Planificadores Inteligentes

Prieto Fontcuberta, José R. Pérez Bernal, Cristian

[1]Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

[2]Planificadores inteligentes

{joprfon, cripeber}@inf.upv.es

Índice

1. Introducción

En esta memoria se evalúa la resolución de un problema de planificación en el dominio de un puerto.

El objetivo en este dominio es dejar disponibles los contenedores marcados como objetivo en el muelle 1 usando las grúas y las cintas disponibles para moverlos entre las diferentes pilas.

2. Ejercicio 1. Dominio proposicional

En el dominio proposicional se resolverá el problema usando solo lógica proposicional. A continuación veremos los tipos, predicados y acciones implementadas y el uso de cada una.

2.1. Tipos

Para resolver el dominio proposicional, sin el uso de funciones numéricas ni control del tiempo, hemos implementado los siguientes tipos:

- **muelle**: representa el lugar donde se colocan las pilas y las grúas.
- **pila**: representa el lugar donde se colocan los contenedores
- **grúa**: las grúas nos permiten mover los contenedores. Están situadas en un muelle y no tienen acceso al resto de muelles.
- **cinta**: las cintas permiten traspasar los contenedores entre los muelles. Es la única forma de comunicar los muelles entre si.
- **contenedor**: los contenedores son los objetos que queremos mover, representan la mercancía del muelle.
- **altura**: este tipo nos ayudará a controlar la altura máxima de cada pila.

2.2. Predicados

Los predicados implementados en el dominio proposicional son los siguientes:

- **(in ?p - (either pila grua cinta) ?m - muelle)**: Con este predicado controlamos la ubicación de las pilas, gruas y cintas. Dicho predicado se mantendrá estático durante todo el problema, sin cambiar una vez definido.
- **(notCinta ?ct - cinta ?m - muelle)**: Nos servirá para evitar que una grúa ponga un contenedor en una cinta que no está en su muelle. Este predicado evitará poner una precondición negativa del predicado anterior
- **(at ?c - contenedor ?x - (either pila cinta))**: Con este predicado indicamos donde estará el contenedor, si en una pila o una cinta.
- **(top ?c - (either pila contenedor) ?m - pila)**: Este predicado nos indicará cual es el contenedor que está más arriba de la pila. Si la pila está vacía el top será la propia pila.
- **(on ?c - contenedor ?p - (either pila contenedor))**: Este predicado nos indcará, una vez colocado un contenedor, sobre que lo está colocando, pudiendo estar sobre otro contenedor o estar en la base de la pila. Va muy ligado al predicado anterior *(top)*.
- **(ocupada ?ct - cinta)**: Nos servirá para indicar cuando la cinta está ocupada o no. En cada cinta solo cabrá un solo contenedor.

- **(not_ocupada ?ct - cinta)**: Nos servirá para lo mismo que la anterior per de forma negativa, evitando así poner precondiciones negadas, dado que el planificador *Optic* no las acepta.
- **(vacia ?g - grua)**: Nos indicará cuando la grúa está vacía, es decir, que no sujeta ningún contenedor.
- **(holding ?g - grua ?c - contenedor)**: Nos indica que la grúa está sujetando el contenedor *c*.
- **(isgoal ?c - contenedor)**: Este predicado nos indicará cuando un contenedor es objetivo, es decir, es un contenedor que ha de quedar en la cima de una pila en el muelle 1.
- **(altura_pila ?p - pila ?n - altura)**: Este predicado nos servirá para indicar cual es la altura actual de la pila.
- **(next ?alt -altura ?sigalt - altura)**: Nos indicará cual es la siguiente altura. Dado que no podemos usar funciones numéricas, este predicado hará la función de éstas, siendo una cadena de variables de tipo altura de la forma: *n0 -¿n1 -¿n2*, siendo *n2* la máxima altura.

2.3. Acciones

Para este dominio hemos implementado un total de 5 acciones, las cuales las explicamos a continuación.

La acción **coger** servirá para que una grua que está libre coja un bloque que está en el *top* de una pila. La acción queda como sigue:

```
(:action coger
:parameters (?base - (either pila contenedor) ?c -
   contenedor ?p1 - pila ?m -muelle ?g - grua  ?antalt -
    altura ?alt - altura ))
```

Como se ve en los parámetros, necesitamos una base, la cual puede ser una pila u otro contenedor. Seguidamente necesitamos el contenedor objetivo a coger, el cual ha de estar encima del todo de la pila y una vez ha sido recogido, dejará de estarlo y, por ello, necesitábamos la base (el bloque o pila que estaba justo debajo), para asignar éste ahora como nuevo *top*. Además, para todo esto, necesitamos saber que pila es con la que estamos trabajando y comprobar que ambos contenedores están en la misma pila, comprobar que la pila está en el mismo muelle que la grúa y, finalmente, tener un control de la altura con los dos últimos parámetros.

Uno de los efectos principales de esta acción es la de poner el bloque en cuestión en la grúa y, a la vez, cambiar el *top* de la pila.

La acción **soltar** nos servirá para, una vez el bloque está en la grúa, soltarlo en el *top* de una pila. La acción queda como sigue:

```
(:action soltar
:parameters (?base - (either pila contenedor) ?c -
   contenedor ?p1 - pila ?m - muelle ?g - grua ?alt -
   altura ?sigalt - altura))
```

En esta acción los parámetros son los mismo que en la anterior. La diferencia está en las precondiciones y efectos, los cuales, en su mayoría, son inversos. Necesitaremos comprobar que la base sobre la que dejamos el contenedor es *top* de la pila objetivo, y como efecto el nuevo *top* será el contenedor que estaba sujetando la grúa. Las comprobaciones de la altura (en este caso es al revés, dado que ha de caber un bloque más) y la pila y grua en el mismo muelle se mantienen.

Una de las precondiciones importantes de esta acción es que la grúa ha de tener un bloque sujetando (holding) para poder dejarlo.

La acción **solta_especial** es una acción similar a la de **soltar** pero con una sola diferencia, y es que esta acción solo se usa cuando el bloque objetivo (el que tiene la grúa) y el bloque *top* (la base donde se va a dejar el bloque objetivo) son ambos bloques objetivos. Esto se comprueba en las precondiciones. En los efectos no eliminamos el bloque base como *top* dado que los bloques objetivos si se pueden poner uno encima de otro y seguir siendo una solución al problema.

La acción **coger_desde_cinta** es una acción que se ejecutará cada vez que se tenga que recoger un contenedor desde una cinta. La regla queda como sigue:

```
(:action coger_desde_cinta
:parameters (?c - contenedor ?ct - cinta ?m - muelle ?g
   - grua))
```

Como se ve en los parámetros, esta acción no tiene una base de donde coger, como si la tenia la acción normal de coger. Esto es dado porque a base siempre será la cinta. Tampoco tenemos que preocuparnos por la altura de la cinta, ya que siempre es binaria (o hay o no hay contenedor). En esta acción, sin contar las precondiciones de que la grúa y la cinta se correspondan, tendremos que tener en cuenta que la grúa ha de estar vacía y la cina ocupada, y los efectos contrarios.

Y en la última acción, **mover_a_cinta**, que queda como sigue:

```
(:action mover_a_cinta
:parameters (?c - contenedor ?ct - cinta ?m - muelle ?g
   - grua))
```

Hace la función inversa a la anterior vista, moviendo un contenedor desde la grúa, la cual no ha de estar vacía, a la cinta, la cual ha de estar vacía.

2.4. Problemas definidos.

Hemos definido un total de 3 problemas a resolver. En orden de dificultad los problemas son: *prob0.PDDL, prob1.PDDL, ejer4.PDDL y probFinal.PDDL*, siendo los dos primeros problemas sin dificultad, el tercero el problema definido en el ejercicio 4 del boletín y el problema final es un problema con un total de 11 contenedores y una alta dificultad. Entre los distintos problemas definidos encontramos modificaciones en la altura, número de pilas y contenedores.

2.5. Ejecuciones.

En este apartado veremos los tiempos de las distintas ejecuciones del problema con los planificadores **Metric-FF, LPG, LPG con timesteps**[1] **y OPTIC**. A continuación se muestra una tabla de resultados con cada problema y cada planificador, y se han adjuntado dichas soluciones a esta memoria.

	Prob0	Prob1	Ejer4	Final
FF	4	6	22	12
LPG	4	6	9	10
LPG -timesteps	4	6	10	20
OPTIC	4	6	11	16

Cuadro 1. Duración de cada problema, medido contando el número de acciones de la mejor solución encontrada en cada caso

Como se puede observar, con el primer y segundo problema, de dificultad nula, ningún planificador tiene problemas en sacar una solución parecida a la del resto, sin haber un vencedor. En el problema **Ejer4** se observa como LPG sin la opción timesteps encuentra la mejor solución del problema con 9 acciones a ejecutar, seguido de LPG con timesteps. En este caso Metric-FF se dista mucho del resto de soluciones.

En el problema más difícil vemos como la mejor solución la vuelve a dar LPG seguido de Metric-FF.

3. Ejercicio 2. Dominio temporal

En este ejercicio vamos a aplicar planificación temporal en nuestro dominio, buscando así aplicar costes temporales y funciones a las diferentes acciones explicadas anteriormente. Para ello se ha indicado al lenguaje *PDDL* que se requiere para este problema tanto *:fluents* como *:durative-actions*.

3.1. Funciones implementadas

Ahora pasamos a explicar las siguientes funciones implementadas, así como, si se diera el caso, las anteriores acciones que sustituyen.

- **(peso ?c - contenedor)** : Esta función ha sido creada para indicarle a cada uno de los contenedores del puerto que peso tienen.
- **(maxaltura ?m - muelle)** : Esta función indica la cantidad máxima de contenedores que puede tener un muelle, no en conjunto de todas las pilas, sino por cada una de ellas.
- **(altura p? - pila)** : Como bien se ha indicado antes, cada una de las pilas contiene su propia altura para poder controlar la cantidad de bloques que hay en ella. Con esta función permitimos indicar cual es la altura actual de la pila. Con esta función y la nombrada en el punto anterior se eliminan los predicados de *altura-pila* y *next* del ejercicio anterior.

[1] Con la opción timesteps le decimos a LPG que intente optimizar el número de pasos para encontrar la solución

- **(distancia ?ct - cinta)** : Para cada una de las cintas vamos a necesitar guardar su la longitud total para posteriores cálculos de duración de las acciones. Dichos cálculos se podrán realizar gracias a los *fluents*.
- **(velocidad ?ct - cinta)** : Como en la anterior función, en esta manejamos la velocidad con la que avanza la cinta, permitiendo usar este valor para posteriores cálculos de la duración de las acciones.

3.2. Cambios en los predicados

Dado que hemos añadido temporalidad al plan hemos modificado ligeramente los predicados. Hemos añadido un predicado para indicar si está **disponible** un contenedor o no. Este se usará cuando se esté moviendo un contenedor de un lado a otro, bloqueándolo y evitando así que el contenedor pueda estar en varios lugares al mismo tiempo.

Se han eliminado los predicados que controlaban la altura (**altura_pila y next**) ya que a partir de ahora usaremos *fluents* para ello, simplificando así la definición de las acciones. Al final de cada acción tendremos un contador que incrementa o disminuye, dependiendo de la acción, y una variable numérica que nos indica el máximo tamaño de las pilas.

3.3. Tiempo de las acciones

Ahora pasamos a explicar cada una de los diferentes cálculos que hemos implementado para medir la duración de las acciones y poder especificar más el problema, viendo como tardaría en un caso más realista.

La siguiente definición de la duración ha sido aplicada a las acciones de *coger, soltar y soltar_especial*. Esta primera aproximación nos ha resultado bastante problemática puesto que ciertos planificadores como puede ser *OPTIC* no aceptan *fluents* en la duración los cuales se modifiquen posteriormente en los efectos de dicha acción.

```
:duration (= ?duration (/ (peso ?c) (altura ?p) ))
```

Por eso, y después de consultarlo con los profesores de la asignatura, decidimos modificar este cálculo por el que se puede ver a continuación. Éste calcula la cantidad de contenedores que tiene que desplazar el gancho de la grúa para llegar al que está en la cima de esa pila, haciendo un cálculo muy parecido al primero que queríamos implementar. A mayor distancia mayor será la duración. Se ha añadido una suma para evitar que el cálculo de una duración igual a cero.

```
:duration (= ?duration (-(+(*(peso ?c) (maxaltura ?m))
   1)  (*(peso ?c) (altura?p)) ))
```

El segundo calculo de coste temporal que vamos a aplicar, es sobre las funciones de *mover_a_cinta y coger_de_cinta*. Para hacer este calculo nos basamos en dividir la distancia de la cinta entre la velocidad para después sumarle el peso, quedando como sigue a continuación:

```
:duration (= ?duration (+ (/ (distancia ?ct) (velocidad
   ?ct)) (peso ?c)))
```

3.4. Momento de las precondiciones y efectos

Dado que ahora el problema es un problema de planificación temporal, las precondiciones pueden darse de tres formas distintas: *at start, over all o at end*. La primera *at start*, indica que la condición ha de darse al principio, es decir, una vez comprobada, si la duración de la acción es mayor a 1, dicha precondición no tendría porque mantenerse durante el resto de la acción, mientras que con la segunda opción, *over all*, si que obligamos a que dicha precondición se mantenga durante toda la acción, hasta el final. Con *at end* obligamos a que dicha precondición se deba cumplir, al menos, al final de la acción, al acabar.

En el caso de los efectos las únicas opciones disponibles son *at start* y *at end*.

En la acción **coger** tenemos dos precondiciones que se han de cumplir durante toda la acción: la pila ha de estar en el mismo muelle que la grúa (dos precondiciones sobre *in*). El resto de precondiciones se han establecido con *at start*, dado que en los efectos cambiamos el valor de algunas de ellas. En los efectos establecemos como *at start* que la grúa pase a estar no vacía y que el contenedor deje de estar en *top*. Si no la precondición sobre la grúa al principio podría pasar que otra acción usase la grúa al mismo tiempo, por lo tanto, bloqueamos la grúa al momento. El resto de efectos se establecen *at end*, dado que son efectos que se producen al finalizar la acción, como el decremento de la altura o decir que el contenedor ya no está en esa pila, para evitar que mientras una grúa mueve un contenedor otra grúa coloque otro a la vez en el mismo lugar.

En la acción **soltar** y **solta_especial**, que como hemos explicado anteriormente las diferencias entre ellas son mínimas, tenemos en la precondiciones que la grúa y la pila han de estar en el mismo muelle durante toda la acción. También tenemos como *over all* el hecho de que durante toda la acción la grúa no puede estar vacía (ha de tener el contenedor hasta el final). Se comprueba solo al principio de la acción el *top* de la pila y la altura, ya que todos los efectos de estas dos acciones se producen al final, como el hecho de cambiar el *top*, incrementar la altura o que la grúa deje de estar ocupada.

En la acción **coger_desde_cinta** tenemos la comprobación de la cinta y el muelle, así como que la cinta esté ocupada durante toda la acción, mientras que solo al principio la grúa ha de estar vacía. En los efectos hemos establecido que al principio la grúa ya deja de estar vacía y el contenedor ya no está en la cinta, así como que al final de la acción la cinta deje de estar ocupada y la grúa manteniendo el contenedor.

Finalmente, en la acción **mover_a_cinta** tenemos todas las precondiciones en *over all*, dado que se han de mantener todas durante la ejecución de la acción. Al principio de la acción el contenedor pasa a estar ya en la cinta, así como la cinta pasa a estar ocupada, evitando que ninguna acción de mover a la cinta se pueda ejecutar simultáneamente. Al final de a acción la grúa pasa a estar vacía.

3.5. Problemas definidos

Para este ejercicio se han definido tres problemas diferentes. El primero, *probLow*, es un problema sencillo, seguido de *ejer4*, problema definido en el ejercicio 4 del boletín, siendo un poco más complicado que el anterior. Finalmente, tenemos *probFinal*, un problema más grande.

3.6. Ejecuciones

En este apartado veremos los tiempos de las distintas ejecuciones de los problemas con los planificadores **LPG y OPTIC**. Se han escogido las mejores soluciones encontradas con cada uno. Dado que como métrica se ha establecido el tiempo total del plan al ser un plan temporal, es lo que mostraremos en la siguiente tabla, junto al número de acciones requeridas. Se han buscado hasta 5 soluciones (n igual a 5 en LPG).

	probLow		**Ejer4**		**ProbFinal**	
	Duración	Acciones	Duración	Acciones	Duración	Acciones
LPG	11	5	30	17	26	20
OPTIC	10	5	18	12	31	18

Cuadro 2. Duración de cada problema, medido contando el tiempo total del plan, y el número de acciones. Se han desestimado los decimales dado que eran valores extremadamente pequeños.

LPG con el problema más pequeño da una solución ligeramente peor a la encontrada con OPTIC, siendo un paso más larga con el mismo número de acciones.

En el problema *Ejer4* con LPG hemos encontrado un plan de duración 30 con 17 acciones mientras que con OPTIC encontramos un plan mejor con una duración mejor, de 18 tiempos, y con solo 12 acciones.

En el problema final LPG ha encontrado una solución mejor a la de OPTIC en este caso, siendo de duración 26 con 20 acciones, mientras que en OPTIC aumentamos a una duración de 31, aunque con dos acciones menos. OPTIC en este caso no ha podido paralelizar mejor que LPG las acciones, aunque es probable que con más tiempo de ejecución OPTIC podría haber encontrado una solución mejor.

Hemos modificado los planes disminuyendo la velocidad de la cinta a 3, siendo el resultado el mostrado en la siguiente tabla.

	probLow		**Ejer4**		**ProbFinal**	
	Duración	Acciones	Duración	Acciones	Duración	Acciones
LPG	12.33	5	29.33	15	39	18
OPTIC	12.335	5	No lo consigue		31.005	18

Cuadro 3. Duración de cada problema, medido contando el tiempo total del plan, y el número de acciones. Se ha disminuido la velocidad de las cintas a 3

Como podemos observar, disminuyendo la velocidad de las cintas suele aumentar la duración de las soluciones. OPTIC, después de varios minutos de ejecución (rondando los 20 minutos), no ha dado con ninguna solución en el segundo problema, mientras que en el primero encuentra una solución peor que con la velocidad más alta y en el problema más grande encuentra una solución con el mismo tiempo. LPG ha ido mejor en el segundo problema y peor en el problema más grande.

4. Ejercicio 3. Dominio con recursos

En esta sección añadiremos un recurso renovable usando variables numéricas, tal y como ya hemos hecho anteriormente. Dicho consumo del recurso será inversamente proporcional al consumo de tiempo; a más consumo menos tiempo. Para ello, se han definido dos versiones del dominio: una versión donde el recurso no es renovable (no se puede recargar) y otra versión donde le recurso si es renovable.

4.1. Dominio con recurso no renovable

Para añadir el recurso hemos añadido dos funciones nuevas:

```
(gasolina ?g -grua)
(total-gas-used
```

La primera, en la que dada una grúa se contará cuanta gasolina le queda, restando gasolina en todo momento que se haga una acción. La segunda nos servirá para saber el total de gasolina consumida por todas las grúas.

4.1.1. Modificaciones en las acciones

Para tener en cuenta el nuevo recurso hemos modificado las acciones añadiendo una nueva condición a cada una de ellas, controlando, durante toda la acción, que dicha grúa que la ejecuta tenga suficiente gasolina (dependiendo de la acción consumirá más o menos). Como efecto hemos añadido dos: al principio (ambos) se decrementará la gasolina de dicha grúa y se incrementará la gasolina del contador total, con los efectos *(at start (decrease (gasolina ?g) X))* y *(at start increase (total-gas-used) Y))* respectivamente.

En la definición de los problemas hemos de definir ahora la gasolina que tiene cada grúa e inicializar el contador del total de gasolina a 0. Además, ahora podremos optar por minimizar el tiempo total (como hasta ahora) o minimizar la nueva función del total de gasolina usada con *(:metric minimize (total-time))*.

Se ha de tener en cuenta de que si el problema no tiene suficientes recursos para terminar el plan no encontrará solución. Esto se arregla haciendo que el plan tenga recursos renovables.

4.2. Dominio con recurso renovable

Para añadir la funcionalidad de que el recurso sea renovable solo se ha añadido una acción al plan anterior, creando otro archivo diferente. La acción es la que sigue:

```
(:durative-action recargar
        :parameters (?g - grua)
        :duration (= ?duration 2)
    :condition  (and
                (at start       (not_cargando ?g))
        )
    :effect (and
                (at start       (not(not_cargando ?g)))
                (at end         (not_cargando ?g))
                (at end         (increase (gasolina ?g)
                   5))

        )
)
```

Se ha añadido el predicado *not_cargando* para indicar que la grúa no está cargando, y, por lo tanto, puede ir a cargar gasolina.

Añadiendo dicha acción, la cual dada una grúa se le añade gasolina pero consumiendo tiempo, hacemos que el plan pase a tener recursos renovables y el planificador tendrá la opción de recargar o no cada grúa. En los efectos vemos como al principio de la acción se bloquea con el predicado *not_cargando* y al final de la acción se desbloquea, aumentando la gasolina de la grúa. También se ha añadido la precondicion, en cada regla, de que la grúa no esté cargando.

Se puede jugar con la duración de la acción para penalizar el hecho de ir a cargar.

4.3. Ejecuciones

Se han lanzado los mismo problemas que en los ejercicios anteriores, este caso añadiendo el recurso. Se ha probado a ejecutar el problema sin tener suficientes recursos y con el dominio sin recursos renovables pero, como era de esperar, el planificador ha concluido que el problema no se puede resolver.

Sin los recursos renovables y usando la métrica del tiempo de las acciones, el resultado es el mostrado a continuación:

Minimizando el tiempo

	probLow		**Ejer4**		**ProbFinal**	
	Duración	Acciones	Duración	Acciones	Duración	Acciones
LPG	11.0	5	21	15	24.0	16
OPTIC	11.0	5	16.0	12	27.011	19

Cuadro 4. Duración y número de acciones de la mejor solución de cada problema con OPTIC y LPG, minimizando el tiempo con *(:metric minimize (total-time))*. Con recursos no renovables

Si como métrica usamos el uso de los recursos, tratando de minimizarlos, los resultados son los siguientes:

Minimizando la gasolina usada

	probLow		Ejer4		ProbFinal	
	Duración	Acciones	Duración	Acciones	Duración	Acciones
LPG	30	5	90	13	110	16
OPTIC	30	5	90	12	125	18

Cuadro 5. Duración y número de acciones de la mejor solución de cada problema con OPTIC y LPG, minimizando el coste con *(:metric minimize (total-gas-used))*. Con recursos no renovables

En la tabla 4 vemos los resultados de minimizar el tiempo con los recursos no renovables, mientras que en la tabla 5 minimizamos la gasolina usada. Comparando ambas vemos que el número de acciones para resolver el problema, aun con objetivos de minimización diferentes, son muy similares o iguales.

Si lanzamos los problemas con recursos renovables, y disminuyendo la gasolina inicial para obligar a que deba recargar, con ambas métricas dan como resultado las dos siguientes tablas:

Minimizando la gasolina usada

	probLow		Ejer4		ProbFinal	
	Duración	Acciones	Duración	Acciones	Duración	Acciones
LPG	30	5	X	X	100	25
OPTIC	X	X	90	18	125	31

Cuadro 6. Duración y número de acciones de la mejor solución de cada problema con OPTIC y LPG, minimizando el coste con *(:metric minimize (total-gas-used))*. Con recursos renovables

Minimizando el tiempo

	probLow		Ejer4		ProbFinal	
	Duración	Acciones	Duración	Acciones	Duración	Acciones
LPG	11	5	26.00	16	25	17
OPTIC	11	5	18	15	34	23

Cuadro 7. Duración y número de acciones de la mejor solución de cada problema con OPTIC y LPG, minimizando el coste con *(:metric minimize (total-time))*. Con recursos renovables

La tabla 6, con recursos renovables y minimizando la gasolina total usada, ha dado problema en el problema pequeño con OPTIC y en el segundo con LPG, no terminando nunca la ejecución, después de varios intentos. En el caso de LPG la ejecución se ha parado sola al llegar a 1800 segundos y en el caso de OPTIC la hemos parado nosotros después de horas de ejecución.

En la tabla 6 vemos como aumentan ligeramente el número de acciones respecto a la tabla 7, en la cual minimizamos el tiempo.

Comparando las soluciones en las cuales se minimiza la gasolina usada, tablas 5 y 6, observamos que en el caso de los recursos renovables (empiezan con menos gasolina también), para llegar a un resultado similar necesita un mayor número de acciones, muy probablemente debido a tener que ir a recargar.

Si comparamos las soluciones en las cuales se minimiza el tiempo, tablas 4 y 7, vemos como en el caso de tener el recurso renovable (y, por lo tanto, haberle impuesto en un inicio menos gasolina) necesita siempre más acciones y, al igual que en el caso anterior, debido a tener que ir recargar. En este caso el tiempo si aumenta en el recurso renovable, ya que el coste de recargar es el tiempo.

5. Ejercicio 4. Desarrollo parcial de un árbol POP

En este ejercicio nos centramos en la creación de un árbol de orden parcial, el cual generamos a partir de un estado inicial y en el cual se expande un único nodo de manera aleatoria. Este ejercicio nos permite ver las dependencias entre las acciones y aquellas precondiciones que generan amenazas.

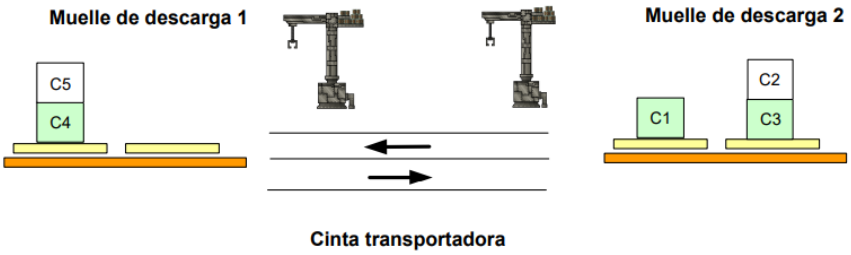

Figura 1. Estado inicial del ejercicio.

La imagen anterior nos muestra el estado inicial del problema el cual vamos a modelar para satisfacer las precondiciones de las diferentes acciones.

```
        (in g2 m2)
        (in p22 m2)
        (vacia g2)
        (on c2 c3)
        (at c2 p22)
        (top c2 p22)
        (altura_pila p22 n2)
        (next n0 n1)
    (COGER C3 C2 P22 M2 G2 N1 N2) --> A
        (top c3 p22)
        (holding g2 c2)
        (altura_pila p22 n1)
        (not (vacia g2))
        (not (top c2 p22))
        (not (on c2 c3))
        (not (at c2 p22))
        (not (altura_pila p22 n2))
```

Como primer nodo expandido tenemos la acción *COGER* la cual vamos a llamar (*A*) de ahora en adelante para simplificar las referencias, las precondiciones se muestran en la parte superior de la acción y los efectos en la parte inferior. Hay varias precondiciones que hay que satisfacer para poder generar esta acción, como:

```
<INIT,(on c2 c3),A>
<INIT,(at c2 p22),A>
<INIT,(top c2 p22),A>
```

Los anteriores *causal links* nos indican que las precondiciones necesarias para expandir este nodo son generadas por el estado anterior, en este caso el estado *init*.

Los otros dos nodos restantes, se expandirán a partir del ejemplo que hemos representado arriba junto a sus *causal links* y las relaciones de orden, el árbol POP completo se adjuntará como anexo a la memoria.

Como punto critico en la resolución de este problema se puede decir que el hecho de que todos los bloques objetivo estén en un mismo muelle generan el problema de mover a través de los dos muelles aquellos contenedores que no se encuentren en el muelle deseado. Por otra parte, las grúas nos permiten paralelizar trabajos haciendo acciones simultaneas, reduciendo el tiempo de ejecución.

Para finalizar el hecho de que las diferentes grúas puedan hacer diferentes acciones al mismo tiempo no genera problemas a la hora de generar nuevas acciones ya que evitan que las precondiciones estén satisfechas y se excluyan de manera mutua.

6. Ejercicio 5. Graphplan

En esta sección se procede a mostrar el grafo de planificación relajado, basándonos en la primera aproximación del ejercicio numero uno, que no utiliza ni dominio temporal ni recursos.Este grafo ha sido realizado sin tener en cuenta los efectos negativos de las acciones a realizar ni las exclusiones mutuas.

6.1. Desarrollo

Antes de empezar a explicar el proceso que hemos usado para desarrollar el grafo y debido a su tamaño, procedemos a adjuntar el grafo para una mejor visualización.

El problema en el cual no basamos para realizar este grafo, el cual está compuesto por dos muelles y cinco cajas, las cuales están repartidas entre las diferentes pilas que tenemos en los muelles y cada uno de estos consta de una grúa y una cinta, como se puede observar en la imagen 1. La situación que acabamos de presentar, será el estado inicial que se le pasa al grafo para su posterior expansión.

En cada unas de las capas de este grafo expandimos los nodos posibles, cumpliendo las precondiciones que tenga cada acción a realizar, generando nuevos efectos para poder alcanzar una meta. Esta meta, que previamente se nos ha proporcionado, nos exige tener los bloques de color verde o bloques objetivo en la cima de cualquier pila del primer muelle, así nos aseguramos de que estén disponibles para cargar.

6.2. Plan relajado

El siguiente plan relajado que se han extraído de las acciones marcadas en rojo del grafo, el cual nos permite crear una solución para el problema dado. El coste de obtener este plan es polinómico ya que vamos marcando cada una de las acciones que generan los estados objetivo y posteriormente marcamos aquellas acciones que tengan como efecto la precondición de estas acciones, esto nos permite ir hasta el principio del grafo sin necesidad de backtracking.

```
(COGER C3 C2 P22 M2 G2 N1 N2)
(MOVER_A_CINTA C2 CT2 M2 G2)
(COGER P22 C3 P22 M2 G2 N0 N1)
(MOVER_A_CINTA C3 CT2 M2 G2)
(COGER_DESDE_CINTA C3 CT2 M1 G1)
(COGER P21 C1 P21 M2 G2 N0 N1)
(SOLTAR P12 C3 P12 M1 G1 N0 N1)
(MOVER_A_CINTA C1 CT2 M2 G2)
(COGER_DESDE_CINTA C1 CT2 M1 G1)
(SOLTA_ESPECIAL C3 C1 P12 M1 G1 N1 N2)
(COGER C4 C5 P11 M1 G1 N1 N2)
```

6.3. Heurísticas

- **h_sum**: el resultado de esta heurística se calcula mediante la suma del numero de capad en las cuales se llega a un estado objetivo. Como resultado obtenemos 1 + 4 + 6 = 11.
- **h_max**: esta heurística muestra la capa donde obtenemos todos los estados objetivos. En el caso de este grafo nos arroja un valor de 6.
- **plan_relajado**: La ultima heurística cuenta la cantidad de acciones que se podrían hacer con el plan relajado. En este caso nos arroja un valor de 20.

Las anteriores heurísticas nos dan a conocer las diferentes acciones que tendría que hacer un planificador para obtener una o varias soluciones con este objetivo. Como heurística mas informada podríamos decir que la del *plan_relajado* es la mas informada por que no indica el numero máximo de acciones a realizar a la hora de encontrar una solución válida.

7. Conclusiones

Para concluir nos gustaría indicar que lo aprendido en *PDDL* nos ha mostrado la potencia que tiene la planificación en la resolución de problemas realmente complejos, como en nuestro caso el problema del puerto.

Principalmente, empezamos con un ejercicio que nos propone modelar nuestro problema de una manera bastante simple, la cual se ha ido complicando cada vez que añadíamos mas opciones como el dominio temporal, funciones o recursos propuestos en el trabajo. Esto nos permite darle bastante potencia a nuestro dominio pero conlleva un problema; y es que a mayor tamaño del árbol y/o mayor número de restricciones, complicado es encontrar una solución que satisfaga un problema dado.

Los resultados arrojados por los diferentes planificadores y opciones que hemos usado para resolver los problemas modelados nos permite también observar diferentes maneras de buscar una solución.