Universidad Internacional Menéndez Pelayo

Máster en Investigación en Inteligencia Artificial

Planificación automática
Práctica 1

AlfonsoLozana/UIMP-PA-Practica1

Autor:
Alfonso Lozana Cueto

1 Introducción

En esta práctica nos enfocamos en problemas PDDL, utilizando como punto de partida dos dominios predefinidos: "Robots" y "Satellite". El objetivo principal es modificar el dominio de los Robots y generar nuevos problemas dentro de este contexto actualizado. Posteriormente, emplearemos el planificador Metric-FF para resolver los problemas en los tres dominios.

Este informe detalla los cambios realizados en el dominio y los problemas, así como los resultados obtenidos mediante las ejecuciones del planificador Metric-FF. En la última sección del documento, se analizarán los resultados obtenidos.

2 Nota importante

Al terminar de realizar la práctica y tras hablar con la profesora Eva Onaindia, me di cuenta de que no se podían utilizar PDDL2.1. Por lo tanto, en la primera parte de la práctica se presenta una solución con PDDL2.1, pero al final de este documento se presenta una solución con PDDL sin ":fluents". A nivel de conclusiones, no es muy distinto utilizar uno u otro.

3 Cambios en el Dominio y Problema

3.1 Cambios en el Dominio

Dado el dominio de *Robots*, se pide realizar una serie de modificaciones:

- Definir dos subtipos de robots: *robots-brazos* y *robots-cesta*.
- *Robots-brazos*: Pueden levantar objetos (*pobject*) de sitios (*location*) y dejarlos en otro lugar. Además, al no llevar una cesta, su capacidad máxima es de dos objetos, uno en cada brazo.
- *Robots-cesta*: Carecen de brazos, por lo que no pueden levantar o dejar objetos directamente. Dependen de los *robots-brazos* para introducir o quitar objetos en la cesta. Además, pueden contener un número infinito de objetos.
- Añadir acción adicional a los *robots-brazo*: dar o tomar objetos de otros robots.

Para llevar a cabo estos cambios, siempre se tuvo en cuenta la simplificación y legibilidad del código desarrollado. Tras varias iteraciones, estos fueron los cambios realizados:

1. Se añadió el tipo *:fluents* para controlar el número de brazos que tiene un robot de tipo robot-brazo.
2. Se introdujeron dos nuevos tipos: *robot-brazo* y *robot-cesta* como subtipos de *robot*.

3. Se implementaron nuevos predicados para dar soporte a las funcionalidades añadidas.

   - `(in-cesta ?r - robot ?p - pobject)`: indica si un objeto está en una cesta.
   - `(has-brazo ?r - robot)`: indica si un robot tiene brazos.

4. También se añadió una nueva función para indicar cuántos brazos tiene disponible un robot:

   ```
   (brazos-libres ?r - robot)
   ```

En cuanto a las acciones, los cambios son los siguientes:

- En la acción `MOVE`, no se tuvo que realizar ninguna modificación.
- En la acción `PICK-UP`, se cambió el tipado de los parámetros, ya que son funciones exclusivas para *robot-brazo*. Además, se añadió una precondición `> (brazos-libres ?rb) 0` para controlar los brazos disponibles que tiene un robot, así como un efecto que ocupa un brazo `(decrease (brazos-libres ?rb) 1)`.

```
(:action PICK-UP
    :parameters (?rb - robot-brazo ?l - location ?p - pobject)
    :precondition (and (at-robot ?rb ?l)
                       (> (brazos-libres ?rb) 0)
                       (at-pobject ?p ?l))
    :effect (and (not (at-pobject ?p ?l))
                 (decrease (brazos-libres ?rb) 1)
                 (holding ?rb ?p))
)
```

- En la acción `PUT-DOWN` se hicieron cambios semejantes. Primero se especificó el tipo de robot y luego en los efectos se añadió `(increase (brazos-libres ?rb) 1)` para liberar el brazo.

```
(:action PUT-DOWN
    :parameters (?rb - robot-brazo ?l - location ?p - pobject)
    :precondition (and (at-robot ?rb ?l)
                       (holding ?rb ?p))
    :effect (and (not (holding ?rb ?p))
                 (increase (brazos-libres ?rb) 1)
                 (at-pobject ?p ?l))
)
```

- Se creó una nueva acción denominada `GIVE`, la cual permite que un *robot-brazo* pase objetos a otro robot, ya sea un *robot-brazo* o un *robot-cesta*.

  La primera condición que debe cumplirse es que ambos robots deben estar en el mismo lugar. Además, si el robot destino tiene brazos, es necesario que el robot destino tenga al menos un brazo libre.

  En cuanto a los efectos, el brazo original deja de sostener el objeto y se libera un brazo. Si el robot destino es un robot-brazo, el objeto pasa a ser sostenido por este robot y se decrementa el número de brazos disponibles. Por otro lado, si el destino es un robot cesta, el objeto pasa a estar dentro de la cesta.

  El código de la acción es el siguiente:

```
(:action GIVE
    :parameters (?rb - robot-brazo ?r - robot ?p - pobject ?l - location)
    :precondition (and (holding ?rb ?p)
                       (at-robot ?rb ?l)
                       (at-robot ?r ?l)
                       (or (not (has-brazo ?r))
                           (and (has-brazo ?r) (> (brazos-libres ?r) 0))))
    :effect (and
                (not (holding ?rb ?p))
                (increase (brazos-libres ?rb) 1)
                (when
                    (has-brazo ?r)
                    (and (decrease (brazos-libres ?r) 1)
                        (holding ?r ?p)))
                (when
                    (not (has-brazo ?r))
                    (in-cesta ?r ?p))
            )
)
```

- La acción `TAKE` permite a un *robot-brazo* tomar un objeto de otro robot.

  Las condiciones son que ambos robots estén en el mismo lugar y que el *robot-brazo* tenga un brazo disponible. Si el otro robot es también un *robot-brazo*, este debe estar sujetando el objeto deseado. En caso contrario, si es un *robot-cesta*, el objeto debe estar dentro de la cesta de ese robot.

  En cuanto a los efectos, el objeto empezara a estar sujeto por el *robot-brazo* y se le ocupará un brazo. Si el origen es otro *robot-brazo*, este dejará de sujetar el objeto y se le liberará un brazo. Si por el contrario, es un *robot-cesta*, el objeto dejará de estar dentro de la cesta del mismo.

  El código de la acción es el siguiente:

```
(:action TAKE
    :parameters (?rb - robot-brazo ?r - robot ?p - pobject ?l - location)
    :precondition (and (at-robot ?rb ?l)
                       (at-robot ?r ?l)
                       (> (brazos-libres ?rb) 0)
                       (or (and (has-brazo ?r) (holding ?r ?p))
                           (and (not (has-brazo ?r)) (in-cesta ?r ?p))))
    :effect (and
                (holding ?rb ?p)
                (decrease (brazos-libres ?rb) 1)
                (when
                    (has-brazo ?r)
                    (and
                        (increase (brazos-libres ?r) 1)
                        (not (holding ?r ?p))))
                (when
                    (not (has-brazo ?r))
                    (not (in-cesta ?r ?p)))
            )
)
```

3.2 Cambios en el problema

Se modificó el dominio del problema de *robots* a *domain-robots* y se añadieron un robot de tipo *robot-brazo* y otro de tipo *robot-cesta*, eliminando el robot de tipo *robot*. También se añadió un nuevo objeto.

Luego, en la inicialización, se estableció en dos el número de brazos disponibles del *robot-brazo*. También se añadió la posición de los objetos y los nuevos robots, junto con la inicialización de *has-brazo* en el caso del *robotB0* (*robot-brazo*). Por último, se añadieron los objetivos.

En el siguiente código, se pueden ver los cambios realizados:

```
(define (problem robots)
  (:domain (domain-robots)
  (:objects l01 l02 l03 l04 l05 l06 l07 l08 l09 l10 - location
  (robotB0 - robot-brazo)
  (robotC0 - robot-cesta)
  (pobject0 - pobject)
  (pobject1 - pobject)
  (:init
     ;FUNCIONES
     (= (brazos-libres robotB0) 2)

     (at-robot robotB0 l01)
     (at-robot robotC0 l10)
```

```
    (has-brazo robotB0)
    (at-pobject pobject0 l02)
    (at-pobject pobject1 l03)
   ;relaciones de las localizaciones
   (connected l01 l02)
   (connected l02 l01)
   (connected l03 l02)
   (connected l02 l03)
   (connected l04 l02)
   ...
 (:goal (and
           (at-pobject pobject0 l08)
           (at-pobject pobject1 l04)
 ))
```

4 Resultados

Se han resuelto todos los problemas a excepción de *pfile22.pddl*, del dominio de satélites, cuyo tiempo de ejecución excedió los 10 minutos. El resto de los problemas se resolvieron satisfactoriamente.

A continuación, se presenta una tabla comparativa que incluye:

- Tiempo de resolución de cada problema.
- Calidad de la solución de cada problema (número de acciones en la solución).
- Número de nodos evaluados en cada solución.
- En el caso del problema de los robots, también se añadió una columna para indicar el número de objetos.

Problema	**Tiempo total (s)**	**Acciones**	**Nodos evaluados**	**Número de objetos**
p01	0.01	5	8	1
p02	0.01	10	19	2
p03	0.01	38	276	6
p04	0.07	37	710	6
p05	0.01	31	64	6
p06	0.29	69	2274	10
p07	1.25	109	7169	15
p08	3.89	149	16564	20
p09	9.81	189	31959	25
p10	21.97	229	54854	30

Table 1: Tabla del dominio de robots

Problema	Tiempo total (s)	Acciones	Nodos evaluados
p15	0	50	375
p16	0.11	52	377
p17	0.11	47	293
p18	0.03	34	174
p19	0.13	72	753
p20	1.21	106	6110
p21	-	-	-

Table 2: Tabla del domino de satélites

4.1 Análisis de los resultados obtenidos

Tras realizar las pruebas se han obtenido diferentes resultados de los que podemos sacar unas cuantas conclusiones. En primer lugar, las soluciones encontradas no son óptimas, por ejemplo para el p03 la solución encontrada es la siguiente:

1. PICK-UP ROBOTB0 L01 POBJECT0
2. MOVE ROBOTB0 L01 L02
3. MOVE ROBOTB0 L02 L05
4. MOVE ROBOTB0 L05 L08
5. PUT-DOWN ROBOTB0 L08 POBJECT0
6. MOVE ROBOTB0 L08 L05
7. MOVE ROBOTB0 L05 L02
8. MOVE ROBOTB0 L02 L01
9. PICK-UP ROBOTB0 L01 POBJECT1
10. MOVE ROBOTB0 L01 L02
11. MOVE ROBOTB0 L02 L05
12. MOVE ROBOTB0 L05 L08
13. PUT-DOWN ROBOTB0 L08 POBJECT1
14. MOVE ROBOTB0 L08 L05
15. MOVE ROBOTB0 L05 L02
16. MOVE ROBOTB0 L02 L01
17. PICK-UP ROBOTB0 L01 POBJECT2

18. MOVE ROBOTB0 L01 L02
19. MOVE ROBOTB0 L02 L05
20. MOVE ROBOTB0 L05 L08
21. PUT-DOWN ROBOTB0 L08 POBJECT2
22. MOVE ROBOTB0 L08 L05
23. MOVE ROBOTB0 L05 L02
24. MOVE ROBOTB0 L02 L01
25. PICK-UP ROBOTB0 L01 POBJECT3
26. MOVE ROBOTB0 L01 L02
27. MOVE ROBOTB0 L02 L05
28. MOVE ROBOTB0 L05 L08
29. PUT-DOWN ROBOTB0 L08 POBJECT3
30. MOVE ROBOTB0 L08 L05
31. MOVE ROBOTB0 L05 L02
32. MOVE ROBOTB0 L02 L01
33. PICK-UP ROBOTB0 L01 POBJECT4
34. PICK-UP ROBOTB0 L01 POBJECT5
35. MOVE ROBOTB0 L01 L02
36. MOVE ROBOTB0 L02 L05
37. MOVE ROBOTB0 L05 L08
38. PUT-DOWN ROBOTB0 L08 POBJECT4
39. PUT-DOWN ROBOTB0 L08 POBJECT5

Pero esa no es la solución óptima. Por ejemplo, la siguiente solución tiene 8 pasos menos:

1. PICK-UP ROBOTB0 L01 POBJECT0
2. GIVE ROBOTB0 ROBOTC0 POBJECT0
3. PICK-UP ROBOTB0 L01 POBJECT1
4. GIVE ROBOTB0 ROBOTC0 POBJECT1

5. PICK-UP ROBOTB0 L01 POBJECT2
6. GIVE ROBOTB0 ROBOTC0 POBJECT2
7. PICK-UP ROBOTB0 L01 POBJECT3
8. GIVE ROBOTB0 ROBOTC0 POBJECT3
9. PICK-UP ROBOTB0 L01 POBJECT4
10. GIVE ROBOTB0 ROBOTC0 POBJECT4
11. PICK-UP ROBOTB0 L01 POBJECT5
12. GIVE ROBOTB0 ROBOTC0 POBJECT5
13. MOVE ROBOTB0 L01 L02
14. MOVE ROBOTB0 L02 L05
15. MOVE ROBOTB0 L05 L08
16. MOVE ROBOTC0 L01 L02
17. MOVE ROBOTC0 L02 L05
18. MOVE ROBOTC0 L05 L08
19. TAKE ROBOTB0 ROBOTC0 POBJECT0
20. PUT-DOWN ROBOTB0 L08 POBJECT0
21. TAKE ROBOTB0 ROBOTC0 POBJECT1
22. PUT-DOWN ROBOTB0 L08 POBJECT1
23. TAKE ROBOTB0 ROBOTC0 POBJECT2
24. PUT-DOWN ROBOTB0 L08 POBJECT2
25. TAKE ROBOTB0 ROBOTC0 POBJECT3
26. PUT-DOWN ROBOTB0 L08 POBJECT3
27. TAKE ROBOTB0 ROBOTC0 POBJECT4
28. PUT-DOWN ROBOTB0 L08 POBJECT4
29. TAKE ROBOTB0 ROBOTC0 POBJECT5
30. PUT-DOWN ROBOTB0 L08 POBJECT5

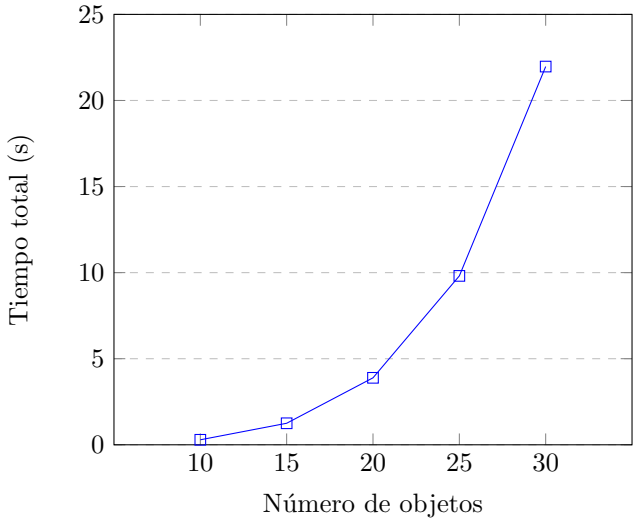
Otra conclusión que podemos extraer al comparar ambas ejecuciones es que el planificador no logra encontrar una solución que utilice las funciones `GIVE` y `TAKE`. Esto se debe a que, aunque estas funciones puedan requerir más pasos intermedios en su ejecución, el planificador considera que es un mejor camino utilizar otras funciones que pueden ser más cortas en apariencia, pero que en realidad involucran más pasos a largo plazo.

La última conclusión que podemos extraer de las pruebas es que cuanto más elevado es el número de nodos evaluados, mayor es el tiempo de ejecución, por lo que existe una correlación directa entre ambos. Sin embargo, no se observa una correlación directa entre el número de objetos (en el caso del dominio de los robots) y las otras dos variables. Esto se debe en gran medida a la complejidad inherente del problema, como podemos observar en los casos de p03, p04 y p06, donde para un mismo número de objetos, el tiempo de ejecución difiere.

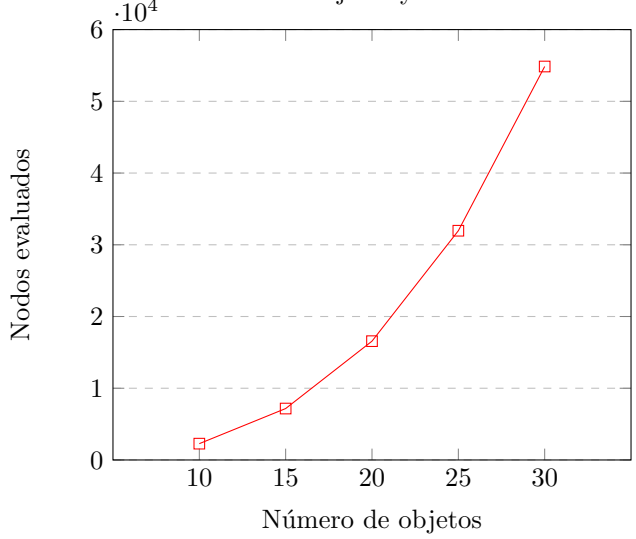
Por otro lado, para problemas de igual complejidad en términos lógicos, como en los casos de p06, p07, p08, p09 y p10, sí existe una correlación directa entre el número de objetos y el tiempo total.

Tomando estos cinco problemas, observemos mediante dos gráficas cómo el aumento del tiempo y el número de nodos evaluados aumenta exponencialmente a medida que se incrementa el número de objetos:

Relación entre el número de objetos y el tiempo total de ejecución



Relación entre el número de objetos y el número de nodos evaluados



5 Solución con PDDL

En este apartado se presentará la solución para resolver el problema sin utilizar PDDL2.1.

5.1 Definición del dominio

Para implementar el problema en PDDL se realizaron varias modificaciones respecto al dominio con PDDL2.1. Este nuevo dominio se puede encontrar en el archivo "domain-robots-sin-fluents".

Los primeros cambios fueron eliminar de los requerimientos ":fluents" y eliminar el apartado de ":functions" del problema.

Una vez eliminados, se definió un nuevo tipo de objeto denominado "capacidad". Este nos servirá para definir el distinto número de brazos que puede tener un robot. En nuestro caso, puede tener 0 o 2, pero podría darse el caso de que se pudieran tener un mayor número de brazos, por lo que la solución planteada sirve para cualquier número de brazos.

En segundo lugar, se añadieron dos nuevos predicados:

```
(:predicates
    (at-robot ?r - robot ?l - location)
    (at-pobject ?p - pobject ?l - location)
    (holding ?r - robot ?p - pobject)
    (connected ?l - location ?l1 - location)
    (in-cesta ?r - robot ?p - pobject)
    (has-brazo ?r - robot)
    ((capacidad_robot ?r - robot ?c - capacidad))
```

```
    ((predecesor_capacidad ?c1 - capacidad ?c2 - capacidad))
)
```

capacidad_robot: asigna una cantidad de brazos a un robot concreto, lo que nos sirve para controlar el número de brazos que tiene disponible un robot.

predecesor_capacidad: indica cuál es la relación entre las capacidades. Ya que no podemos utilizar números, creamos una relación de orden entre las distintas capacidades.

5.1.1 Modificaciones en las acciones

Los mayores cambios en el problema se encuentran en el apartado de acciones, ya que ahora tenemos que controlar el número de brazos disponibles con objetos de tipo capacidad.

El funcionamiento es sencillo, por cada robot pasado por parámetro, se le añaden dos nuevas propiedades "capacidad_actual" y "capacidad_final". En el caso de pasar más de un robot por parámetro, se deberán añadir dos nuevos parámetros.

Capacidad actual: debe coincidir con la capacidad actual del robot, por lo que la condición (capacidad_robot ?r ?capacidad_actual) debe ser siempre verdadera.

Capacidad final: tiene dos casos:

1. La acción va a implicar una liberación del brazo: esto implica que la capacidad actual tiene que ser el predecesor de la capacidad final, lo que se traduce en (predecesor_capacidad ?capacidad_actual ?capacidad_final).

2. La acción va a implicar la ocupación de un brazo: en este caso, primero tenemos que comprobar que la capacidad del robot es mayor que cero (not((capacidad_robot ?r capacidad0))). Una vez que hacemos esta comprobación, hay que verificar que la capacidad_final sea el antecesor de la capacidad actual (predecesor_capacidad ?capacidad_final ?capacidad_actual).

Teniendo estas consideraciones en cuenta, ya tenemos todas las condiciones iniciales. En cuanto a los efectos, únicamente tendremos que eliminar la relación del robot con la capacidad inicial y crear una relación con la capacidad final.

Todos estos cambios se aplicaron en las distintas acciones del dominio y el resultado fue el siguiente:

```
(:predicates
    (at-robot ?r - robot ?l - location)
    (at-pobject ?p - pobject ?l - location)
    (holding ?r - robot ?p - pobject)
    (connected ?l - location ?l1 - location)
    (in-cesta ?r - robot ?p - pobject)
    (has-brazo ?r - robot)
    ((capacidad_robot ?r - robot ?c - capacidad))
    ((predecesor_capacidad ?c1 - capacidad ?c2 - capacidad))
```

```
)


(:action PICK-UP
    :parameters (?rb - robot-brazo ?l - location ?p - pobject
    (?capacidad_actual ?final_capacidad - capacidad)
    :precondition (and (at-robot ?rb ?l)
        (not(capacidad_robot ?rb capacidad0))
        ((capacidad_robot ?rb ?capacidad_actual)
        ((predecesor_capacidad ?final_capacidad ?capacidad_actual)
        (at-pobject ?p ?l))
    :effect (and (not (at-pobject ?p ?l))
        ((not(capacidad_robot ?rb ?capacidad_actual))
        ((capacidad_robot ?rb ?final_capacidad)
        (holding ?rb ?p))
)

(:action PUT-DOWN
    :parameters (?rb - robot-brazo ?l - location ?p - pobject
    ?capacidad_actual ?final_capacidad - capacidad)
    :precondition (and (at-robot ?rb ?l)
        (capacidad_robot ?rb ?capacidad_actual)
        (predecesor_capacidad ?capacidad_actual ?final_capacidad)
        (holding ?rb ?p))
    :effect (and (not (holding ?rb ?p))
        (not(capacidad_robot ?rb ?capacidad_actual))
        (capacidad_robot ?rb ?final_capacidad)
        (at-pobject ?p ?l))
)

(:action GIVE
    :parameters (?rb - robot-brazo ?r - robot ?p - pobject ?l - location
    ?capacidad_actual_rb ?capacidad_final_rb
    ?capacidad_actual_r ?final_capacidad_r - capacidad)
    :precondition (and (holding ?rb ?p)
        (at-robot ?rb ?l)
        (at-robot ?r ?l)
        (capacidad_robot ?rb ?capacidad_actual_rb)
        (predecesor_capacidad ?capacidad_actual_rb ?capacidad_final_rb)
        (or (not (has-brazo ?r))
            (and (not(capacidad_robot ?r capacidad0))
                (capacidad_robot ?r ?capacidad_actual_r)
                (predecesor_capacidad ?final_capacidad_r ?capacidad_actual_r))))
    :effect (and
        (not (holding ?rb ?p))
        (not(capacidad_robot ?rb ?capacidad_actual_rb))
```

```
        (capacidad_robot ?rb ?capacidad_final_rb)
        (when
            (has-brazo ?r)
            (and (not(capacidad_robot ?r ?capacidad_actual_r))
                capacidad_robot ?r ?final_capacidad_r)
                (holding ?r ?p)))
        (when
            (not (has-brazo ?r))
            (in-cesta ?r ?p))
    )
)

(:action TAKE
    :parameters (?rb - robot-brazo ?r - robot ?p - pobject ?l - location
    ?capacidad_actual_rb ?capacidad_final_rb
    ?capacidad_actual_r ?final_capacidad_r - capacidad)
    :precondition (and (at-robot ?rb ?l)
        (at-robot ?r ?l)
        (not(capacidad_robot ?rb capacidad0))
        (capacidad_robot ?rb ?capacidad_actual_rb)
        (predecesor_capacidad ?capacidad_final_rb ?capacidad_actual_rb)
        (or (and (has-brazo ?r)
                (holding ?r ?p)
                (capacidad_robot ?r ?capacidad_actual_r)
                (predecesor_capacidad ?capacidad_actual_r ?final_capacidad_r))
            (and (not (has-brazo ?r)) (in-cesta ?r ?p))))
    :effect (and
        (holding ?rb ?p)
        (not(capacidad_robot ?rb ?capacidad_actual_rb))
        (capacidad_robot ?rb ?capacidad_final_rb)
        (when
            (has-brazo ?r)
            (and
                (not(capacidad_robot ?r ?capacidad_actual_r))
                (capacidad_robot ?r ?final_capacidad_r)
                (not (holding ?r ?p))))
        (when
            (not (has-brazo ?r))
            (not (in-cesta ?r ?p)))
    )
)
```

5.2 Cambios en la definición del problema

Para adaptar los problemas a este nuevo dominio, se tuvieron que realizar algunos cambios:

- Eliminación de las referencias a funciones.
- Se definieron objetos de tipo capacidad: "capacidad0 capacidad1 capacidad2 - capacidad".
- Se definió la capacidad de cada robot.
- Se definieron los antecesores de cada capacidad.

Para el archivo `p02.pddl`, las modificaciones realizadas fueron las siguientes:

```
(define (problem robots)
  (:domain domain-robots-sin-fluents)
  (:objects
    l01 l02 l03 l04 l05 l06 l07 l08 l09 l10 - location
    capacidad0 capacidad1 capacidad2 - capacidad
    robotB0 - robot-brazo
    robotC0 - robot-cesta
    pobject0 - pobject
    pobject1 - pobject
  )
  (:init
    ;FUNCTIONS
    (capacidad_robot robotB0 capacidad2)
    (capacidad_robot robotC0 capacidad0)

    (predecesor_capacidad capacidad0 capacidad1)
    (predecesor_capacidad capacidad1 capacidad2)

    (at-robot robotB0 l01)
    (at-robot robotC0 l10)
    (has-brazo robotB0)
    (at-pobject pobject0 l02)
    (at-pobject pobject1 l03)
    (connected l01 l02)
    (connected l02 l01)
    (connected l03 l02)
    (connected l02 l03)
    (connected l04 l02)
    (connected l02 l04)
    (connected l05 l02)
    (connected l02 l05)
    (connected l06 l05)
    (connected l05 l06)
    (connected l07 l05)
    (connected l05 l07)
    (connected l08 l05)
```

```
    (connected l05 l08)
    (connected l09 l08)
    (connected l08 l09)
    (connected l10 l08)
    (connected l08 l10)
  )
  (:goal
    (and
      (at-pobject pobject0 l08)
      (at-pobject pobject1 l04))
  )
)
```

5.3 Resultados

Problema	Tiempo total (s)	Acciones	Nodos evaluados	Número de objetos
p02_sin_fluents	0.02	10	19	2
p03_sin_fluents	0.08	26	192	6
p04_sin_fluents	0.88	31	1388	6
p05_sin_fluents	0.06	29	78	6
p06_sin_fluents	1.95	51	2261	10
p07_sin_fluents	8.36	79	6713	15
p08_sin_fluents	26.24	101	14676	20
p09_sin_fluents	68.69	129	27608	25
p10_sin_fluents	149.41	152	46216	30

Table 3: Tabla del dominio de robots

6 Comparación de resultados

Se ha realizado una comparativa de tiempo, número de acciones y nodos evaluados, como se puede ver en los gráficos debajo.

En cuanto al tiempo, la implementación realizada con PDDL2.1 ofrece mejores resultados en todos los escenarios. Sin embargo, las soluciones aportadas por la implementación de PDDL son mejores, ya que conllevan un menor número de acciones. Por último, a los nodos expandidos, aunque hay diferencias, prácticamente es irrelevantes.

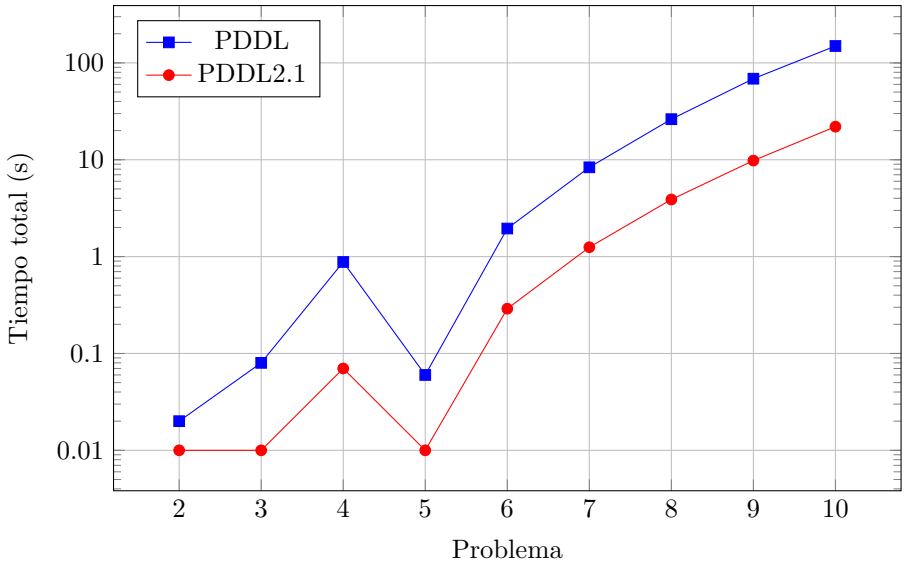


Figure 1: Comparación de Tiempos Totales entre PDDL y PDDL2.1

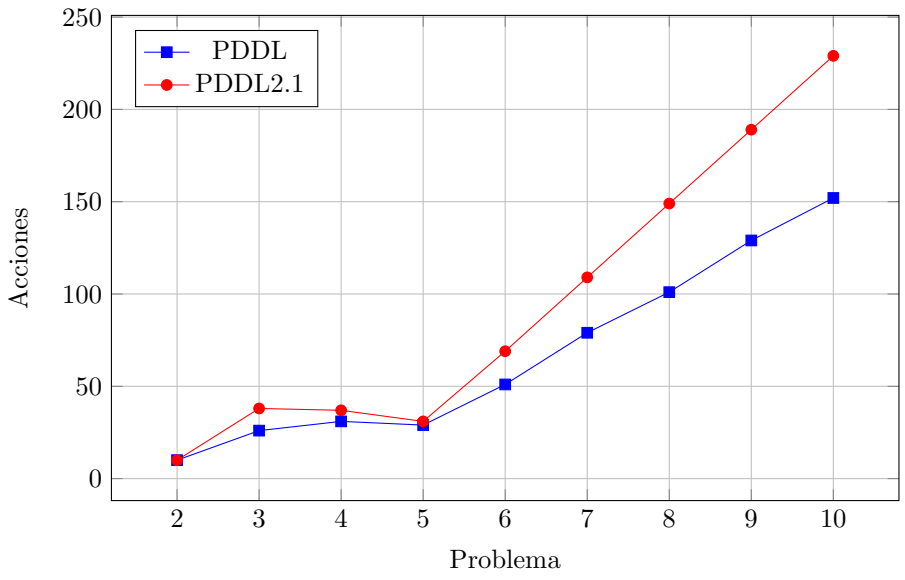


Figure 2: Comparación de Número de Acciones entre PDDL y PDDL2.1

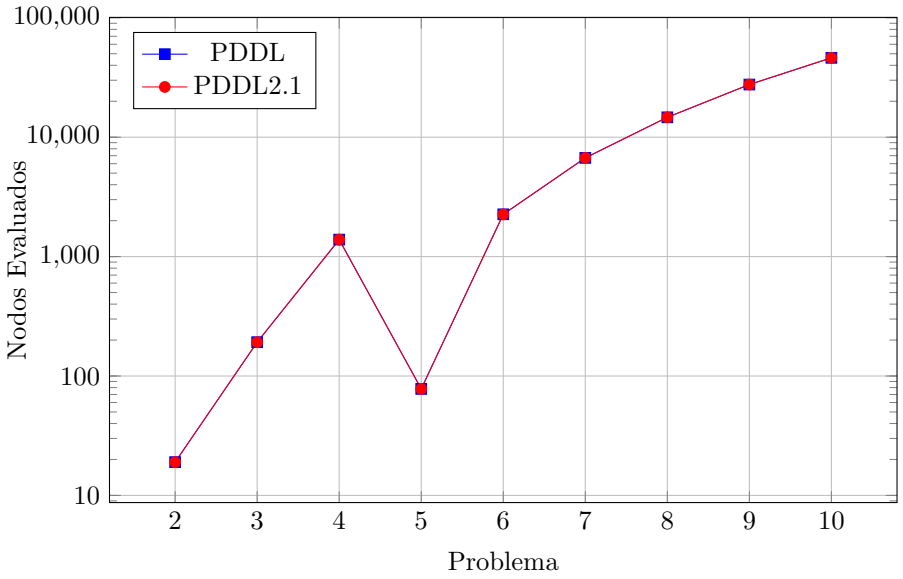


Figure 3: Comparación de Nodos Evaluados entre PDDL y PDDL2.1

7 Conclusión

Este trabajo permitió adentrarse en la complejidad de la planificación automática mediante la manipulación de problemas PDDL y el uso del planificador Metric-FF. Si bien se logró resolver la mayoría de los problemas, se evidenció una brecha entre las soluciones óptimas y las obtenidas por el planificador.Además, se puso de manifiesto cómo el aumento de la complejidad y/o el número de variables involucradas puede incrementar significativamente el tiempo de ejecución en este tipo de problemas.