Asignación de tareas a programadores usando PDDL

IA FIB @ UPC

Carlos Bergillos, Adrià Cabeza, Roger Vilaseca

Junio 3, 2019

Índice

1 Introducción

En esta práctica tendremos que resolver un problema de planificación usando *Metric FastForward* (*Metric-FF*). Éste es un planificador que nos permite resolver problemas de planificación definidos en lenguaje PDDL (*Planning Domain Definition Language*).

La modelización de los problemas PDDL se divide conceptualmente en dos partes:

- **Modelización del dominio:** Contiene las definiciones del "mundo" del problema. Definiremos aquí todo aquello que es invariante para cualquier instanciación del problema, por ejemplo los tipos, las acciones y los predicados disponibles.
- **Modelización del problema:** Contiene la información concreta del problema para el dominio dado. Pueden existir infinitos problemas para un mismo dominio. Por ejemplo, aquí definiremos los objetos que intervienen en el problema concreto, y definiremos el estado inicial y el estado objetivo.

Dada un modelizacion del dominio y una modelizacion del problema, el planificador se encargará de buscarnos un plan válido (si existiera), que contendrá una secuencia de acciones que nos llevarán del estado inicial al estado inicial.

Además, para ciertas extensiones de la práctica, no solo nos interesará buscar un plan válido cualquiera, sino que nos interesará encontrar un plan que optimice un criterio dado (una métrica).

2 Descripción del problema

En esta práctica nos encargaremos de planificar un proyecto de programación de gran envergadura. Deberemos repartir un conjunto de tareas a realizar entre los programadores disponibles.

En concreto, disponemos de T tareas de programación, cada una de ellas tiene un grado de dificultad asignado (de 1 a 3), y un tiempo estimado de realización estimado (en horas).

También disponemos de un conjunto P de programadores. Cada uno de ellos tiene asignado un grado de habilidad (de 1 a 3), que nos indica lo mucho o poco que éste está capacitado para resolver las tareas.

No queremos asignar a los programadores tareas mucho más difíciles de lo que para ellos están capacitados. En concreto, a un programador solo le podremos asignar tareas de como mucho una unidad más de dificultad de lo que nos indique su habilidad. En caso de que sea necesario asignar a un programador una tarea más difícil que su capacidad, la duración de la realización de la tarea se verá incrementada en 2 horas.

Además, todas las tareas deberán ser revisadas, para ello, hará falta una nueva tarea adicional. Esta nueva tarea de revisión será de la misma dificultad que la tarea original. Los programadores tienen también asociada una calidad (de 1 a 2). Si la tarea original es realizada por un programador de calidad 1, la nueva tarea de revisión durará 1 hora, si en cambio el programador era de calidad 2, la nueva tarea de revisión durará 2 horas. Para evitar una recursividad infinita, las nuevas tareas de revisión no requerirán a su vez de revisión, y su tiempo de realización no se verá penalizado por la habilidad del programador que la realiza.

3 Nivel básico

En esta primera versión problema, solamente tendremos como objetivo que todas las tareas queden asignadas a algún programador, sin tener en cuenta las tareas de revisión asociadas, y sin intentar optimizar ningún criterio.

3.1 Dominio

3.1.1 Variables

Para la correcta resolución de este problema de planificación hemos visto conveniente trabajar con variables con tipo. En concreto, hemos necesitado 2 tipos, los cuáles hemos llamado **programador** y **tarea**.

- **programador:** Se utilizará para las variables que correspondan a cada programador del conjunto P.
- **tarea:** Se utilizará para las variables que correspondan a cada tarea del conjunto T.

Para las próximas extensiones ya no se requieren más cambios en las variables, esta será la configuración definitiva.

3.1.2 Funciones

- **habilidadProgramador:** Esta función nos servirá para definir y conocer la habilidad de un programador determinado. La habilidad de un programador se representa con un número entero 1, 2 o 3.
- **dificultadTarea:** Esta función nos servirá para definir y conocer la dificultad de una tarea determinada. La dificultad de una tarea se representa con un número entero 1, 2 o 3.

3.1.3 Predicados

- `(asignacion ?x - programador ?y - tarea)`

  Al programador `x` se le ha sido asignada la tarea `y`, por lo tanto, éste deberá ser el encargado de realizarla.

- `(tareaAsignada ?x - tarea)`

  Nos indica que la tarea `x` ha sido asignada a algún programador. Nos servirá para evitar asignar dos veces la misma tarea.

3.1.4 Acciones

- **asignar:** Esta acción nos sirve para asignar una tarea a un programador.

  Parámetros

  - `?p - programador`
  - `?t - tarea`

  Precondición

  - La tarea `t` no está asignada.
  - La habilidad del programador `p` debe ser más grande o igual que la dificultad de la tarea `p` más uno.

  Postcondición

  - La tarea `t` ha sido asignada al programador `p`.

3.2 Problema

3.2.1 Objetos

Para la modelización del problema tenemos que definir el conjunto de tareas y el conjunto de programadores de los que disponemos, declarando tantas variables (objetos) como requiramos.

3.2.2 Estado inicial

Para cada tarea:

- Definimos la dificultad de la tarea (1, 2 o 3).

Y para cada programador:

- Definimos la habilidad del programador (1, 2 o 3).

3.2.3 Estado final

Para el estado final, requerimos que todas las tareas estén asignadas:

```
(:goal (forall (?t - tarea) (tareaAsignada ?t))))
```

3.3 Juegos de prueba

4 Extensión 1

En esta extensión, vamos a añadir a nuestro programa la revisión de tareas. La revisión de tareas consiste en que cada vez que algún programador realice una tarea, esta deberá ser revisada por otro programador. La dificultad de la revisión será la misma que de la tarea y por lo tanto utilizaremos el mismo criterio para asignar un programador, para revisar que en la sección 3.

4.1 Dominio

4.1.1 Funciones

Para esta extensión vamos a utilizar las mismas funciones que en el apartado anterior.

4.1.2 Predicados

- **asignacionTarea:** Equivalente al predicado "asignacion" en la sección 3.1.3.
- **asignacionRevision:** Predicado utilizado para asignar a una tarea un programador para que este la revise.
- **tareaAsignada:** Igual que en la sección 3.1.3.
- **tareaRevisada:** Predicado para conocer si la tarea en questión ha sido revisada por algún programador.

4.1.3 Acciones

- **asignar:** Esta acción és equivalente a la acción "asignar" explicada en la sección 3.1.4. Donde el único cambio es el cambio de nombre del predicado de "asignacion" a "asignacionTarea".
- **revisar:** El objetivo de esta acción es a partir de las tareas ya asignadas, asignar una revisión a todas la tareas con un programador diferente del que ha tenido asignada la tarea, el cual debe tener habilidad suficientemente grande para poder revisarla.

  Parámetros

  - `?p - programador`
  - `?t - tarea`

  Precondición

  Para poder realizar esta acción se debe cumplir:

  - La tarea debe estar asignada.
  - La tarea no debe estar revisada
  - La habilidad del programador debe ser más grande o igual que la dificultad de la tarea más uno.
  - El programador que revisa la tarea no puede ser el mismo que el que le ha sido asignada.

(and (tareaAsignada ?t) (not (tareaRevisada ?t)) (¿= (habilidadProgramador ?p) (- (dificultadTarea ?t) 1)) (not (asignacionTarea ?p ?t)))

Postcondición

- Un programador será el revisor de la tarea.
- La tarea estará revisada.

(and (asignacionRevision ?p ?t) (tareaRevisada ?t))

4.2 Problema

4.2.1 Estado inicial

En este caso el estado inicial será equivalente al de la sección 3.2.2.

4.2.2 Estado final

El estado final remplazaremos la comprobación de de que todas las tareas estén asignadas, por la comprobación de que todas las tareas estén revisadas.

(forall (?t - tarea) (tareaRevisada ?t))

4.3 Juegos de prueba

5 Extensión 2

En la siguiente extensión, partiendo de la anterior extensión, tenemos como objetivo minimizar el tiempo total que se usa en resolver todas las tareas. El tiempo total se interpreta como la suma de las horas de las tareas, tanto las principales como las de revisión.
En este caso, minimizar significa otorgar las tareas dependiendo de la calidad del programador para que las tareas de revisión sean menos duraderas.

5.1 Dominio

5.1.1 Funciones

Puesto a que esta es la primera extensión donde observamos el concepto de tiempo, hemos tenido que añadir funciones además de las mencionadas anteriormente.

- **tiempoTarea**: Esta función nos servirá para definir y conocer el tiempo asociado a una tarea T.
- **tiempoTotal**: Esta función nos servirá para definir y conocer la suma de duración de todas las tareas. Es decir el tiempo total que representa el conjunto de tareas.

5.1.2 Predicados

En cuanto a lo que concierne a los predicados, utilizamos los mismos que en la extensión anterior.

5.1.3 Acciones

- **asignarDificil**
- **asignar**

5.2 Problema

5.2.1 Estado inicial

Para cada tarea:

- Definimos cuanto tiempo tarda en hacerse.

Además inicializamos el valor de *tiempoTotal*.

5.2.2 Estado final

Para el estado final, de la misma manera que en la anterior extensión, requerimos que todas las tareas esten revisadas. No obstante, en este caso añadimos una métrica: **minimizar el valor de tiempoTotal**.

Anexos

A Generador de juegos de prueba

Generar a mano varios casos de prueba es una tarea ardua y monótona que puede ser fácilmente automatizable, así pues, para poder probar varios valores rápidamente, decidimos crear un generador de juegos de prueba. En nuestro caso hemos decidido utilizar el lenguaje de scripting **Python** para generar casos de prueba automáticamente.

Nuestro generador de pruebas admite tres parámetros distintos:

- *–programadores P*, donde P es el número de programadores que queremos
- *–tareas T*, donde T es el número de programadores que queremos
- *–extension E* donde E es la extensión que deseamos utilizar.

Finalmente se genera un documento llamado *fichero_generadoN.pddl* donde la N es la extensión que hemos seleccionado en los argumentos del programa.