

:precondition (not(taskAssigned ?t))

Aquesta predicat ens ha fet falta en totes les extensions que hem desenvolupat ja que en totes elles duem a terme assignacions de tasques. Per tant, sempre és crucial comprovar abans de fer qualsevol nova assignació d'una tasca, que aquesta no ha estat assignada ja, doncs altrament seria un resultat incoherent.

En extensions diferents de la versió inicial, a més a més, ens ha fet falta per un altre motiu diferent. Quan es dur a terme la assignació de la tasca de revisió, també és imprescindible assegurar-nos que ja hem assignat la seva tasca original. Això és necessari, perquè en el moment que assignem la tasca original, per exemple, sabem quin programador no se li podrà assignar a la tasca de revisió d'una tasca. Per tant assignar primer la tasca de revisió que la seva tasca original, amb el nostre plantejament, seria incorrecte.

Assignació de tasca de revisió (taskReviewAssigned ?t - task)

Aquest predicat dur a terme una funció molt semblant a la anterior, que serà de indicar si una tasca de revisió ja ha estat assignada. Tal i com hem comentat anteriorment, nosaltres hem decidit que no hi hagi dos objectes diferenciats task i taskReview, doncs ens ha semblat que tenia més sentit que tot girés al voltant de la mateixa tasca. Al crear un sol objecte tasca, per tant, és necessària crear dos predicats diferents que ens indiquin si aquest objecte tasca ja té un programador per la tasca original i ja té un programador per la tasca de revisió (objectiu en totes les extensions a partir de la 1).

Aleshores, aquest predicat no ha sigut necessari a la versió inicial del programa, doncs no tenim en compte cap tasca de revisió, però si en tota la resta. A la acció d'assignar un programador a una tasca de revisió, seguint el mateix raonament anterior, caldrà assegurar-nos que la tasca de revisió que escau no ha sigut encara assignada, i un cop l'assignem, assegurar que aquest predicat per aquella tasca és cert.

Programador fa una tasca (programmerDoesTask ?p - programmer ?t - task)

Arribats en aquest punt, ens feia falta un predicat que ens indiqués pròpiament l'assignació d'un programador a una tasca. Per tant, aquest predicat, indicaria que el programador ?p està assignat a la tasca original ?t. Aquest predicat, constarà en totes les extensions com a un efecte de la acció de assignar un programador a una tasca normal, doncs un cop hem comprovat que el programador és compatible amb una tasca i que la tasca encara no ha estat assignada, duem a terme pròpiament la assignació. Així fariem, per tant, l'assignació d'un programador.

:effect (programmerDoestTask ?p ?t)

Programador fa una tasca de revisió (programmerReviewsTask ?p - programmer ?t - task)

Tal i com ha passat abans, i a causa de que no hi ha pròpiament dos objectes tasca i tascaRevisió, hem hagut d'implementar aquesta diferenciació a través de predicats. Aquest predicat ens indicarà que un programador ?p ha estat assignat a la tasca de revisió associada a la tasca ?t.
Aquest predicat ha sigut necessari en totes les extensions que contemplen la necessitat d'assignar la tasca de revisió (totes menys la inicial), i tal hi com a passat abans, serà el efecte de la acció d'assignar un programador a una tasca de revisió. Per tant, en aquesta acció, un cop comprovat que el programador és compatible amb la tasca de revisió i que aquesta encara no ha estat assignada, assignarem aquest programador de la següent manera:

:effect (programmerReviewsTask ?p ?t)

Programador està treballant (programmerWorking ?p - programmer)

Finalment ens ha fet falta aquest predicat ja que en la extensió 4, ens cal maximitzar la mitjana ponderada de hores totals dedicades i del nombre de programadors que tenen alguna tasca assignada. Aleshores, per tant, ens ha fet falta mantenir una funció que ens indiqui el nombre de programadors que tenen alguna tasca assignada. Aquesta funció, s'anirà incrementant en 1 cada vegada que assignem un programador que encara no havia estat assignada a ninguna tasca. Tot i així, això tan sols és possible si mantenim una sèrie de predicats que ens indiquin que un programador ja ha estat assignat o no a una tasca com a mínim.

Per tant, aquest predicat ens estaria indicant que el programador ?p ja està treballant.

Un exemple del seu ús seria el següent (en el efecte d'una acció, en el que després de fer l'assignació del programador, mirem si ja estava treballant o no):

:effect (when (not(programmerWorking ?p)) (and (programmerWorking ?p) (increase workingProgrammers) 1)))

3.4 Els operadors

A continuació anem a parlar una mica sobre els diferents operadors que hem implementat per a aquesta pràctica, que en el nostre cas tan sols són 2. Voldríem destacar que hem usat tan sols un operadors per a la versió més bàsica de la pràctica, i tota la resta consten dels 2 mateixos operadors, tot i que cada versió té d'operadors sensiblement diferents que els d'altres extensions. En la majoria de casos, de fet, a mesura que anem avançant en l'extensió en

concret, els operadors incorporen tot el codi anterior i a més a més, n'afegeixen d'altres predicats i funcions.

Nivell bàsic (domain0.pddl)

Aquesta versió del domini compte només amb una sola acció que serà la d'assignar una tasca (assignTask). Aquest operador durà a terme una assignació d'un programador a una tasca.

Els paràmetres que rebrà aquesta acció serà un programador i una tasca.

Aleshores, establim una sèrie de precondicions que seran que:

1) La tasca encara no hagi estat assignada
2) La habilitat del programadors sigui com a molt inferior en 1 respecte la complexitat de la tasca.

Finalment, hem indicat els effectes que té aquesta acció d'assignar un programador a una tasca (en el cas de complir totes les precondicions), que són els següents:

1) La tasca passa a estar assignada.
2) El programador se li assigna aquesta tasca (fent servir el predicat programmerDoesTask)
3) En el cas que la habilitat del programador sigui inferior que la dificultat de la tasca, la duració de la tasca passa a ser 2 hores superior.

Extensió 1 (domain1.pddl)

Tal i com hem comentat breument en apartats anteriors, aquesta versió consistirà en la mateixa dinàmica que la anterior, amb la única diferència que ara sí que ens caldrà tenir en compte les tasques de revisió associades. Per tant, ens veiem obligats a afegir un nou operador a part de l'operador prèviament explicat a l'apartat anterior.

Aquest nou operador consistirà en assignar un operador a la revisió associada d'una tasca (assignReviewTask), i d'aquesta manera, el planificador ja serà capaç de fer totes les assignacions necessàries de totes les tasques a considerar en el nostre problema.

Aquest nou operador rebrà 3 paràmetres diferents, que seran, per un cantó el nou programador (?p) a assignar a la tasca de revisió. També se li ha de passar la tasca que estem considerant (?t) , i finalment el programador que realitza la tasca original (?pAnterior), la qual ara caldrà revisar.
Per tant, les precondicions que ara ens caldrà comprovar seran les següents:

1) La tasca de revisió associada no està assignada.
2) El programador ?pAnterior té assignada la tasca original.
3) La tasca original ja ha estat assignada.

4) Els dos programadors ?p i ?pAnterior són diferents (per al restricció que un mateix programador no pot revisar la seva pròpia feina).
5) Que la habilitat del programador ?p és com a molt inferior a la dificultat de la tasca de revisió, que coincideix amb la dificultat de la tasca original.

En el cas de que s'aconsegueixi que totes aquestes restriccions es compleixin, definim els efectes que tindria aquesta acció que seran els següents:

1) La tasca de revisió associada a la tasca ?t passa a estar assignada.
2) Assignem el programador ?p a la tasca de revisió associada a ?t (fent servir el predicat programmerReviewsTask)

L'altre operador d'assignar una tasca serà quelcom igual que en el cas anterior, amb una petita modificació que ens cal introduir. Ara, un cop assignem un programador a una tasca, ens cal guardar-nos la durada que tindrà la seva tasca de revisió associada. Aquesta durada dóna el cas que coincidirà amb la qualitat del programador. Per tant, establirem que la funció (reviewDuration ?t) serà igual a la funció que ens indica la qualitat del programador (quality ?p).

Extensió 2 (domain2.pddl)

En aquesta extensió ens fa falta minimitzar el nombre total d'hores necessàries per a realitzar totes les tasques. Aleshores, hem hagut de crear una nova funció que anomenarem (totalHours) que ens servirà com a comptador del nombre d'hores totals. Per tant, les accions tindran exactament la mateixa forma que per a l'extensió anterior, amb la única diferència que ara hem de ser capaços d'actualitzar aquest comptador. Per a fer-ho, el que hem fet és afegir una acció més en els efectes del nostres 2 operadors.

Pel cas d'assignar la tasca original, aquest tindrà aquesta forma:
(increase (totalHours) (duration ?t))

Per tant, donat que tindrem una funció (duration ?t) que ens indica quan es triga a realitzar una tasca, tan sols ens cal actualitzar el nombre total d'hores amb aquest valor.

Pel cas del segon operador d'assignar una tasca de revisió, farem exactament el mateix però aquest cop incrementarem la funció de (totalHours) amb la duració de la tasca de revisió, que també tenim guardada en una funció. Així doncs, tindria la forma següent:
(increase (totalHours) (reviewDuration ?t))

Extensió 3 (domain3.pddl)

En aquesta extensió volem afegir la restricció de que cap treballador pugui participar en més de 2 tasques. Per aquest motiu varem introduir una nova funció (numTasks ?p) que ens indica el nombre de tasques en les que participa un programador. Aleshores això, òbviament, implica uns pocs canvis respecte els operadors descrits anteriorment, per tal de poder anar actualitzant aquesta funció. Per tant, el que caldrà fer serà incrementar en 1 aquesta funció cada vegada que assignem un programador a una tasca tan original com de revisió.

Aleshores, tan sols hem hagut d'afegir un efecte més en el dos operadors que faci aquest increment. Aquest increment, doncs, tindrà la següent sintaxis:
(increase (numTasks ?p) 1)

Extensió 4 (domain4.pddl)

Finalment, en aquesta última extensió se'ns demana ser capaços de maximitzar la suma ponderada del nombre d'hores totals dedicades i el nombre total de programadors que treballen (que tenen com a mínim una tasca assignada). Per tant, els que ens interessa és ser capaços de mantenir aquestes dues informacions per tal que arribats al goal, poguem crear una funció de maximització en funció d'aquests valors.
El nombre total d'hores dedicades ja el tenim correctament calculat, doncs és el que hem fet a l'extensió 2. Per tant, el que ens falta introduir en aquesta nova extensió serà tenir guardat en una funció el nombre total de programadors que treballen. Per a fer-ho seguirem una dinàmica semblant a la de l'extensió 2.
Hem creat una nova funció que ho representarà, i a més, hem introduït el nou predicat que ens indica si un programador ja estava treballant o no.
Dit això, tant en el primer operador com el segon, ens faltarà afegir un últim efecte que serà que en el cas que encara no estigui treballant aquest programador, el marqui com a que ara sí que treballa, i incrementi la funció creada en 1.

Si duem a terme aquesta acció, assegurem que arribats al final del programa, tindrem guardats a la funció el nombre total dels programadors que treballen, que en definitiva és tot el que necessitem.

4. Modelat del problema

Fins ara ens hem centrat en tot el que implicava el domini, però en aquest apartat anem a intentar veure com modelem el problema per a les diferents versions del problema que se'ns han plantejat. Per a fer-ho veurem molt breument quins són els objectes que creem en cada cas, el cas inicial i finalment l'estat final. També inclourem, en cas d'haver-n'hi, la funció de minimització o maximització.

Versió inicial

Per a la versió inicial, tan sols ens cal crear n programadors i k tasques. Aleshores, el cas inicial consistirà en que cadascún d'aquests programadors se'ls hi assigni una habilitat i una qualitat, i ha totes les tasques una complexitat i una duració. Tot això ho farem a través de donar valors a les diferents funcions, que ja hem vista en apartats anteriors que estaven dedicats a emmagatzemar aquesta informació.

Pel que fa l'estat final, donat que no se'ns demana tenir en compte les tasques de revisió, tan sols hem de comprovar que per tota tasca ?t, el predicat (taskAssigned ?t) és cert.

Extensió 1

Aquesta versió durà a terme el modelat del problema d'una forma pràcticament idèntica que la versió inicial. La creació del cas inicial i dels objectes serà igual. La única diferència serà en el cas final, ja que ara, no volem assegurar tan sols que tota tasca té el predicat (taskAssigned ?t) com a cert, sinó que ara també ens caldrà comprovar que (taskReviewAssigned ?t) també ho és de cert. D'aquesta manera garantim que totes les tasques normals i la seva corresponent revisió estan assignades.

Extensió 2

Com en les dues versions anteriors, els objectes que haurem de crear seran només els programadors amb els que contem, així com les tasques que hem d'assignar. La inicialització del problema serà igual que en la extensió 1, amb la única diferència que ara hem d'inicialitzar la funció (totalHours) per tal que valgui 0. D'aquesta manera, podem dur a terme al domini increments en funció de la duració de cada tasca, i que el resultat final sigui l'esperat. L'objectiu final del programa serà igual que per l'extensió anterior, tot i que ara hem hagut d'afegir quelcom una funció de minimització que consistirà en minimitzar el nombre total d'hores a realitzar les tasques (valor de la funció (totalHours)).

Extensió 3

En aquesta versió, tal i com hem estat fent fins ara, partirem del modelat del problema i durem a terme unes poques modificacions per a adaptar-lo al nou problema. Pel que els objectes respecte no hi ha cap diferència. El cas inicial serà idèntic tot i que ens caldrà igualar la funció de (numTasks ?p) a 0 per tots els programadors que hem creat.
Aleshores el cas final serà igual que abans, incloënt la funció de minimització, amb la diferència que hem de comprovar quelcom que per tot programador, la seva funció numTasks no retorna un valor superior a 2.

Extensió 4

En aquesta extensió ens caldrà afegir al cas inicial la inicialització del nombre de programadors que estan treballant inicialment, que serà 0. Tota la resta serà exactament igual que per a l'extensió 3 excepte la funció de minimització.
En aquest cas se'ns demana eliminar la funció de minimització, i substituir-ho per a una funció de maximització de la suma ponderada entre el nombre de programadors que treballen i les hores totals dedicades.

Així doncs, aquesta funció de maximització seguiria la següent fórmula:

*Maximize (10 * numProgramadorsQueTreballen + 1* (sum(duració de cada tasca del problema)))*

Pel que fa els coeficients, hem decidit multiplicar el nombre de programadors que treballen per 10 donat que aquest valor serà clarament més petit que el sumatori de les duracions de les tasques. Aleshores, després de dur a terme tota una sèrie d'experimentació, hem pogut veure com aquest valor resultava ser el millor per tal de maximitzar aquesta suma, i que ambdós paràmetres es tinguessin en compte amb percentatges semblants.

5.Jocs de proves

Per aquesta pràctica hem ideat uns jocs de proves més o menys aleatoris pel que fa a nombre de programadors i de tasques, tenint en compte que tingués sentit pel domini proposat (per exemple, a partir de l'extensió 3 com a molt poden haver-hi tantes tasques com el nombre de programadors, ja que hi ha una restricció on cada programador només pot fer 2 tasques, contant que cada tasca implica una de revisió i aquestes computen com a tasca). Les inicialitzacions dels fluents (característiques dels programadors i de les tasques) sí que són aleatòries fent servir la funció *randint(1,n)* del mòdul random de python3, sent n el nombre de programadors.
Comptem amb un total de 15 jocs de proves, 3 per a cada extensió (5 extensions comptant la versió bàsica).

5.1. Generador joc de proves

Per a generar tots els jocs de proves per a cada un dels dominis que se'ns demanaven, pel cas base i per les extensions, hem fet un script en python. Només li hem d'indicar el domini que estem utilitzant (nivell bàsic, extensió 1, extensió 2, extensió 3 o extensió 4), el nombre de programadors disponibles i el nombre de tasques a assignar.
Depenent de tots aquests inputs l'script genera tres problemes amb la sintaxis correcte en PDDL i inicialitzant els valors de forma aleatòria. Tenen com a nom del fitxer: problemaPX.pddl on dom(P)={B,E1,E2,E3,E4} i dom(X)={a,b,c}, i per exemple un primer problema per a l'extensió 3 té el nom: problemE3a.pddl.
Per executar-lo només hem d'estar a la carpeta on tenim l'script executar la comanda python3 script.py i ens generarà a la carpeta actual els tres jocs de proves que hem creat per a aquella versió del domini.

5.2. Jocs de prova Nivell Bàsic

5.2.1.Problema Bàsic A

En aquest primer joc de prova per a la versió bàsica tenim 4 programadors i 3 tasques amb les seves característiques definides a continuació:

Joc de prova
(define (problem Ba) (:domain tasking) (:objects p1 p2 p3 p4 - programmer t1 t2 t3 - task

```
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 5)
         (= (ability p2) 5)
         (= (ability p3) 1)
         (= (ability p4) 1)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 2)
         (= (quality p4) 2)

        ;;Complexity tasks
         (= (complexity t1) 2)
         (= (complexity t2) 1)
         (= (complexity t3) 2)

        ;;Duration tasks
         (= (duration t1) 2)
         (= (duration t2) 4)
         (= (duration t3) 4)

        )
        ;; The goal is:
        (:goal (forall (?t - task) (taskAssigned ?t)))
)
```

Com veiem en la sortida del planificador la tasca 3, la tasca 2 i la tasca 1 han estat assignades al programador 1 perquè és l'únic que compleix les restriccions de complexitat de les tasques.

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T3
        1: ASSIGNTASK P1 T2
        2: ASSIGNTASK P1 T1


time spent:     0.00 seconds total time
```

5.2.2.Problema Bàsic B

En aquest segon joc de proves per a la versió bàsica tenim 5 programadors i 3 tasques amb les característiques per a cada un a continuació:

Joc de prova
(define (problem Bb) (:domain tasking) (:objects p1 p2 p3 p4 p5 - programmer t1 t2 t3 - task) (:init ;;Ability programmer (= (ability p1) 2) (= (ability p2) 4) (= (ability p3) 3) (= (ability p4) 4) (= (ability p5) 2) ;;Quality programmer (= (quality p1) 2) (= (quality p2) 2) (= (quality p3) 2) (= (quality p4) 2) (= (quality p5) 2) ;;Complexity tasks (= (complexity t1) 4) (= (complexity t2) 1) (= (complexity t3) 3) ;;Duration tasks (= (duration t1) 1) (= (duration t2) 4) (= (duration t3) 3)) ;; The goal is: (:goal (forall (?t - task) (taskAssigned ?t))))

Com podem veure a la sortida del planificador s'han assignat la tasca 3 i tasca 2 al programador 1, i la tasca 1 al programador 2 per a satisfer les restriccions de complexitat i comprovem que aquestes assignacions compleixen totes les restriccions perfectament.

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T3
        1: ASSIGNTASK P1 T2
        2: ASSIGNTASK P2 T1


time spent:      0.00 seconds total time
```

5.2.3.Problema Bàsic C

En aquest problema tenim dues tasques i tres programadors. Podem veure les seves característiques a continuació:

Joc de prova

```
(define (problem Bc)
        (:domain tasking)
        (:objects
        p1 p2 p3 - programmer
        t1 t2 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 1)
         (= (ability p2) 1)
         (= (ability p3) 3)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 2)
```

```
        ;;Complexity tasks
         (= (complexity t1) 2)
         (= (complexity t2) 3)

        ;;Duration tasks
         (= (duration t1) 2)
         (= (duration t2) 3)

        )
        ;; The goal is:
        (:goal (forall (?t - task) (taskAssigned ?t)))
)
```

Assignem la tasca amb complexitat 3 al programador amb habilitat 3 i la tasca amb complexitat 2 al programador 1 amb habilitat 1.

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P3 T2
        1: ASSIGNTASK P1 T1


time spent:      0.00 seconds total time
```

5.3. Jocs de prova Extensió 1

5.3.1.Problema Extensió 1 A

En aquesta extensió s'inclou una tasca de revisió per cada tasca el temps de la qual dependrà de la qualitat del programador que l'ha dut a terme, i amb complexitat igual a la tasca inicial. Per tant haurem de veure com els programadors que duen a terme les tasques de revisió

(partim de la base que en l'apartat anterior ja hem vist com els programadors s'assignen correctament a cada tasca) també compleixen que la seva habilitat sigui com a molt 1 inferior a la complexitat de la tasca de revisió. En aquest joc de proves tenim 3 programadors i 2 tasques amb les següents característiques:

Joc de prova

```
(define (problem E1a)
        (:domain tasking)
        (:objects
        p1 p2 p3 - programmer
        t1 t2 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 1)
         (= (ability p2) 4)
         (= (ability p3) 1)

        ;;Quality programmer
         (= (quality p1)2)
         (= (quality p2) 1)
         (= (quality p3) 2)

        ;;Complexity tasks
         (= (complexity t1) 1)
         (= (complexity t2) 1)

        ;;Duration tasks
         (= (duration t1) 1)
         (= (duration t2) 3)

        )
        ;; The goal is:
        (:goal (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))))
)
```

S'assigna la tasca de revisió de la tasca 1 (complexitat 1) al programador 2 (habilitat 1) i la tasca de revisió de la tasca 2 (complexitat 1) al programador 2 (habilitat 1). Per tant fa una assignació que no viola cap de les restriccions abans mencionades.

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T2
        1: ASSIGNTASK P1 T1
        2: ASSIGNREVIEWTASK P2 T1 P1
        3: ASSIGNREVIEWTASK P2 T2 P1


time spent:    0.00 seconds total time
```

5.3.2.Problema Extensió 1 B

Joc de prova

```
(define (problem E1b)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 - programmer
        t1 t2 t3 t4 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 4)
         (= (ability p2) 3)
         (= (ability p3) 1)
         (= (ability p4) 5)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 2)
         (= (quality p4) 1)

        ;;Complexity tasks
         (= (complexity t1) 5)
         (= (complexity t2) 5)
         (= (complexity t3) 2)
         (= (complexity t4) 3)

        ;;Duration tasks
         (= (duration t1) 1)
         (= (duration t2) 4)
```

```
        (= (duration t3) 2)
        (= (duration t4) 5)

        )
        ;; The goal is:
        (:goal (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))))
)
```

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T4
        1: ASSIGNTASK P1 T3
        2: ASSIGNTASK P1 T2
        3: ASSIGNTASK P1 T1
        4: ASSIGNREVIEWTASK P4 T1 P1
        5: ASSIGNREVIEWTASK P4 T2 P1
        6: ASSIGNREVIEWTASK P2 T3 P1
        7: ASSIGNREVIEWTASK P2 T4 P1


time spent:     0.00 seconds total time
```

5.3.3.Problema Extensió 1 C

Joc de prova

```
(define (problem E1c)
        (:domain tasking)
        (:objects
        p1 p2 p3 - programmer
        t1 t2 t3 t4 - task
        )
        (:init
        ;;Ability programmer
        (= (ability p1) 4)
        (= (ability p2) 1)
        (= (ability p3) 2)
```

```
        ;;Quality programmer
         (= (quality p1) 1)
         (= (quality p2) 2)
         (= (quality p3) 2)

        ;;Complexity tasks
         (= (complexity t1) 3)
         (= (complexity t2) 3)
         (= (complexity t3) 4)
         (= (complexity t4) 1)

        ;;Duration tasks
         (= (duration t1) 1)
         (= (duration t2) 2)
         (= (duration t3) 1)
         (= (duration t4) 2)

        )
        ;; The goal is:
        (:goal (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))))
)
```

Sortida del planificador

```
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 total time
```

5.4. Jocs de prova Extensió 2

5.4.1.Problema Extensió 2 A

En aquesta extensió ens interessa minimitzar el temps total que s'utilitza en resoldre totes les tasques (suma de les hores). Per tant hauriem de veure com s'han de minimitzar les tasques assignades a un programador de menys dificultat, i les tasques dutes a terme per un programador amb qualitat baixa. En aquest joc de proves tenim quatre programadors i quatre tasques amb les següents característiques:

Joc de prova

```
(define (problem E2a)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 - programmer
        t1 t2 t3 t4 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 5)
         (= (ability p2) 2)
         (= (ability p3) 4)
         (= (ability p4) 2)

        ;;Quality programmer
         (= (quality p1) 1)
         (= (quality p2) 2)
         (= (quality p3) 1)
         (= (quality p4) 2)

        ;;Complexity tasks
         (= (complexity t1) 1)
         (= (complexity t2) 3)
         (= (complexity t3) 3)
         (= (complexity t4) 4)

        ;;Duration tasks
         (= (duration t1) 3)
         (= (duration t2) 3)
         (= (duration t3) 3)
         (= (duration t4) 3)

         ;;TotalHours
         (= (totalHours) 0)

        )
        ;; The goal is:
        (:goal (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))))
        (:metric minimize(totalHours))
)
```

Podem veure com minimitza el temps ja que assigna les tasques al programador amb més habilitat (tal que no es sumen hores innecessàries) i menys qualitat (la seva correcció serà més fàcil).

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T4
        1: ASSIGNTASK P1 T3
        2: ASSIGNTASK P1 T2
        3: ASSIGNTASK P1 T1
        4: ASSIGNREVIEWTASK P2 T1 P1
        5: ASSIGNREVIEWTASK P2 T2 P1
        6: ASSIGNREVIEWTASK P2 T3 P1
        7: ASSIGNREVIEWTASK P3 T4 P1


time spent:     0.00 seconds total time
```

5.4.2.Problema Extensió 2 B

Joc de prova

```
(define (problem E2b)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 p5 - programmer
        t1 t2 t3 t4 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 4)
         (= (ability p2) 6)
         (= (ability p3) 6)
         (= (ability p4) 4)
         (= (ability p5) 3)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
```

```
        (= (quality p3) 2)
        (= (quality p4) 1)
        (= (quality p5) 2)

        ;;Complexity tasks
        (= (complexity t1) 4)
        (= (complexity t2) 2)
        (= (complexity t3) 2)
        (= (complexity t4) 2)

        ;;Duration tasks
        (= (duration t1) 2)
        (= (duration t2) 2)
        (= (duration t3) 2)
        (= (duration t4) 1)

        ;;TotalHours
        (= (totalHours) 0)

        )
        ;; The goal is:
        (:goal (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))))
        (:metric minimize(totalHours))
)
```

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T4
        1: ASSIGNTASK P1 T3
        2: ASSIGNTASK P1 T2
        3: ASSIGNTASK P1 T1
        4: ASSIGNREVIEWTASK P2 T1 P1
        5: ASSIGNREVIEWTASK P2 T2 P1
        6: ASSIGNREVIEWTASK P2 T3 P1
        7: ASSIGNREVIEWTASK P2 T4 P1


time spent:     0.00 seconds total time
```

5.4.3.Problema Extensió 2 C

Joc de prova

```
(define (problem E2c)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 - programmer
        t1 t2 t3 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 1)
         (= (ability p2) 2)
         (= (ability p3) 2)
         (= (ability p4) 1)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 2)
         (= (quality p4) 2)

        ;;Complexity tasks
         (= (complexity t1) 1)
         (= (complexity t2) 4)
         (= (complexity t3) 3)

        ;;Duration tasks
         (= (duration t1) 3)
         (= (duration t2) 3)
         (= (duration t3) 2)

         ;;TotalHours
         (= (totalHours) 0)

        )
        ;; The goal is:
        (:goal (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))))
        (:metric minimize(totalHours))
)
```

Sortida del planificador
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 total time

5.5. Jocs de prova Extensió 3

5.5.1.Problema Extensió 3 A

En aquesta extensió volem limitat el nombre de tasques (incloent les tasques de revisió) que pot fer una persona a 2 per tal de paral·lelitzar al màxim el temps. Així doncs en les solucions d'aquests jocs de prova no hauriem d'assignar a cap programador més de dues tasques. En aquest joc de prova tenim 4 programadors i 2 tasques amb les següents característiques:

Joc de prova
(define (problem E3a) (:domain tasking) (:objects p1 p2 p3 p4 - programmer t1 t2 - task) (:init ;;Ability programmer (= (ability p1) 3) (= (ability p2) 4) (= (ability p3) 5) (= (ability p4) 1) ;;Quality programmer (= (quality p1) 2) (= (quality p2) 2) (= (quality p3) 1) (= (quality p4) 1) ;;Complexity tasks

```
        (= (complexity t1) 3)
        (= (complexity t2) 1)

       ;;Duration tasks
        (= (duration t1) 3)
        (= (duration t2) 1)

        ;;TotalHours
        (= (totalHours) 0)

        ;;NumTasks programmer
        (= (numTasks p1) 0)
        (= (numTasks p2) 0)
        (= (numTasks p3) 0)
        (= (numTasks p4) 0)

       )
       ;; The goal is:
       (:goal (and (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))) (forall
(?p - programmer) (<= (numTasks ?p) 2))) )
       (:metric minimize(totalHours))
)
```

Com es pot veure cap programador té més de dues tasques assignades per tant complim la restricció.

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T2
        1: ASSIGNTASK P1 T1
        2: ASSIGNREVIEWTASK P2 T1 P1
        3: ASSIGNREVIEWTASK P2 T2 P1


time spent:    0.00 seconds total time
```

5.5.2.Problema Extensió 3 B

Joc de prova

```
(define (problem E3b)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 - programmer
        t1 t2 t3 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 4)
         (= (ability p2) 4)
         (= (ability p3) 1)
         (= (ability p4) 3)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 2)
         (= (quality p4) 2)

        ;;Complexity tasks
         (= (complexity t1) 4)
         (= (complexity t2) 3)
         (= (complexity t3) 1)

        ;;Duration tasks
         (= (duration t1) 4)
         (= (duration t2) 1)
         (= (duration t3) 4)

         ;;TotalHours
         (= (totalHours) 0)

         ;;NumTasks programmer
         (= (numTasks p1) 0)
         (= (numTasks p2) 0)
         (= (numTasks p3) 0)
         (= (numTasks p4) 0)

        )
        ;; The goal is:
        (:goal (and (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))) (forall
(?p - programmer) (<= (numTasks ?p) 2))) )
        (:metric minimize(totalHours))
)
```

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T3
        1: ASSIGNTASK P1 T2
        2: ASSIGNTASK P2 T1
        3: ASSIGNREVIEWTASK P4 T1 P2
        4: ASSIGNREVIEWTASK P2 T2 P1
        5: ASSIGNREVIEWTASK P3 T3 P1


time spent:     0.00 seconds total time
```

5.5.3.Problema Extensió 3 C

Joc de prova

```
(define (problem E3c)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 p5 - programmer
        t1 t2 t3 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 2)
         (= (ability p2) 6)
         (= (ability p3) 4)
         (= (ability p4) 4)
         (= (ability p5) 1)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 2)
         (= (quality p4) 1)
         (= (quality p5) 2)
```

```
        ;;Complexity tasks
         (= (complexity t1) 4)
         (= (complexity t2) 2)
         (= (complexity t3) 2)

        ;;Duration tasks
         (= (duration t1) 2)
         (= (duration t2) 2)
         (= (duration t3) 3)

         ;;TotalHours
         (= (totalHours) 0)

         ;;NumTasks programmer
         (= (numTasks p1) 0)
         (= (numTasks p2) 0)
         (= (numTasks p3) 0)
         (= (numTasks p4) 0)
         (= (numTasks p5) 0)

        )
        ;; The goal is:
        (:goal (and (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))) (forall
(?p - programmer) (<= (numTasks ?p) 2))) )
        (:metric minimize(totalHours))
)
```

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T3
        1: ASSIGNTASK P1 T2
        2: ASSIGNTASK P2 T1
        3: ASSIGNREVIEWTASK P3 T1 P2
        4: ASSIGNREVIEWTASK P2 T2 P1
        5: ASSIGNREVIEWTASK P3 T3 P1


time spent:     0.00 seconds total time
```

5.6. Jocs de prova Extensió 4

5.6.1.Problema Extensió 4 A

En aquesta extensió se'ns demana mantenir el nombre màxim de tasques per programador (dues) però a més a més, maximitzar la suma ponderada entre el nombre de persones que estan fent les tasques i el temps total que es triga a resoldre-les.

Joc de prova

```
(define (problem E4a)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 p5 p6 p7 p8 - programmer
        t1 t2 t3 t4 t5 t6 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 3)
         (= (ability p2) 8)
         (= (ability p3) 2)
         (= (ability p4) 9)
         (= (ability p5) 6)
         (= (ability p6) 1)
         (= (ability p7) 7)
         (= (ability p8) 7)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 2)
         (= (quality p4) 2)
         (= (quality p5) 2)
         (= (quality p6) 2)
         (= (quality p7) 2)
         (= (quality p8) 1)

        ;;Complexity tasks
         (= (complexity t1) 6)
         (= (complexity t2) 3)
         (= (complexity t3) 7)
         (= (complexity t4) 7)
         (= (complexity t5) 5)
         (= (complexity t6) 4)
```

```
        ;;Duration tasks
         (= (duration t1) 3)
         (= (duration t2) 3)
         (= (duration t3) 2)
         (= (duration t4) 5)
         (= (duration t5) 4)
         (= (duration t6) 2)

         ;;TotalHours
         (= (totalHours) 0)

         ;;NumTasks programmer
         (= (numTasks p1) 0)
         (= (numTasks p2) 0)
         (= (numTasks p3) 0)
         (= (numTasks p4) 0)
         (= (numTasks p5) 0)
         (= (numTasks p6) 0)
         (= (numTasks p7) 0)
         (= (numTasks p8) 0)

         ;;WorkingProgrammers
         (= (workingProgrammers) 0)

        )
        ;; The goal is:
        (:goal (and (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))) (forall
(?p - programmer) (<= (numTasks ?p) 2))) )
        (:metric maximize (+ (* 10 (workingProgrammers)) (* 1 (totalHours))))
)
```

Garantim que el codi comprova i maximitza la suma ponderada demanda, pero és inasumible justificar la optimalitat de la solució.

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T6
        1: ASSIGNTASK P2 T5
        2: ASSIGNTASK P2 T4
        3: ASSIGNTASK P4 T3
        4: ASSIGNTASK P1 T2
        5: ASSIGNTASK P4 T1
```

```
        6: ASSIGNREVIEWTASK P5 T1 P4
        7: ASSIGNREVIEWTASK P3 T2 P1
        8: ASSIGNREVIEWTASK P5 T3 P4
        9: ASSIGNREVIEWTASK P7 T4 P2
        10: ASSIGNREVIEWTASK P7 T5 P2
        11: ASSIGNREVIEWTASK P8 T6 P1


time spent:     0.01 seconds total time
```

5.6.1.Problema Extensió 4 B

Joc de prova

```
(define (problem E4b)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 p5 p6 p7 - programmer
        t1 t2 t3 t4 t5 t6 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 4)
         (= (ability p2) 7)
         (= (ability p3) 2)
         (= (ability p4) 4)
         (= (ability p5) 8)
         (= (ability p6) 5)
         (= (ability p7) 1)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 2)
         (= (quality p4) 2)
         (= (quality p5) 2)
         (= (quality p6) 2)
         (= (quality p7) 1)

        ;;Complexity tasks
         (= (complexity t1) 4)
         (= (complexity t2) 6)
```

```
        (= (complexity t3) 7)
        (= (complexity t4) 7)
        (= (complexity t5) 5)
        (= (complexity t6) 7)

       ;;Duration tasks
        (= (duration t1) 4)
        (= (duration t2) 4)
        (= (duration t3) 2)
        (= (duration t4) 7)
        (= (duration t5) 6)
        (= (duration t6) 1)

        ;;TotalHours
        (= (totalHours) 0)

        ;;NumTasks programmer
        (= (numTasks p1) 0)
        (= (numTasks p2) 0)
        (= (numTasks p3) 0)
        (= (numTasks p4) 0)
        (= (numTasks p5) 0)
        (= (numTasks p6) 0)
        (= (numTasks p7) 0)

        ;;WorkingProgrammers
        (= (workingProgrammers) 0)

       )
       ;; The goal is:
       (:goal (and (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))) (forall
(?p - programmer) (<= (numTasks ?p) 2))) )
       (:metric maximize (+ (* 10 (workingProgrammers)) (* 1 (totalHours))))
)
```

Sortida del planificador

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

time spent:      7.12 seconds total time
```

5.6.1.Problema Extensió 4 C

Joc de prova

```
(define (problem E4c)
        (:domain tasking)
        (:objects
        p1 p2 p3 p4 p5 p6 p7 - programmer
        t1 t2 t3 t4 t5 - task
        )
        (:init
        ;;Ability programmer
         (= (ability p1) 8)
         (= (ability p2) 4)
         (= (ability p3) 5)
         (= (ability p4) 6)
         (= (ability p5) 1)
         (= (ability p6) 5)
         (= (ability p7) 1)

        ;;Quality programmer
         (= (quality p1) 2)
         (= (quality p2) 2)
         (= (quality p3) 1)
         (= (quality p4) 2)
         (= (quality p5) 1)
         (= (quality p6) 2)
         (= (quality p7) 2)

        ;;Complexity tasks
         (= (complexity t1) 4)
         (= (complexity t2) 3)
         (= (complexity t3) 4)
         (= (complexity t4) 2)
         (= (complexity t5) 3)

        ;;Duration tasks
         (= (duration t1) 4)
         (= (duration t2) 3)
         (= (duration t3) 6)
         (= (duration t4) 4)
         (= (duration t5) 5)

         ;;TotalHours
         (= (totalHours) 0)
```

```
        ;;NumTasks programmer
        (= (numTasks p1) 0)
        (= (numTasks p2) 0)
        (= (numTasks p3) 0)
        (= (numTasks p4) 0)
        (= (numTasks p5) 0)
        (= (numTasks p6) 0)
        (= (numTasks p7) 0)

        ;;WorkingProgrammers
        (= (workingProgrammers) 0)

      )
      ;; The goal is:
      (:goal (and (forall (?t - task) (and (taskReviewAssigned ?t)(taskAssigned ?t))) (forall
(?p - programmer) (<= (numTasks ?p) 2))) )
      (:metric maximize (+ (* 10 (workingProgrammers)) (* 1 (totalHours))))
)
```

Sortida del planificador

```
ff: found legal plan as follows

step    0: ASSIGNTASK P1 T5
        1: ASSIGNTASK P1 T4
        2: ASSIGNTASK P2 T3
        3: ASSIGNTASK P2 T2
        4: ASSIGNTASK P3 T1
        5: ASSIGNREVIEWTASK P4 T1 P3
        6: ASSIGNREVIEWTASK P3 T2 P2
        7: ASSIGNREVIEWTASK P4 T3 P2
        8: ASSIGNREVIEWTASK P5 T4 P1
        9: ASSIGNREVIEWTASK P6 T5 P1


time spent:     0.01 seconds total time
```