Planificación de Transporte en Aeropuertos con PDDL

Sergio Madrid Pérez, Jorge Haces Fuertes

Índice

1. Introducción

El presente documento describe un dominio PDDL modelado para el problema asignado de transporte en la terminal de un aeropuerto. Este dominio se basa en la gestión de trenes de equipaje mediante máquinas y vagones, asegurando que los equipajes sean transportados de manera eficiente y aquellos considerados sospechosos sean inspeccionados antes de ser entregados. La distribución de la terminal se muestra en la Figura 1.

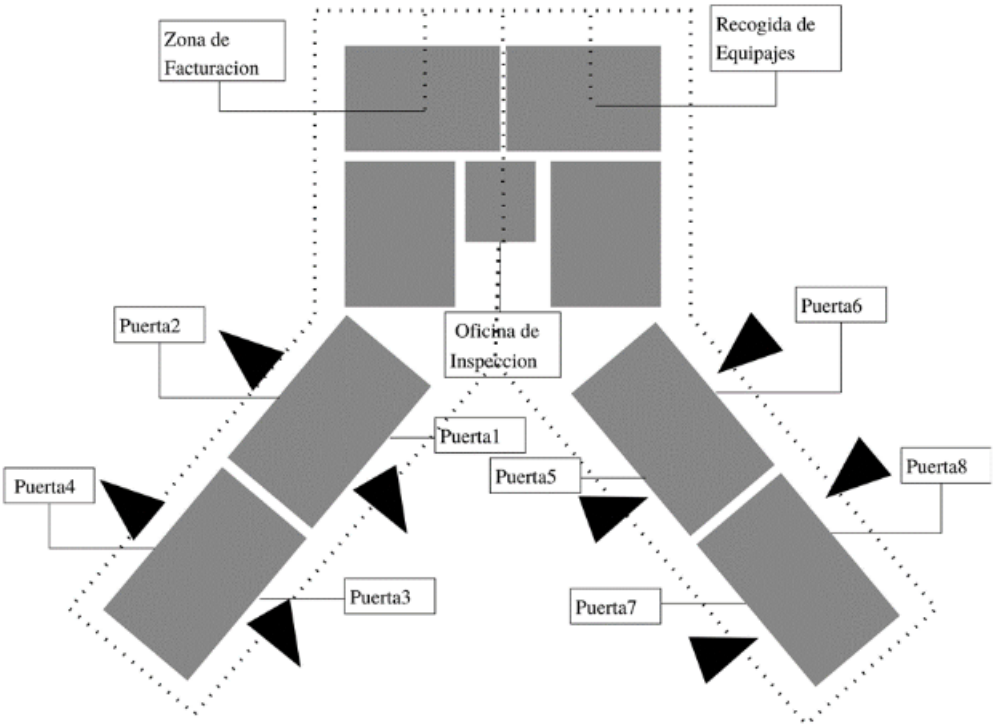

Figura 1: Representación del dominio del problema a resolver.

2. Dominio PDDL

En esta sección se describen los elementos básicos del dominio, incluyendo los tipos de objetos, predicados y acciones utilizadas.

2.1. Tipos

- **maquina**: Vehículos que mueven los vagones.

- **vagon**: Contenedores que se enganchan y son arrastrados por máquinas. Almacenan los equipajes.
- **equipaje**: Elementos que deben ser transportados a una localización específica.
- **posicion**: Representación de los puntos de ubicación dentro del aeropuerto.
- **numero**: Representación de números para ocuparnos de la capacidad de equipajes en un vagón.

2.2. Predicados

Los predicados permiten definir el estado del sistema. Algunos de los principales predicados empleados son:

- `(at ?x - (either maquina vagon equipaje) ?y - posicion)`: Indica que el objeto `?x` está en la posición `?y`.
- `(adjacent ?x - posicion ?y - posicion)`: Indica que las posiciones `?x` y `?y` son adyacentes.
- `(libre ?x - maquina)`: Indica que la máquina `?x` no tiene vagones enganchados.
- `(sospechoso ?x - equipaje)`: Indica que el equipaje `?x` es sospechoso.
- `(no-sospechoso ?x - equipaje)`: Indica que el equipaje `?x` ha sido inspeccionado.
- `(es-oficina-inspeccion ?x - posicion)`: Indica que la posición `?x` es una oficina de inspección.
- `(vacio ?n - numero)`: Indica que el número `?n` representa la cantidad vacía (cero).
- `(enganchado-a-maquina ?x - maquina ?y - vagon)`: Indica que el vagón `?y` está enganchado a la máquina `?x`.
- `(enganchado-a-vagon ?x - vagon ?y - vagon)`: Indica que el vagón `?y` está enganchado a otro vagón `?x`.

- `(ultimo-vagon ?x - maquina ?y - vagon)`: Indica que el vagón `?y` es el último vagón enganchado a la máquina `?x`.
- `(primer-vagon ?x - maquina ?y - vagon)`: Indica que el vagón `?y` es el primer vagón enganchado a la máquina `?x`.
- `(equipaje-cargado-en ?x - vagon ?y - equipaje)`: Indica que el equipaje `?y` está cargado en el vagón `?x`.
- `(carga-actual ?v - vagon ?n - numero)`: Indica la cantidad actual de carga en el vagón `?v`.
- `(next ?v - vagon ?n1 - numero ?n2 - numero)`: Indica la relación de orden entre los números `?n1` y `?n2` en el vagón `?v`.

2.3. Acciones

Las acciones permiten modificar el estado del sistema. Las acciones definidas en el dominio son:

2.3.1. Acción `move`

```
(:action move
    :parameters (?x - maquina ?y - posicion ?z - posicion)
    :precondition (and (at_maquina ?x ?y) (adjacent ?y ?z))
    :effect (and (not (at_maquina ?x ?y))
        (at_maquina ?x ?z))
)
```

- **Descripción**: Mueve una máquina de una posición a otra adyacente.
- **Precondiciones**: La máquina debe estar en la posición `?y` y las posiciones `?y` y `?z` deben ser adyacentes.
- **Efectos**: La máquina deja de estar en `?y` y ahora está en `?z`.

2.3.2. Acción `enganchar-primer-vagon`

```
(:action enganchar-primer-vagon
    :parameters (?x - maquina ?y - vagon ?z - posicion ?n - numero)
    :precondition (and (at_maquina ?x ?z) (at_vagon ?y ?z)
        (carga-actual ?y ?n)
        (vacio ?n)
```

```
        (libre ?x))
    :effect (and (not (libre ?x))
        (not(at_vagon ?y ?z))
        (enganchado-a-maquina ?x ?y)
        (ultimo-vagon ?x ?y)
        (primer-vagon ?x ?y))
)
```

- **Descripción**: Engancha un vagón como el primer vagón de la máquina.
- **Precondiciones**: El vagón debe estar en la misma posición que la máquina y estar vacío. La máquina debe estar libre.
- **Efectos**: El vagón se engancha a la máquina, se marca como primer y último vagón, y la máquina deja de estar libre.

2.3.3. Acción `enganchar-vagon`

```
(:action enganchar-vagon
    :parameters (?x - maquina ?y - vagon ?z - vagon ?p - posicion ?n - numero)
    :precondition (and (at_maquina ?x ?p) (at_vagon ?z ?p)
        (carga-actual ?z ?n)
        (vacio ?n)
        (ultimo-vagon ?x ?y))
    :effect (and (enganchado-a-maquina ?x ?z)
        (enganchado-a-vagon ?y ?z)
        (ultimo-vagon ?x ?z)
        (not(ultimo-vagon ?x ?y))
        (not(at_vagon ?z ?p)))
)
```

- **Descripción**: Engancha un nuevo vagón al último vagón de la máquina.
- **Precondiciones**: El vagón `?z` debe estar en la misma posición que la máquina y estar vacío. `?y` debe ser el último vagón.
- **Efectos**: `?z` se engancha a `?y`, se actualiza el último vagón y se elimina `?z` de su posición original.

2.3.4. Acción `desenganchar-primer-vagon`

```
(:action desenganchar-primer-vagon
    :parameters (?x - maquina ?y - vagon ?p - posicion ?n - numero)
    :precondition (and (at_maquina ?x ?p)
        (ultimo-vagon ?x ?y)
        (primer-vagon ?x ?y)
        (carga-actual ?y ?n)
        (vacio ?n)
        (enganchado-a-maquina ?x ?y))
    :effect (and (at_vagon ?y ?p)
        (not (ultimo-vagon ?x ?y))
        (not (primer-vagon ?x ?y))
        (not (enganchado-a-maquina ?x ?y))
        (libre ?x)
    )
)
```

- **Descripción**: Desengancha el primer vagón de la máquina.
- **Precondiciones**: El vagón debe estar vacío y ser tanto el primer como el último vagón enganchado a la máquina.
- **Efectos**: El vagón queda libre en la posición y se eliminan sus conexiones con la máquina. La máquina queda libre.

2.3.5. Acción `desenganchar-vagon`

```
(:action desenganchar-vagon
    :parameters (?x - maquina ?y - vagon ?z - vagon
                 ?p - posicion ?n - numero)
    :precondition (and (at_maquina ?x ?p)
        (ultimo-vagon ?x ?z)
        (carga-actual ?z ?n)
        (vacio ?n)
        (enganchado-a-maquina ?x ?y)
        (enganchado-a-maquina ?x ?z)
        (enganchado-a-vagon ?y ?z))
    :effect (and (at_vagon ?z ?p)
        (not (ultimo-vagon ?x ?z))
        (ultimo-vagon ?x ?y)
        (not (enganchado-a-maquina ?x ?z))
```

```
        (not (enganchado-a-vagon ?y ?z))
    )
)
```

- **Descripción**: Desengancha el último vagón de la máquina y lo deja en la posición actual.
- **Precondiciones**: El vagón debe estar vacío, ser el último vagón enganchado a la máquina, y estar enganchado a otro vagón.
- **Efectos**: El vagón queda libre en la posición y la máquina queda libre. Este deja de ser el último vagón y se actualiza al que estaba conectado como último. Se eliminan sus conexiones con la máquina.

2.3.6. Acción `cargar-equipaje`

```
(:action cargar-equipaje
    :parameters (?x - maquina ?y - vagon ?e - equipaje ?p - posicion
                ?n1 - numero ?n2 - numero)
    :precondition (and (at_maquina ?x ?p) (at_equipaje ?e ?p)
        (carga-actual ?y ?n1)
        (next ?y ?n1 ?n2)
        (enganchado-a-maquina ?x ?y))
    :effect (and (carga-actual ?y ?n2)
        (equipaje-cargado-en ?y ?e)
        (not (carga-actual ?y ?n1))
        (not(at_equipaje ?e ?p)))
)
```

- **Descripción**: Carga un equipaje en un vagón.
- **Precondiciones**: La máquina y el equipaje deben estar en la misma posición. El vagón debe estar enganchado a la máquina y tener un estado de carga que permita incrementar su nivel (es decir, que exista un predicado `next` con su carga como la menor. Si está en su límite, este predicado no existe, por lo cual no puede superarlo).
- **Efectos**: El equipaje se almacena en el vagón, el nivel de carga del vagón se actualiza y el equipaje desaparece de la posición.

2.3.7. Acción `descargar-un-equipaje`

```
(:action descargar-un-equipaje
    :parameters (?x - maquina ?y - vagon ?e - equipaje ?p - posicion
                 ?n1 - numero ?n2 - numero)
    :precondition (and (at_maquina ?x ?p)
        (no-sospechoso ?e)
        (carga-actual ?y ?n1)
        (next ?y ?n2 ?n1)
        (enganchado-a-maquina ?x ?y)
        (equipaje-cargado-en ?y ?e))
    :effect (and (at_equipaje ?e ?p)
        (carga-actual ?y ?n2)
        (not(carga-actual ?y ?n1))
        (not(equipaje-cargado-en ?y ?e)))
)
```

- **Descripción**: Descarga un equipaje de un vagón en una posición determinada.
- **Precondiciones**: La máquina y el vagón deben estar en la misma posición, el equipaje no debe ser sospechoso y el vagón debe tener el equipaje cargado.
- **Efectos**: El equipaje se coloca en la posición `?p`, el nivel de carga del vagón se actualiza y se elimina la asociación del equipaje con el vagón.

2.3.8. Acción `dejar-equipaje-en-oficina-de-inspeccion`

```
(:action dejar-equipaje-en-oficina-de-inspeccion
    :parameters (?x - maquina ?y - vagon ?e - equipaje ?p - posicion
                 ?n1 - numero ?n2 - numero)
    :precondition (and (at_maquina ?x ?p)
        (es-oficina-inspeccion ?p)
        (sospechoso ?e)
        (carga-actual ?y ?n1)
        (next ?y ?n2 ?n1)
        (enganchado-a-maquina ?x ?y)
        (equipaje-cargado-en ?y ?e))
    :effect (and (at_equipaje ?e ?p)
        (carga-actual ?y ?n2)
        (not(carga-actual ?y ?n1))
```

```
        (not(equipaje-cargado-en ?y ?e)))
)
```

- **Descripción**: Deja un equipaje sospechoso en una oficina de inspección.
- **Precondiciones**: La máquina y el vagón deben estar en la misma posición, que debe ser una oficina de inspección. El equipaje debe estar identificado como sospechoso y estar cargado en el vagón.
- **Efectos**: El equipaje es descargado en la oficina de inspección y se actualiza el nivel de carga del vagón.

2.3.9. Acción `investigar-equipaje`

```
(:action investigar-equipaje
    :parameters (?e - equipaje ?p - posicion)
    :precondition (and (at_equipaje ?e ?p)
        (es-oficina-inspeccion ?p))
    :effect (and (not (sospechoso ?e)) (no-sospechoso ?e))
)
```

- **Descripción**: Investiga un equipaje en la oficina de inspección.
- **Precondiciones**: El equipaje debe estar en una oficina de inspección.
- **Efectos**: El equipaje deja de ser sospechoso y se marca como no sospechoso.

3. Problema

El problema específico a resolver consiste en transportar seis equipajes desde su ubicación inicial hasta sus destinos finales. Dos de estos paquetes son sospechosos, por lo que antes de entregarlos a su destino deben pasar por la Oficina de Inspección y ser revisados en esta. La capacidad máxima de los vagones se limita a 2 equipajes. La Figura 3 muestra un esquema del estado inicial del problema a resolver.

Los objetos especificados para el problema dado por el boletín son:

- Posiciones: P1, P2, P3, P4, P5, P6, P7, P8, ZF, RE, OI.
- Máquinas: M1, M2.

- Vagones: V1, V2, V3, V4, V5.
- Equipajes: E1, E2, E3, E4, E5, E6.
- Números: N0, N1, N2.

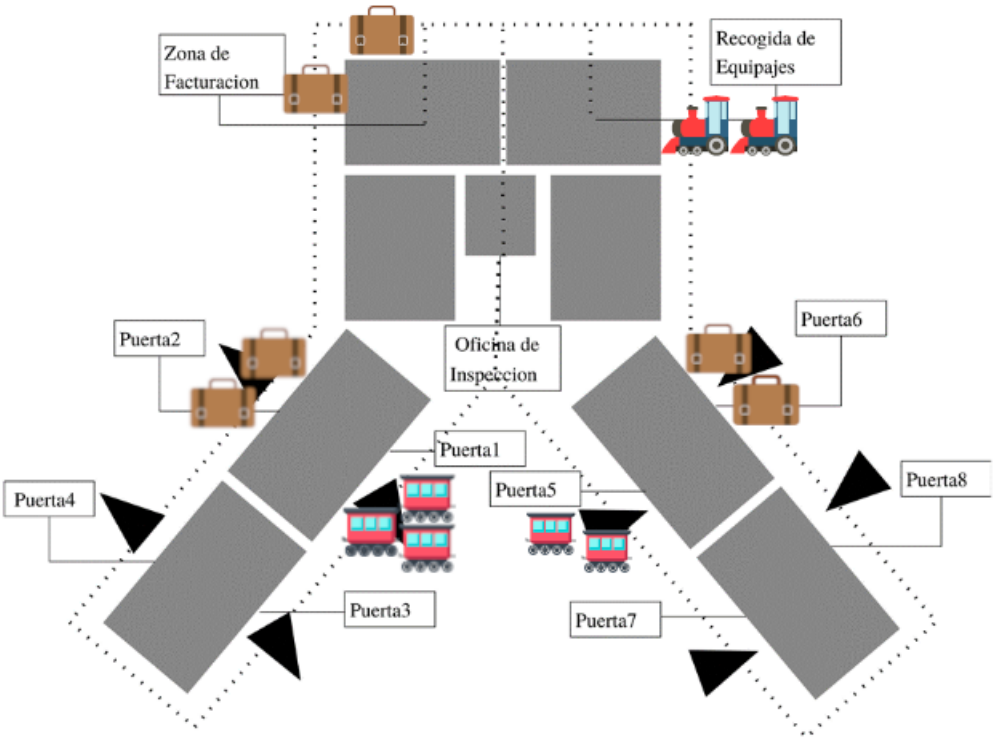

Figura 2: Representación del problema original a resolver.

3.1. Estado Inicial y Objetivo

- Posición inicial de máquinas, vagones y equipajes.
- Estado de los equipajes (sospechoso o no sospechoso):
  `(sospechoso E3) (sospechoso E6)`
  `(no-sospechoso E1) (no-sospechoso E2)`
- Conexiones entre posiciones adyacentes:
  `(adjacent P1 P2) (adjacent P2 P1),`
  `(adjacent P2 P3) (adjacent P3 P2)`...
- Capacidad de los vagones:

- : El número 0 está representado con un literal constante:
  `(vacio N0)`
- : La capacidad actual de los vagones se representan como:
  `(carga-actual V1 N0) (carga-actual V2 N2)`
  `(carga-actual V3 N1)`
- : Mediante predicados definimos la relacion de orden de los números:
  `(next V1 N0 N1) (next V1 N1 N2)`
  `(next V2 N0 N1) (next V2 N1 N2)`

El objetivo del problema es ubicar los equipajes en sus destinos designados, asegurando que aquellos sospechosos sean inspeccionados previamente. Lo definimos de la siguiente forma en el archivo de problema:

```
(:goal (and (at E1 P4) (at E2 P8) (at E3 RE)
(at E4 RE) (at E5 RE) (at E6 RE)))
```

3.2. Experimentos

Para evaluar el comportamiento de nuestro dominio y comparar distintas estrategias de planificación, partimos del **problema base** definido anteriormente (al que denominamos *Original*) y definimos diversas variantes de este modificando el número de localizaciones, máquinas, vagones y equipajes, así como la posibilidad de cargar varios equipajes en un solo vagón. En la Tabla 1 se muestran los problemas utilizados en los experimentos:

Problema	**Loc.**	**Máq.**	**Vag.**	**Equip.**	**Capacidad**
Original	11	2	5	6	2
Más equipajes	11	2	5	12	2
Más capacidad	11	2	5	6	4
Más máquinas	11	5	5	6	2
Más vagones	11	2	10	6	2
Más Todo	11	5	5	12	4

Tabla 1: Problemas utilizados en los experimentos.

En todos los casos, se persigue el mismo objetivo: ubicar los equipajes en sus destinos designados, garantizando la inspección previa de aquellos marcados como *sospechosos*. Para cada problema, se probaron distintos planificadores (`MetricFF`, `LPG` y `Optic`), comparando tanto el tiempo de cómputo necesario para encontrar un plan como el número total de acciones en dicho plan. La Tabla 2 resume los resultados obtenidos:

Problema	MetricFF		LPG		Optic	
	Tiempo	**Acciones**	**Tiempo**	**Acciones**	**Tiempo**	**Acciones**
Original	0.08s	49	0.25s	65	12.78s	61
Más equip.	0.5s	87	2.3s	238	50.30	105
Más cap.	0.1s	46	0.54s	65	31.54s	67
Más máq.	0.66s	48	0.41s	42	-	-
Más vag.	0.2s	49	17.29s	109	42.26s	61
Más todo	6.10s	65	696.4s	213	-	-

Tabla 2: Resultados obtenidos en los distintos problemas y planificadores.

Los datos evidencian que, al incrementar el tamaño del problema, el tiempo de planificación crece de forma notable en todos los planificadores. Sin embargo, se observan diferencias en cuanto al número total de acciones y la rapidez de generación de planes.

3.3. Análisis de Resultados

A partir de los experimentos realizados, podemos extraer las siguientes conclusiones y comentarios:

- **Escalabilidad del problema.** Como era de esperar, al incrementar la complejidad del dominio (ya sea aumentando la cantidad de equipajes, la capacidad de los vagones o el número de máquinas), el tiempo de planificación crece de manera considerable. Esto se observa especialmente en el caso de *Más Todo*, donde la combinación de todos los factores incrementados hace que algunos planificadores requieran un tiempo muy alto o incluso aborten (reflejado con `-`).
- **Diferencias entre planificadores.**
  - `MetricFF` resulta ser el planificador más rápido en la gran mayoría de los problemas planteados y, al mismo tiempo, el que genera la menor cantidad de acciones. Solo en un caso específico `LPG` obtuvo un plan con menos acciones, pero habitualmente `MetricFF` conserva esa ventaja tanto en velocidad de ejecución como en número de acciones.
  - `LPG` aprovecha la posibilidad de ejecutar acciones de forma paralela, pero su enfoque de *local search* puede incrementar considerablemente los tiempos de cómputo cuando el problema presenta

muchos objetos o factores a considerar. En esos casos (*Más equipajes* o *Más vagones*), tanto el tiempo como el número de acciones se incrementan de forma significativa.

- `Optic` alcanza resultados notables en algunos problemas, pero en los más complejos puede presentar tiempos muy elevados e incluso no finalizar la planificación (tal como se ve en *Más máquinas* o *Más todo*).

- **Síntesis.** En términos generales, `MetricFF` destaca como el planificador más eficiente al obtener planes de forma rápida y con menos acciones en casi todos los casos. `LPG` presenta la ventaja de la ejecución paralela, pero se ve penalizado en entornos de mayor complejidad. Por su parte, `Optic` también muestra resultados favorables en determinados escenarios, aunque su rendimiento se ve afectado cuando crece de forma considerable la complejidad del problema.

4. Desarrollo parcial de un árbol POP

En esta sección se aplicarán tres iteraciones del algoritmo de un planificador POP a nuestro problema original. La única modificación será seleccionar solo un objetivo, el cual será mover un paquete de un lugar a otro. Se trabajará con un dominio totalmente instanciado, lo que significa que no se generarán *flaws* por la instanciación de una variable. Por tanto al seleccionar un *flaw*, se generarán tantos nodos hijo como opciones haya de resolver el *flaw* con todas las variables involucradas instanciadas a un valor concreto.

Nivel 1

El nivel 1 del árbol, representado en la Figura 4, contendrá únicamente al nodo raíz el cual consiste en el plan inicial, formado solo por el estado inicial y el estado final, dichos estados se muestran en la Figura 3. El único *flaw* a resolver en el plan inicial es el objetivo `(at E3 RE)`.

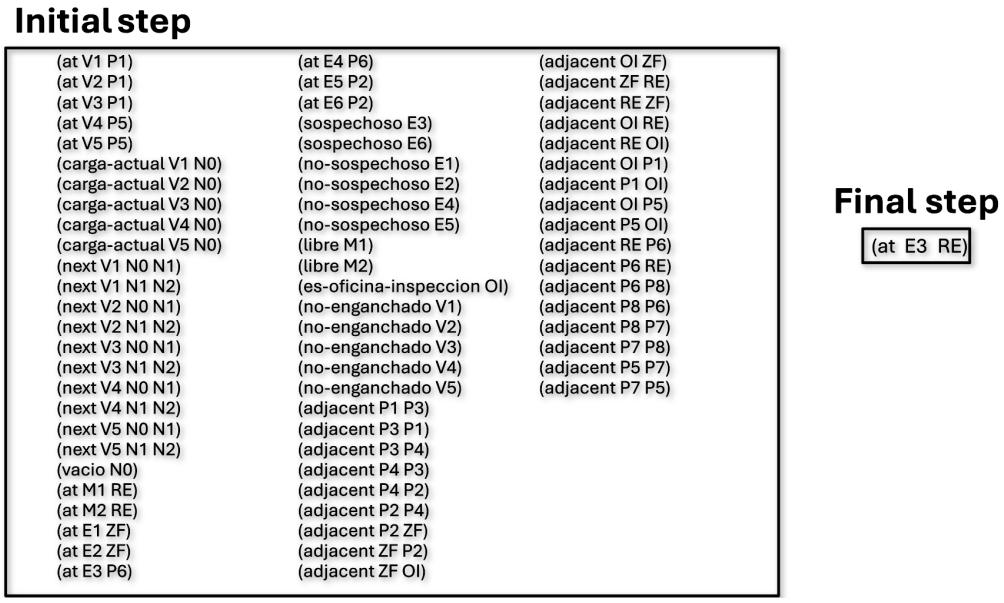

Figura 3: Estado inicial y estado final del problema.

Nivel 2

Para resolver el objetivo, la única acción válida es `(descargar-equipaje ?m ?v E3 ?n1 ?n2)`. Esta acción se añade al plan junto a su enlace causal con el estado final, como se puede ver en la Figura 5. Además, se elimina

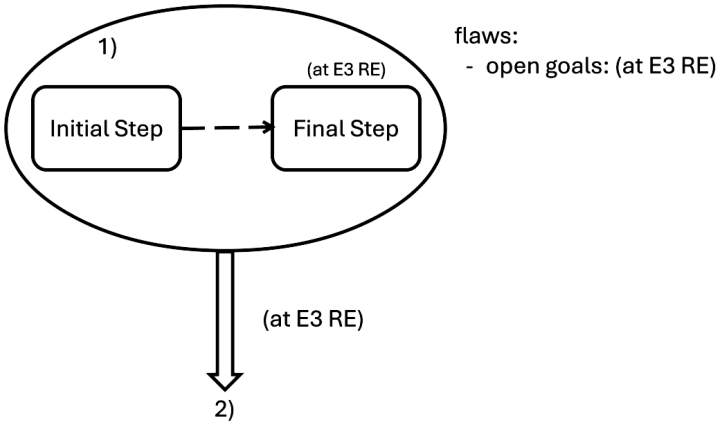

Figura 4: Nodo raíz del árbol POP.

el objetivo resuelto de la lista de *flaws* y se añaden las precondiciones de la nueva acción a la lista. Para el siguiente paso del algoritmo se seleccionó `(no-sospechoso E3)` como *flaw* a resolver. Esta selección se hizo de forma aleatoria.

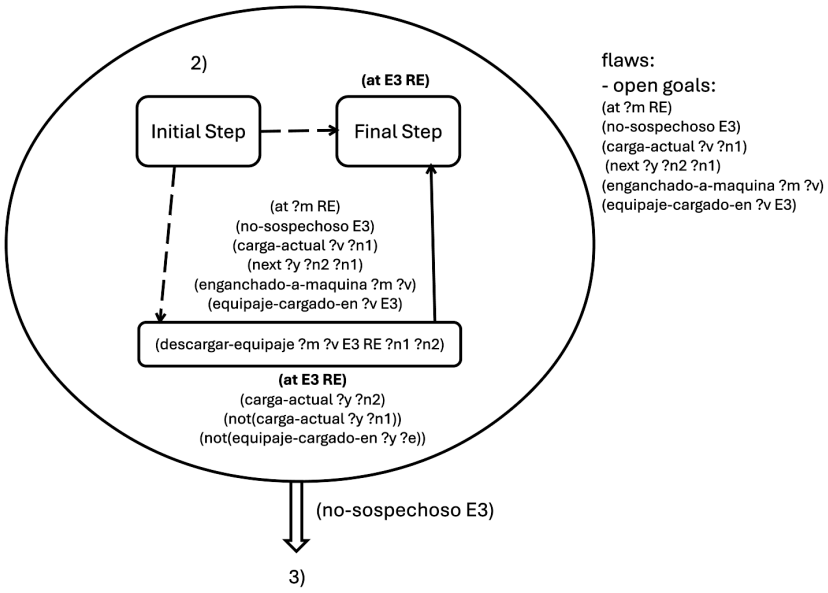

Figura 5: Nivel 2 del árbol POP.

Nivel 3

A continuación, se aplicó la acción `(investigar-equipaje E3 ?p)` para resolver el *flaw*. Se creó un enlace causal entre la acción de investigar un equipaje y la de descargar un equipaje, tal y como se muestra en la Figura 6. Además, se añadieron las precondiciones de la nueva acción la lista de *flaws*. Para el siguiente paso del algoritmo, se decidió resolver el *flaw* `(at ?m RE)`. Dicho *flaw* contiene una variable del tipo *máquina*, por lo que generará tantos hijos como máquinas haya (en este caso tenemos 2 máquinas).

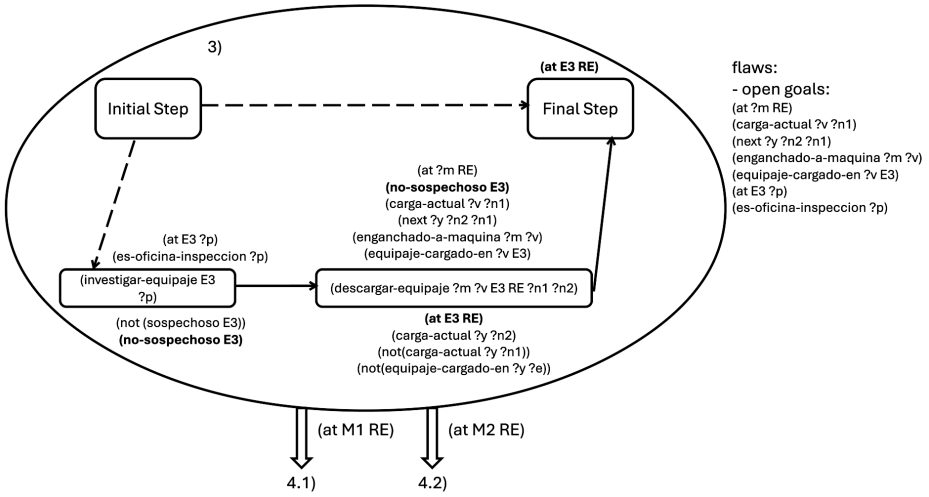

Figura 6: Nivel 3 del árbol POP.

Nivel 4

En este nivel del árbol se decidió seguir por el camino en el que la máquina es M1. Para resolver el *flaw* se agregó al plan la acción `(move M1 ?p RE)`, creando un enlace causal entre esta acción y la de descargar un equipaje. Por último, se añadieron las precondiciones de dicha acción a la lista de *flaws* pendientes de resolver. Este nivel del árbol se representa en la Figura 7, aunque solo se muestra una de las ramas.

Si continuáramos con la expansión del árbol, llegaría un momento en el que la acción de investigar un equipaje tendría que aplicarse antes que la de mover la máquina M1 a la zona de recogida de equipajes. Esto se debe a que para descargar un equipaje, la máquina tiene que estar en el punto de entrega (zona de recogida) y para investigar el equipaje este tiene que estar en la oficina de inspección. Por tanto, en el plan final la máquina deberá moverse primero a la oficina de inspección y luego al punto de entrega.

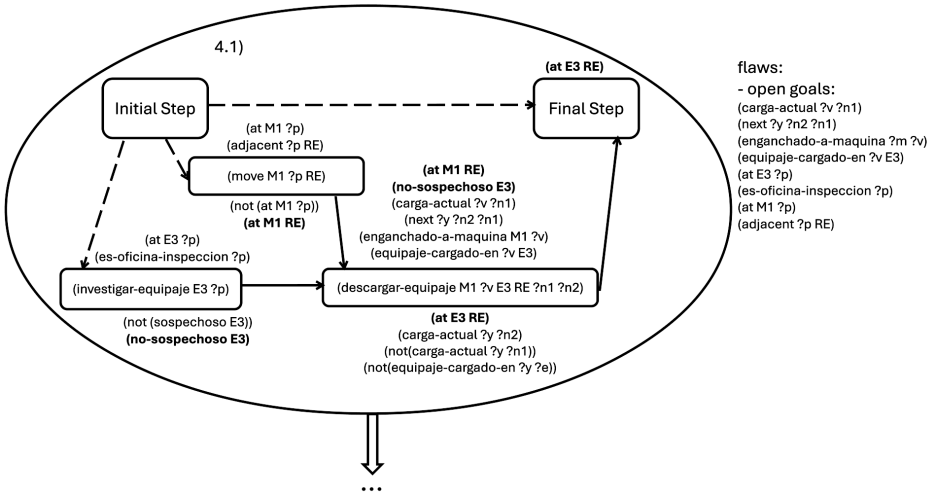

Figura 7: Nivel 4 del árbol POP.

Además, al finalizar la ejecución del algoritmo, se verá que la únicas acciones paralelizables son las que involucran a máquinas distintas. Lo que quiere decir que todas las acciones que involucren a una misma máquina deberán establecer una relación de orden.

5. Planificación en grafos

El grafo correspondiente a este ejercicio se encuentra en el archivo de Excel que acompaña la entrega. En él se listan, para cada nivel, las acciones necesarias hasta alcanzar los objetivos propuestos, junto con sus precondiciones y efectos. Dada la gran cantidad de acciones, se reducen los índices de todas las precondiciones y efectos, además de las acciones posibles no vitales para alcanzar los objetivos del problema.

En la segunda hoja del documento se indican los niveles en los que se cumplen los objetivos y se muestran los valores de las heurísticas h_{sum}, h_{max} y la del plan relajado.

5.1. Cálculo de las heurísticas h_{sum} y h_{max}

- **Heurística h_{sum}:** Para hallarla, se suman los niveles en que se satisfacen ambos objetivos. El primero se cumple en el nivel 5 y el segundo en el nivel 7, resultando:

$$h_{\text{sum}} = 7 + 5 = 13.$$

- **Heurística h_{max}:** Se obtiene tomando el nivel más alto en el que se alcanza alguno de los objetivos. Dado que el último se logra en el nivel 7, se tiene:

$$h_{\text{max}} = \text{máx}(5, 7) = 7.$$

5.2. Extracción de un plan relajado para los objetivos

El plan relajado se ilustra en el documento, donde las acciones relevantes se destacan en color rojo. Para construirlo, se parte del nivel final, eligiendo las acciones que cumplen los objetivos. Cada precondición de dichas acciones se convierte en un nuevo objetivo a satisfacer. Este proceso se repite hasta llegar al estado inicial.

Como en el plan relajado no se consideran exclusiones mutuas ni efectos negados, una vez alcanzado un efecto, este se mantiene en los niveles sucesivos. Gracias a ello, se pueden trazar rutas desde los objetivos hasta el estado inicial sin necesidad de hacer *backtracking*, manteniendo la complejidad en tiempo polinómico.

El plan relajado resultante está formado por 13 acciones.

5.3. Heurística más informada

La heurística que consideramos más informada para este problema es la del plan relajado, ya que su estimación está más próxima al coste real de alcanzar todos los objetivos. Hemos ejecutado el problema en el planificador FF y nos ha proporcionado un plan de 20 acciones, por lo cual, la heurística de plan relajado es, efectivamente, la más cercana (13 acciones).