Documentació de la pràctica de Planificació

Problema:

El problema plantejat per aquesta pràctica, és la distribució de tasques en un projecte informàtic de gran escala, mitjançant planificació automàtica.
Donades certes Tasques amb les seves propietats, i una certa quantitat de programadors, dels quals sabem també certa informació, el nostre planificador haurà de buscar les assignacions de tasques a programadors complint els requisits del model, i diverses extensions.
L'exercici proposat, serà resoldre el problema bàsic, en el que només s'ha de satisfer que els programadors resolguin les tasques, i a mesura que anem afegint extensions, augmentarà la complexitat del problema. En la primera extensió, afegirem el fet de que quan una tasca sigui completada, es generin tasques de revisió, que també s'hauran de resoldre tot i que no generaran noves tasques. A l'extensió dos, afegirem un punt de complexitat, ja que demanarem minimitzar el temps utilitzat en la resolució. A la següent extensió, la tercera, es restringeix les tasques que poden resoldre els programadors, a dos per cap. A la darrera extensió, la quarta, es restringirà la solució optimitzant respecte en nombre de programadors i el temps que s'ha tardat a resoldre.

Domini

Com és el domini?

En aquest problema trobem tasques i programadors.
Per cada Tasca, tenim definides les propietats de Dificultat, que en la descripció del problema podrà tenir els valors 1, 2 o 3, i Temps de realització, que representarem amb un valor positiu, ja que serà el temps necessari per ser resoltes en hores.
Per cada Programador, tenim la seva habilitat, que en aquest problema també representarem amb els valors 1, 2, o 3, i la qualitat d'aquest com a programador, que representarem amb els valors 1 o 2.
A més, per satisfer les diferents restriccions, i extensions del problema, utilitzarem el concepte de tasca per realitzar, que serà una tasca que es generarà en resoldre una tasca de les anteriorment esmentades, però no generarà tasques noves al ser resolta. Afegirem al programador la quantitat de tasques que encara podem assignar-li, i treballarem amb els conceptes de temps total, i quantitat de programadors utilitzats per resoldre el problema segons la nostra planificació.

Genís Bayona, Gonzalo Diez, Alejandro Polo

Elements del domini:

Predicats:

- (tBasica ?t - Tarea)

Tasca a realitzar, que haurà de ser assignada a un programador.
Aquestes tasques són les que en primera instància volem resoldre amb els programadors disponibles. En ser resolta, generarà una tasca per revisar.

- (tPorRevisar ?t - Tarea ?p - Programador)

Tasca derivada de realitzar una tasca simple, que conté el programador que ha realitzat la tasca simple, és a dir el programador que no pot resoldre aquesta nova tasca. Per resoldre el problema necessitem distingir entre les tasques bàsiques, que generen tasques per revisar o les tasques per revisar, que no generaran cap tasca addicional en ser realitzades. Ens cal recordar quin programador ha resolt la tasca que ha generat aquesta tasca per realitzar, ja que un programador no pot resoldre la tasca per revisar que ha generat una tasca completada per ell mateix.

- (finalizada ?t - Tarea)

Tasca que ja ha estat resolta i revisada.
Per poder modelar el Problema, utilitzarem aquest predicat per no proposar solucions que resolguin més d'una vegada una mateixa tasca.

- (por-contar ?p - Programador)

Quan es genera una tasca amb un programador aquest passa a l'estat "per contar". Podem entendre que és un indicador de si un programador ha resolt alguna tasca i encara el tenim que desmarcar.
Aquest predicat serà necessari per minimitzar el nombre de programadors de la solució proposada (a partir de la extensió 3).

- (contado ?p - Programador)

Indica que un Programador ha estat contat per el programa.
Predicat necessari, utilitzat per no contar més de una vegada cada Programador que hagi realitzat alguna tasca.

Functions:

- (tiempo-tarea ?t - Tarea)

Temps requerit per realitzar la tasca assignada. Pot tenir qualsevol valor, però degut a que representa el temps, haurien de ser positius.
Modela una part de la informació bàsica del problema.

- (nivel-tarea ?t - Tarea)

Nivell de dificultat de la tasca assignada, en el problema té els nivells 1, 2, 3.
Modela una part de la informació bàsica del problema.

- (nivel-programador ?p - Programador)

Nivell de expertesa del programador assignat, en el problema pot tenir els nivells 1, 2 i 3.
Modela una part de la informació bàsica del problema.

- (calidad ?p - Programador)

Qualitat del programador assignat, en el model pot ser 1 o 2.
Modela una part de la informació bàsica del problema.

- (tareas-programador ?p - Programador)

Quantitat de tasques que pot realitzar un programador. A mesura que resolgui una tasca, utilitzarem aquest fluent per saber que pot resoldre una tasca menys. En aquest problema, tots els programadors començaran amb el valor 2 en aquest fluent. Si arriba al valor 0, ja no podrà realitzar més tasques.
Aquest fluent serà necessari doncs per assegurar la restricció de l'extensió tres.

- (ttime)

Numero de hores totals utilitzades per resoldre el problema amb la planificació proposada.
S'utilitzarà per poder minimitzar el resultat d'acord amb les extensions dos i quatre.

- (nprog)

Numero de programadors utilitzats per resoldre el problema amb la planificació proposada.
S'utilitzarà per poder minimitzar el resultat d'acord amb l'extensio quatre.

Operadors

A continuació anunciarem i explicarem, justificadament, els diferents operadors que hem utilitzat com a accions per aquest planificador.

(:action **completar-tarea-basica**
Operador que agafa com a paràmetres una Tasca i un Programador, i si la tasca és bàsica, el programador pot agafar més tasques, i el nivell del programador és més gran o igual que el de la tasca, elimina la tasca bàsica, i en genera una per revisar amb el programador que l'ha resolt i incrementa els valors necessaris per mantenir l'estat del programa.
Amb aquesta funció, resoldrem la part bàsica del programa de resoldre tasques amb programadors.

(:action **completar-tarea-basica-con-suplemento**
Operador que agafa com a paràmetres una Tasca i un Programador, i si la tasca és bàsica, el programador pot agafar més tasques, però el nivell de la tasca és una unitat més gran que el nivell del programador, es resoldrà la tasca bàsica, i es generarà la conseqüent tasca per realitzar, amb un suplement de dues hores de més (que es sumen pel fet que un programador resolgui una tasca de mes nivell).
Aquest Operador podríem no haver-lo inclòs, fent alguna modificació en l'accio de completar-tarea-basica, però creiem que així s'entén més bé i fa de més bon llegir i programar, i sobretot de interpretar el planning proposat per el sistema.
Amb aquesta funció, resoldrem la part bàsica del programa de resoldre tasques amb programadors.

(:action **completar-tarea-por-revisar**
Operador que agafa com a paràmetres una Tasca i dos programadors,i si és compleix que la tasca és per revisar, generad aper un dels dos programadors, són diferents els dos programadors, i l'altre programador té nivell per resoldre la tasca, dona la tasca per finalitzda, i actualitza l'estat del programa.
Per poder resoldre el problema a partir de la segona extensió, necessitarem aquest operador

(:action **contar-programador**
Operador que agafa com a paràmetre un Programador, i si ha realitzat alguna tasca i no ha estat contat anteriorment, actualitza l'estat del programa per tenir la quantitat de programadors utilitzats.
Aquest Operador serà necessari per la darrera extensió del problema.

(:action **desmarcar-programador**
Operador que agafa com a paràmetre un Programador, i si ha realitzat alguna tasca i ha estat contat anteriorment, el desmarca per a que pugui tornar a fer alguna tasca. Aquest Operador es necessari, ja que per forçar que el planificador conti a tots el programadors, posarem com a goal que cap programador estigui per contar.

Modelado de los problemas

Los problemas se modelan de la siguiente manera:
En cada problema tenemos que tener ciertos programadores y ciertas tareas. Su número no tiene porqué coincidir. Estos serán los únicos objetos que tendremos.
Para definir el estado inicial del problema debemos primero de todo inicializar el tiempo total (ttime) y número de programadores (nprog) a 0, ya que son los fluentes que trataremos de minimizar.
Seguido de esto inicializaremos las tareas. Todas las tareas deben de tener las siguientes predicados e inicialización de fluentes:

- (tBasica ?t) → esto indica que la tarea es básica. Todas las tareas son básicas inicialmente.
- (= (nivel-tarea ?t) N) → N =[1..3]
- (= (tiempo-tarea ?) N) → N positivo.

Por otro lado, por cada programador habrá que declarar lo siguiente:

- (= (nivel-programador ?p) N) → N = [1..3]
- (= (calidad ?p) N) → N = [1,2]
- (= (tareas-programador ?p) 2)

Como objetivo siempre deberemos poner que todas las tareas han sido realizadas, es decir, (forall (?t - Tarea) (finalizada ?t)). Además, también deberemos poner que no puede quedar ningún programador por contar. De esta manera forzamos a que el planificador cuenta a todos los programadores que realizan alguna tarea: (forall (?p - Programador) (not (por-contar ?p))).

Por último, pondremos que la métrica minimice. En esta métrica deberemos poner una suma ponderada de los dos fluentes. Si quisiéramos utilizar este planificador sin querer tener en cuenta los dos fluentes no tendría mucho sentido, ya que se ha ideado para que minimice los dos.

Desarrollo de los modelos

Para el desarrollo de este proyecto hemos utilizado el método de prototipado rápido e incremental. Primero resolvimos el problema básico, y a partir de ahí fuimos

haciendo un prototipo para cada extensión, construyendo sobre el anterior y siguiendo la planificación que habíamos decidido.

Juegos de prueba

- Nivel Básico

Al no minimizar, estos juegos de prueba únicamente mirarán que exista un programador que tenga nivel como mínimo un nivel inferior al nivel máximo de todas las tareas. Si existe ese programador, seguramente haga todas las tareas.
No hace falta explicar mucho los juegos de prueba que metemos porque son triviales.

- **juego de pruebas 1**

Entrada:

```
(define (problem test0)

 (:domain mintask)

 (:objects
  p1 p2 p3 p4 - Programador
  t1 t2 t3 - Tarea
 )

 (:init
  (tBasica t1)
  (tBasica t2)
  (tBasica t3)

  ;Programador 1
  (= (nivel-programador p1) 3)
  ;Programador 2
  (= (nivel-programador p2) 2)
  ;Programador 3
  (= (nivel-programador p3) 2)
  ;Programador 4
  (= (nivel-programador p4) 2)

  ;Tarea 1
  (= (nivel-tarea t1) 3)
  ;Tarea 2
  (= (nivel-tarea t2) 2)
  ;Tarea 3
  (= (nivel-tarea t3) 1)

 )
```

```
 (:goal
  (forall (?t - Tarea) (finalizada ?t))
 )
)
```

Salida:

```
ff: parsing domain file
domain 'MINTASK' defined
 ... done.
ff: parsing problem file
problem 'TEST0' 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: COMPLETAR-TAREA-BASICA-CON-SUPLEMENTO T1 P4
        1: COMPLETAR-TAREA-BASICA T2 P4
        2: COMPLETAR-TAREA-BASICA T3 P4


time spent:    0.00 seconds instantiating 24 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 6 facts and 12 actions
                 0.00 seconds creating final representation with 6 relevant facts, 0
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 7 states, to a max depth of 0
               0.00 seconds total time
```

En este caso vemos como un único programador ha hecho todas las tareas, y le ha dado igual que la tarea tuviera un nivel más alto que el nivel del programador. Se puede ver el código en la carpeta de entrega. En este caso nos abstendremos de hacer otro juego de pruebas para este nivel, ya que el planificador en esta fase de desarrollo no muestra casi ningún signo de inteligencia.

- **Extensión 1**

En esta extensión necesitaremos que se cumpla la misma condición que en la primera, pero en lugar de que exista un solo programador que la cumpliera, que existan mínimo dos, ya que si solo existiera uno, no habría ningún otro que pudiera revisar esa tarea.

- **Primer juego de pruebas**

Este juego de pruebas es el mismo al anterior. Como existe más de un programador que con nivel de habilidad a 2 o mayor, elegirá a cualquiera de ellos para hacer las tareas. Como todavía no minimizamos nada, le dará igual la calidad de los que utilice.

Entrada:

```
(define (problem test0)

 (:domain mintask)

 (:objects
  p1 p2 p3 p4 - Programador
  t1 t2 t3 - Tarea
 )

 (:init
  (= (ttime) 0)
  (tBasica t1)
  (tBasica t2)
  (tBasica t3)

  ;Programador 1
  (= (nivel-programador p1) 3)
  (= (calidad p1) 2)
  ;Programador 2
  (= (nivel-programador p2) 2)
  (= (calidad p2) 2)
  ;Programador 3
  (= (nivel-programador p3) 2)
  (= (calidad p3) 2)
  ;Programador 4
  (= (nivel-programador p4) 2)
  (= (calidad p4) 1)

  ;Tarea 1
  (= (nivel-tarea t1) 3)
  (= (tiempo-tarea t1) 5)
  ;Tarea 2
  (= (nivel-tarea t2) 2)
  (= (tiempo-tarea t2) 5)
  ;Tarea 3
```

```
   (= (nivel-tarea t3) 1)
   (= (tiempo-tarea t3) 5)

 )

 (:goal
   (forall (?t - Tarea) (finalizada ?t))
 )

)
```

Salida:

```
ff: parsing domain file
domain 'MINTASK' defined
 ... done.
ff: parsing problem file
problem 'TEST0' 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: COMPLETAR-TAREA-BASICA T1 P1
        1: COMPLETAR-TAREA-POR-REVISAR T1 P1 P4
        2: COMPLETAR-TAREA-BASICA T2 P1
        3: COMPLETAR-TAREA-POR-REVISAR T2 P1 P4
        4: COMPLETAR-TAREA-BASICA T3 P1
        5: COMPLETAR-TAREA-POR-REVISAR T3 P1 P4


time spent:    0.00 seconds instantiating 60 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 18 facts and 48 actions
                0.00 seconds creating final representation with 18 relevant facts, 1
relevant fluents
               0.00 seconds computing LNF
```

```
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 40 states, to a max depth of 0
0.00 seconds total time
```

En este caso vemos que utiliza los programadores 1 y 4, pero podría haber utilizado a cualquiera de ellos.

- **Segundo juego de pruebas**

En este juego de pruebas vamos a hacer que la condición no se cumpla. Únicamente vamos a poner a un programador de nivel 3 y todos los demás de nivel 1, y una tarea de nivel 3. Al no haber nadie para poder revisar esa tarea, ya que el programador que la realiza no puede revisarla, fallará.

Entrada:

```
(define (problem test1)
 (:domain mintask)
 (:objects
  p1 p2 p3 p4 - Programador
  t1 t2 t3 - Tarea
 )

 (:init
  (= (ttime) 0)
  (tBasica t1)
  (tBasica t2)
  (tBasica t3)

  ;Programador 1
  (= (nivel-programador p1) 3)
  (= (calidad p1) 2)
  ;Programador 2
  (= (nivel-programador p2) 1)
  (= (calidad p2) 2)
  ;Programador 3
  (= (nivel-programador p3) 1)
  (= (calidad p3) 2)
  ;Programador 4
  (= (nivel-programador p4) 1)
  (= (calidad p4) 1)

  ;Tarea 1
  (= (nivel-tarea t1) 3)
  (= (tiempo-tarea t1) 5)
  ;Tarea 2
  (= (nivel-tarea t2) 2)
  (= (tiempo-tarea t2) 5)
  ;Tarea 3
```

```
   (= (nivel-tarea t3) 1)
   (= (tiempo-tarea t3) 5)

 )

 (:goal
   (forall (?t - Tarea) (finalizada ?t))
 )

)
```

Salida:

```
ff: parsing domain file
domain 'MINTASK' defined
 ... done.
ff: parsing problem file
problem 'TEST1' 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 60 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 18 facts and 36 actions
                0.00 seconds creating final representation with 18 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
```

Como esperábamos, no tiene solución.

- **Extensión 2**

En esta extensión ya es cuando el planificador empezará a hacer algo inteligente. En esta extensión minimizaremos el número de horas totales. De esta manera, el planificador tratará de elegir para solucionar una tarea, un programador que tenga un nivel de habilidad igual o superior a una tarea y calidad 1, seguido del mismo con

calidad 2. Si no existiera ninguno que tuviera un nivel de habilidad igual o superior, elegiría a los programadores con un solo nivel inferior y de calidad 1, seguido de los de calidad 2.

- **Primer juego de pruebas**

En este juego de pruebas tenemos la misma entrada que en el anterior, solo que vamos a minimizar el tiempo total. Tenemos una tarea de cada nivel, un programador con nivel 3 y tres con nivel 2. De esos tres programadores con nivel 2, solo uno tiene calidad 1. Así que esperamos que el programador de nivel 3 resuelva la tarea de nivel 3, y las otras dos tareas las resuelva el programador con mejor calidad. Como todavía no nos importa la cantidad de tareas que haga cada programador, siempre se utilizará el mejor programador posible.

Entrada:

```
(define (problem test0)

 (:domain mintask)

 (:objects
   p1 p2 p3 p4 - Programador
   t1 t2 t3 - Tarea
 )

 (:init
   (= (ttime) 0)
   (tBasica t1)
   (tBasica t2)
   (tBasica t3)

   ;Programador 1
   (= (nivel-programador p1) 3)
   (= (calidad p1) 2)
   ;Programador 2
   (= (nivel-programador p2) 2)
   (= (calidad p2) 2)
   ;Programador 3
   (= (nivel-programador p3) 2)
   (= (calidad p3) 2)
   ;Programador 4
   (= (nivel-programador p4) 2)
   (= (calidad p4) 1)

   ;Tarea 1
   (= (nivel-tarea t1) 3)
   (= (tiempo-tarea t1) 5)
   ;Tarea 2
   (= (nivel-tarea t2) 2)
```

```
   (= (tiempo-tarea t2) 5)
   ;Tarea 3
   (= (nivel-tarea t3) 1)
   (= (tiempo-tarea t3) 5)

 )

 (:goal
   (forall (?t - Tarea) (finalizada ?t))
 )
 (:metric minimize (ttime))

)
```

Salida:

```
ff: parsing domain file
domain 'MINTASK' defined
 ... done.
ff: parsing problem file
problem 'TEST0' defined
 ... done.


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

advancing to distance:    6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: COMPLETAR-TAREA-BASICA T1 P1
        1: COMPLETAR-TAREA-BASICA T2 P4
        2: COMPLETAR-TAREA-BASICA T3 P4
        3: COMPLETAR-TAREA-POR-REVISAR T3 P4 P3
        4: COMPLETAR-TAREA-POR-REVISAR T2 P4 P3
        5: COMPLETAR-TAREA-POR-REVISAR T1 P1 P4


time spent:    0.00 seconds instantiating 60 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 18 facts and 48 actions
```

```
                0.00 seconds creating final representation with 18 relevant facts, 1
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 34 states, to a max depth of 0
               0.00 seconds total time
```

No es difícil ver que siempre se usa el programador que tenía mejor calidad pero siempre con un nivel de habilidad mayor o igual a la tarea.
Como el problema sigue siendo trivial, no pondremos otro juego de pruebas, ya que o veremos como usa los programadores tal y como lo hemos explicado. En el caso que no pueda elegirlos de esa manera, es que no hay solución posible.

- Extensión 3

En esta extensión se introduce el concepto de número de tareas que puede realizar un programador. Tal y como pone en el problema, el número de tareas siempre será dos, pero el dominio está modelado de manera que cada programador tiene el número de tareas que podría hacer.

- **Primer juego de pruebas**

En este juego de pruebas tendremos una tarea de nivel 3 y un solo programador de nivel 3, además de diferentes tareas y programadores. De esta manera tendríamos que forzar que la tarea de nivel se haga obligatoriamente por el programador de nivel 3. También habrá que tener en cuenta que para que exista solución hace falta que como mínimo haya el mismo número de programadores que de tareas. Te las tareas no descritas todavía, hay dos que son de nivel 1, así que también esperamos que el programador de nivel 1 y calidad 1 las haga para que minimice el tiempo.

Entrada:

```
(define (problem test0)

 (:domain mintask)

 (:objects
  p1 p2 p3 p4 - Programador
  t1 t2 t3 t4 - Tarea
 )

 (:init
  (= (ttime) 0)
  (tBasica t1)
  (tBasica t2)
  (tBasica t3)
  (tBasica t4)
```

```
   ;Programador 1
   (= (nivel-programador p1) 3)
   (= (calidad p1) 2)
   (= (tareas-programador p1) 2)
   ;Programador 2
   (= (nivel-programador p2) 2)
   (= (calidad p2) 2)
   (= (tareas-programador p2) 2)
   ;Programador 3
   (= (nivel-programador p3) 2)
   (= (calidad p3) 2)
   (= (tareas-programador p3) 2)
   ;Programador 4
   (= (nivel-programador p4) 1)
   (= (calidad p4) 1)
   (= (tareas-programador p4) 2)

   ;Tarea 1
   (= (nivel-tarea t1) 3)
   (= (tiempo-tarea t1) 5)
   ;Tarea 2
   (= (nivel-tarea t2) 2)
   (= (tiempo-tarea t2) 5)
   ;Tarea 3
   (= (nivel-tarea t3) 1)
   (= (tiempo-tarea t3) 5)
   ;Tarea 4
   (= (nivel-tarea t4) 1)
   (= (tiempo-tarea t4) 5)

 )

 (:goal
   (forall (?t - Tarea) (finalizada ?t))
 )
 (:metric minimize (ttime))

)
```

Salida:

```
ff: parsing domain file
domain 'MINTASK' defined
 ... done.
ff: parsing problem file
problem 'TEST0' defined
 ... done.


metric established (normalized to minimize): ((1.00*[RF4](TTIME)) - () + 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*[RF4](TTIME)) - () + 0.00)

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

ff: found legal plan as follows

step    0: COMPLETAR-TAREA-BASICA T1 P1
        1: COMPLETAR-TAREA-BASICA T3 P4
        2: COMPLETAR-TAREA-BASICA T4 P4
        3: COMPLETAR-TAREA-BASICA T2 P1
        4: COMPLETAR-TAREA-POR-REVISAR T1 P1 P2
        5: COMPLETAR-TAREA-POR-REVISAR T2 P1 P2
        6: COMPLETAR-TAREA-POR-REVISAR T3 P4 P3
        7: COMPLETAR-TAREA-POR-REVISAR T4 P4 P3


time spent:    0.00 seconds instantiating 80 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 24 facts and 60 actions
                0.00 seconds creating final representation with 24 relevant facts, 5
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 71 states, to a max depth of 0
               0.00 seconds total time
```

Como se puede observar, la solución obtenida era la esperada y nadie ha hecho más de 2 tareas.

- **Segundo juego de pruebas**

Como podríamos pensar que fue pura casualidad que en el juego de pruebas anterior todos los programadores hicieran como máximo dos tareas, le agregaremos una tarea más de nivel 3 y otra mas de nivel 1. Para que siga pudiendo encontrar una solución, agregaremos dos programadores que serán de nivel 2 y calidad 2. De esta manera, si no estuvieran forzados a hacer 2 tareas como máximo, el programador 4, que es el único que tiene calidad 1 y nivel 1, haría las tres tareas de nivel 1. Veremos que no.

Entrada:

```
(define (problem test0)

 (:domain mintask)

 (:objects
   p1 p2 p3 p4 p5 p6 - Programador
   t1 t2 t3 t4 t5 t6 - Tarea
 )

 (:init
   (= (ttime) 0)
   (tBasica t1)
   (tBasica t2)
   (tBasica t3)
   (tBasica t4)
   (tBasica t5)
   (tBasica t6)

   ;Programador 1
   (= (nivel-programador p1) 3)
   (= (calidad p1) 2)
   (= (tareas-programador p1) 2)
   ;Programador 2
   (= (nivel-programador p2) 2)
   (= (calidad p2) 2)
   (= (tareas-programador p2) 2)
   ;Programador 3
   (= (nivel-programador p3) 2)
   (= (calidad p3) 2)
   (= (tareas-programador p3) 2)
   ;Programador 4
   (= (nivel-programador p4) 1)
   (= (calidad p4) 1)
   (= (tareas-programador p4) 2)
    ;Programador 5
   (= (nivel-programador p5) 2)
   (= (calidad p5) 2)
   (= (tareas-programador p5) 2)
   ;Programador 6
   (= (nivel-programador p6) 2)
   (= (calidad p6) 2)
   (= (tareas-programador p6) 2)

   ;Tarea 1
   (= (nivel-tarea t1) 3)
   (= (tiempo-tarea t1) 5)
   ;Tarea 2
   (= (nivel-tarea t2) 2)
   (= (tiempo-tarea t2) 5)
```

```
  ;Tarea 3
  (= (nivel-tarea t3) 1)
  (= (tiempo-tarea t3) 5)
  ;Tarea 4
  (= (nivel-tarea t4) 1)
  (= (tiempo-tarea t4) 5)
  ;Tarea 5
  (= (nivel-tarea t5) 1)
  (= (tiempo-tarea t5) 5)
  ;Tarea 6
  (= (nivel-tarea t6) 3)
  (= (tiempo-tarea t6) 5)

 )

 (:goal
  (forall (?t - Tarea) (finalizada ?t))
 )
 (:metric minimize (ttime))

)
```

Salida:

```
ff: parsing domain file
domain 'MINTASK' defined
 ... done.
ff: parsing problem file
problem 'TEST0' defined
 ... done.


metric established (normalized to minimize): ((1.00*[RF6](TTIME)) - () + 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*[RF6](TTIME)) - () + 0.00)

advancing to distance:   12
                         11
                         10
                          9
                          8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0
```

```
ff: found legal plan as follows

step    0: COMPLETAR-TAREA-BASICA T1 P1
        1: COMPLETAR-TAREA-BASICA T6 P1
        2: COMPLETAR-TAREA-BASICA T3 P4
        3: COMPLETAR-TAREA-BASICA T4 P4
        4: COMPLETAR-TAREA-BASICA T2 P2
        5: COMPLETAR-TAREA-BASICA T5 P2
        6: COMPLETAR-TAREA-POR-REVISAR T1 P1 P3
        7: COMPLETAR-TAREA-POR-REVISAR T6 P1 P3
        8: COMPLETAR-TAREA-POR-REVISAR T2 P2 P5
        9: COMPLETAR-TAREA-POR-REVISAR T5 P2 P5
       10: COMPLETAR-TAREA-POR-REVISAR T3 P4 P6
       11: COMPLETAR-TAREA-POR-REVISAR T4 P4 P6


time spent:    0.00 seconds instantiating 252 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 48 facts and 204 actions
                0.00 seconds creating final representation with 48 relevant facts, 7
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 220 states, to a max depth of 0
               0.00 seconds total time
```

Podemos observar que en este caso, aunque el programador 4 era el mas afin para completar la tarea 5, no la ha podido realizar ya que ya habí cumplido su cupo de tareas.

- Extensión 4

Por la forma en la que trabaja el planificador, aunque no le pongamos que minimice el número de programadores, afortunadamente siempre minimiza el número de programadores que va a usar. Igualmente veremos como el planificador utiliza los operadores para contar a los programadores.

- **Primer juego de pruebas**

En este juego de pruebas probamos a poner más programadores que tareas. También elegimos la ponderación entre número de horas totales y programadores de la siguiente manera: tres cuartos el número de programadores mas un cuarto del número de horas totales, ya que el número de horas sube mucho más rápido.

Entrada:

(define (problem test0)

```
(:domain mintask)

(:objects
  p1 p2 p3 p4 p5 p6 - Programador
  t1 t2 t3 t4 - Tarea
)

(:init
  (= (ttime) 0)
  (= (nprog) 0)
  (tBasica t1)
  (tBasica t2)
  (tBasica t3)
  (tBasica t4)

  ;Programador 1
  (= (nivel-programador p1) 3)
  (= (calidad p1) 2)
  (= (tareas-programador p1) 2)
  ;Programador 2
  (= (nivel-programador p2) 2)
  (= (calidad p2) 2)
  (= (tareas-programador p2) 2)
  ;Programador 3
  (= (nivel-programador p3) 1)
  (= (calidad p3) 2)
  (= (tareas-programador p3) 2)
  ;Programador 4
  (= (nivel-programador p4) 1)
  (= (calidad p4) 1)
  (= (tareas-programador p4) 2)
  ;Programador 5
  (= (nivel-programador p5) 1)
  (= (calidad p5) 2)
  (= (tareas-programador p5) 2)
  ;Programador 6
  (= (nivel-programador p6) 1)
  (= (calidad p6) 1)
  (= (tareas-programador p6) 2)

  ;Tarea 1
  (= (nivel-tarea t1) 3)
  (= (tiempo-tarea t1) 5)
  ;Tarea 2
  (= (nivel-tarea t2) 2)
  (= (tiempo-tarea t2) 5)
  ;Tarea 3
  (= (nivel-tarea t3) 2)
  (= (tiempo-tarea t3) 5)
  ;Tarea 4
  (= (nivel-tarea t4) 1)
```

```
   (= (tiempo-tarea t4) 5)

 )

 (:goal
  (and
     (forall (?t - Tarea) (finalizada ?t))
     (forall (?p - Programador) (not (por-contar ?p)))
  )
 )
 (:metric minimize (+ (* (nprog) 15) (* (ttime) 5)))
)
```

Salida:

```
ff: parsing domain file
domain 'MINTASK' defined
 ... done.
ff: parsing problem file
problem 'TEST0' defined
 ... done.


metric established (normalized to minimize): ((15.00*[RF6](NPROG)5.00*[RF7](TTIME))
- () + 0.00)

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((15.00*[RF6](NPROG)5.00*[RF7](TTIME)) - () + 0.00)

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

ff: found legal plan as follows

step    0: COMPLETAR-TAREA-BASICA-CON-SUPLEMENTO T2 P6
        1: COMPLETAR-TAREA-BASICA T3 P2
        2: COMPLETAR-TAREA-BASICA T1 P1
        3: CONTAR-PROGRAMADOR P1
        4: COMPLETAR-TAREA-BASICA T4 P1
        5: CONTAR-PROGRAMADOR P2
        6: CONTAR-PROGRAMADOR P6
        7: DESMARCAR-PROGRAMADOR P1
        8: COMPLETAR-TAREA-POR-REVISAR T4 P1 P3
```

```
 9: CONTAR-PROGRAMADOR P3
10: COMPLETAR-TAREA-POR-REVISAR T1 P1 P2
11: COMPLETAR-TAREA-POR-REVISAR T2 P6 P3
12: COMPLETAR-TAREA-POR-REVISAR T3 P2 P6
13: DESMARCAR-PROGRAMADOR P6
14: DESMARCAR-PROGRAMADOR P3
15: DESMARCAR-PROGRAMADOR P2


time spent:    0.00 seconds instantiating 180 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 56 facts and 132 actions
                0.00 seconds creating final representation with 56 relevant facts, 8
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.01 seconds searching, evaluating 469 states, to a max depth of 0
               0.01 seconds total time
```

Como ponderamos mas el numero de programadores que el numero de horas, vemos como el planificador no ha encontrado el planning optimo. Esto se puede ver al darse cuenta que para la tarea 4, no ha elegido al programador 4, que tenía una calidad superior al 1.

- **Segundo juego de pruebas**

En este juego de pruebas será el mismo que el anterior, pero cambiando la ponderación de los fluentes. En este caso le daremos más peso al numero de horas que al número de programadores.

Entrada:

```
(define (problem test0)

 (:domain mintask)

 (:objects
  p1 p2 p3 p4 p5 p6 - Programador
  t1 t2 t3 t4 - Tarea
 )

 (:init
  (= (ttime) 0)
  (= (nprog) 0)
  (tBasica t1)
  (tBasica t2)
  (tBasica t3)
```

```
  (tBasica t4)

  ;Programador 1
  (= (nivel-programador p1) 3)
  (= (calidad p1) 2)
  (= (tareas-programador p1) 2)
  ;Programador 2
  (= (nivel-programador p2) 2)
  (= (calidad p2) 2)
  (= (tareas-programador p2) 2)
  ;Programador 3
  (= (nivel-programador p3) 1)
  (= (calidad p3) 2)
  (= (tareas-programador p3) 2)
  ;Programador 4
  (= (nivel-programador p4) 1)
  (= (calidad p4) 1)
  (= (tareas-programador p4) 2)
  ;Programador 5
  (= (nivel-programador p5) 1)
  (= (calidad p5) 2)
  (= (tareas-programador p5) 2)
  ;Programador 6
  (= (nivel-programador p6) 1)
  (= (calidad p6) 1)
  (= (tareas-programador p6) 2)

  ;Tarea 1
  (= (nivel-tarea t1) 3)
  (= (tiempo-tarea t1) 5)
  ;Tarea 2
  (= (nivel-tarea t2) 2)
  (= (tiempo-tarea t2) 5)
  ;Tarea 3
  (= (nivel-tarea t3) 2)
  (= (tiempo-tarea t3) 5)
  ;Tarea 4
  (= (nivel-tarea t4) 1)
  (= (tiempo-tarea t4) 5)

 )

 (:goal
  (and
     (forall (?t - Tarea) (finalizada ?t))
     (forall (?p - Programador) (not (por-contar ?p)))
  )
 )
 (:metric minimize (+ (* (nprog) 5) (* (ttime) 15)))
)
```

Salida:

```
ff: parsing domain file
domain 'MINTASK' defined
 ... done.
ff: parsing problem file
problem 'TEST0' defined
 ... done.


metric established (normalized to minimize): ((5.00*[RF6](NPROG)15.00*[RF7](TTIME))
- () + 0.00)

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((5.00*[RF6](NPROG)15.00*[RF7](TTIME)) - () + 0.00)

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

ff: found legal plan as follows

step    0: COMPLETAR-TAREA-BASICA-CON-SUPLEMENTO T2 P6
        1: COMPLETAR-TAREA-BASICA T3 P2
        2: COMPLETAR-TAREA-BASICA T1 P1
        3: COMPLETAR-TAREA-BASICA T4 P4
        4: CONTAR-PROGRAMADOR P4
        5: CONTAR-PROGRAMADOR P1
        6: CONTAR-PROGRAMADOR P2
        7: CONTAR-PROGRAMADOR P6
        8: COMPLETAR-TAREA-POR-REVISAR T3 P2 P1
        9: COMPLETAR-TAREA-POR-REVISAR T1 P1 P2
       10: COMPLETAR-TAREA-POR-REVISAR T2 P6 P4
       11: COMPLETAR-TAREA-POR-REVISAR T4 P4 P6
       12: DESMARCAR-PROGRAMADOR P6
       13: DESMARCAR-PROGRAMADOR P4
       14: DESMARCAR-PROGRAMADOR P2
       15: DESMARCAR-PROGRAMADOR P1


time spent:    0.00 seconds instantiating 180 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 56 facts and 132 actions
```

```
                0.00 seconds creating final representation with 56 relevant facts, 8
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 165 states, to a max depth of 0
               0.00 seconds total time
```

En este caso podemos ver que si ha elegido al programador 4 para la tarea 4, minimizando el número de horas totales a la vez que el número de programadores.

Generador aleatorio

Los generadores aleatorios que tenemos están dentro los directorios **extension 2** y la carpeta **extension 4**, ya que no sabíamos si el generador debía ser solo para la extensión 2.
Se compilan con el comando g++ -std=c++11 -o generador generador.cpp
Para ejecutarlos se puede hacer con un argumento, número usado para decidir el número de tareas que tendrá el problema generado aleatoriamente, o sin argumento, que en ese caso el número de tareas será aleatorio.