Práctica de Planificación

Planificador para Rovers en Marte

Ariosa Hernández, Roberto
Bosch Llufriu, Júlia
Gómez Sánchez, Joaquín

Inteligencia Artificial
QT - Curso 2018/19

Índice

El Problema

En el enunciado de esta práctica se nos describe un escenario particular: estamos asentados en Marte, en un futuro lejano, y necesitamos cubrir las necesidades de transporte de los colonos.

Para cubrir dichas necesidades necesitamos desarrollar un planificador (en PDDL) para los rovers, que permiten transportar suministros y personas, entre las diferentes bases que pueden ser de tipo asentamiento o almacén.

Para ello deberemos realizar un trabajo de planificación con PDDL que nos ayude a organizar los desplazamientos de Rovers con los debidos suministros que van a cubrir las necesidades anteriormente mencionadas con distintas limitaciones.

Dominio

El dominio del presente problema está compuesto de las siguientes partes. Para cada elemento se indicará en qué extensión aparece.

- **Funciones / Variables**
  - *[EXTENSIÓN 1]* (**supplies** ?r - rover):Cantidad de carga que tiene un rover r, siendo la carga máxima 2 personas (+1 por cada persona) o 1 suministro (+2 por suministro).
  - *[EXTENSIÓN 2]* (**fuel** ?r - rover): Combustible del que dispone un rover r.
  - *[EXTENSIÓN 2]* (**fuel-used**): Combustible total utilizado por todos los rovers.
  - *[EXTENSIÓN 3]* (**violations**): Acumula el número de violaciones que se han realizado en el orden de los servicios según las prioridades hasta el momento.

- **Predicados**
  - (**parked** ?c - rover ?b - base): Indica que el rover *c* está estacionado en la base *b*.
  - (**is-in** ?c - cargo ?b - place): Indica que la carga *c* (persona o suministro) está en el rover o en la base *b*.
  - (**needs** ?c - cargo ?b - base): Indica que la carga *c* (persona o suministro) se necesita en la base *b*.
    - *[EXTENSIÓN 3]* (**needs-p1** ?c - cargo ?b - base): Indica que la carga *c* (persona o suministro) se necesita en la base *b* con prioridad 1.
    - *[EXTENSIÓN 3]* (**needs-p2** ?c - cargo ?b - base): Indica que la carga *c* (persona o suministro) se necesita en la base *b* con prioridad 2.
    - *[EXTENSIÓN 3]* (**needs-p3** ?c - cargo ?b - base): Indica que la carga *c* (persona o suministro) se necesita en la base *b* con prioridad 3.
  - (**served** ?c - cargo): Indica que la carga *c* (persona o suministro) está servida.

- **Acciones**
  - (:action **pick-person** :parameters (?p - person ?b - settlement ?r - rover)): El rover *r* recoge un persona *c* de una base *b.*

- `(:action` **`pick-supply`** `:parameters (?s - supply ?b - warehouse ?r - rover))`: El rover *r* recoge un suministro *s* de una base *b*.
- `(:action` **`leave-person`** `:parameters (?p - person ?r - rover ?b - settlement))`: La persona *p* abandona la base *b* en el rover *r*.
- `(:action` **`leave-supply`** `:parameters (?s - supply ?b - base ?r - rover))`: El suministro *s* abandona la base *b* en el rover *r*.
- `(:action` **`move`** `:parameters (?r - rover ?b1 - base ?b2 - base))`: El rover *r* se mueve desde la base *b1* hasta la base *b2*.
- `(:action` **`perform`** `:parameters (?c - cargo ?b - base))`: La carga *c* está en la base *b*.
  - *`[EXTENSIÓN 3]`* `(:action` **`perform-p1`** `:parameters (?c - cargo ?b - base))`: La carga *c* con prioridad 1 está en la base *b*.
  - *`[EXTENSIÓN 3]`* `(:action` **`perform-p2`** `:parameters (?c - cargo ?b - base))`: La carga *c* con prioridad 2 está en la base *b*.
  - *`[EXTENSIÓN 3]`* `(:action` **`perform-p3`** `:parameters (?c - cargo ?b - base))`: La carga *c* con prioridad 3 está en la base *b*.

Modelización

Para resolver los distintos problemas, proponemos las siguientes modelizaciones. Como se trata de una resolución incremental del problema, para cada extensión sólo describiremos qué cambia respecto a la anterior y qué elementos del problema utilizamos para resolver dicha extensión.

Nivel básico

En este primer nivel básico se deben cubrir las solicitudes de transporte de suministros y personal, con lo que hay disponible, sin ningún tipo de limitación.

- **Objetos**
  - **`Rover:`** es el medio de transporte de los suministros y del personal especializado (cargas). Pueden moverse entre bases (de dos tipos: almacenes y asentamientos). En este nivel del problema tienen capacidad ilimitada y pueden transportar a la vez tantos suministros y personal especializado como se desee.
  - **`Supply:`** representan los suministros que llegan de la Tierra hasta los almacenes marcianos. Cada almacén es responsable de hacer tantas peticiones de suministros como necesite.
  - **`Person:`** representan las personas especializadas que los colonos necesitan. Junto con los suministros, son uno de los tipos de carga que transportan los rovers. Estas personas viajan entre asentamientos según las peticiones de estos.
  - **`Warehouse:`** almacenes donde se guardan los suministros desde que son enviados de la Tierra hasta que algún almacén hace una petición y los rovers se disponen a transportarlos.
  - **`Settlement:`** asentamientos en los cuales viven los colonos. En ellos también se aloja el personal especializado del que precisan los colonos en sus asentamientos y, si estos no se corresponden, son los rovers los encargados de su transporte hasta el asentamiento que los haya solicitado.
    Cada asentamiento es el responsable de hacer las peticiones que necesite al organismo centralizado que se ocupa del suministro de las cargas de los rovers.

- **Estado inicial**
  - El estado inicial tiene todos los rovers definidos en *objects* aparcados en alguna base (ya sea asentamiento o almacén).
  - Por ejemplo, sea r1 un rover y b1 una base:
    `(parked r1 b1)`

- Las cargas que van a transportar los rovers, ya sean suministros o personas, también se encuentran inicialmente en alguna base. Las personas van a estar en asentamientos mientras que los suministros se encontrarán en almacenes.
- Por ejemplo, sean s1 un suministro concreto, p1 una persona, b1 un almacén y b4 un asentamiento:

```
(is-in s1 b1)
(is-in p1 b4)
```

- Finalmente, en el estado inicial tienen que encontrarse definidas las necesidades de cada base, de manera que los almacenes hacen peticiones de suministros y los asentamientos de personal especializado.
- Por ejemplo, sean p1 una persona, s1 un suministro, b2 y b3 almacenes y b5 un asentamiento:

```
(needs p1 b5)
(needs s1 b2)
(needs s1 b3)
```

- **Estado final**
  -
```
(forall (?c - cargo)
     (served ?c)
```
  - Conforme al objetivo descrito en el enunciado, podemos representar el estado final con las dos líneas de código anteriores, ya que sólo tenemos que cumplir que **todos** los cargamentos (tanto suministros como personal) de los que disponemos sean utilizados para satisfacer alguna de las necesidades descritas en el estado inicial.

Primera Extensión

En esta primera extensión del problema inicial, el único cambio que tenemos que tener en cuenta es que, ahora, los rovers sólo podrán transportar dos personas o un suministro a la vez (nunca mezclando personas y suministros). Dicho esto, los elementos que se añaden y/o cambian en nuestra representación del problema son los siguientes:

- **Estado inicial**
  - El estado inicial tiene todos los rovers definidos en *objects* aparcados en alguna base (ya sea asentamiento o almacén), al igual que en la implementación anterior. Ahora, además, tenemos que inicializar la variable numérica `supplies` del rover correspondiente a 0, ya que al inicio de la ejecución todos los rovers estarán vacíos de carga. No nos hace falta definir nada más, ya que la condición de no mezclar tipos de

carga queda satisfecha por definición de la variable usada (y explicada en el apartado anterior del informe)

- Por ejemplo, sea r1 un rover y b1 una base:

```
(parked r1 b1)
(= (supplies r1) 0)
```

Segunda Extensión - Versión sin optimizar

En esta segunda extensión del problema inicial, el problema tiene las mismas características que en la primera extensión y, además, los rovers tienen un límite de combustible (lo que se traduce en un número limitado de movimientos, que corresponde al número que representa el combustible disponible).

- **Estado inicial**
  - Para esta versión de la segunda extensión, necesitamos inicializar una variable numérica adicional:
    - `fuel`
      Se inicializa una vez por cada rover que tenemos disponible en nuestro estado inicial. El valor numérico que se le asigne representará la cantidad de movimientos que tiene permitido hacer el vehículo correspondiente.
  - Por ejemplo, sea r1 un rover y b1 una base, una posible inicialización de r1 (que podría realizar 5 movimientos) sería:

```
(parked r1 b1)
(= (supplies r1) 0)
(= (fuel r1) 5)
```

Segunda Extensión - Versión optimizada

En esta versión de la segunda extensión, nos piden minimizar el combustible total gastado.

- **Estado inicial**
  - Para esta versión necesitamos inicializar otra variable numérica adicional:
    - `fuel-used`
      Esta se inicializa a cero ya que, como hemos explicado anteriormente, representa el combustible total consumido en los movimientos de los rovers. Solo tenemos que asignarle un valor una vez ya que es común para todo el escenario.

```
(= (fuel-used) 0)
```

Esta variable vamos a usarla en la cláusula `:metric`, que nos permite dar a nuestro programa un criterio de optimización. En este caso en concreto, como nos interesa minimizar:

- **Metric**
  - `(:metric minimize (fuel-used))`

Tercera Extensión - Versión con una optimización

En esta tercera y última extensión del problema inicial, el problema tiene las mismas características que en la segunda extensión y, además, le añadimos una prioridad a tener en cuenta a las peticiones (1, 2 o 3, de mínima a máxima, respectivamente). Así pues, los elementos que cambian y/o se añaden son los siguientes:

- **Estado inicial**
  - Como para esta versión no nos interesa minimizar el combustible total gastado, la presencia de la variable `fuel-used` es innecesaria y podríamos prescindir de su inicialización
  - Para esta versión necesitamos inicializar otra variable numérica adicional:
    - `violations`
      Esta se inicializa a cero ya que, como hemos explicado anteriormente, representa el número de violaciones de prioridad ocurridas.
      `(= (violations) 0)`

  - En la representación de las prioridades hemos optado por modificar el predicado `needs`, que pasa a ser `needs-px`, siendo x el número de prioridad que le damos a la petición. Así pues, con el mismo ejemplo que el que habíamos puesto en la versión básica del problema, una posible representación de las necesidades quedaría tal que así:

```
(needs-p1 p1 b5)
(needs-p3 s1 b2)
(needs-p3 s1 b3)
```

- **Metric**
  - Como ahora el objetivo es maximizar la prioridad de las peticiones servidas (sin minimizar el combustible gastado), lo que buscaremos en la definición de :`metric` será minimizar las violaciones.
    `(:metric minimize (violations))`

Tercera Extensión - Versión con dos optimizaciones

En esta versión de la tercera extensión, nos interesa optimizar tanto la satisfacción de peticiones según prioridades como el combustible usado, pudiendo personalizar el peso que queremos darle a cada una de las optimizaciones.

- **Estado inicial**
  - Como para esta versión sí nos interesa minimizar el combustible total gastado, la presencia de la variable `fuel-used` es necesaria y tenemos que inicializarla a 0 como en la anterior extensión.

- **Metric**
  - Como ahora el objetivo es optimizar ambos parámetros, lo que buscaremos en la definición de `:metric` será minimizar las violaciones y el combustible utilizado según una combinación lineal de las variables. Con pesos 1 y 2 en las violaciones y el combustible usado, respectivamente:

    ```
    (:metric minimize (+ (* (violations) 1) (* (fuel-used) 2)))
    ```

Desarrollo de los modelos

Para el desarrollo de los distintos modelos hemos optado por un diseño incremental basado en prototipos tal y como se plantea en el enunciado.

Hemos empezado por el nivel básico y hemos ido realizando las diferentes extensiones sobre el modelo anterior una vez este estaba acabado y funcionando. En cada prototipo hemos ido añadiendo funcionalidades hasta obtener las de la última extensión.

Para probar los diferentes modelos se han hecho problemas adecuados para cada dominio de forma manual, y al final se ha realizado un generador de problemas aleatorios donde se puede escoger el dominio para el que se va a generar.

Juegos de Prueba

Para comprobar el correcto funcionamiento del modelo inicial y sus diferentes extensiones hemos realizado los siguientes juegos de prueba que se han generado aleatoriamente mediante un script que hemos desarrollado para la generación de problemas aleatorios.

Primer Juego de Prueba - Nivel Básico

Juego de prueba para el nivel básico del problema dado. Este ha sido generado aleatoriamente mediante el script generador y se encuentra en el fichero *problem-basic-random.pddl*.

- **Entrada**

```
(define (problem rovers)

     ; Domain
     (:domain mars)

     ; Objects
     (:objects
          r0 - rover
          s0 - supply
          p0 - person
          b0 b1 - settlement
          b2 b3 b4 - warehouse
     )

     ; Init
     (:init
          ; Rovers
          (parked r0 b3)

          ; Supplies
       (is-in s0 b2)

          ; Staff
       (is-in p0 b1)

          (needs s0 b4)
          (needs p0 b0)
          (needs s0 b3)
     )

     ; Goal
     (:goal
          (forall (?c - cargo)
               (served ?c)
```

```
            )
        )

    )
```

- **Salida**

```
step 0: MOVE R0 B3 B1
     1: PICK-PERSON P0 B1 R0
     2: MOVE R0 B1 B2
     3: PICK-SUPPLY S0 B2 R0
     4: MOVE R0 B2 B0
     5: LEAVE-PERSON P0 R0 B0
     6: MOVE R0 B0 B4
     7: LEAVE-SUPPLY S0 R0 B4
     8: PERFORM P0 B0
     9: PERFORM S0 B4
```

- **Justificación**

La intención del juego de prueba es desarrollar un plan para los rovers tal que al final el suministro 0 esté en el almacén 4, el suministro 0 en el almacén 3 y la persona 0 en la base 0.

Segundo Juego de Prueba - Extensión 1

Juego de prueba para el nivel básico con la primera extensión. Este ha sido generado aleatoriamente mediante el script generador y se encuentra en el fichero *problem-ext1-random.pddl*.

- **Entrada**

```
(define (problem rovers)

    ; Domain
    (:domain mars)

    ; Objects
    (:objects
        r0 r1 r2 r3 - rover
        s0 s1 - supply
        p0 p1 p2 - person
        b0 b1 b2 - settlement
        b3 b4 b5 b6 - warehouse
    )

    ; Init
    (:init
        ; Rovers
        (parked r0 b4)
```

```
            (= (supplies r0) 0)

            (parked r1 b6)
            (= (supplies r1) 0)

            (parked r2 b3)
            (= (supplies r2) 0)

            (parked r3 b4)
            (= (supplies r3) 0)

            ; Supplies
          (is-in s0 b3)
          (is-in s1 b6)

            ; Staff
          (is-in p0 b1)
          (is-in p1 b0)
          (is-in p2 b1)

            (needs s1 b4)
            (needs p1 b0)
            (needs p0 b0)
            (needs p1 b1)
            (needs p2 b0)
            (needs s0 b5)
        )

        ; Goal
        (:goal
            (forall (?c - cargo)
                (served ?c)
            )
        )

    )
```

- **Salida**

```
step 0: MOVE R3 B4 B1
     1: PICK-PERSON P2 B1 R3
     2: PICK-PERSON P0 B1 R3
     3: MOVE R3 B1 B0
     4: PICK-SUPPLY S1 B6 R1
     5: MOVE R1 B6 B4
     6: PICK-SUPPLY S0 B3 R2
     7: MOVE R2 B3 B5
     8: LEAVE-PERSON P2 R3 B0
     9: LEAVE-PERSON P0 R3 B0
```

```
10: LEAVE-SUPPLY S1 R1 B4
11: LEAVE-SUPPLY S0 R2 B5
12: PERFORM P2 B0
13: PERFORM P1 B0
14: PERFORM P0 B0
15: PERFORM S1 B4
16: PERFORM S0 B5
```

- **Justificación**

El objetivo de este juego de pruebas es llevar una serie de personas y suministros a las bases y almacenes correspondientes teniendo en cuenta que se pueden transportar un máximo de 2 personas a la vez o una carga, y que no se pueden mezclar.

Tercer Juego de Prueba - Extensión 2 sin Optimización

Juego de prueba para la primera extensión más la segunda extensión sin tener en cuenta ninguna optimización. Este ha sido generado aleatoriamente mediante el script generador y se encuentra en el fichero *problem-ext2-random.pddl*.

- **Entrada**

```
(define (problem rovers)

    ; Domain
    (:domain mars)

    ; Objects
    (:objects
      r0 - rover
      s0 - supply
      p0 p1 p2 - person
      b0 b1 b2 - settlement
      b3 b4 b5 - warehouse
    )

    ; Init
    (:init
      ; Rovers
      (= (fuel-used) 0)

      (parked r0 b0)
      (= (supplies r0) 0)
      (= (fuel r0) 5)

      ; Supplies
      (is-in s0 b4)
```

```
        ; Staff
        (is-in p0 b2)
        (is-in p1 b1)
        (is-in p2 b2)

        (needs s0 b4)
        (needs p2 b1)
        (needs p2 b0)
        (needs p0 b1)
        (needs p1 b0)
        (needs p2 b2)
    )

    ; Goal
    (:goal
        (forall (?c - cargo)
            (served ?c)
        )
    )

)
```

- **Salida**

```
step 0: PERFORM S0 B4
     1: PERFORM P2 B2
     2: MOVE R0 B0 B1
     3: PICK-PERSON P1 B1 R0
     4: MOVE R0 B1 B0
     5: LEAVE-PERSON P1 R0 B0
     6: MOVE R0 B0 B1
     7: PERFORM P1 B0
     8: MOVE R0 B1 B2
     9: PICK-PERSON P0 B2 R0
     10: MOVE R0 B2 B1
     11: LEAVE-PERSON P0 R0 B1
     12: PERFORM P0 B1
```

- **Justificación**

La intención de este juego de pruebas es comprobar el correcto funcionamiento de la primera extensión más la segunda extensión, pero sin tener en cuenta la minimización del combustible gastado.

Cuarto Juego de Prueba - Extensión 2 con Optimización

Juego de prueba para la primera extensión más la segunda extensión, teniendo en cuenta la optimización del `fuel-used`. Este ha sido generado aleatoriamente mediante el script generador y se encuentra en el fichero *problem-ext2-optimize-random.pddl*.

- **Entrada**

```
(define (problem rovers)

    ; Domain
    (:domain mars)

    ; Objects
    (:objects
      r0 r1 - rover
      s0 - supply
      p0 p1 p2 - person
      b0 b1 b2 - settlement
      b3 b4 b5 - warehouse
    )

    ; Init
    (:init
      ; Rovers
      (= (fuel-used) 0)

      (parked r0 b5)
      (= (supplies r0) 0)
      (= (fuel r0) 5)

      (parked r1 b0)
      (= (supplies r1) 0)
      (= (fuel r1) 5)

      ; Supplies
      (is-in s0 b5)

      ; Staff
      (is-in p0 b2)
      (is-in p1 b0)
      (is-in p2 b2)
```

```
        (needs p1 b2)
        (needs p2 b0)
        (needs s0 b3)
        (needs p0 b2)
        (needs p0 b1)
    )

    ; Goal
    (:goal
        (forall (?c - cargo)
            (served ?c)
        )
    )

    ; Metric
    (:metric minimize (fuel-used))
)
```

- **Salida**

```
step 0: PICK-SUPPLY S0 B5 R0
     1: MOVE R0 B5 B3
     2: PICK-PERSON P1 B0 R1
     3: PERFORM P0 B2
     4: MOVE R1 B0 B2
     5: PICK-PERSON P2 B2 R1
     6: LEAVE-PERSON P1 R1 B2
     7: MOVE R1 B2 B0
     8: LEAVE-PERSON P2 R1 B0
     9: PERFORM P2 B0
     10: PERFORM P1 B2
     11: LEAVE-SUPPLY S0 R0 B3
     12: PERFORM S0 B3
```

- **Justificación**

El objetivo de este juego de pruebas es comprobar que se realiza una optimización del combustible utilizado por los rovers.

Quinto Juego de Prueba - Extensión 3 Versión 1

Juego de prueba para la segunda extensión más la tercera extensión, teniendo en cuenta la optimización del número de violaciones en las prioridades de las solicitudes. Este ha sido generado aleatoriamente mediante el script generador y se encuentra en el fichero *problem-ext3-vers1-random.pddl*.

- **Entrada**

```
(define (problem rovers)

    ; Domain
    (:domain mars)

    ; Objects
    (:objects
        r0 r1 r2 - rover
        s0 s1 - supply
        p0 p1 - person
        b0 b1 - settlement
        b2 b3 - warehouse
    )

    ; Init
    (:init
        ; Rovers
        (= (fuel-used) 0)
        (= (violations) 0)

        (parked r0 b1)
        (= (supplies r0) 0)
        (= (fuel r0) 4)

        (parked r1 b1)
        (= (supplies r1) 0)
        (= (fuel r1) 2)

        (parked r2 b0)
        (= (supplies r2) 0)
        (= (fuel r2) 2)

        ; Supplies
        (is-in s0 b3)
        (is-in s1 b3)
```

```
            ; Staff
            (is-in p0 b0)
            (is-in p1 b0)

            (needs-p1 p1 b1)
            (needs-p1 p0 b1)
            (needs-p1 s1 b2)
            (needs-p2 s0 b2)
        )

        ; Goal
        (:goal
            (forall (?c - cargo)
                (served ?c)
            )
        )

        ; Metric
        (:metric minimize (violations))
    )
```

- **Salida**

```
step 0: PICK-PERSON P0 B0 R2
     1: PICK-PERSON P1 B0 R2
     2: MOVE R0 B1 B0
     3: MOVE R0 B0 B1
     4: MOVE R0 B1 B0
     5: MOVE R1 B1 B3
     6: PICK-SUPPLY S0 B3 R1
     7: MOVE R1 B3 B2
     8: LEAVE-SUPPLY S0 R1 B2
     9: PERFORM-P2 S0 B2
    10: LEAVE-PERSON P1 R2 B0
    11: PICK-PERSON P1 B0 R0
    12: LEAVE-PERSON P0 R2 B0
    13: PICK-PERSON P0 B0 R0
    14: MOVE R0 B0 B1
    15: MOVE R2 B0 B3
    16: PICK-SUPPLY S1 B3 R2
    17: MOVE R2 B3 B2
    18: LEAVE-SUPPLY S1 R2 B2
```

```
 19: PICK-SUPPLY S1 B2 R1
 20: LEAVE-PERSON P0 R0 B1
 21: PERFORM-P1 P0 B1
 22: LEAVE-SUPPLY S1 R1 B2
 23: PICK-SUPPLY S1 B2 R2
 24: LEAVE-PERSON P1 R0 B1
 25: PERFORM-P1 P1 B1
 26: LEAVE-SUPPLY S1 R2 B2
 27: PERFORM-P1 S1 B2
```

- **Justificación**

En este juego de pruebas pretende testear que dado que las solicitudes tienen unas prioridades, se van a intentar violar el orden de estas el mínimo posible.

Sexto Juego de Prueba - Extensión 3 Versión 2

Juego de prueba para la segunda extensión más la tercera extensión, teniendo en cuenta la optimización del número de violaciones en las prioridades de las solicitudes, y la cantidad de combustible consumido. Este ha sido generado aleatoriamente mediante el script generador y se encuentra en el fichero *problem-ext3-vers2-random.pddl*.

- **Entrada**

```
(define (problem rovers)

    ; Domain
    (:domain mars)

    ; Objects
    (:objects
      r0 r1 r2 r3 r4 - rover
      s0 s1 - supply
      p0 p1 - person
      b0 b1 b2 b3 - settlement
      b4 b5 - warehouse
    )

    ; Init
    (:init
      ; Rovers
      (= (fuel-used) 0)
      (= (violations) 0)
```

```
  (parked r0 b0)
  (= (supplies r0) 0)
  (= (fuel r0) 21)

  (parked r1 b5)
  (= (supplies r1) 0)
  (= (fuel r1) 29)

  (parked r2 b4)
  (= (supplies r2) 0)
  (= (fuel r2) 21)

  (parked r3 b3)
  (= (supplies r3) 0)
  (= (fuel r3) 26)

  (parked r4 b1)
  (= (supplies r4) 0)
  (= (fuel r4) 22)

  ; Supplies
  (is-in s0 b4)
  (is-in s1 b5)

  ; Staff
  (is-in p0 b2)
  (is-in p1 b2)

  (needs-p3 p1 b2)
  (needs-p3 p0 b1)
  (needs-p1 s0 b5)
  (needs-p1 s1 b4)
)

; Goal
(:goal
  (forall (?c - cargo)
       (served ?c)
  )
)

; Metric
```

```
    (:metric minimize (+ (* (violations) 5) (*
(fuel-used) 3)))
)
```

- **Salida**

```
step 0: MOVE R1 B5 B2
     1: PICK-PERSON P0 B2 R1
     2: MOVE R1 B2 B1
     3: PICK-SUPPLY S0 B4 R2
     4: MOVE R0 B0 B4
     5: MOVE R2 B4 B5
     6: LEAVE-SUPPLY S0 R2 B5
     7: PICK-SUPPLY S1 B5 R2
     8: MOVE R2 B5 B4
     9: LEAVE-SUPPLY S1 R2 B4
     10: PICK-SUPPLY S1 B4 R0
     11: LEAVE-PERSON P0 R1 B1
     12: PICK-PERSON P0 B1 R4
     13: LEAVE-SUPPLY S1 R0 B4
     14: PICK-SUPPLY S1 B4 R2
     15: PERFORM-P3 P1 B2
     16: LEAVE-PERSON P0 R4 B1
     17: PERFORM-P3 P0 B1
     18: LEAVE-SUPPLY S1 R2 B4
     19: PERFORM-P1 S1 B4
     20: PERFORM-P1 S0 B5
```

- **Justificación**

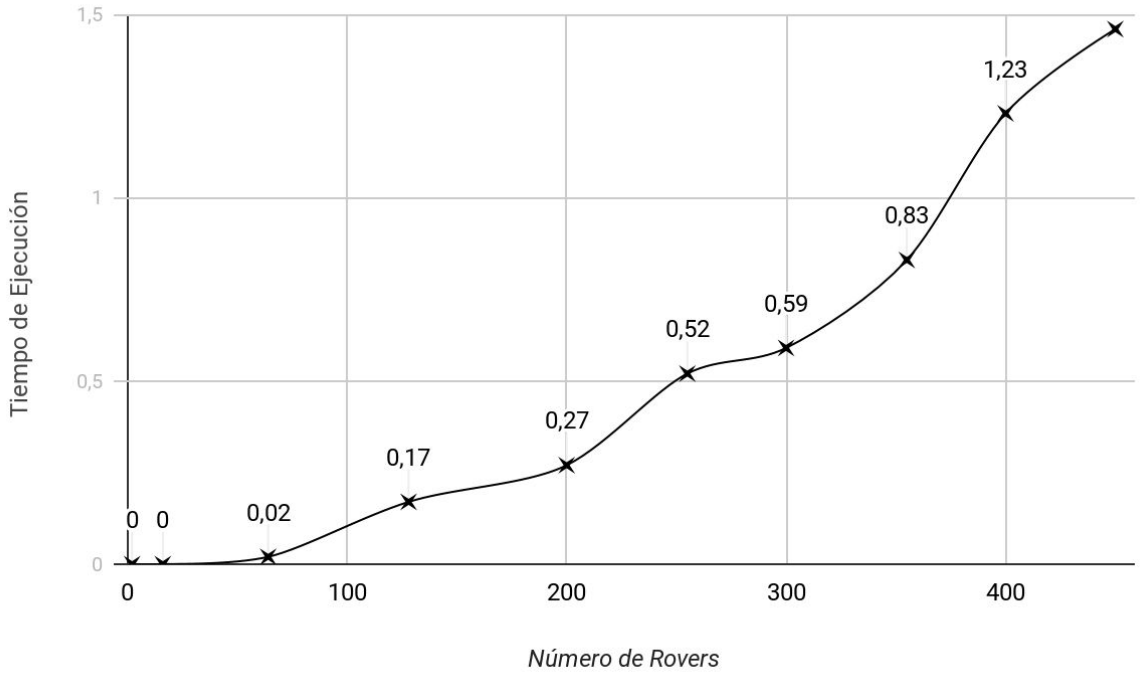
El objetivo de este juego de pruebas es comprobar que dado que las solicitudes tienen unas prioridades, se van a intentar violar el orden de estas el mínimo posible, y que a su vez, el consumo de combustible de los rovers sea el mínimo.

Análisis del Tiempo de Ejecución

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

Para simplificar la complejidad, realizaremos el análisis con la primera extensión, generando los problemas mediante el script generador que hemos realizado.

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



Hemos observado mediante los datos que el planificador nos devuelve que la mayor parte del tiempo la emplea en “computing LNF”, es decir, computando el algoritmo Left Neighbor First, y una parte ínfima en construir el grafo de resolución interno y en la búsqueda de la solución.

A partir de esto último podemos deducir, que si aumentamos el número de peticiones el tiempo de resolución aumentaría, y que si en cambio aumentamos el número de personal/suministros a transportar también el “computing LNF” y posiblemente también la búsqueda de la solución, si incluimos más solicitudes.

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

Número de Rovers	**Tiempo de Ejecución**
2	0,00
16	0,00
64	0,02
128	0,17
200	0,27
255	0,52
300	0,59
355	0,83
400	1,23
450	1,46

Conclusiones

Podemos extraer varias conclusiones de la presente práctica. En primer lugar, que hemos observado una relación intrínseca en los SBC y en los Planificadores, puesto que ambos trabajan bajo el paraguas de la lógica, sólo que estos últimos en cierta manera son más autónomos utilizando heurísticos para determinar qué acciones emplear para obtener de mejor forma una solución correcta.

También hemos observado claras diferencias entre la planificación simple y la planificación métrica, siendo la segunda mucho más costosa de evaluar y “menos natural” para el tipo de objetivos a los que están enfocados planificadores como FF.

En cuanto al contenido en sí de la práctica, ha sido interesante poder evaluar diferentes puntos de vista o diferentes complejidades para el mismo tipo de problema.