Planificación Inteligente

Dominio puerto

Jose Arias Moncho
Victoria Beltrán Domínguez

Febrero 2022

Contents

1 Introducción **1**

2 Ejercicios **1**
2.1 Ejercicio 1: Dominio proposicional 1
2.2 Ejercicio 2: Dominio temporal 12
2.3 Ejercicio 3: Dominio con recursos numéricos 14
2.4 Ejercicio 4: Desarrollo parcial de un árbol POP 18
2.5 Ejercicio 5: Graphplan 21

1 Introducción

Esta memoria tiene el objetivo de explicar cómo hemos desarrollado los distintos ejercicios propuestos para el problema del puerto planteado en la asignatura de *Planificación Inteligente*.

Antes de empezar, queremos recordar brevemente este problema: nos encontramos en el puerto de una ciudad donde existen dos muelles de descarga que contienen varios contenedores cada uno. En un momento dado, una compañía solicita al puerto que disponga de un conjunto determinado de contenedores en el Muelle de descarga 1 para su posterior recogida. El objetivo del problema es dejar disponibles en el muelle de descarga 1 los contenedores objetivo.

2 Ejercicios

2.1 Ejercicio 1: Dominio proposicional

Con el fin de resolver este problema, primero se han definido distintos tipos en lenguaje PDDL. El tipo *dock* representará los muelles, *pile* las pilas, *container* los contenedores, *crane* las grúas, *conveyor* para las cintas transportadoras y *height* para la altura de las pilas.

```
(:types dock pile container crane conveyor height)
```

Una vez tenemos claros los tipos, pasamos a definir los predicados que se han utilizado.

- `(belong ?x - (either crane conveyor pile) ?y - dock)`: se utiliza para saber en qué muelle están situados los distintos objetos (a excepción de los contenedores, que se inferirán a través de las pilas).
- `(on ?x - container ?y - (either container pile conveyor))`: empleado para distinguir la posición relativa de los contenedores. El contenedor o bien está encima de otro, o sólo en una pila o encima de la cinta. Para el caso en el que el contenedor está siendo cogido por la grúa existe un predicado diferente con el objetivo de que sea más intuitivo.
- `(top ?x - (either container pile) ?y - pile)`: define el objeto más alto de una pila (normalmente contenedores, pero la propia pila si esta está vacía).
- `(available ?x - (either container pile) ?m - dock)`: declara qué contenedor (o pila si está vacía) está disponible en el muelle especificado.
- `(empty ?x - (either conveyor crane))`: este predicado se utiliza para ver si un objeto de tipo cinta o grúa está vacío.
- `(holding ?x - container ?y - crane)`: se utiliza para representar una grúa sujetando un contenedor.
- `(pick ?ct - conveyor ?m - dock)`: este predicado se utiliza para saber a qué muelle se dirige la cinta. Imaginemos que la cinta c1 va del muelle m1 al muelle m2. Entonces, tendremos un predicado `(belong c m1)` y un `(pick c m2)`. Estos dos predicados definirán el sentido de la cinta.
- `(green ?x - container)`: define si el contenedor es objetivo.
- `(no-green ?x - container)`: define si el contenedor es corriente.
- `(next ?m - dock ?x - height ?y - height)`: predicado auxiliar para poder manejar la altura de la pila sin utilizar operadores aritmeticos.
- `(current-height ?p - pile ?x - height)`: define la altura actual de una pila.

Una vez definidos los predicados, pasamos a definir que acciones se han distinguido. En este caso, vemos que lo único que nosotros podemos hacer es coger y dejar contenedores con la grúa. Sin embargo, a causa de que las repercusiones son diferentes, hemos tenido que desglosar aún más las acciones, hasta que finalmente han quedado:

- **take**: coger el contenedor de una pila. Debemos cerciorarnos de que el contenedor está encima de una pila (y disponible). También debemos asegurar que la grua esté vacía y en el mismo muelle que la pila, y que la altura puede ser actualizada (no excedemos un mínimo). Una vez cumplimos con todas estas precondiciones, actualizamos la base de hechos: el contenedor ya no estará encima de la pila ni disponible, la grúa no estará vacía y la nueva cima de la pila será ?c2, que además estará disponible. También será necesario actualizar la altura de la pila.

```
(:action take
    :parameters (?c1 - container ?p - pile ?c2 - (either container pile) ?m - dock ?g - crane ?alt - height ?new_alt - height)
    :precondition
            (and
                (top ?c1 ?p)
                (on ?c1 ?c2)
                (belong ?p ?m)
                (belong ?g ?m)
                (empty ?g)
                (available ?c1 ?m)
                (current-height ?p ?alt)
                (next ?m ?new_alt ?alt)
            )
    :effect
            (and
                (not (top ?c1 ?p))
                (not (on ?c1 ?c2))
                (not (available ?c1 ?m)) ; container isnt available when the crane is holding it
                (not (empty ?g))
                (holding ?c1 ?g)
                (top ?c2 ?p)
                (available ?c2 ?m)
                (not (current-height ?p ?alt)) ; update current pile height
                (current-height ?p ?new_alt)
            )
)
```

Este regla, previamente la habíamos diseñado en dos reglas diferentes, para distinguir los casos en que la pila estaba vacía y la que estaba llena. Sin embargo, haciendo que la cima de una pila sea ella misma, conseguimos unificarlo.

- **leave-green-container**: dejar un contenedor verde en una pila. Este caso ha sido separado del caso en el que el contenedor es corriente, porque cambian los contenedores resultantes disponibles. Para este, simplemente debemos asegurarnos de que la grúa este sujetando el contenedor verde, y que esa misma grua pertenece al mismo muelle que la grua y que la pila. Además, la altura de la pila debe ser actualizable (para asegurar que no hemos llegado a la máxima).

  Una vez cubiertas todas las precondiciones, actualizamos la cima de la pila, y la altura de esta y el estado de la grúa (que ahora está vacía). Destacar que en este caso, el contenedor inferior sigue estando disponible, al haber puesto encima un contenedor objetivo.

```
(:action leave-green-container
    :parameters (?c1 - container ?p - pile ?c2 - (either container pile) ?m - dock ?g - crane ?alt - height ?new_alt - height)
    :precondition
            (and
                (top ?c2 ?p)
                (belong ?p ?m)
                (belong ?g ?m)
                (holding ?c1 ?g)
                (green ?c1)
                (current-height ?p ?alt)
                (next ?m ?alt ?new_alt)
            )
    :effect
            (and
                (top ?c1 ?p)
                (not (top ?c2 ?p))
                (on ?c1 ?c2)
                (available ?c1 ?m)
                (empty ?g)
                (not (holding ?c1 ?g))
                (not (current-height ?p ?alt))
                (current-height ?p ?new_alt)
            )
)
```

- **leave-ordinary-container**: dejar un contenedor corriente en una pila. Es exactamente igual que el caso anterior pero debemos cerciocarnos de que el contenedor no es verde y por lo tanto, el contenedor inferior ya no está disponible.

```
(:action leave-ordinary-container
    :parameters (?c1 - container ?p - pile ?c2 - (either container pile) ?m - dock ?g - crane ?alt - height ?new_alt - height)
    :precondition
            (and
                (top ?c2 ?p)
                (belong ?p ?m)
                (belong ?g ?m)
                (holding ?c1 ?g)
                (no-green ?c1)
                (current-height ?p ?alt)
                (next ?m ?alt ?new_alt)
            )
    :effect
            (and
                (top ?c1 ?p)
                (not (top ?c2 ?p))
                (on ?c1 ?c2)
                (available ?c1 ?m)
                (not (available ?c2 ?m))
                (empty ?g)
                (not (holding ?c1 ?g))
                (not (current-height ?p ?alt))
                (current-height ?p ?new_alt)
            )
)
```

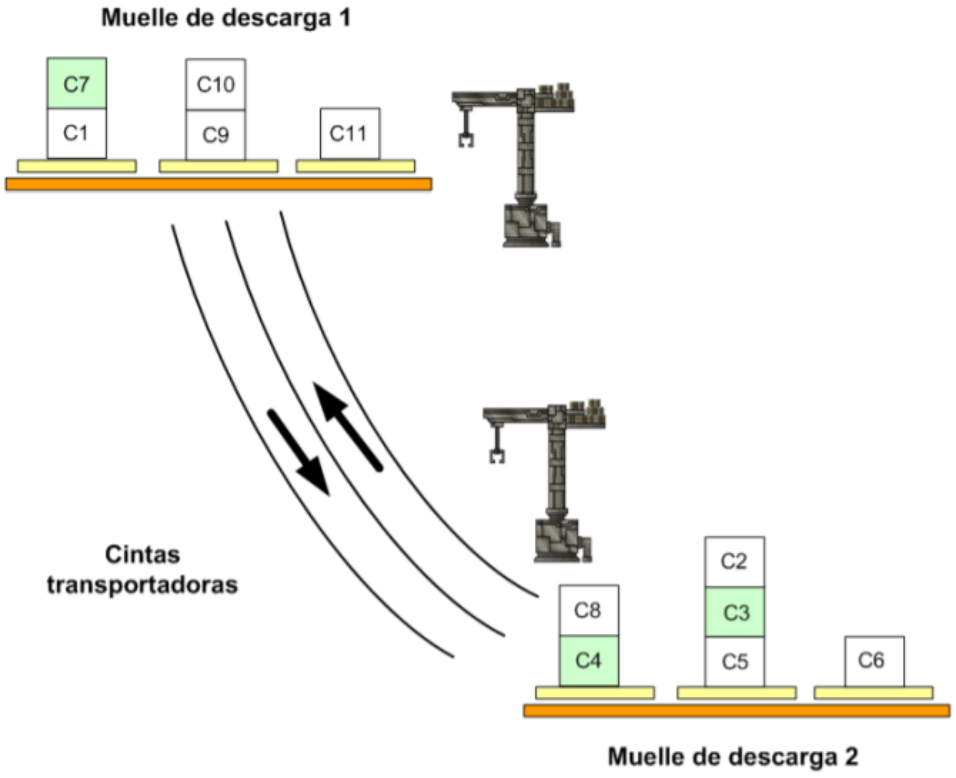
- **take-from-conveyor**: coger el contenedor de la cinta transportadora. Debemos asegurarnos de que la grua está vacía, que pertenece al mismo muelle del que se puede recoger de la cinta y que en la cinta hay un contenedor. El efecto de esta acción es que la grúa está sujetando el paquete y que la cinta está vacía.

```
(:action take-from-conveyor
    :parameters (?c1 - container ?ct - conveyor ?m - dock ?g - crane)
    :precondition
                (and
                    (empty ?g)
                    (pick ?ct ?m)
                    (belong ?g ?m)
                    (on ?c1 ?ct)
                )
    :effect
            (and
                (empty ?ct)
                (not (on ?c1 ?ct))
                (not (empty ?g))
                (holding ?c1 ?g)
            )
)
```

- **leave-in-conveyor**: dejar el contenedor en la cinta transportadora. Muy similar a la acción anterior. Debemos cerciorarnos que la grua esta sujetando el paquete y que la cinta está vacía. Si es el caso, pone el paquete en la cinta y marca la grúa como vacía.

```
(:action leave-in-conveyor
    :parameters (?c1 - container ?ct - conveyor ?m - dock ?g - crane)
    :precondition
                (and
                    (holding ?c1 ?g)
                    (belong ?g ?m)
                    (belong ?ct ?m)
                    (empty ?ct)
                )
    :effect
            (and
                (on ?c1 ?ct)
                (not (empty ?ct))
                (empty ?g)
                (not (holding ?c1 ?g))
            )
)
```

Una vez ya hemos definido todos los predicados y acciones que hemos utilizado para diseñar el problema en PDDL, pasamos a definir la instancia del problema que se muestra a continuación:



La representación de esta situación en lenguaje PDDL sería:

```
(define (problem puerto1)

(:domain puerto)
```

```
(:objects
    M1 M2 - dock
    P1 P2 P3 P4 P5 P6 - pile
    C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 - container
    G1 G2 - crane
    CT1 CT2 - conveyor
    n0 n1 n2 n3 - height
)

(:init
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;definir localización de los objetos grua, cinta y pila
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    (belong G1 M1)
    (belong G2 M2)

    (belong CT1 M1)
    (belong CT2 M2)

    (belong P1 M1)
    (belong P2 M1)
    (belong P3 M1)
    (belong P4 M2)
    (belong P5 M2)
    (belong P6 M2)

    (pick CT2 M1)
    (pick CT1 M2)


    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;definir situacion inicial de los contenedores
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    (on C1 P1)
    (on C7 C1)
    (on C9 P2)
    (on C10 C9)
    (on C11 P3)
    (on C4 P4)
    (on C8 C4)
    (on C5 P5)
    (on C3 C5)
    (on C2 C3)
    (on C6 P6)
```

```
(top C7 P1)
(top C10 P2)
(top C11 P3)
(top C8 P4)
(top C2 P5)
(top C6 P6)

(available C7 M1)
(available C1 M1)
(available C10 M1)
(available C11 M1)
(available C8 M2)
(available C2 M2)
(available C6 M2)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;definir tipo de cada contenedor
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(green C3)
(green C4)
(green C7)
(no-green C1)
(no-green C2)
(no-green C5)
(no-green C6)
(no-green C8)
(no-green C9)
(no-green C10)
(no-green C11)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;definir situación inicial de cintas y gruas
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(empty CT1)
(empty CT2)

(empty G1)
(empty G2)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;definir la sucesión de la altura para cada uno de los muelles
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
        (next M1 n0 n1)
        (next M1 n1 n2)
        (next M1 n2 n3)

        (next M2 n0 n1)
        (next M2 n1 n2)
        (next M2 n2 n3)

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;definir altura actual de cada pila
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        (current-height P1 n2)
        (current-height P2 n2)
        (current-height P3 n1)
        (current-height P4 n2)
        (current-height P5 n3)
        (current-height P6 n1)

    )
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;definir objetivos
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    (:goal
        (and
            (available C3 M1)
            (available C4 M1)
            (available C7 M1)
        )
    )
)
```

Cómo podemos ver, hemos especificado la localización de todos los objetos, además de su disponibilidad y tipo para los contenedores. También hemos definido la situación inicial tanto de las cintas como de las gruas, la altura de las pilas, y la sucesión de alturas para cada muelle. Nuestro objetivo consistirá en que cada uno de los distintos contenedores verdes estén disponibles en el muelle uno.

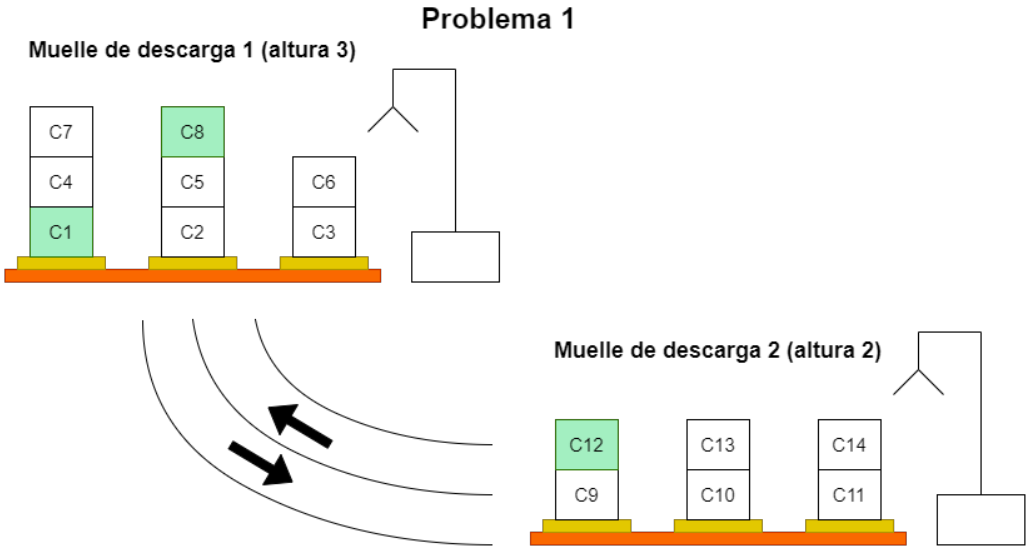
Una vez ya hemos definido tanto el dominio como el problema, pasamos a ejecutar los diferentes planificadores para comprobar si el plan ejecutado resuelve el problema especificado. Todos los planificadores logran resolver el problema especificado. La siguiente tabla muestra los resultados obtenidos:

Planificador	**# Reglas**	**Time(s)**
FF	14	0.02
LPG	12	125
OPTIC	22	1

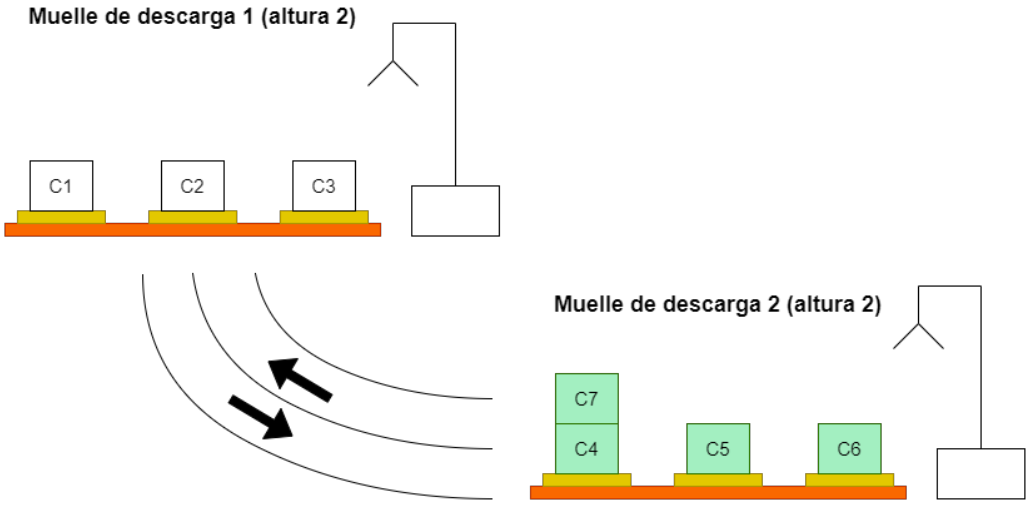
El mejor plan (aquel obtenido por LPG) es el siguiente:

```
0:   (TAKE C8 P4 C4 M2 G2 N2 N1) [1]
1:   (LEAVE-ORDINARY-CONTAINER C8 P6 C6 M2 G2 N1 N2) [1]
2:   (TAKE C4 P4 P4 M2 G2 N1 N0) [1]
3:   (LEAVE-IN-CONVEYOR C4 CT2 M2 G2) [1]
4:   (TAKE-FROM-CONVEYOR C4 CT2 M1 G1) [1]
4:   (TAKE C2 P5 C3 M2 G2 N3 N2) [1]
5:   (LEAVE-GREEN-CONTAINER C4 P3 C11 M1 G1 N1 N2) [1]
5:   (LEAVE-ORDINARY-CONTAINER C2 P4 P4 M2 G2 N0 N1) [1]
6:   (TAKE C3 P5 C5 M2 G2 N2 N1) [1]
7:   (LEAVE-IN-CONVEYOR C3 CT2 M2 G2) [1]
8:   (TAKE-FROM-CONVEYOR C3 CT2 M1 G1) [1]
9:   (LEAVE-GREEN-CONTAINER C3 P3 C4 M1 G1 N2 N3) [1]
```

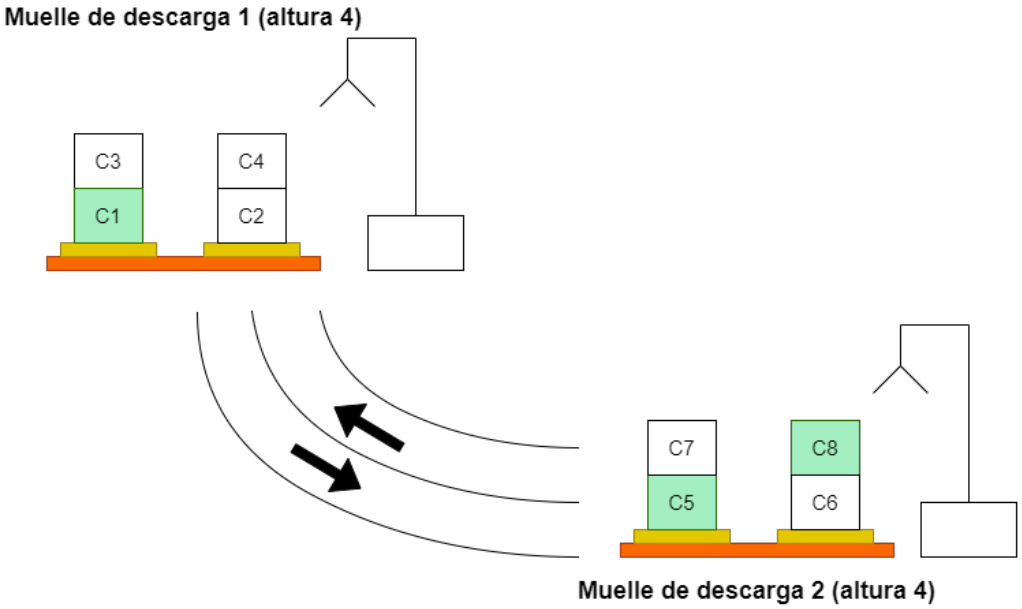
A continuación vamos a definir cuatro instancias diferentes del problema para así poder comparar el rendimiento de los diferentes planificadores. De esta forma, los problemas que utilizaremos serán:



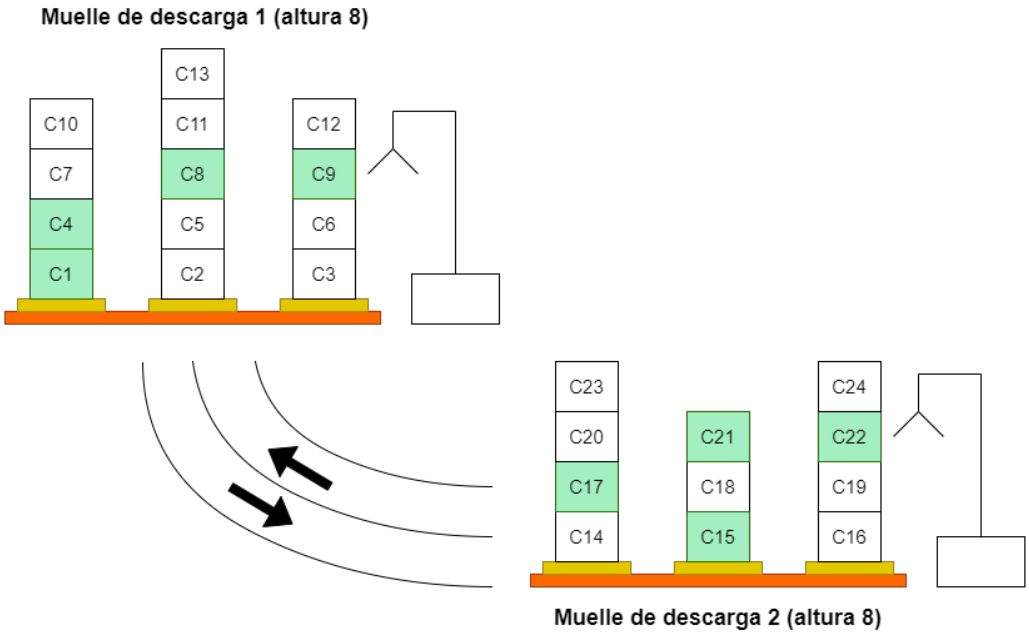
Problema 2



Problema 3



Problema 4



En la siguiente tabla se muestran los resultados obtenidos en cuanto a coste temporal:

Tiempo de ejecución	**FF**	**OPTIC**	**LPG**	**LPG -timesteps**
Problema 1	0.02	49.16	48.68	5.66
Problema 2	0.10	1.92	14.86	71.98
Problema 3	0.02	0.16	193.40	247.50
Problema 4	10.14	×	3396.92	3243.60

Podemos ver que FF es el planificador que menos tiempo tarda en obtener resultados. Cabe destacar que para el problema 4 el único planificador que encuentra una solución es FF.

Otro aspecto a destacar de las pruebas realizadas con los diferentes planificadores es la calidad de las soluciones. Podemos observar los resultados en la siguiente tabla y nótese que en esta hemos indicado el número de reglas y la duración total entre paréntesis para los planificadores que admiten concurrencia en sus planes:

Número de reglas	**FF**	**OPTIC**	**LPG**	**LPG -timesteps**
Problema 1	7	7 (5)	7 (8)	7(5)
Problema 2	30	18 (12)	18 (10)	18 (10)
Problema 3	14	18 (10)	11 (9)	11 (9)
Problema 4	60	×	71 (55)	71 (55)

En este caso, vemos que generalmente es LPG con la opción de -timesteps el planificador que mejores soluciones obtiene del problema independientemente de las características de este, quitando el Problema 4, que solo lo resuelve FF.

2.2 Ejercicio 2: Dominio temporal

Con el fin de resolver este problema dentro del dominio temporal ha sido necesario añadir la acción de desplazamiento de los contenedores para las cintas transporadoras. Para ello, se han añadido predicados a los ya definidos previamente:

En primer lugar tenemos `(needs-transport ?c - container)`, que indicará que un paquete se haya depositada en la cinta transportadora, pero no se ha desplazado al otro puerto para ser recogido.

En segundo y último lugar tenemos `(transported ?c - container)`, que indica que ya se ha desplazado al puerto y que puede ser recogido.

Una vez definidos, se ha añadido una acción de desplazamiento del contenedor en sí, que sería la siguiente:

```
(:action transport-conveyor
    :parameters (?c1 - container ?ct - conveyor)
    :precondition
            (and
                (needs-transport ?c1)
                (on ?c1 ?ct)
            )
    :effect
            (and
                (not (needs-transport ?c1))
                (transported ?c1)
            )
)
```

Una vez tenemos definidas todas las precondiciones y acciones necesarias, pasamos a introducir los cambios necesarios para realizar el dominio temporal.

Para ello, añadiremos a los requerimientos las opciones `:fluents` y `:durative-actions` para poder trabajar con este dominio. Además, definiremos un nuevo apartado de funciones donde vamos a establecer:

- `(weight ?c1 - container)` establecerá el peso de nuestras cajas.
- `(time-height ?x - height)` se encargará de fijar un ratio que afectará al tiempo.
- Por último, tendremos `(conveyor-length ?ct -conveyor)` y `(conveyor-speed ?ct -conveyor)` para encargarnos de modelar el transporte en la cinta de forma temporal.

Finalmente, se altera las diferentes acciones, que en nuestro dominio son todas ellas, añadiendo el coste temporal que tienen y añadiendo también el momento temporal en el que tienen que cumplirse tanto las condiciones como cuando se aplicarán los efectos.

Dentro de la definición del problema, tendremos que instanciar las funciones definidas, dando un valor concreto para cada objeto.

Así, vamos a establecer para todos los problemas que:

- El peso de las cajas será de 20 para todas ellas.
- La primera cinta tendrá siempre una longitud de 15 y la segunda de 25 y ambas tendrán una velocidad de 5.
- Por último lugar, estableceremos para cada altura un factor $f \in]0, 1]$ que cumplirá que $\forall a_1, a_2 : a_1 \neq a_2 \ \wedge \ a_1 > a_2, \ \ f_{a_1} < f_{a_2}$.

Añadiendo estos cambios a nuestro problema del dominio proposicional y ejecutando los planificadores, se obtienen los siguientes resultados:

Planificador	**Coste temporal del plan**	**Time(s)**
OPTIC	149	89.12
LPG	149	2.26

Como podemos observar en la tabla 2.2, los resultados obtenidos son los mismos en cuanto a calidad. Sin embargo, LPG tarda una cantidad de tiempo bastante inferior en comparación a OPTIC.

A continuación, vamos a realizar estos cambios para los problemas establecidos en la subsección 2.1. Para ello, añadimos los valores indicados anteriormente, y al ejecutar los planificadores, se obtiene:

Tiempo de ejecución	**OPTIC**	**LPG**
Problema 1	15.46	0.76
Problema 2	7.28	62.12
Problema 3	×	946.00
Problema 4	×	×

Como podemos observar, los resultados obtenidos son bastante variados, de forma que OPTIC obtiene mejores resultados para el Problema 2, pero LPG es más rápido con el Problema 1 y encuentra una solución en el Problema 3. Por otro lado, la calidad de las soluciones es:

Coste temporal del plan	**OPTIC**	**LPG**
Problema 1	75	80
Problema 2	240	220
Problema 3	×	141
Problema 4	×	×

En este aspecto OPTIC es superior debido a que los planes tienen un coste inferior. Sin embargo, sigue sin obtener una solución para el Problema 3, pero LPG sí lo hace.

2.3 Ejercicio 3: Dominio con recursos numéricos

Una vez terminado el dominio temporal, procedemos a añadir una variable numérica que represente el consumo de un recurso. Particularmente, pasamos a tener en cuenta el consumo de combustible que hacen las cintas transportadoras, de tal manera en la que ahora la cinta transportadora o bien podrá moverse más despacio con el fin de gastar menos combustible, o bien podrá moverse más deprisa a cambio de gastar más combustible. Además, se pide definir dos versiones diferentes del dominio, una en la que el combustible no sea renovable y otra dónde sí lo sea.

Una vez claro nuestro objetivo, procedemos a explicar qué hemos modificado para conseguir manejar este recurso.

Lo primero que hemos hecho ha sido definir varias funciones:

- `(conveyor-slow-speed ?ct -conveyor)` y `(conveyor-fast-speed ?ct -conveyor)` en vez de `(conveyor-speed ?ct -conveyor)` con el fin de diferenciar entre la velocidad rápida y lenta.
- `(conveyor-slow-burn ?c - conveyor)` y `(conveyor-fast-burn ?c - conveyor)` definen la tasa de combustible que gasta la cinta por cada unidad de distancia (dependiendo del modo).
- `(fuel ?c - conveyor)` especifica el combustible que queda en el tanque de combustible de la cinta transportadora c.
- `(total-fuel-used)` nos permite seguir la cuenta de todo el combustible que hemos gastado.
- `(fuel-capacity ?c - conveyor)` especifica la capacidad que tiene el tanque de combustible de la cinta transportadora c.
- `(refuel-rate ?c - conveyor)` define la tasa de combustible que es capaz de recargar la cinta transportadora por segundo.

`(refuel-rate ?c - conveyor)` y `(fuel-capacity ?c - conveyor)` solamente van a ser utilizadas en la versión en la que el recurso es renovable.

Haciendo uso de las funciones definidas, pasamos a desglosar la acción `transport-conveyor` en `transport-conveyor-fast` y `transport-conveyor-slow`.

En la siguiente imagen podemos ver la definición que hemos hecho de `transport-conveyor-fast`:

```
(:durative-action transport-conveyor-fast
    :parameters (?c1 - container ?ct - conveyor)
    :duration (= ?duration (/ (conveyor-length ?ct) (conveyor-fast-speed ?ct)))
    :condition
            (and
                (at start (>= (fuel ?ct) (* (conveyor-length ?ct) (conveyor-fast-burn ?ct))))
                (at start (needs-transport ?c1))
                (over all (on ?c1 ?ct))
            )
    :effect
            (and
                (at start (not (needs-transport ?c1)))
                (at end (transported ?c1))
                (at end (increase total-fuel-used (* (conveyor-length ?ct) (conveyor-fast-burn ?ct))))
                (at end (decrease (fuel ?ct) (* (conveyor-length ?ct) (conveyor-fast-burn ?ct))))
            )
)
```

Las diferencias con respecto a `transport-conveyor` son bastante evidentes. La duración está definida como la longitud de la cinta transportadora entre la velocidad de la cinta (en este caso rápida). Además, hemos añadido una condición para asegurar que solo transportamos de manera rápida si la cinta tiene suficiente combustible (si el combustible de la cinta es mayor o igual a la longitud de esta por la tasa de consumo rápido). De esta manera, si realizamos esta acción, como consecuencia tendremos que incrementar la variable del combustible total utilizado y decrementar la cantidad de combustible que tiene el tanque de combustible de la cinta.

Para el caso `transport-conveyor-slow` tenemos los mismos cambios pero utilizando las funciones lentas correspondientes.

```
(:durative-action transport-conveyor-slow
    :parameters (?c1 - container ?ct - conveyor)
    :duration (= ?duration (/ (conveyor-length ?ct) (conveyor-slow-speed ?ct)))
    :condition
            (and
                (at start (>= (fuel ?ct) (* (conveyor-length ?ct) (conveyor-slow-burn ?ct))))
                (at start (needs-transport ?c1))
                (over all (on ?c1 ?ct))
            )
    :effect
            (and
                (at start (not (needs-transport ?c1)))
                (at end (transported ?c1))
                (at end (increase total-fuel-used (* (conveyor-length ?ct) (conveyor-slow-burn ?ct))))
                (at end (decrease (fuel ?ct) (* (conveyor-length ?ct) (conveyor-slow-burn ?ct))))
            )
)
```

Para el dominio dónde el combustible no es renovable, con que definamos las funciones descritas anteriormente (a excepción de `refuel-fate` y `fuel-capacity`), las dos acciones mencionadas e inicializemos todas las funciones del problema, es más que suficiente.

Sin embargo, para el dominio dónde el combustible es renovable, deberemos utilizar además la acción `refuel`:

```
(:durative-action refuel
    :parameters (?c - conveyor)
    :duration (= ?duration (/ (- (fuel-capacity ?c) (fuel ?c)) (refuel-rate ?c)))
    :condition
        (and
            (at start (< (fuel ?c) (/ (fuel-capacity ?c) 2)))
        )
    :effect
        (and
            (at end (assign (fuel ?c) (fuel-capacity ?c)))
        )
)
```

En esta, si la capacidad del tanque de combustible de la cinta está a menos de la mitad, la llenaremos. Esto nos costará tanto tiempo como el fuel que quepa para llenar completamente el tanque por lo que tarda por segundo en rellenar (`refuel-fate`).

Una vez definidos tanto el dominio no renovable como el renovable, pasamos a hacer experimentos con LPG. Para los experimentos vamos a utilizar estas dos instancias del problema:

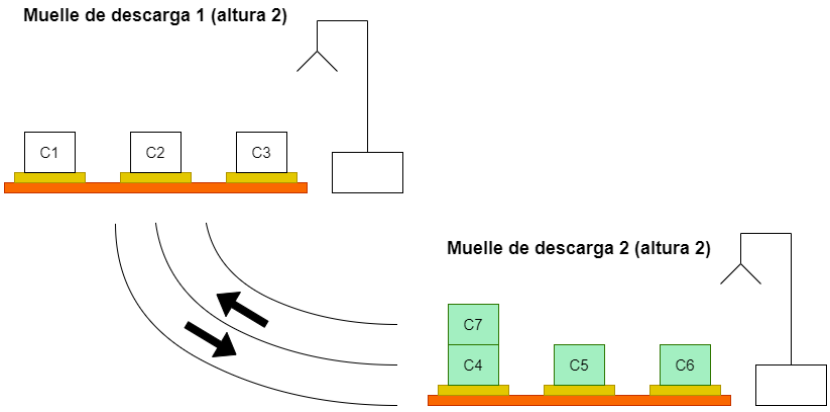


Figure 1: Instancia 1

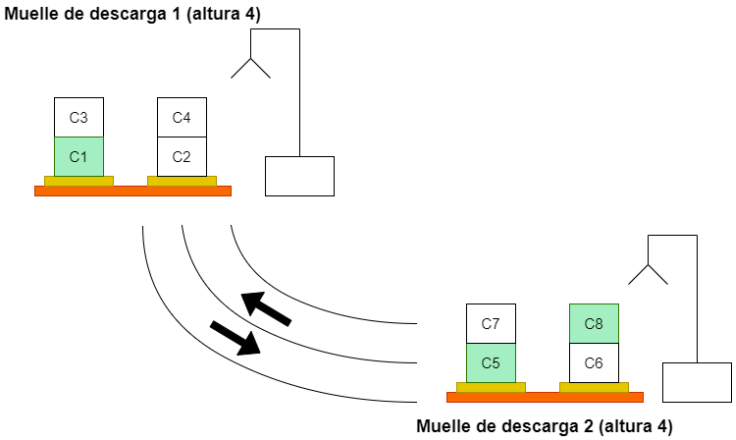


Figure 2: Instancia 2

Además, vamos a definir las siguientes funciones para todas los problemas:

- `(conveyor-slow-speed ?ct -conveyor)` = 1
- `(conveyor-fast-speed ?ct -conveyor)` = 5
- `(conveyor-slow-burn ?c - conveyor)` = 1
- `(conveyor-fast-burn ?c - conveyor)` = 2
- `(fuel ?c - conveyor)` = 100
- `(total-fuel-used)` = 0
- `(fuel-capacity ?c - conveyor)` = 200
- `(refuel-rate ?c - conveyor)` = 5

Experimentando con los problemas al minimizar el combustible, obtenemos los siguientes resultados:

	Coste temporal	**Combustible utilizado**
Instancia 1 no renovable	×	×
Instancia 1 renovable	695.00	180.00
Instancia 2 no renovable	145.00	50.00
Instancia 2 renovable	145.00	50.00

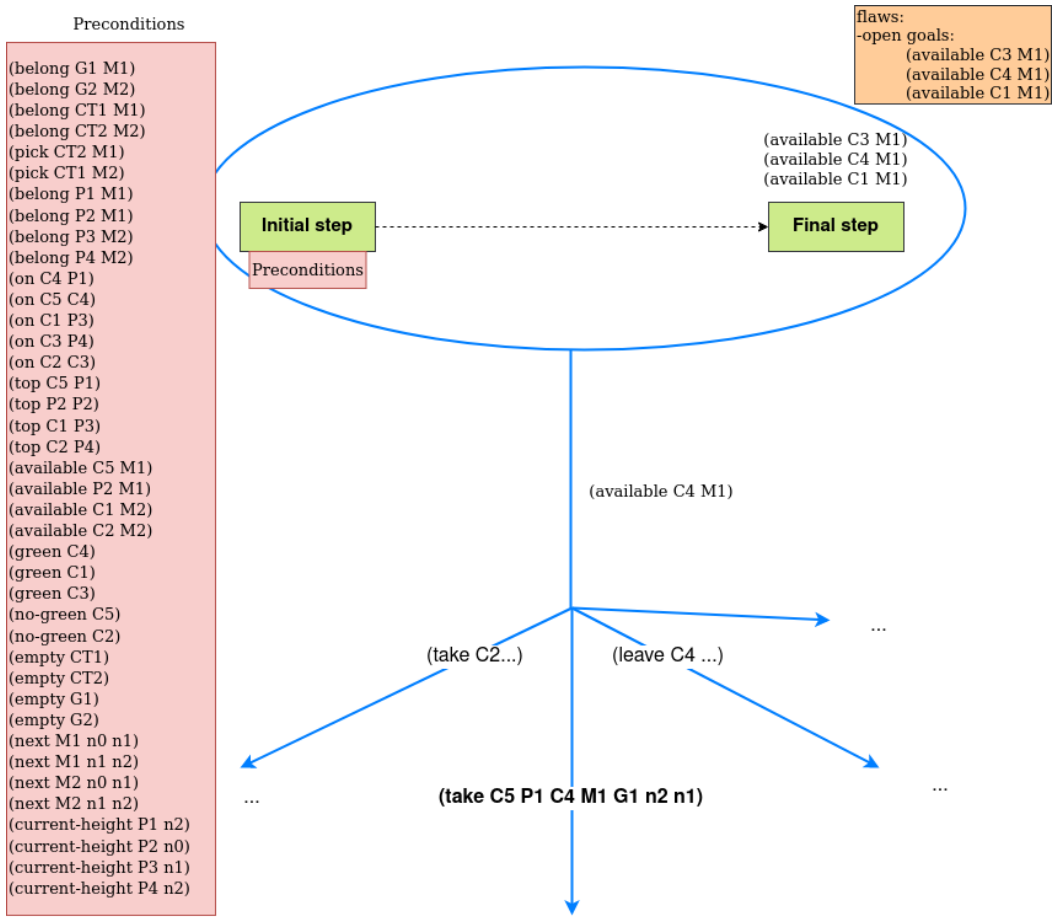
Sin embargo, si minimizamos el tiempo, obtenemos unos resultados muy diferentes:

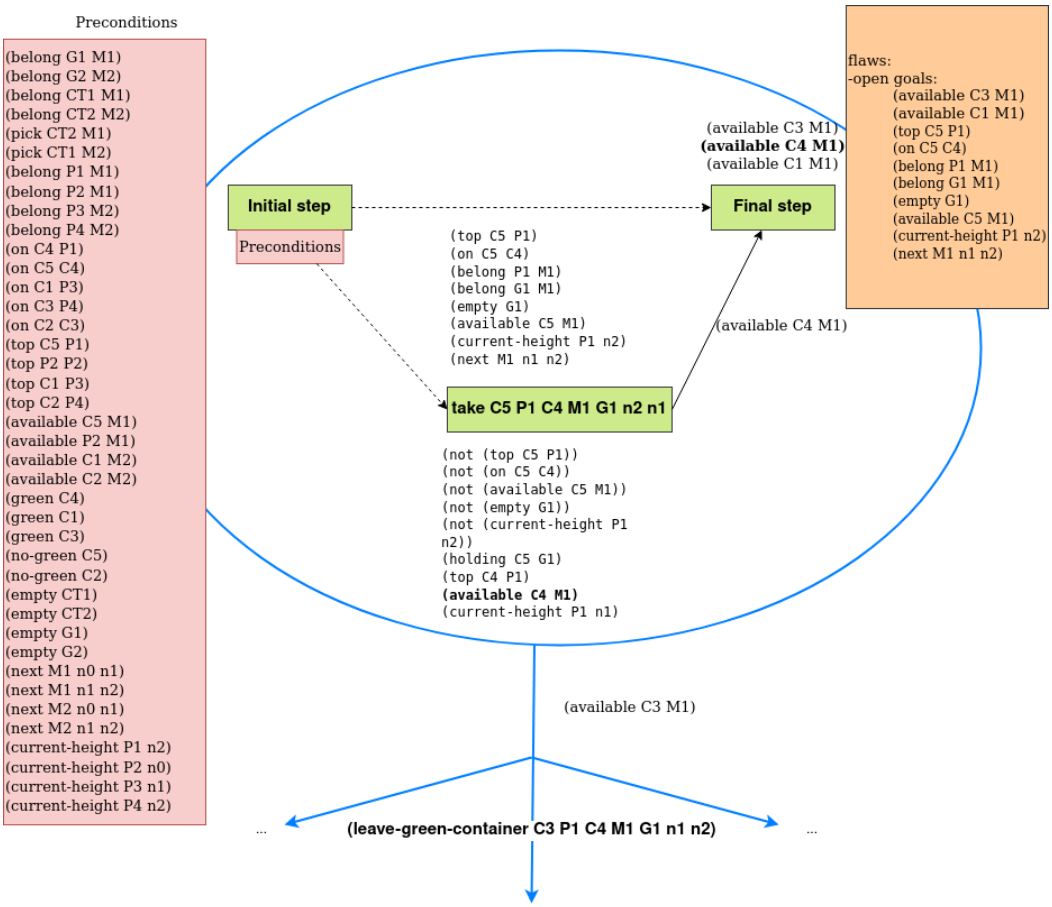
	Coste temporal	**Combustible utilizado**
Instancia 1 no renovable	×	×
Instancia 1 renovable	240.00	200.00
Instancia 2 no renovable	150.00	50.00
Instancia 2 renovable	130.00	100.00

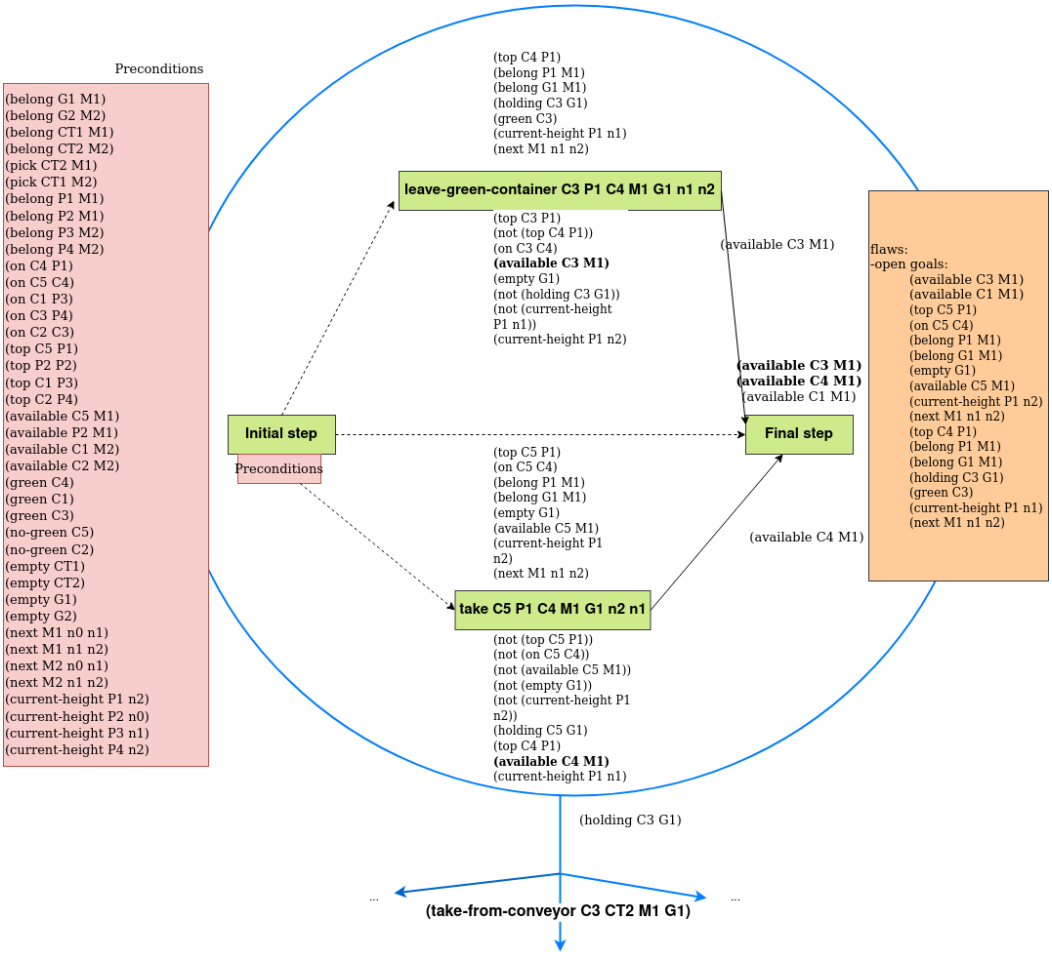
Por una parte, minimizando el combustible utilizado podemos ver como la instancia 1 no renovable no tiene suficiente combustible como para conseguir resolver el problema. Por otro lado, vemos como la instancia 2 no renovable tiene el combutible mínimo necesario para poder cumplir con el problema (y por lo tanto no importa si es renovable o no, porque al querer minimizar el combustible, encuentra la misma solución que con el no renovable). Por otro lado, si minimizamos el tiempo, vemos que generalmente conseguimos reducirlo, aumentando a su vez el combustible utilizado.

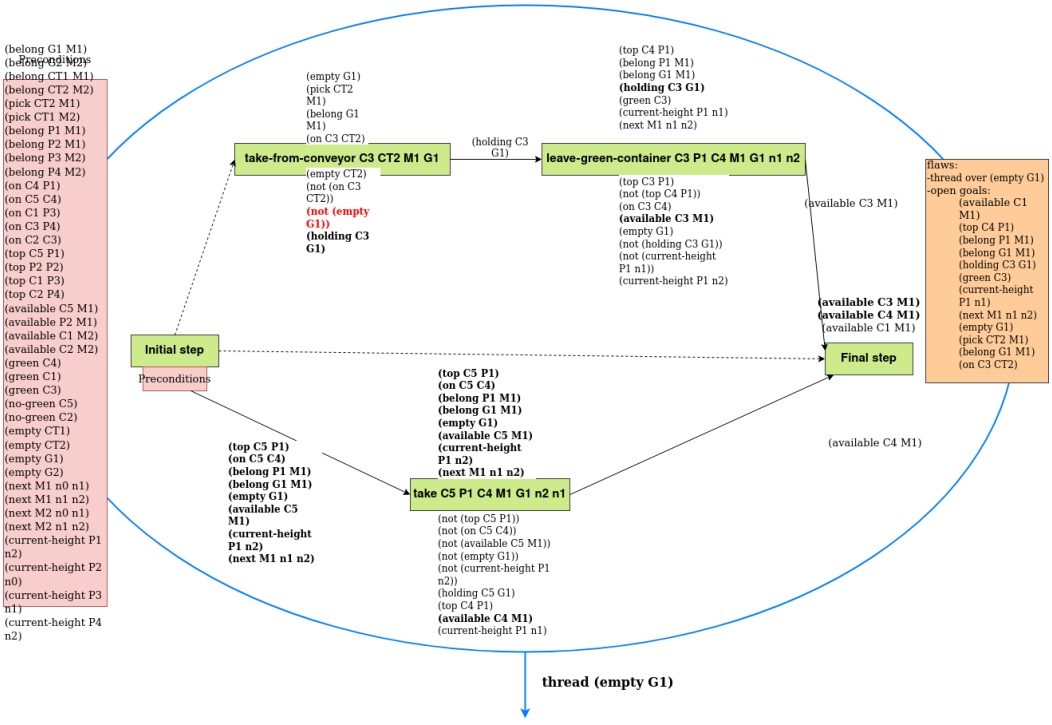
2.4 Ejercicio 4: Desarrollo parcial de un árbol POP

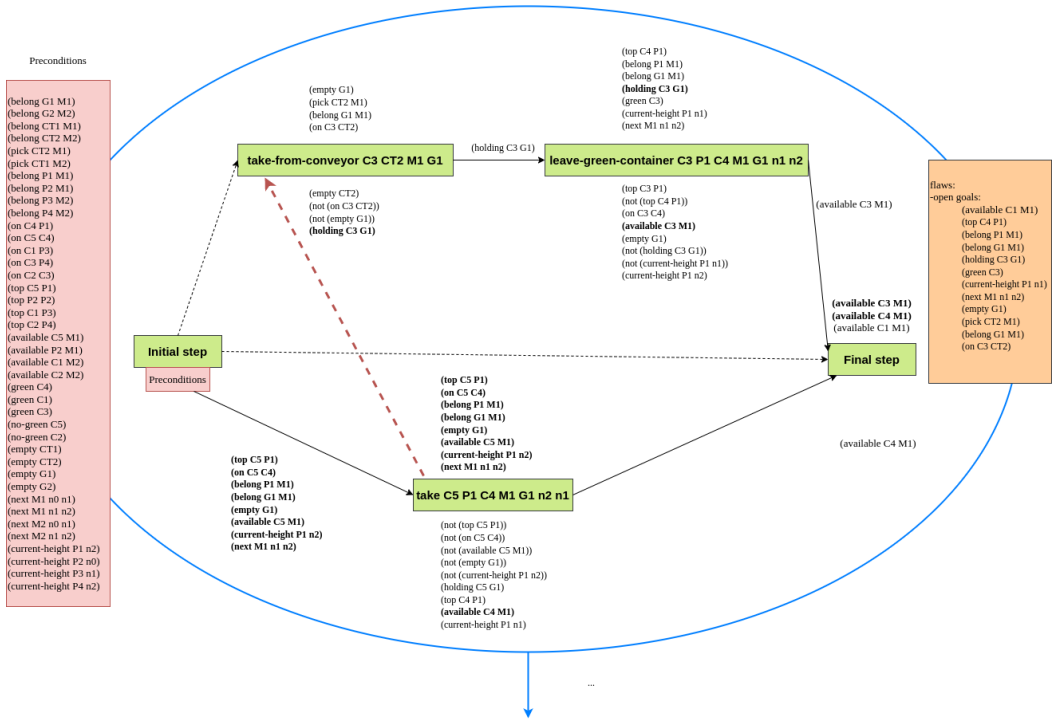
Para realizar este ejercicio, hemos escogido un dominio totalmente instanciado, y una heurística aleatoria (puesto que hemos elegido los nodos con el objetivo de que generen un thread).




Preconditions
(belong G1 M1)
(belong G2 M2)
(belong CT1 M1)
(belong CT2 M2)
(pick CT2 M1)
(pick CT1 M2)
(belong P1 M1)
(belong P2 M1)
(belong P3 M2)
(belong P4 M2)
(on C4 P1)
(on C5 C4)
(on C1 P3)
(on C3 P4)
(on C2 C3)
(top C5 P1)
(top P2 P2)
(top C1 P3)
(top C2 P4)
(available C5 M1)
(available P2 M1)
(available C1 M2)
(available C2 M2)
(green C4)
(green C1)
(green C3)
(no-green C5)
(no-green C2)
(empty CT1)
(empty CT2)
(empty G1)
(empty G2)
(next M1 n0 n1)
(next M1 n1 n2)
(next M2 n0 n1)
(next M2 n1 n2)
(current-height P1 n2)
(current-height P2 n0)
(current-height P3 n1)
(current-height P4 n2)
flaws:
-open goals:
(available C3 M1)
(available C1 M1)
(top C5 P1)
(on C5 C4)
(belong P1 M1)
(belong G1 M1)
(empty G1)
(available C5 M1)
(current-height P1 n2)
(next M1 n1 n2)
(available C3 M1)
(available C4 M1)
(available C1 M1)
Initial step
Preconditions
Final step
(top C5 P1)
(on C5 C4)
(belong P1 M1)
(belong G1 M1)
(empty G1)
(available C5 M1)
(current-height P1 n2)
(next M1 n1 n2)
(available C4 M1)
take C5 P1 C4 M1 G1 n2 n1
(not (top C5 P1))
(not (on C5 C4))
(not (available C5 M1))
(not (empty G1))
(not (current-height P1
n2))
(holding C5 G1)
(top C4 P1)
(available C4 M1)
(current-height P1 n1)
(available C3 M1)
...
(leave-green-container C3 P1 C4 M1 G1 n1 n2)
...


Preconditions
(belong G1 M1)
(belong G2 M2)
(belong CT1 M1)
(belong CT2 M2)
(pick CT2 M1)
(pick CT1 M2)
(belong P1 M1)
(belong P2 M1)
(belong P3 M2)
(belong P4 M2)
(on C4 P1)
(on C5 C4)
(on C1 P3)
(on C3 P4)
(on C2 C3)
(top C5 P1)
(top P2 P2)
(top C1 P3)
(top C2 P4)
(available C5 M1)
(available P2 M1)
(available C1 M2)
(available C2 M2)
(green C4)
(green C1)
(green C3)
(no-green C5)
(no-green C2)
(empty CT1)
(empty CT2)
(empty G1)
(empty G2)
(next M1 n0 n1)
(next M1 n1 n2)
(next M2 n0 n1)
(next M2 n1 n2)
(current-height P1 n2)
(current-height P2 n0)
(current-height P3 n1)
(current-height P4 n2)
(top C4 P1)
(belong P1 M1)
(belong G1 M1)
(holding C3 G1)
(green C3)
(current-height P1 n1)
(next M1 n1 n2)
leave-green-container C3 P1 C4 M1 G1 n1 n2
(top C3 P1)
(not (top C4 P1))
(on C3 C4)
(available C3 M1)
(empty G1)
(not (holding C3 G1))
(not (current-height P1 n1))
(current-height P1 n2)
(available C3 M1)
flaws:
-open goals:
(available C3 M1)
(available C1 M1)
(top C5 P1)
(on C5 C4)
(belong P1 M1)
(belong G1 M1)
(empty G1)
(available C5 M1)
(current-height P1 n2)
(next M1 n1 n2)
(top C4 P1)
(belong P1 M1)
(belong G1 M1)
(holding C3 G1)
(green C3)
(current-height P1 n1)
(next M1 n1 n2)
(available C3 M1)
(available C4 M1)
(available C1 M1)
Initial step
Preconditions
Final step
(top C5 P1)
(on C5 C4)
(belong P1 M1)
(belong G1 M1)
(empty G1)
(available C5 M1)
(current-height P1 n2)
(next M1 n1 n2)
(available C4 M1)
take C5 P1 C4 M1 G1 n2 n1
(not (top C5 P1))
(not (on C5 C4))
(not (available C5 M1))
(not (empty G1))
(not (current-height P1 n2))
(holding C5 G1)
(top C4 P1)
(available C4 M1)
(current-height P1 n1)
(holding C3 G1)
...
...
(take-from-conveyor C3 CT2 M1 G1)


(belong G1 M1)
Preconditions
(belong G2 M2)
(belong CT1 M1)
(belong CT2 M2)
(pick CT2 M1)
(pick CT1 M2)
(belong P1 M1)
(belong P2 M1)
(belong P3 M2)
(belong P4 M2)
(on C4 P1)
(on C5 C4)
(on C1 P3)
(on C3 P4)
(on C2 C3)
(top C5 P1)
(top P2 P2)
(top C1 P3)
(top C2 P4)
(available C5 M1)
(available P2 M1)
(available C1 M2)
(available C2 M2)
(green C4)
(green C1)
(green C3)
(no-green C5)
(no-green C2)
(empty CT1)
(empty CT2)
(empty G1)
(empty G2)
(next M1 n0 n1)
(next M1 n1 n2)
(next M2 n0 n1)
(next M2 n1 n2)
(current-height P1 n2)
(current-height P2 n0)
(current-height P3 n1)
(current-height P4 n2)
(empty G1)
(pick CT2 M1)
(belong G1 M1)
(on C3 CT2)
take-from-conveyor C3 CT2 M1 G1
(empty CT2)
(not (on C3 CT2))
(not (empty G1))
(holding C3 G1)
(holding C3 G1)
(top C4 P1)
(belong P1 M1)
(belong G1 M1)
(holding C3 G1)
(green C3)
(current-height P1 n1)
(next M1 n1 n2)
leave-green-container C3 P1 C4 M1 G1 n1 n2
(top C3 P1)
(not (top C4 P1))
(on C3 C4)
(available C3 M1)
(empty G1)
(not (holding C3 G1))
(not (current-height P1 n1))
(current-height P1 n2)
(available C3 M1)
flaws:
-thread over (empty G1)
-open goals:
(available C1 M1)
(top C4 P1)
(belong P1 M1)
(belong G1 M1)
(holding C3 G1)
(green C3)
(current-height P1 n1)
(next M1 n1 n2)
(empty G1)
(pick CT2 M1)
(belong G1 M1)
(on C3 CT2)
(available C3 M1)
(available C4 M1)
(available C1 M1)
Initial step
Preconditions
Final step
(top C5 P1)
(on C5 C4)
(belong P1 M1)
(belong G1 M1)
(empty G1)
(available C5 M1)
(current-height P1 n2)
(next M1 n1 n2)
(top C5 P1)
(on C5 C4)
(belong P1 M1)
(belong G1 M1)
(empty G1)
(available C5 M1)
(current-height P1 n2)
(next M1 n1 n2)
(available C4 M1)
take C5 P1 C4 M1 G1 n2 n1
(not (top C5 P1))
(not (on C5 C4))
(not (available C5 M1))
(not (empty G1))
(not (current-height P1 n2))
(holding C5 G1)
(top C4 P1)
(available C4 M1)
(current-height P1 n1)
thread (empty G1)



Las mayores dificultades del árbol POP han sido tanto tener en cuenta todos los predicados, puesto que se hacía algo lioso, como la detección de flaws, pues la intuición humana se adelanta a lo que va a pasar y parece que se den antes de lo que realmente se dan.

2.5 Ejercicio 5: Graphplan

Este ejercicio se ha realizado y se adjunta en un fichero Excel.

En concreto, se ha creado un grafo de planificación relajado para el que se han obtenido las heurísticas indicadas.

En este caso, las heurísticas serían:

	Valor
Heurística suma	10
Heurística máximo	5
Heurística plan relajado	9

Para obtener el valor de la heurística del plan relajado se ha obtenido los diferentes objetivos a conseguir y se ha seleccionado aquellas acciones que obtenían precondiciones necesarias.

Así, se ha seleccionado la acción `(LEAVE-GREEN C3 P2 C1 M1 G1 n1 n2)` para obtener `(available C3 M1)`. En el nivel anterior (nivel 3) se realizará `(TAKE-CONVEYOR C3,CT1,M1,G1)` para cumplir las precondiciones y elegimos `(LEAVE-GREEN C1,P2,P2,M1,G1,n0,n1)` como acción para obtener `(available C1 M1)`. En el nivel 2 se realizarán `(LEAVE-CONVEYOR C3,CT2,M2,G2)` y `(TAKE-CONVEYOR C1,CT1,M1,G1)` para cumplir las precondiciones necesarias en el nivel 3.

Posteriormente en el nivel 1 se realizan las acciones `(TAKE C3,P4,P4,M2,G2,n1,n0)` y `(LEAVE-CONVEYOR C1,CT2,M2,G2)`.

Por último en el nivel inicial tendremos que realizar las acciones `(TAKE C5,P1,C4,M1,G1,n2,n1)` y `(TAKE C1,P3,P3,M2,G2,n1,n0)`.

Finalmente, consideramos que la heurística más informada para este problema sería la heurística de suma, ya que la solución óptima se realiza en 12 acciones.