UNIVERSITAT POLITÈCNICA DE CATALUNYA

Práctica 3
INTELIGENCIA ARTIFICIAL

# Práctica de Planificación

Autores:
Joan Caballero Castro
Jeremy Comino Raigón
Marc González Vidal

Profesor:
Javier Béjar Alonso

Q1 2022/2023


UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

# ÍNDICE

# 1. Descripción del Problema

En esta práctica el problema que se nos presenta es básicamente la necesidad de planificar rutas de rovers de transporte que tenemos en Marte y que nos permiten desplazar tanto personal autorizado como cargas de suministros según una serie de peticiones que tengamos a asentamientos.

Además, de la forma en que se han montado las bases, no tiene por qué existir camino directo de un asentamiento o almacén a otro. Es decir, si modelamos el mapa con un grafo, no tenemos un grafo completo, de lo único que estamos seguros es que tenemos un grafo conexo, ya que de lo contrario no podríamos llegar a todas las bases. Además, ya se nos advierte en el enunciado que no se van a poder servir todas las peticiones que se nos pidan, ya que hay un mayor número de peticiones que de cargas.

# 2. Dominio

## Tipos

- Base: Tipo que indica el lugar donde pueden estar situados los elementos del problema. En este caso los suministros y personal. Lo creamos con el objetivo de poder representar los lugares donde debía estar situado el escenario del problema
- Rover: Tipo que indica la unidad que permite el transporte de material y personal. Lo creamos con el objetivo de poder tratar con diferentes unidades.
- Carga: Tipo que representa el objeto que puede llevar el Rover. Lo creamos con el objetivo de poder situar en el problema diferentes cargas y así representar de mejor manera el problema dado
- Asentamiento: Subtipo de Base que identifica las zonas donde se pueden quedar tanto personal como suministro.
- Almacén: Subtipo de Base que identifica las zonas donde se pueden quedar solo suministro.

  Estos dos subtipos nos ayuda a para que a la hora de hacer entregas, solo necesitemos hacer matching con solo los Asentamientos. A la vez que nos ayuda a decir que en un Asentamiento solo puede haber al principio Personal y en un Almacen Suministro.
- Suministro: Subtipo de Carga que representa una carga de suministro.
- Personal: Subtipo de Carga que representa una carga de personal.

  Estos tipos nos ayudan a diferenciar las características respecto al volumen que ocupa dentro de un Rover.

## Funciones

- (capacidad ?r - Rover): Fluente que representa la capacidad de un rover. Esto está creado para poder dar un límite a lo que el rover puede llevar.
- (combustible ?r - Rover): Fluente que representa el nivel de combustible que tiene un rover. Esto está creado para poder dar un límite a lo que el rover se puede mover.
- (prioridad ?c - Carga ?l - Asentamiento): Fluente que represente la prioridad que hay en la entrega de la carga ?c a un asentamiento ?l. Creado para definir la prioridad de entregar una cierta carga a un cierto Asentamiento.
- (combustible-total): Fluente que representa el combustible total de todos los rover. Creado para hacer la minimización de combustible.
- (penalidad): Fluente que representa la suma de todas las penalizaciones que se dan cuando un rover da una cierta Carga. Creado para la maximización de prioridad.

# Predicados

- (Aparcado ?r - Rover ?l - Base): Relación que expresa en qué Base está situado un rover. Está creado para poder situar en una base determinada a los rovers.
- (Accesible ?l1- Base ?l2 - Base): Relación que expresa que la base l2 es accesible desde l1. Está creado para poder definir los caminos entre bases.
- (esta ?c - Carga ?l - Base): Relación que expresa donde está situado una carga. Está creado para poder situar en bases las cargas.
- (transportando ?r - Rover ?c - Carga): Relación que expresa que un rover está transportando una Carga. Está creado para que durante el proceso de transporte no se recoja más de una vez el mismo paquete.
- (objetivo ?p - Carga ?l - Asentamiento): Relación que expresa a donde tiene que ir una cierta Carga.  Está creado con el objetivo de poder decir a donde tiene que ir cada carga.
- (entregada ?c - Carga): Relación que expresa que una Carga ya ha sido entregada. Está creado con el objetivo de que cuando una carga sea entregada. Ya no se pueda volver a utilizar como carga para otra entrega.

# Acciones

- mover (?r - Rover ?l1 - Base ?l2 - Base): Acción que representa el paso de un rover de una Base a otra. Creada para poder expresar la acción de mover dentro del mapa.
- recogerS (?r - Rover ?c - Suministro ?l - Almacen): Acción de recoger un Suministro.
- recogerP (?r - Rover ?c - Personal ?l - Asentamiento): Acción de recoger un miembro de Personal

Estas dos acciones sirven para hacer matching no tan extenso ya que utilizamos subtipos específicos. Pero las dos expresan el hecho de recoger paquetes para así poder entregarlos más tarde.

- entregarS(?r - Rover ?c - Suministro ?l - Asentamiento): Acción de entregar un Suministro
- entregarP(?r - Rover ?c - Personal ?l - Asentamiento): Acción de entregar un miembro de Personal.

Estas dos acciones sirven para hacer matching no tan extenso ya que utilizamos subtipos específicos. Pero las dos expresan el hecho de entregar paquetes para así poder completar las tareas dadas

# 3. Modelización

En este apartado se describen los diferentes elementos que aparecen en los distintos modelos desarrollados para cada una de las versiones del problema que han sido consideradas. En cada extensión se han añadido nuevas funcionalidades al modelo, así que en cada descripción solo se añadirán los nuevos cambios.

## Nivel básico

En este primer nivel básico se realiza el transporte de todos los suministros y personal especializado que tenemos disponible. Tenemos rovers que se desplazan entre bases y tienen que transportar las cargas de suministros y de personal a los asentamientos que han hecho las peticiones.

- Objetos

Con tal de representar los elementos del problema hemos creado los tipos siguientes:

  - Rover: Es el medio de transporte de los suministros y del personal autorizado. Se puede mover entre bases, recoger y entregar cargas.
  - Asentamiento: Es un subtipo de Base donde viven los colonos. En ellos también se almacena el personal autorizado y es la destinación de todas las peticiones.
  - Almacen: Es un subtipo de Base donde llegan y se almacenan los suministros de la Tierra.
  - Suministro: Es un subtipo de Carga que representa una carga de suministro.
  - Personal: Es un subtipo de Carga que representa una carga de personal.
  - pSuministro: Es un subtipo de Petición que pide suministros.
  - pPersonal: Es un subtipo de Petición que pide personal especializado.

Para representar las características de los elementos hemos creado los predicados siguientes:

  - (aparcado ?r - Rover ?l - Base): Indica si el rover está aparcado en la base.
  - (accesible ?l1 - Base ?l2 - Base): Indica si se puede acceder a la base l2 desde la base l1.
  - (esta ?c - Carga ?l - Base): Indica si la carga se encuentra situada en la base.
  - (transportando ?r - Rover ?c - Carga): Indica si el rover está transportando la carga.
  - (objetivo ?p - Peticion ?l - Asentamiento): Indica si la petición tiene como base destino el asentamiento.
  - (servida ?p - Peticion): Indica si la petición ya está servida.
  - (entregada ?c - Carga): Indica si la carga ya se ha entregado. Nos servirá para definir nuestro estado GOAL.

Para representar las tareas de los elementos hemos creado las acciones siguientes:

  - mover (?r - Rover ?l1 - Base ?l2 - Base): Acción que representa el paso de un rover de una Base a otra.
  - recogerS (?r - Rover ?c - Suministro ?l - Almacen): Acción de recoger un Suministro.

    - recogerP (?r - Rover ?c - Personal ?l - Asentamiento): Acción de recoger un miembro de Personal
    - entregarS(?r - Rover ?c - Suministro ?l - Asentamiento ?p - pSuministro): Acción de entregar un Suministro a la Petición pSuministro.
    - entregarP(?r - Rover ?c - Personal ?l - Asentamiento ?p - pPersonal): Acción de entregar un miembro de Personal a la Petición pPersonal.

- Estado inicial
    - Los rovers estarán aparcados en bases.
    - Las bases serán accesibles a otras bases.
    - Las cargas estarán situadas en bases.
    - Habrá peticiones de suministros y personal que tendrán como lugar de destino asentamientos.

- Estado final

Nuestro estado final será aquel en el que hayamos repartido todos los suministros y personal autorizado. Con la siguiente línea de código podemos representar nuestro objetivo:

(:goal (forall (?c - Carga) (entregada ?c)))

# Extensión 1

Esta primera extensión está construida a partir del nivel básico. Se añade una limitación de capacidad a los rovers, en la que pueden transportar a la vez un máximo de dos personas o una sola carga de suministros, y no se pueden mezclar personas y suministros.

- Objetos

Con tal de representar esta limitación en nuestro programa hemos añadido un fluente que nos indicará la capacidad de carga que tiene cada rover en su momento. El rover tendrá una capacidad inicial máxima de 2 unidades, por cada carga que recoja se decrementará su valor y por cada carga que entregue se incrementará.

Los suministros ocuparán dos unidades de capacidad y el personal una unidad. Por tanto, si el rover recoge un suministro, su capacidad se reducirá en dos unidades y cuando lo entregue se incrementará en dos. En cambio, con el personal solo se decrementará e incrementará en una sola unidad, siendo capaz de llevar dos personales a la vez.

(capacidad ?r - Rover)

Los cambios que hemos hecho sobre las acciones han sido añadir en la precondición de recogerS y recogerP una comprobación de la capacidad del rover, y en su efecto la disminución de su capacidad. Para entregarS y entregarP también hemos modificado su efecto incrementando la capacidad del rover.

- Estado inicial
    - Deberemos inicializar el fluente de capacidad a 2 unidades para cada rover.

# Extensión 2

La segunda extensión está construida a partir de la primera extensión. Se añade una capacidad de combustible a los rovers, la cual se carga al principio del día y se decrementa por cada desplazamiento entre dos bases.

- Objetos

Con tal de representar esta limitación en nuestro programa hemos añadido un fluente que indica la cantidad de combustible restante que tiene cada rover. El rover tendrá una capacidad inicial y por cada acción de mover que haga se reducirá en una unidad.

(combustible ?r - Rover)

Como también nos piden hacer una versión que minimice el total de combustible gastado, hemos añadido otro fluente que se inicializará a 0 en el principio del programa y se irá incrementando cada vez que un rover haga un movimiento.

(combustible-total)

Los cambios que hemos hecho sobre las acciones han sido añadir en la precondición de mover una comprobación del combustible del rover y en su efecto la disminución del combustible del rover en una unidad.
Para el caso de la versión donde queremos minimizar el combustible total gastado, hemos añadido en su efecto un incremento del fluente combustible-total en una unidad.

También hemos añadido una métrica al problema que intentará minimizar el fluente de combustible-total.

(:metric minimize (combustible-total))

- Estado inicial
    - En cuanto al primer fluente, inicializaremos la cantidad de combustible de cada rover a un valor arbitrario que nos dará el problema.
    - También deberemos inicializar el fluente combustible-total a 0.

# Extensión 3

La tercera extensión está construida a partir de la segunda extensión. Se añade una prioridad a las peticiones, y se intenta buscar un plan que maximice la prioridad de las peticiones servidas.

- Objetos

Intentamos realizar esta extensión con el tipo Peticion tal como habíamos hecho con las extensiones anteriores, pero nos dimos cuenta que nuestro programa era muy lento y que necesitaba de más información.

Antes nuestro programa intentaba decidir por su cuenta cuáles eran las mejores cargas que se tenían que entregar para cada petición, y ahora le ayudamos proporcionándole la información de qué carga tiene que ir con cada petición.

Para ello hemos eliminado el tipo Peticion y lo hemos suplantado modificando el predicado de objetivo, que pasa de estar relacionado con la petición y el asentamiento de destino

- (objetivo ?p - Peticion ?l - Asentamiento): Indica si la petición tiene como base destino el asentamiento.

a estar relacionado con la carga y el asentamiento de destino:

- (objetivo ?c - Carga ?l - Asentamiento): Indica qué base tiene la carga como objetivo (puede haber diferentes bases para una misma carga).

Con tal de representar las prioridades de las peticiones hemos añadido un fluente que nos indicará para cada objetivo (dada una carga y un asentamiento), qué prioridad (con valores del 1 al 3) tiene:

(prioridad ?c - Carga ?l - Asentamiento): Indica la prioridad del pedido de la carga al asentamiento.

Para maximizar la prioridad de las peticiones servidas hemos añadido un fluente que utilizaremos como penalizador. Por cada petición entregada restaremos 3 con la prioridad de la petición entregada, y el resultado de la resta lo incrementaremos al fluente.

(penalidad)

Así, al entregar una petición de prioridad 3 no incrementaremos la penalidad, pero al entregar una petición de prioridad 1 le sumaremos 2 y de prioridad 2 le sumaremos 1. Nuestro objetivo es que este penalizador tenga el menor valor posible, y para ello hemos añadido una métrica.

(:metric minimize (penalidad))

Para la versión donde queremos optimizar una combinación entre prioridades y combustible total, hemos hecho una métrica aplicando diferentes pesos a cada criterio.

(:metric minizime (+ (* α (penalidad)) (* β (combustible-total))))

- Estado inicial
  - Deberemos inicializar el fluente penalizador a 0.
  - Cada petición la deberemos representar con el predicado objetivo y otorgarle la prioridad deseada con el fluente prioridad.

# 4. Desarrollo de los modelos

Para el desarrollo de los distintos problemas hemos optado por un diseño incremental basado en prototipos siguiendo el guion del enunciado de la práctica. Empezamos por el problema básico y fuimos ampliando poco a poco, teniendo en cuenta las características únicas que nos propone cada extensión.

En cada prototipo hemos ido añadiendo nuevos elementos del problema hasta llegar a la última extensión. En cambio, en la última extensión tuvimos que realizar una serie de cambios respecto al prototipo anterior para mejorar el tiempo de ejecución de nuestro programa.

Aparte de los modelos en PDDL, también hemos desarrollado un generador de entradas para los juegos de prueba de nuestro problema.

# 5. Juegos de prueba

En este apartado vamos a explicar tanto como hemos generado los diferentes juegos de prueba como también la documentación de cada uno.

## Generación de los juegos de prueba

Para hacer un generador de ficheros de prueba automático hemos creado un programa en C++ que recibe los siguientes parámetros como inputs:

- Semilla: semilla que se utilizará para la randomización del output.
- Número de rovers
- Número de asentamientos
- Número de almacenes
- Número de suministros
- Número de personal

Posteriormente se modela el mapa mediante un grafo bidireccional que tiene como número de nodos la suma de almacenes y suministros, es decir, las bases totales. Para esto se tiene en cuenta que si numeramos los nodos del 1 al total, primero estamos contando asentamientos y posteriormente almacenes.

Una vez creado el grafo, nos aseguramos que este sea conexo, para hacerlo unimos bidireccionalmente el nodo 1 con el nodo 2 y así sucesivamente hasta tener un grafo conexo. Para añadir aleatoriedad al problema y no siempre tener el mismo mapa conexo vamos a añadir aristas bidireccionales de una forma aleatoria según una probabilidad, y nos es muy difícil demostrar que si un grafo es conexo con la adición de una arista, del grafo sigue siendo conexo.

Posteriormente pasamos a una fase de quitado de aristas random, aquí tenemos que ir con más cuidado ya que puede ser que el grafo no sea conexo, entonces para hacerlo procedemos a quitar la arista bidireccionalmente y comprobamos que el grafo siga siendo conexo mediante un algoritmo (DFS) en caso de que sea conexo continuamos, en caso contrario volvemos a añadir las aristas que hemos quitado ya que són aristas de corte.
Con este procedimiento vamos a ser capaces de generar mapas aleatorios para poder tener inputs del problema de planificación.

Finalmente, el programa genera los output que nuestro fichero del problema necesita para funcionar.

# Juego de prueba 1

## Contextualización

Para esta prueba hemos utilizado. 2 Rovers, 4 Asentamientos, 5 Almacenes, 2 Suministros, 1 carga de Personal, 3 peticiones de suministros y 1 petición de personal. Lo interesante de esta prueba es que el mapa es un grafo arbol. Es decir que por cada par de nodos, solo existe un camino. Por lo tanto queríamos ver si funcionan las métricas de optimización. A continuación ponemos a modo de descripción más detallada el input de la extensión 3.

```
(define (problem ext1)
(:domain ext1)
(:objects       r1 r2 - Rover
                        as1 as2 as3 as4 - Asentamiento
                        al1 al2 al3 al4 al5 - Almacen
                        s1 s2 - Suministro
                        pers1 - Personal
)

(:init
        (= (capacidad r1) 2)
        (= (capacidad r2) 2)
        (= (combustible r1) 20)
        (= (combustible r2) 20)
        (= (combustible-total) 0)
        (=(penalidad) 0)
        (aparcado r1 as1)
        (aparcado r2 as1)

        (accesible as1 al3) (accesible al3 as1)
        (accesible al3 as4) (accesible as4 al3)
        (accesible as4 al4) (accesible al4 as4)
        (accesible al3 al5) (accesible al5 al3)
        (accesible al3 as3) (accesible as3 al3)
        (accesible as3 al1) (accesible al1 as3)
        (accesible al1 as2) (accesible as2 al1)
        (accesible as3 al2) (accesible al2 as3)

        (esta pers1 as3)
        (esta s1 al4)
        (esta s2 al4)

        (objetivo s1 as4)
        (=(prioridad s1 as4) 1)
        (objetivo s1 as1)
        (=(prioridad s1 as1) 3)
```

```
        (objetivo pers1 as2)
        (=(prioridad pers1 as2) 2)
        (objetivo s2 as3)
        (=(prioridad s2 as3) 2)
)

(:goal (forall (?c - Carga) (entregada ?c)))

(:metric minimize (penalidad))
)
```

## Resultados

Hemos puesto las mismas condiciones en cada extensión. A continuación enseñamos los resultados.

Output extensión 1:

```
step    0: MOVER R1 AS1 AL3
        1: MOVER R1 AL3 AS3
        2: RECOGERP R1 PERS1 AS3
        3: MOVER R1 AS3 AL1
        4: MOVER R1 AL1 AS2
        5: ENTREGARP R1 PERS1 AS2 P2
        6: MOVER R2 AS1 AL3
        7: MOVER R2 AL3 AS4
        8: MOVER R2 AS4 AL4
        9: RECOGERS R2 S1 AL4
       10: MOVER R1 AS2 AL1
       11: MOVER R1 AL1 AS3
       12: MOVER R2 AL4 AS4
       13: MOVER R1 AS3 AL3
       14: MOVER R1 AL3 AS4
       15: MOVER R1 AS4 AL4
       16: RECOGERS R1 S2 AL4
       17: MOVER R1 AL4 AS4
       18: MOVER R1 AS4 AL3
       19: MOVER R1 AL3 AS3
       20: ENTREGARS R1 S2 AS3 P3
       21: ENTREGARS R2 S1 AS4 P4
```

Principalmente lo que podemos destacar es que principalmente todas las entregas las ha hecho el Rover 1, todo esto debido a que no tiene asignada aún el tema del combustible. A su vez, no ha tenido criterio en elegir los pedidos, por lo tanto los ha escogido de manera random.

Output 2 sin optimizar combustible:

```
step    0: MOVER R1 AS1 AL3
        1: MOVER R1 AL3 AS3
        2: RECOGERP R1 PERS1 AS3 P2
        3: MOVER R1 AS3 AL1
        4: MOVER R1 AL1 AS2
        5: ENTREGARP R1 PERS1 AS2 P2
        6: MOVER R2 AS1 AL3
        7: MOVER R2 AL3 AS4
        8: MOVER R2 AS4 AL4
        9: RECOGERS R2 S1 AL4 P4
       10: MOVER R1 AS2 AL1
       11: MOVER R1 AL1 AS3
       12: MOVER R1 AS3 AL3
       13: MOVER R2 AL4 AS4
       14: MOVER R1 AL3 AS4
       15: MOVER R1 AS4 AL4
       16: RECOGERS R1 S2 AL4 P3
       17: MOVER R1 AL4 AS4
       18: MOVER R1 AS4 AL3
       19: MOVER R1 AL3 AS3
       20: ENTREGARS R1 S2 AS3 P3
       21: ENTREGARS R2 S1 AS4 P4
```

Output extensión 2 con optimización de combustible:

```
step    0: MOVER R1 AS1 AL3
        1: MOVER R1 AL3 AS3
        2: RECOGERP R1 PERS1 AS3 P2
        3: MOVER R1 AS3 AL1
        4: MOVER R1 AL1 AS2
        5: ENTREGARP R1 PERS1 AS2 P2
        6: MOVER R2 AS1 AL3
        7: MOVER R2 AL3 AS4
        8: MOVER R2 AS4 AL4
        9: RECOGERS R2 S2 AL4 P4
       10: MOVER R1 AS2 AL1
       11: MOVER R1 AL1 AS3
       12: MOVER R1 AS3 AL3
       13: MOVER R2 AL4 AS4
       14: MOVER R1 AL3 AS4
       15: MOVER R1 AS4 AL4
       16: RECOGERS R1 S1 AL4 P3
       17: MOVER R1 AL4 AS4
       18: MOVER R1 AS4 AL3
       19: MOVER R1 AL3 AS3
       20: ENTREGARS R1 S1 AS3 P3
       21: ENTREGARS R2 S2 AS4 P4
```

Vemos que no hay diferencia entre los 2 inputs debido a que al estar un en un grafo árbol. Solo hay un camino a seguir por lo tanto siempre se hará el camino óptimo.

Output extensión 3 optimizando solo prioridad:

step 0: MOVER R1 AS1 AL3
1: MOVER R1 AL3 AS3
2: RECOGERP R1 PERS1 AS3
3: MOVER R1 AS3 AL1
4: MOVER R1 AL1 AS2
5: ENTREGARP R1 PERS1 AS2
6: MOVER R1 AS2 AL1
7: MOVER R1 AL1 AS3
8: MOVER R1 AS3 AL2
9: MOVER R2 AS1 AL3
10: MOVER R2 AL3 AS1
11: MOVER R1 AL2 AS3
12: MOVER R1 AS3 AL1
13: MOVER R1 AL1 AS2
14: MOVER R2 AS1 AL3
15: MOVER R2 AL3 AS1
16: MOVER R1 AS2 AL1
17: MOVER R1 AL1 AS3
18: MOVER R1 AS3 AL2
19: MOVER R2 AS1 AL3
20: MOVER R2 AL3 AS1
21: MOVER R1 AL2 AS3
22: MOVER R1 AS3 AL1
23: MOVER R1 AL1 AS2
24: MOVER R2 AS1 AL3
25: MOVER R2 AL3 AS1
26: MOVER R1 AS2 AL1
27: MOVER R1 AL1 AS3
28: MOVER R1 AS3 AL2
29: MOVER R2 AS1 AL3
30: MOVER R2 AL3 AS1
31: MOVER R1 AL2 AS3
32: MOVER R2 AS1 AL3
33: MOVER R2 AL3 AS1
34: MOVER R2 AS1 AL3
35: MOVER R2 AL3 AS4
36: MOVER R2 AS4 AL4
37: RECOGERS R2 S1 AL4
38: MOVER R2 AL4 AS4

39: ENTREGARS R2 S1 AS4
40: MOVER R2 AS4 AL4
41: RECOGERS R2 S2 AL4
42: MOVER R2 AL4 AS4
43: MOVER R2 AS4 AL3
44: MOVER R2 AL3 AS3
45: ENTREGARS R2 S2 AS3

Output extensión 3 optimizando tanto prioridad como combustible:

step 0: MOVER R1 AS1 AL3
1: MOVER R2 AS1 AL3
2: MOVER R1 AL3 AS3
3: RECOGERP R1 PERS1 AS3
4: MOVER R1 AS3 AL1
5: MOVER R1 AL1 AS2
6: ENTREGARP R1 PERS1 AS2
7: MOVER R2 AL3 AS4
8: MOVER R2 AS4 AL4
9: RECOGERS R2 S1 AL4
10: MOVER R2 AL4 AS4
11: ENTREGARS R2 S1 AS4
12: MOVER R2 AS4 AL4
13: RECOGERS R2 S2 AL4
14: MOVER R2 AL4 AS4
15: MOVER R2 AS4 AL3
16: MOVER R2 AL3 AS3
17: ENTREGARS R2 S2 AS3

Inesperadamente hemos observado que si sólo optimizamos la prioridad. El planificador parece que se olvida del consumo. Haciendo que todo un rover gaste todo su combustible y el otro haga todo el trabajo. Además, cabe resaltar no han encontrado el óptimo en prioridad a escogido el de prioridad 1 antes que los de prioridad 3.

# Juego de prueba 2

## Contextualización

Para esta prueba hemos utilizado 2 rovers con una autonomía de 20 unidades (nivel del combustible). El mapa está conformado por 3 asentamientos y 4 almacenes. Hemos decidido que hay 7 cargas en total y un total de 8 pedidos. Todo esto con el objetivo de probar inputs de nuestro generador de problemas a la vez de ver hasta qué tamaño es escalable los problemas. A continuación ponemos a modo de descripción más detallada el input de la extensión 3.

```
(define (problem ext1)
(:domain ext1)
(:objects
        r1 r2  - Rover
        as1 as2 as3 - Asentamiento
        al1 al2 al3 al4 - Almacen
        s1 s2 s3 - Suministro
        pers1 pers2 pers3 pers4 - Personal
)

(:init
        (= (capacidad r1) 2)
        (= (capacidad r2) 2)

        (=(combustible r1) 20)
        (=(combustible r2) 20)

        (=(combustible-total) 0)
        (=(penalidad) 0)
        (aparcado r1 al1)
        (aparcado r2 as1)

        (accesible as1 as2)
        (accesible as1 as3)
        (accesible as1 al1)
        (accesible as2 as1)
        (accesible as2 as3)
        (accesible as2 al3)
        (accesible as3 as1)
        (accesible as3 as2)
        (accesible as3 al1)
        (accesible as3 al2)
        (accesible as3 al3)
        (accesible al1 as1)
        (accesible al1 as3)
```

```
                (accesible al1 al2)
                (accesible al1 al3)
                (accesible al2 as3)
                (accesible al2 al1)
                (accesible al2 al4)
                (accesible al3 as2)
                (accesible al3 as3)
                (accesible al3 al1)
                (accesible al3 al4)
                (accesible al4 al2)
                (accesible al4 al3)

                (esta pers1 as1)
                (esta pers2 as1)
                (esta pers3 as2)
                (esta pers4 as2)
                (esta s1 al4)
                (esta s2 al3)
                (esta s3 al2)

                (objetivo s1 as1)
                (= (prioridad s1 as1) 1)
                (objetivo s1 as3)
                (= (prioridad s1 as3) 3)
                (objetivo s2 as3)
                (= (prioridad s2 as3) 2)
                (objetivo s3 as3)
                (= (prioridad s3 as3) 2)
                (objetivo pers1 as3)
                (= (prioridad pers1 as3) 3)
                (objetivo pers2 as3)
                (= (prioridad pers2 as3) 2)
                (objetivo pers3 as3)
                (= (prioridad pers3 as3) 3)
                (objetivo pers4 as3)
                (= (prioridad pers4 as3) 2)
)

(:goal (forall (?c - Carga) (entregada ?c)))
(:metric minimize (+(combustible-total) (penalidad)))
)
```

## Resultados

Hemos probado por extensión estas mismas condiciones. A continuación compararemos los resultados por cada extensión:

Output extensión 1:

```
step    0: RECOGERP R2 PERS1 AS1
        1: RECOGERP R2 PERS2 AS1
        2: MOVER R2 AS1 AS3
        3: ENTREGARP R2 PERS1 AS3 PP4
        4: ENTREGARP R2 PERS2 AS3 PP1
        5: MOVER R1 AL1 AL3
        6: RECOGERS R1 S2 AL3
        7: MOVER R1 AL3 AS3
        8: ENTREGARS R1 S2 AS3 PS1
        9: MOVER R1 AS3 AS2
       10: RECOGERP R1 PERS3 AS2
       11: RECOGERP R1 PERS4 AS2
       12: MOVER R1 AS2 AS3
       13: ENTREGARP R1 PERS3 AS3 PP3
       14: ENTREGARP R1 PERS4 AS3 PP2
       15: MOVER R2 AS3 AL2
       16: RECOGERS R2 S3 AL2
       17: MOVER R2 AL2 AS3
       18: ENTREGARS R2 S3 AS3 PS3
       19: MOVER R1 AS3 AL3
       20: MOVER R1 AL3 AL4
       21: RECOGERS R1 S1 AL4
       22: MOVER R1 AL4 AL3
       23: MOVER R1 AL3 AS3
       24: ENTREGARS R1 S1 AS3 PS2
```

Debido a que en esta extensión, nuestro dominio comprendía también el tipo Petición, podemos ver ha escogido las peticiones sin tener ningún tipo de criterio.

Output extensión 2 sin optimizar

```
step    0: RECOGERP R2 PERS1 AS1 PP1
        1: RECOGERP R2 PERS2 AS1 PP2
        2: MOVER R2 AS1 AS3
        3: ENTREGARP R2 PERS1 AS3 PP2
        4: ENTREGARP R2 PERS2 AS3 PP1
        5: MOVER R1 AL1 AL3
        6: RECOGERS R1 S2 AL3 PS1
        7: MOVER R1 AL3 AS3
        8: ENTREGARS R1 S2 AS3 PS1
```

9: MOVER R1 AS3 AS2
10: RECOGERP R1 PERS3 AS2 PP3
11: RECOGERP R1 PERS4 AS2 PP4
12: MOVER R1 AS2 AS3
13: ENTREGARP R1 PERS3 AS3 PP4
14: ENTREGARP R1 PERS4 AS3 PP3
15: MOVER R2 AS3 AL2
16: RECOGERS R2 S3 AL2 PS2
17: MOVER R2 AL2 AS3
18: ENTREGARS R2 S3 AS3 PS2
19: MOVER R2 AS3 AL3
20: MOVER R2 AL3 AL4
21: RECOGERS R2 S1 AL4 PS3
22: MOVER R2 AL4 AL3
23: MOVER R2 AL3 AS3
24: ENTREGARS R2 S1 AS3 PS3

Output extensión 2 optimizado:
step 0: MOVER R1 AL1 AL3
1: RECOGERS R1 S2 AL3 PS1
2: MOVER R1 AL3 AS3
3: RECOGERP R2 PERS1 AS1 PP1
4: RECOGERP R2 PERS2 AS1 PP2
5: MOVER R2 AS1 AS3
6: ENTREGARP R2 PERS1 AS3 PP2
7: ENTREGARP R2 PERS2 AS3 PP1
8: ENTREGARS R1 S2 AS3 PS1
9: MOVER R2 AS3 AS2
10: RECOGERP R2 PERS3 AS2 PP3
11: RECOGERP R2 PERS4 AS2 PP4
12: MOVER R2 AS2 AS3
13: ENTREGARP R2 PERS3 AS3 PP4
14: ENTREGARP R2 PERS4 AS3 PP3
15: MOVER R1 AS3 AL2
16: MOVER R2 AS3 AS1
17: RECOGERS R1 S3 AL2 PS3
18: MOVER R1 AL2 AS3
19: ENTREGARS R1 S3 AS3 PS3
20: MOVER R1 AS3 AL3
21: MOVER R1 AL3 AL4
22: RECOGERS R1 S1 AL4 PS2
23: MOVER R1 AL4 AL3
24: MOVER R1 AL3 AS3
25: ENTREGARS R1 S1 AS3 PS2

Lo primero que podemos ver comparando estos 2 outputs. Es que el plan que intenta optimizar el consumo de combustible obtiene más pasos a hacer aunque tampoco es una desviación considerable, teniendo en cuenta que solo ha hecho un paso extra. Entendemos que esto es debido al algoritmo que utiliza para optimizar ya que no hace una exploración sobre todo los posibles descendientes en el fast-forward. A su vez, podemos ver que en esta optimización se han comprobado más estados que en el plan no optimizado (11005 estados en comparación con 358 estados).

Output extensión 3 optimizando la prioridad.

```
step    0: RECOGERP R2 PERS1 AS1
        1: RECOGERP R2 PERS2 AS1
        2: MOVER R1 AL1 AL3
        3: RECOGERS R1 S2 AL3
        4: MOVER R1 AL3 AL1
        5: MOVER R1 AL1 AL3
        6: MOVER R1 AL3 AL1
        7: MOVER R1 AL1 AL3
        8: MOVER R1 AL3 AL1
        9: MOVER R1 AL1 AL3
       10: MOVER R1 AL3 AL1
       11: MOVER R1 AL1 AL3
       12: MOVER R1 AL3 AL1
       13: MOVER R1 AL1 AL3
       14: MOVER R1 AL3 AL1
       15: MOVER R1 AL1 AL3
       16: MOVER R1 AL3 AL1
       17: MOVER R1 AL1 AL3
       18: MOVER R1 AL3 AL1
       19: MOVER R1 AL1 AL3
       20: MOVER R1 AL3 AL1
       21: MOVER R1 AL1 AL3
       22: MOVER R2 AS1 AL1
       23: MOVER R1 AL3 AS3
       24: ENTREGARS R1 S2 AS3
       25: MOVER R2 AL1 AL3
       26: MOVER R2 AL3 AL4
       27: MOVER R2 AL4 AL3
       28: MOVER R2 AL3 AL4
       29: MOVER R2 AL4 AL3
       30: MOVER R2 AL3 AL4
       31: MOVER R2 AL4 AL3
       32: MOVER R2 AL3 AL4
       33: MOVER R2 AL4 AL3
```

34: MOVER R2 AL3 AL1
35: MOVER R2 AL1 AS3
36: ENTREGARP R2 PERS2 AS3
37: ENTREGARP R2 PERS1 AS3
38: MOVER R2 AS3 AL3
39: MOVER R2 AL3 AL4
40: RECOGERS R2 S1 AL4
41: MOVER R2 AL4 AL3
42: MOVER R2 AL3 AS3
43: ENTREGARS R2 S1 AS3
44: MOVER R2 AS3 AL2
45: RECOGERS R2 S3 AL2
46: MOVER R2 AL2 AS3
47: ENTREGARS R2 S3 AS3
48: MOVER R2 AS3 AS2
49: RECOGERP R2 PERS3 AS2
50: RECOGERP R2 PERS4 AS2
51: MOVER R2 AS2 AS3
52: ENTREGARP R2 PERS4 AS3
53: ENTREGARP R2 PERS3 AS3

Output extensión 3 optimizando la prioridad y el consumo.

step 0: MOVER R1 AL1 AL3
1: RECOGERS R1 S2 AL3
2: MOVER R1 AL3 AS3
3: ENTREGARS R1 S2 AS3
4: RECOGERP R2 PERS1 AS1
5: RECOGERP R2 PERS2 AS1
6: MOVER R2 AS1 AS3
7: ENTREGARP R2 PERS2 AS3
8: ENTREGARP R2 PERS1 AS3
9: MOVER R1 AS3 AS2
10: RECOGERP R1 PERS3 AS2
11: RECOGERP R1 PERS4 AS2
12: MOVER R1 AS2 AS3
13: ENTREGARP R1 PERS4 AS3
14: ENTREGARP R1 PERS3 AS3
15: MOVER R1 AS3 AL2
16: MOVER R2 AS3 AS1
17: RECOGERS R1 S3 AL2
18: MOVER R1 AL2 AS3
19: ENTREGARS R1 S3 AS3
20: MOVER R1 AS3 AL3
21: MOVER R1 AL3 AL4

22: RECOGERS R1 S1 AL4
23: MOVER R1 AL4 AL3
24: MOVER R1 AL3 AS3
25: ENTREGARS R1 S1 AS3

Podemos ver principalmente 2 observaciones. La primera de todas es la influencia de la optimización del consumo. En el primer output podemos observar como un Rover (r1) se mueve principalmente en un bucle hasta agotar el combustible y posteriormente se hace el la tarea de entregar por el otro Rover.

La segunda observación es que al igual que en la segunda extensión no ha encontrado la solución óptima en el término de consumo. Pero si que nos ha sorprendido que sí que ha encontrado la solución óptima para la prioridad. Escogiendo los paquetes que van a as3 y no coger ese paquete de prioridad 1 que va a as1.

# Juego de prueba 3

## Contextualización

Para esta prueba hemos hecho un mapa principal en forma de línea que donde los 4 Cargas (2 Suministros y 2 de Personal) están situados a lo largo de esta. Los Rovers están situados en el medio de esta línea y el objetivo es entregar estás cargas en Zonas que están en las puntas de está línea formada por diferentes Bases. Aquí mostramos como ejemplo el input del Dominio1:

```
(define (problem ext1)
(:domain ext1)
(:objects
        r1 r2 r3 - Rover
        base as1 as2 as3 as4 as5 en1 en2 en3 en4 en5 en6 - Asentamiento
        al1 al2 al3 al4 al5 - Almacen
        s1 s2 - Suministro
        pers1 pers2 - Personal
        ps1 ps2  - pSuministro
        pp1 pp2 - pPersonal
)

(:init
        (= (capacidad r1) 2)
        (= (capacidad r2) 2)
        (= (capacidad r3) 2)

        (aparcado r1 base)
        (aparcado r2 base)
        (aparcado r3 base)

        (accesible base as1) (accesible as1 base)
```

```
                (accesible as1 as2) (accesible as2 as1)
                (accesible as2 as3) (accesible as3 as2)
                (accesible as3 as4) (accesible as4 as3)
                (accesible as4 as5) (accesible as5 as4)
                (accesible as5 en1) (accesible en1 as5)
                (accesible en1 en2) (accesible en2 en1)
                (accesible en1 en3) (accesible en3 en1)
                (accesible en2 en3) (accesible en3 en2)
                (accesible base al1) (accesible al1 base)
                (accesible al1 al2) (accesible al2 al1)
                (accesible al2 al3) (accesible al3 al2)
                (accesible al3 al4) (accesible al4 al3)
                (accesible al4 al5) (accesible al5 al4)
                (accesible al5 en4) (accesible en4 al5)
                (accesible en4 en5) (accesible en5 en4)
                (accesible en4 en6) (accesible en6 en4)
                (accesible en5 en6) (accesible en6 en5)

                (esta pers1 en3)
                (esta pers2 en2)
                (esta s1 al5)
                (esta s2 al1)

                (objetivo ps1 en1)
                (objetivo ps2 en3)
                (objetivo pp1 en4)
                (objetivo pp2 en5)
)

(:goal (forall (?c - Carga) (entregada ?c)))
)
```

## Resultados

Output extensión 1:

```
step    0: MOVER R2 BASE AL1
        1: RECOGERS R2 S2 AL1
        2: MOVER R2 AL1 BASE
        3: MOVER R1 BASE AL1
        4: MOVER R3 BASE AS1
        5: MOVER R2 BASE AS1
        6: MOVER R2 AS1 AS2
        7: MOVER R2 AS2 AS3
        8: MOVER R2 AS3 AS4
```

9: MOVER R2 AS4 AS5
10: MOVER R2 AS5 EN1
11: MOVER R2 EN1 EN3
12: MOVER R3 AS1 BASE
13: ENTREGARS R2 S2 EN3 PS2
14: MOVER R3 BASE AL1
15: MOVER R3 AL1 AL2
16: MOVER R1 AL1 AL2
17: MOVER R1 AL2 AL3
18: MOVER R3 AL2 AL3
19: MOVER R1 AL3 AL4
20: MOVER R3 AL3 AL4
21: MOVER R1 AL4 AL5
22: MOVER R3 AL4 AL5
23: RECOGERS R3 S1 AL5
24: MOVER R3 AL5 AL4
25: MOVER R3 AL4 AL3
26: MOVER R3 AL3 AL2
27: MOVER R3 AL2 AL1
28: MOVER R3 AL1 BASE
29: MOVER R3 BASE AS1
30: MOVER R3 AS1 AS2
31: MOVER R3 AS2 AS3
32: MOVER R3 AS3 AS4
33: MOVER R3 AS4 AS5
34: MOVER R3 AS5 EN1
35: ENTREGARS R3 S1 EN1 PS1
36: MOVER R1 AL5 EN4
37: MOVER R3 EN1 EN3
38: RECOGERP R3 PERS1 EN3
39: MOVER R3 EN3 EN2
40: RECOGERP R3 PERS2 EN2
41: MOVER R3 EN2 EN1
42: MOVER R3 EN1 AS5
43: MOVER R3 AS5 AS4
44: MOVER R3 AS4 AS3
45: MOVER R3 AS3 AS2
46: MOVER R3 AS2 AS1
47: MOVER R3 AS1 BASE
48: MOVER R3 BASE AL1
49: MOVER R3 AL1 AL2
50: MOVER R3 AL2 AL3
51: MOVER R3 AL3 AL4
52: MOVER R3 AL4 AL5

53: MOVER R3 AL5 EN4
54: ENTREGARP R3 PERS1 EN4 PP1
55: MOVER R3 EN4 EN5
56: ENTREGARP R3 PERS2 EN5 PP2

Como es de esperar, obtenemos un gran número de pasos correspondiente al mover a cada a los rovers, primero del centro de la línea hasta uno de los lados, posteriormente coger las cargas y finalmente ir hasta la punta contraría a su posición para dar las cargas. Un dato que nos ha impresionado es el número de estados visitados para poder llegar a la solución. Siendo un total de 18431 estados. Vemos que entonces tener un gran número de Bases también es un factor a tener en cuenta para el tiempo de ejecución del algoritmo.

Posteriormente hicimos las versiones de la extensión 2 y 3. Por desgracia estás no daban una solución, al estar principalmente utilizando métricas de optimización añade de forma considerable complejidad al problema. Esto hace que el tiempo de ejecución se incremente de manera significativa.

# 6. Análisis del Tiempo de Ejecución

## Según el Número de Rovers

En este apartado vamos a analizar cómo evoluciona el tiempo de ejecución del planificador para un grafo fijo, manteniendo el mismo número de peticiones y personal y suministros a transportar, modificando únicamente el número de rovers.

Para simplificar la complejidad realizaremos el análisis con la primera extensión del planificador, generando los problemas mediante nuestro generador de juegos de prueba.

Para este test hemos utilizado la semilla 123, un número de 10 asentamientos y 10 almacenes en total, 5 unidades de suministros y 5 de personal, y 15 peticiones de suministro y 15 peticiones de personal.

Aumentando únicamente el número de rovers podemos observar que el tiempo crece de manera desigual, pero se aproxima a un crecimiento lineal.

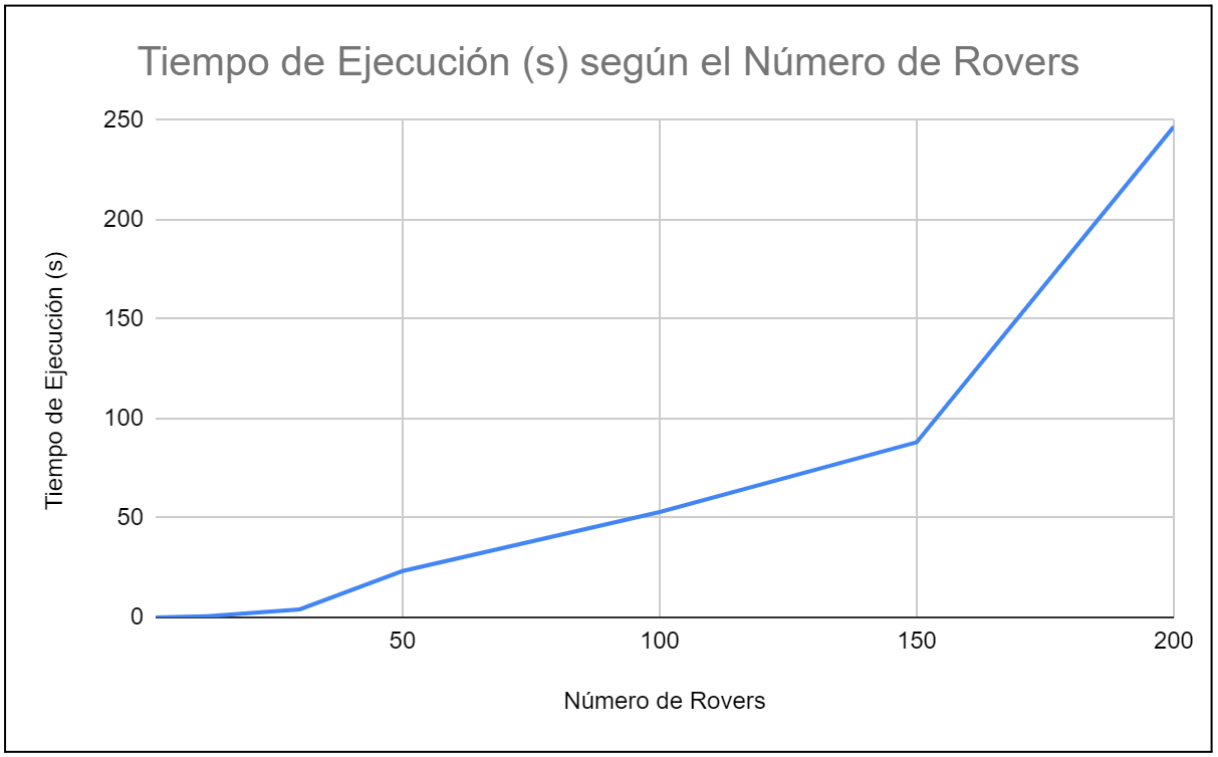


Figura 6.1: Gráfico de la Evolución del Tiempo de Ejecución según el Número de Rovers

El gráfico lo hemos obtenido mediante los problemas generados aleatoriamente que se encuentran en la carpeta *TestRover*, y los datos son los siguientes:

| Número de Rovers | Tiempo de Ejecución (s) |
|---|---|
| 2 | 0,01 |
| 12 | 0,62 |

| | |
|---|---|
| 30 | 4,02 |
| 50 | 23,3 |
| 100 | 52,97 |
| 150 | 88,08 |
| 200 | 246,77 |

## Según el Número de Peticiones

En este apartado vamos a analizar cómo evoluciona el tiempo de ejecución del planificador para un grafo fijo, manteniendo el mismo número de rovers y personal y suministros a transportar, modificando únicamente el número de peticiones.

Para simplificar la complejidad realizaremos el análisis con la primera extensión del planificador y generando los problemas mediante nuestro generador de juegos de prueba, tal como hemos hecho en el estudio anterior.

Para este test hemos utilizado la semilla 123, 3 rovers, un número de 10 asentamientos y 10 almacenes en total y 5 unidades de suministros y 5 de personal. Hemos utilizado los mismos parámetros que en el estudio anterior para así tener una referencia del tiempo de ejecución.

Nuestra hipótesis es que los tiempos que obtendremos serán cercanos a 0,01s y 0,62s, pudiendo superar este último pero no con mucha diferencia. A medida que aumentemos el número de peticiones aumentará el tiempo de ejecución, pero de manera muy lenta.

Nuestro razonamiento es el siguiente: en el estudio anterior obtuvimos un tiempo de ejecución de 0.01s utilizando 2 rovers y los mismos parámetros que ahora (menos el número de peticiones), por eso pensamos que el tiempo de ejecución en este caso se encontrará en torno a ese valor.

En el caso de la extensión 1, las peticiones solo tienen importancia a la hora de hacer las entregas. Cuando un rover tenga una carga y decida dónde tiene que entregarla, el número de peticiones influirá en el número de posibles opciones que tendrá el rover para entregar. Cuantas más peticiones haya más probable será que exista una petición muy cercana donde se encuentre el rover, y por eso el número de movimientos que tendrá que hacer el rover disminuirá.

Es por eso que si tenemos en cuenta el incremento de posibles candidatos a entregar que tendrá el rover (lo cual aumentaría el tiempo de ejecución), pero la gran disminución de movimientos que tendrá que hacer ya que se encontrará con peticiones muy cerca de donde esté él (lo cual reduciría el tiempo de ejecución), creemos que los tiempos que obtendremos serán muy constantes respecto al número de peticiones, muy diferente si lo hiciéramos respecto al número de rovers o al número de cargas de suministro y personal.

El número de peticiones se ha distribuido equitativamente entre peticiones de suministros y peticiones de personal. Por tanto, un juego de prueba con 100 peticiones en total tendrá 50 peticiones de suministros y 50 peticiones de personal.

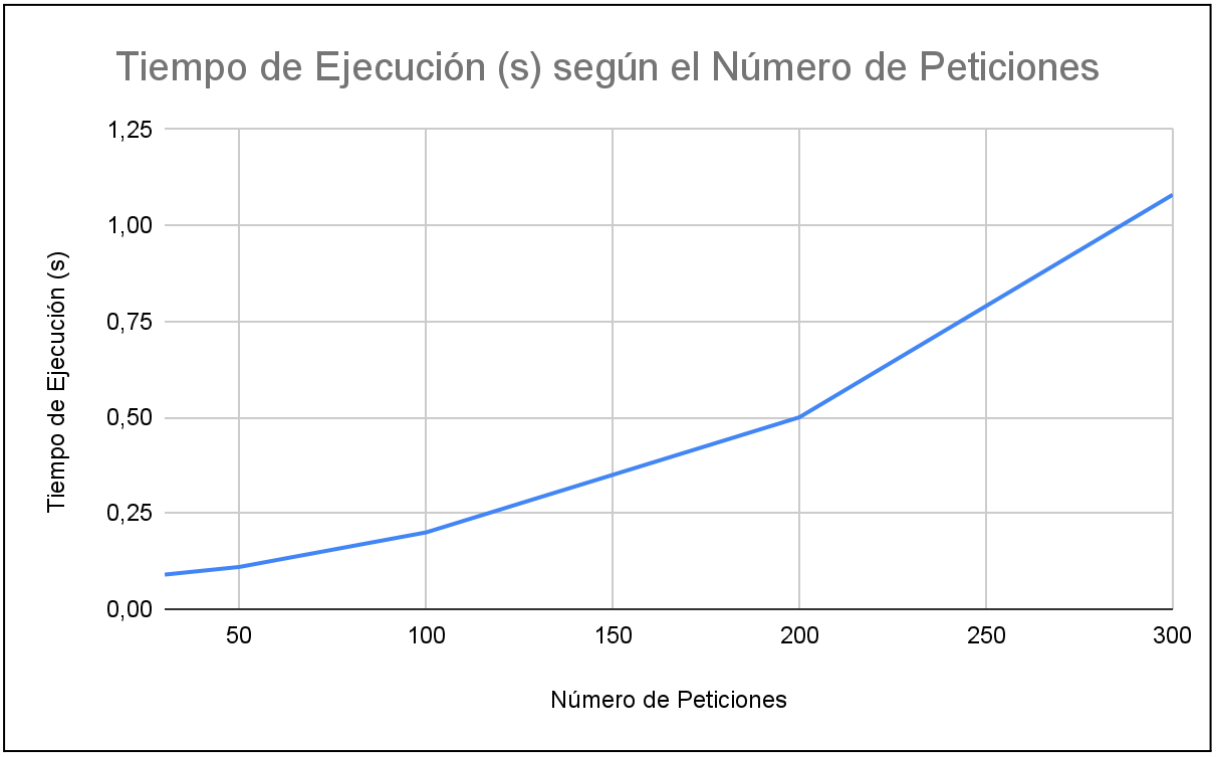


Figura 6.2: Gráfico de la Evolución del Tiempo de Ejecución según el Número de Peticiones

El gráfico lo hemos obtenido mediante los problemas generados aleatoriamente que se encuentran en la carpeta *TestPeticion*, y los datos son los siguientes:

| Número de Peticiones | Tiempo de Ejecución (s) |
|---|---|
| 30 | 0,09 |
| 50 | 0,11 |
| 100 | 0,20 |
| 200 | 0,50 |
| 300 | 1,08 |

Podemos ver como nuestra hipótesis es correcta. El tiempo de ejecución se incrementa de forma muy lenta respecto al número de peticiones, estando sobre un rango de valores entre 0s y 1s con una gran diferencia de 270 peticiones entre el valor mínimo y el máximo.

## Según el Número de Cargas

Para este estudio no vamos a generar juegos de prueba y a hacer una gráfica según los diferentes valores que obtengamos. En vez de eso, vamos a justificar qué es lo que esperaríamos de realizar este estudio de la manera en que lo habríamos hecho como en los estudios anteriores.

Nos hemos de dar cuenta que el objetivo de nuestro planificador es encontrar un estado en el que todas las cargas de suministro y personal estén entregadas. Es por eso que es lógico que si aumentamos el número de cargas nuestro tiempo de ejecución aumentará, ya que tendremos que realizar más movimientos y más decisiones para cada nueva carga que añadamos.

Probablemente el tiempo de ejecución evolucione linealmente respecto al número de cargas o puede que incluso más, dependiendo de la implementación de las acciones y de cómo funciona internamente el planificador.

# 7. Conclusiones

Con este trabajo hemos podido comprobar de forma extensa toda la capacidad que tiene toda el área de planificación. Sobre todo, hemos comprobado el potencial del planificador Metric FF que, a pesar de ser uno de los primeros planificadores que se hicieron en las competiciones de planificación (Metric FF fue presentado en la 3.ª Edición de esta competición en el año 2002), demuestra qué tan potente pueden llegar a ser estas invenciones como son los Planificadores.

Por desgracia, en este proceso de creación, nos hemos encontrado barreras a la hora de hacer nuestras extensiones por las complejidades que pueden presentar, como se ha podido ver tanto en los juegos de prueba como en el análisis de tiempos. La manera en que están construidos los dominios afectan de manera notoria en comprobar si el problema es resoluble o no. Si bien los problemas encontrados en la realización de los dominios en parte son atribuidos por la falta de experiencia por parte de nosotros al programar en este nuevo paradigma y lenguaje nuevo, tampoco hay que menospreciar los límites de Metric FF y el problema dado.

Ha pasado un tiempo considerable desde la invención de este planificador y por lo tanto los planificadores han seguido evolucionando. A su vez, el problema tiene el objetivo de optimizar una serie de parámetros y por consiguiente le estás aportando complejidad al problema.

En resumen, gracias a este trabajo hemos aprendido cómo es trabajar en PDDL y lo que podemos llegar a conseguir. A su vez nos ha hecho entender la importancia de construir bien el dominio del problema, para que así se puedan solucionar los problemas de manera eficiente.