Asignatura	Datos del alumno	Fecha
Razonamiento y planificación automática	Apellidos: Domínguez Espinoza Nombre: Edgar Uriel	5 de mayo de 2022

Planificación en STRIPS/PDDL

1. Introducción

En trabajos anteriores se ha explorado el estado del arte en el área de planificadores automáticos para la Inteligencia Artificial (IA). En este texto se retoma el uso de los planificadores para resolver problemas expresados en STRIPS/PDDL. PDDL es una familia de lenguajes que permiten definir un problema de planificación. Esta familia ha evolucionado junto con el desarrollo de la planificación, por lo que hay numerosas versiones disponibles con diferentes niveles de expresividad. (Green, 2021b)

Es importante señalar que los lenguajes PDDL comparten elementos entre sí, hay otros que son ampliamente utilizados pero no universales y algunos más son exclusivos de alguna versión de PDDL. En este texto se exploran *grosso modo* los fundamentos de PDDL a través de tres problemas de planificación automática.

2. Marco teórico

En IA la planificación explora por medio de métodos lógicos y de programación un proceso que pretende resolverse gracias a técnicas autónomas. En un problema de planificación existe un estado inicial que debe transformarse en un estado meta mediante la realización de un conjunto de acciones. (Ghallab et al., 2004; Green, 2021a)

En PDDL, se deben elaborar dos programas: dominio y problema. El dominio describe las determinaciones que forman el contexto, mientras que el problema detalla el estado inicial, final así como los participantes particulares de la situación.

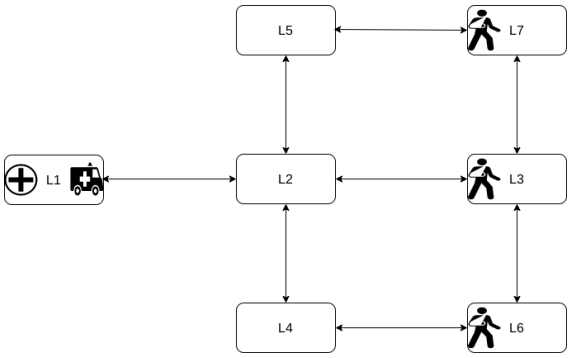
Ambas descripciones, dominio y problema, serán los datos de entrada para que un planificador automático busque cual es la ruta de acción a seguir para pasar del estado inicial al estado final. Generalmente, los planificadores reducen el problema a un contexto de búsqueda dentro de un grafo, de esta forma se expresa la solución como una ruta a seguir.

3. Marco referencial

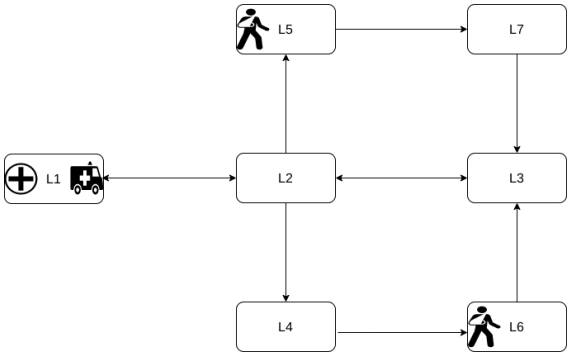
En este texto se utilizarán los siguientes planificadores: FF, Metis (Sievers and Katz, 2018), LPG-td y el Solver Planning Domains (SPD). También se consideraron otros como IBaCoP (Cenamor et al., 2018), un conjunto de planificadores, pero fueron descartados por su bajo desempeño en tiempo de ejecución. El resultado de otro planificador, Scorpion (Seipp, 2018), se menciona durante el texto.

En el dominio del problema a resolver se presenta un mapa con una serie de localizaciones, algunas de ellas son hospitales, otras tienen pacientes. Se asume que un robot conduce una ambulancia, originalmente localizada en un hospital, y debe llegar a la ubicación de los pacientes, subirlos a la ambulancia y llevarlos al hospital. La ambulancia solo puede atender un paciente a la vez.

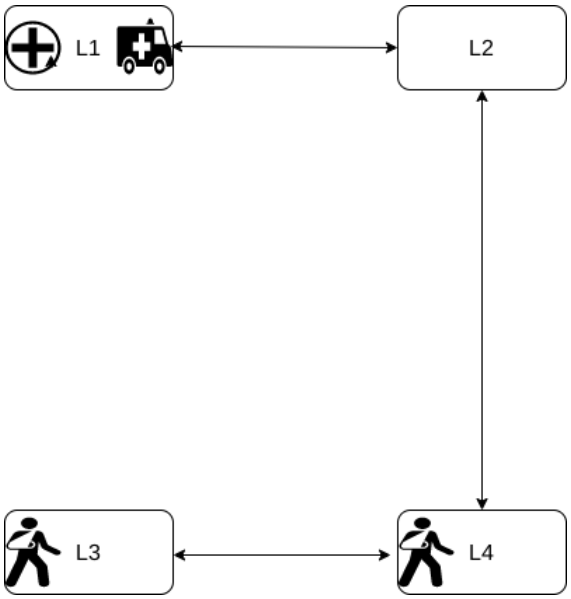
Los problemas variarán según el número de ubicaciones, pacientes y caminos disponibles en el mapa. La figura 1 ilustra los tres problemas considerados, uno de ellos es el original u obligatorio para este texto (1c), los otros dos son variaciones del primero (1a y 1b).



(a) Variante 1



(b) Variante 2



(c) Problema original

Figura 1: Problemas resueltos

4. Dominio y problemas

Tanto STRIPS como PDDL son lenguajes declarativos (Domínguez Espinoza, 2017). En el dominio (ver listing 1) se distinguirá primero el nombre del mismo: *emergency* en este caso. Posteriormente, habrá listas de propiedades: el primer elemento de la lista es un parámetro (*:nombre del parámetro*) que indica el tipo de predicados que tiene la lista, Estos parámetros pueden repetirse según el nombre del parámetro, si la lista corresponde a acciones (*:action*), habrá tres parámetros adicionales: *:parameters*, *:precondition* y *:effect*. La primera de estas listas serán las variables que intervienen en la acción; la segunda serán los predicados que deben cumplirse para ejecutar la acción; la última los cambios consecuentes a realizar la acción.

Listing 1: domain.pddl

```
(define (domain emergency)
  (:predicates
   (ambulance ?a) ; Ambulancia
   (at ?a ?l) ; Si la ambulancia/
      paciente está en un lugar
   (carry ?p ?a) ; Si el paciente
      está en la ambulancia
   (current ?l) ; La ubicación
      current de la ambulancua
   (free ?a) ; Si está vacía la
      ambulancia
   (location ?l) ; Los lugares
   (path ?f ?t) ; Camino entre dos
      lugares
   (patient ?p)) ; Paciente
  (:action move
          :parameters (?from ?to)
          :precondition (and
                        (current ?from
                           )
                        (location ?
                           from)
                        (location ?to)
                        (path ?from ?
```

```
                                to))
18          :effect (and
19                  (current ?to)
20                  (not (current ?from
                        ))))
21  (:action getinto
22          :parameters (?p ?l ?a)
23          :precondition (and
24                      (ambulance ?a)
25                      (at ?p ?l)
26                      (current ?l)
27                      (free ?a)
28                      (location ?l)
29                      (patient ?p))
30          :effect (and
31                  (carry ?p ?a)
32                  (not (at ?p ?l))
33                  (not (free ?a))))
34  (:action getout
35          :parameters (?p ?l ?a)
36          :precondition (and
37                      (ambulance ?a)
38                      (carry ?p ?a)
39                      (current ?l)
40                      (location ?l)
41                      (patient ?p))
42          :effect (and
43                  (at ?p ?l)
44                  (free ?a)
45                  (not (carry ?p ?a))
                    )))
```

En el problema se encuentra el nombre del mismo. Después se leen cuatro listas de propiedades: *:domain*, *:objects*, *:init* y *:goal*. La lista *:objects* consiste en un conjunto de átomos que nombran los participantes del problema, mientras *:init* y *:goal* listan predicados que describen la situación inicial y final del problema. El listing 2 corresponde a la escritura en STRIPS del problema original.

Listing 2: problem01.pddl

```
1   (define (problem original)
2     (:domain emergency)
3     (:objects ambulance hospital
         L2 L3 L4 patient1 patient2)
4     (:init
5      (ambulance ambulance)
6      (at patient1 L3)
7      (at patient2 L4)
8      (current hospital)
9      (free ambulance)
10     (location L2)
11     (location L3)
12     (location L4)
13     (location hospital)
14     (path L2 L4)
15     (path L2 hospital)
16     (path L3 L4)
17     (path L4 L2)
18     (path L4 L3)
19     (path hospital L2)
20     (patient patient1)
21     (patient patient2))
22    (:goal (and
23            (at patient1 hospital)
24            (at patient2 hospital)
                 )))
```

Basado en el problema original se establecieron dos problemas adicionales: primero se cambió el mapa, a uno simétrico donde ahora se encuentran disponibles siete ubicaciones con caminos en dos direcciones y una sola ruta final al hospital. Se colocaron tres pacientes (ver figura 1a y listing 3). En realidad sería igual el resultado sin importar el orden en que se recoja a

los pacientes, lo interesante es saber si todos los planificadores tendrán la misma elección. En el segundo mapa solamente se limitaron los sentidos de los caminos (ver figura 1b y listing 4), y se colocaron dos pacientes, con el propósito de saber si algún planificador sugiere una ruta más larga.

Listing 3: problem02.pddl

```
(define (problem variante01)
  (:domain emergency)
  (:objects ambulance hospital
     L2 L3 L4 L5 L6 L7 patient1
     patient2 patient3)
  (:init
   (ambulance ambulance)
   (at patient1 L3)
   (at patient2 L6)
   (at patient3 L7)
   (current hospital)
   (free ambulance)
   (location L2)
   (location L3)
   (location L4)
   (location L5)
   (location L6)
   (location L7)
   (location hospital)
   (path L2 L3)
   (path L2 L4)
   (path L2 L5)
   (path L2 hospital)
   (path L3 L2)
   (path L3 L6)
   (path L3 L7)
   (path L4 L2)
   (path L4 L6)
   (path L5 L2)
   (path L5 L7)
   (path L6 L3)
   (path L6 L4)
   (path L7 L3)
   (path L7 L5)
   (path hospital L2)
   (patient patient1)
   (patient patient2)
   (patient patient3))
  (:goal (and
          (at patient1 hospital)
          (at patient2 hospital)
          (at patient3 hospital)
             )))
```

Listing 4: problem03.pddl

```
(define (problem variante02)
  (:domain emergency)
  (:objects ambulance hospital
     L2 L3 L4 L5 L6 L7 patient1
     patient2)
  (:init
   (ambulance ambulance)
   (at patient1 L5)
   (at patient2 L6)
   (current hospital)
   (free ambulance)
   (location L2)
   (location L3)
   (location L4)
   (location L5)
   (location L6)
   (location L7)
   (location hospital)
   (path L2 L3)
   (path L2 L4)
   (path L2 L5)
   (path L2 hospital)
   (path L3 L2)
   (path L4 L6)
```

```
23      (path L5 L7)
24      (path L6 L3)
25      (path L7 L3)
26      (path hospital L2)
27      (patient patient1)
28      (patient patient2))
29     (:goal (and
30             (at patient1 hospital)
31             (at patient2 hospital)
                )))
```

5. Resultados y comparación

A continuación se encontrarán los resultados del problema original (ver figura 1c) de cada uno de los planificadores probados. Los planificadores hacen el mismo número de pasos, catorce, salvo LPG-td que registró 16 pasos. Si se observa con atención entre los pasos 1 y 3 el planificador regresa innecesariamente entre los lugares L2 y L4 (ver líneas 13 a 15 en listing 6), por lo que en realidad también ofrece 14 pasos efectivos, quizá exista un error en el planificador.

Dentro del plan entregado, los planificadores FF y Metis coinciden en el resultado final, en ambos se recoge primero al paciente de la ubicación L3 (ver listing 5 y 9); los planificadores LPG-td y SPD por su parte decidieron recoger primero al paciente de la ubicación L4, dicha ubicación es la más cercana al hospital, mismo que define el estado inicial del robot (ver listing 7 y 6). Sin embargo, esta diferencia no hace más costoso ninguno de los dos planes posibles.

Respecto al tiempo de ejecución, FF y LPG-td marcaron en cero el contador de dos decimales. Si se recurre al comando `time`, se puede ver que FF tarda $0.008s$ y LPG-td en promedio tarda $0.040s$. Metis es el segundo más rápido, incluso cuando requiere un contenedor Singularity, con un tiempo de $0.001s$, aunque el más rápido fue SPD, que se corrió desde un servidor externo mediante una API (Muise and Taylor-Muise, 2015), requirió tan solo $1.8999x10^{-10}s$ (ver listing 8). Al probar en otros planificadores, Scorpion tardó $200.014s$ (ver listing 10).

Listing 5: Resultado de FF

```
1  ff: parsing domain file
2  domain 'EMERGENCY' defined
3   ... done.
4  ff: parsing problem file
5  problem 'ORIGINAL' defined
6   ... done.
7
8  Cueing down from goal distance: 7 into depth
       [1][2][3][4][5]
9                                  6 [1]
10                                 5 [1][2]
11                                 4 [1][2][3]
12                                 3 [1]
13                                 2 [1]
14                                 1 [1]
15                                 0
16
17 ff: found legal plan as follows
18
19 step 0: MOVE HOSPITAL L2
20        1: MOVE L2 L4
21        2: MOVE L4 L3
22        3: GETINTO PATIENT1 L3 AMBULANCE
23        4: MOVE L3 L4
24        5: MOVE L4 L2
25        6: MOVE L2 HOSPITAL
26        7: GETOUT PATIENT1 HOSPITAL AMBULANCE
27        8: MOVE HOSPITAL L2
28        9: MOVE L2 L4
29       10: GETINTO PATIENT2 L4 AMBULANCE
30       11: MOVE L4 L2
31       12: MOVE L2 HOSPITAL
32       13: GETOUT PATIENT2 HOSPITAL AMBULANCE
33
34 time spent: 0.00 seconds instantiating 22 easy, 0
       hard action templates
35             0.00 seconds reachability analysis,
                   yielding 15 facts and 22 actions
36             0.00 seconds creating final
                   representation with 15 relevant
                   facts
37             0.00 seconds building connectivity
```

```
                graph
0.00 seconds searching, evaluating 21
     states, to a max depth of 5
0.00 seconds total time
```

Listing 6: Resultado de LPG-td

```
; Version LPG-td-1.4
; Seed 33654337
; Command line: ./planners/LPG-td-1.4/lpg-td
    -o domain.pddl -f problem01.pddl -n 1 -
   out out/p1-lpg-td
; Problem problem01.pddl
; Actions having STRIPS duration
; Time 0.00
; Search time 0.00
; Parsing time 0.00
; Mutex time 0.00
; NrActions 16

0:   (MOVE HOSPITAL L2) [1]
1:   (MOVE L2 L4) [1]
2:   (MOVE L4 L2) [1]
3:   (MOVE L2 L4) [1]
4:   (GETINTO PATIENT2 L4 AMBULANCE) [1]
5:   (MOVE L4 L2) [1]
6:   (MOVE L2 HOSPITAL) [1]
7:   (GETOUT PATIENT2 HOSPITAL AMBULANCE)
    [1]
8:   (MOVE HOSPITAL L2) [1]
9:   (MOVE L2 L4) [1]
10:   (MOVE L4 L3) [1]
11:   (GETINTO PATIENT1 L3 AMBULANCE) [1]
12:   (MOVE L3 L4) [1]
13:   (MOVE L4 L2) [1]
14:   (MOVE L2 HOSPITAL) [1]
15:   (GETOUT PATIENT1 HOSPITAL AMBULANCE)
    [1]
```

Listing 7: Plan de Solver Planning Domain

```
(move hospital l2)
(move l2 l4)
(getinto patient2 l4 ambulance)
(move l4 l2)
(move l2 hospital)
(getout patient2 hospital ambulance)
(move hospital l2)
(move l2 l4)
(move l4 l3)
(getinto patient1 l3 ambulance)
(move l3 l4)
(move l4 l2)
(move l2 hospital)
(getout patient1 hospital ambulance)
```

Listing 8: Resultado de Solver Planning Domain

```
 --- OK.
 Match tree built with 22 nodes.

PDDL problem description loaded:
        Domain: EMERGENCY
        Problem: ORIGINAL
        #Actions: 22
        #Fluents: 15
Landmarks found: 2
Starting search with IW (time budget is 60
    secs)...
rel_plan size: 7
#RP_fluents 8
Caption
{#goals, #UNnachieved,  #Achieved} -> IW(
    max_w)

{2/2/0}:IW(1) -> [2][3][4][5][6][7]rel_plan
    size: 5
#RP_fluents 6
{2/1/1}:IW(1) -> [2][3][4][5][6][7][8][9]
    rel_plan size: 0
#RP_fluents 0Plan found with cost: 14
Total time: -1.8999e-10
Nodes generated during search: 58
Nodes expanded during search: 51
IW search completed
```

Listing 9: Resultado de Metis

```
  Solution found!
  Actual search time: 0s [t=0.00119607s]
  move hospital l2 (1)
  move l2 l4 (1)
  move l4 l3 (1)
  getinto patient1 l3 ambulance (1)
  move l3 l4 (1)
  move l4 l2 (1)
  move l2 hospital (1)
  getout patient1 hospital ambulance (1)
  move hospital l2 (1)
  move l2 l4 (1)
  getinto patient2 l4 ambulance (1)
  move l4 l2 (1)
  move l2 hospital (1)
  getout patient2 hospital ambulance (1)
  Plan length: 14 step(s).
  Plan cost: 14
  Expanded 30 state(s).
  Reopened 0 state(s).
```

```
    Evaluated 40 state(s).
    Evaluations: 80
    Generated 69 state(s).
    Dead ends: 0 state(s).
    Expanded until last jump: 25 state(s).
    Reopened until last jump: 0 state(s).
    Evaluated until last jump: 33 state(s).
    Generated until last jump: 58 state(s).
    Number of registered states: 40
    total successors before partial-order
        reduction: 69
    total successors after partial-order
        reduction: 69
    Search time: 0s
    Total time: 0.00119607s
    Solution found.
    Peak memory: 4960 KB
```

Listing 10: Resultado de Scorpion

```
    Solution found!
    Actual search time: 0s [t=200.014s]
    move hospital l2 (1)
    move l2 l4 (1)
    move l4 l3 (1)
    getinto patient1 l3 ambulance (1)
    move l3 l4 (1)
    move l4 l2 (1)
    move l2 hospital (1)
    getout patient1 hospital ambulance (1)
    move hospital l2 (1)
    move l2 l4 (1)
    getinto patient2 l4 ambulance (1)
    move l4 l2 (1)
    move l2 hospital (1)
    getout patient2 hospital ambulance (1)
    Plan length: 14 step(s).
    Plan cost: 14
    Expanded 24 state(s).
    Reopened 0 state(s).
    Evaluated 37 state(s).
    Evaluations: 37
    Generated 58 state(s).
    Dead ends: 0 state(s).
    Expanded until last jump: 14 state(s).
    Reopened until last jump: 0 state(s).
    Evaluated until last jump: 24 state(s).
    Generated until last jump: 36 state(s).
    Number of registered states: 37
    Int hash set load factor: 37/64 = 0.578125
    Int hash set resizes: 6
    total successors before partial-order
        reduction: 58
    total successors after partial-order
        reduction: 58
    Number of times each order was the best
        order: [16, 11, 0, 6, 3]
    Probably useful orders: 4/5 = 80%
    Search time: 0.000681004s
    Total time: 200.014s
    Solution found.
    Peak memory: 5512 KB
    exitcode: 0
```

Al describir los problemas variantes se observó que para Metis los tiempos de ejecución aumentaron a $0.0046s$ en la primera variante del problema (ver listing 12), mientras que para los otros tres planificadores los tiempos se mantuvieron sin cambios, incluso SPD mantuvo el tiempo de ejecución completamente igual. La diferencia negativa la marcó LPG-td que en la primera variante del problema volvió a registrar un número mayor de pasos: 26 pasos; mientras que los otros planificadores necesitaron solamente 22 pasos en la solución. En la segunda variante, al tener caminos en una sola dirección, LPG-td por fin logró obtener 16 pasos, mismos que los otros planificadores.

Los listings 11 y 13 muestran los planes obtenidos para resolver los dos problemas adicionales planteados.

Listing 11: Plan de la primera variante con Solver Planning Domain

```
(move hospital l2)
(move l2 l3)
(getinto patient1 l3 ambulance)
(move l3 l2)
(move l2 hospital)
(getout patient1 hospital ambulance)
(move hospital l2)
(move l2 l3)
(move l3 l6)
(getinto patient2 l6 ambulance)
(move l6 l3)
(move l3 l2)
(move l2 hospital)
(getout patient2 hospital ambulance)
(move hospital l2)
(move l2 l3)
```

```
(move l3 l7)
(getinto patient3 l7 ambulance)
(move l7 l3)
(move l3 l2)
(move l2 hospital)
(getout patient3 hospital ambulance)
```

Listing 12: Plan de la primera variante con Metis

```
Solution found!
Actual search time: 0.00299427s [t
    =0.00460844s]
move hospital l2 (1)
move l2 l3 (1)
move l3 l6 (1)
getinto patient2 l6 ambulance (1)
move l6 l3 (1)
move l3 l2 (1)
move l2 hospital (1)
getout patient2 hospital ambulance (1)
move hospital l2 (1)
move l2 l3 (1)
move l3 l7 (1)
getinto patient3 l7 ambulance (1)
move l7 l3 (1)
move l3 l2 (1)
move l2 hospital (1)
getout patient3 hospital ambulance (1)
move hospital l2 (1)
move l2 l3 (1)
getinto patient1 l3 ambulance (1)
move l3 l2 (1)
move l2 hospital (1)
getout patient1 hospital ambulance (1)
Plan length: 22 step(s).
Plan cost: 22
Expanded 226 state(s).
Reopened 0 state(s).
Evaluated 286 state(s).
Evaluations: 572
Generated 689 state(s).
Dead ends: 0 state(s).
Expanded until last jump: 221 state(s).
Reopened until last jump: 0 state(s).
Evaluated until last jump: 278 state(s).
Generated until last jump: 674 state(s).
Number of registered states: 286
total successors before partial-order
    reduction: 689
total successors after partial-order
    reduction: 689
Search time: 0.00314171s
Total time: 0.00460844s
Solution found.
Peak memory: 5088 KB
```

Listing 13: Plan de la segunda variante con Solver Planning Domain

```
(move hospital l2)
(move l2 l4)
(move l4 l6)
(getinto patient2 l6 ambulance)
(move l6 l3)
(move l3 l2)
(move l2 hospital)
(getout patient2 hospital ambulance)
(move hospital l2)
(move l2 l5)
(getinto patient1 l5 ambulance)
(move l5 l7)
(move l7 l3)
(move l3 l2)
(move l2 hospital)
(getout patient1 hospital ambulance)
```

En general se ha descrito un programa dominio, el cual es invariable. También se escribieron varios problemas que corresponden a dicho dominio, una vez comprendidas las determinaciones del dominio resulta sencillo describir los problemas a resolver. Pese a ello, hay dos dificultades principales al crear estos programas: 1) La documentación sobre el lenguaje; 2) La verificación del programa escrito.

Sobre la documentación del lenguaje, no fue posible encontrar un estándar o documentación oficial sobre cualquiera de las versiones de PDDL. Cada versión tiene diferentes opciones de parámetros posibles, aun así tan solo con el uso del parámetro *:typing*, el más común entre versiones, se presentaron problemas. También se dice que la sintaxis es la misma de Lisp, mas al usar una expresión S del conector lógico OR, fue igualmente complicado hacer funcionar el programa. Los programas presentados tienen la sintaxis más básica de STRIPS, el antecesor de PDDL y con mayor correspondencia a Lisp, una

familia de lenguajes conocida y bien documentada. El listing 14 contiene un problema en el cual hay dos hospitales, y los pacientes deben ser llevados a cualquiera de los hospitales. Dicho código es funcional, hay planificadores que encuentran solución, sin embargo, algunos de aquellos incorporados en IBacop tienen problemas con la sintaxis aquí presentada. Sumado a ello, SPD resuelve el problema, pero no detalla el plan como lo hace en otros casos. Si bien el planificador no tiene un mensaje concreto, es presumible que el elemento sintáctico nuevo, el conector lógico OR (líneas 30 a 35), es el causante del problema, por esa razón este código no fue incluido en las pruebas metodológicas.

Listing 14: problem04.pddl

```
1  (define (problem variante03)
2    (:domain emergency)
3    (:objects ambulance hospital1
        L2 L3 hospital2 L5 L6 L7
        patient1 patient2)
4    (:init
5     (ambulance ambulance)
6     (at patient1 L5)
7     (at patient2 L6)
8     (current hospital1)
9     (free ambulance)
10    (location L2)
11    (location L3)
12    (location hospital2)
13    (location L5)
14    (location L6)
15    (location L7)
16    (location hospital1)
17    (path L2 L3)
18    (path L2 hospital2)
19    (path L2 L5)
20    (path L2 hospital1)
21    (path L3 L2)
22    (path hospital2 L6)
23    (path L5 L7)
24    (path L6 L3)
25    (path L7 L3)
26    (path hospital1 L2)
27    (patient patient1)
28    (patient patient2))
29   (:goal (and
30            (or
31                (at patient1
                     hospital1)
32                (at patient1
                     hospital2))
33            (or
34                (at patient2
                     hospital1)
35                (at patient2
                     hospital2))))
                     )
```

Agregado a lo anterior se encuentra la falta de un verificador léxico/semántico para el programa. La única forma de comprobación es la ejecución sobre el planificador, el cual no da detalles adecuados para la depuración del programa, ya que no es su propósito. Para esta labor existe el editor en línea de planning.domains; tiene las mismas carencias que el modo PDDL de GNU/Emacs: no corrige la sintaxis y no permite escoger con antelación la versión de PDDL que se desea emplear, por lo que no es posible saber si los errores son ocasionados por el programador o por limitaciones del planificador que se usa para probar el programa. Aunque existe documentación sobre otros programas escritos en Java, no se han podido probar por la dificultad al encontrar el código fuente correspondiente.

6. Conclusión

Durante este texto se ha explorado la escritura de dominios y problemas de planificación automática en lenguajes STRIPS/PDDL, y su verificación por medio de diversos planificadores de código abierto. Se han descrito los elementos básicos obligatorios para la escritura de código y los resultados obtenido de los planificadores.

Se ha visto que los planificadores tienen rendimiento constante aunque existe variación entre ellos. Si bien los resultados son muy similares, un tiempo de ejecución mejorado puede ser importante al resolver problemas de planificación más grandes.

Se ha hecho énfasis en las dificultades provocadas por la falta de documentación clara y herramientas de desarrollo avanzadas. Quizá el trabajo a futuro más interesante se encuentra en estos dos puntos, pues permitiría una mejor colaboración entre aquellos interesados en la planificación automática.

Los detalles adicionales con resultados más detallados que no pudieron ser incluidos en este documento se encuentran en el repositorio de la actividad: https://-gitlab.com/genomorro/unir/-/tree/RPA-03.

Referencias

Cenamor, I., de la Rosa, T., and Fernández, F. (2018). IBaCoP-2018 and IBaCoP2-2018. *Ninth International Planning Competition (IPC 2018)*.

Domínguez Espinoza, E. U. (2017). *Procesador de lenguas naturales*. Escuela Nacional de Antropología e Historia.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Elsevier.

Green, A. (2021a). What is ai planning?

Green, A. (2021b). What is planning domain definition language (pddl)?

Muise, C. and Taylor-Muise, K. (2015). Pddl solver.

Seipp, J. (2018). Scorpion (planner abstract). *Ninth International Planning Competition (IPC 2018)*.

Sievers, S. and Katz, M. (2018). Metis. *Ninth International Planning Competition (IPC 2018)*.