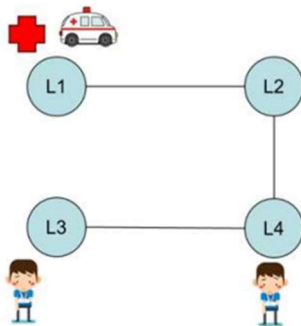


### Abstract

El presente trabajo muestra la ejecución y entendimiento de 4 planificadores que fueron mostrados en la conferencia ICAPS 2018, dichos planificadores fueron ejecutados con un Domain y un Problem que resuelve lo siguiente:



Una vez comprobada la ejecución de forma correcta, se procedió a realizar modificaciones en el problema con la finalidad de resolver uno nuevo y analizar los resultados.

### Introducción

La planificación y programación automatizadas, a veces denominada simplemente planificación de IA, es una rama de la inteligencia artificial que se ocupa de la realización de estrategias o secuencias de acción, normalmente para que las ejecuten agentes inteligentes, robots autónomos y vehículos no tripulados. A diferencia de los problemas clásicos de control y clasificación, las soluciones son complejas y deben descubrirse y optimizarse en un espacio multidimensional. La planificación también está relacionada con la teoría de la decisión.

En entornos conocidos con modelos disponibles, la planificación se puede realizar sin conexión. Las soluciones se pueden encontrar y evaluar antes de la ejecución.

En entornos dinámicamente desconocidos, la estrategia a menudo debe revisarse en línea. Los modelos y las políticas deben adaptarse. Las soluciones suelen recurrir a procesos iterativos de prueba y error comúnmente vistos en inteligencia artificial. Estos incluyen programación dinámica, aprendizaje por refuerzo y optimización combinatoria. Los lenguajes utilizados para describir la planificación y la programación a menudo se denominan lenguajes de acción.

Estos procesos iterativos de prueba y error comúnmente vistos en inteligencia artificial. Estos incluyen programación dinámica, aprendizaje por refuerzo y optimización combinatoria. Los lenguajes utilizados para describir la planificación y la programación a menudo se denominan lenguajes de acción.

En la planificación de IA, los planificadores suelen introducir un modelo de dominio (una descripción de un conjunto de acciones posibles que modelan el dominio), así como el problema específico que se resolverá especificado por el estado inicial y el objetivo, en contraste con aquellos en los que no hay dominio de entrada especificado.

Dichos planificadores se denominan "independientes del dominio" para enfatizar el

hecho de que pueden resolver problemas de planificación de una amplia gama de dominios.

Los ejemplos típicos de dominios son el apilamiento de bloques, la logística, la gestión del flujo de trabajo y la planificación de tareas de robots. Por lo tanto, se puede utilizar un solo planificador independiente del dominio para resolver problemas de planificación en todos estos diversos dominios. Por otro lado, un planificador de rutas es típico de un planificador de dominio específico.

## Desarrollo

### DecStar

DecStar es una técnica recientemente introducida en la planificación de IA, la cual explota la independencia entre los componentes de una tarea de planificación para reducir el tamaño del espacio-estado de la representación; dividiendo las variables de estado en componentes, de tal manera que la interacción entre estos toma la forma de una **topología estrella**, la búsqueda desacoplada solo se busca sobre secuencias de acción afectando el componente central de la topología, y enumera asignaciones accesibles para cada componente por separado.

Esto puede conducir a una reducción exponencial del tamaño de la búsqueda-espacio. No siempre es fácil encontrar una partición para la tarea planificada dada, aun así, amplía el algoritmo STD (topología estrella) por una opción, que ejecuta una búsqueda estándar siempre que haya fallado en encontrar una partición (buena).

```
ald@ald: ~  
Solution found!  
Actual search time: 0s [t=0s]  
mover a l1 l2 (1)  
mover a l2 l4 (1)  
mover a l4 l3 (1)  
subir p1 l3 a (1)  
mover a l3 l4 (1)  
mover a l4 l2 (1)  
mover a l2 l1 (1)  
bajar p1 l1 a (1)  
mover a l1 l2 (1)  
mover a l2 l4 (1)  
subir p2 l4 a (1)  
mover a l4 l2 (1)  
mover a l2 l1 (1)  
bajar p2 l1 a (1)  
Plan length: 14 step(s).  
Plan cost: 14  
Initial state h value: 7.  
Expanded 55 state(s).  
Reopened 0 state(s).  
Evaluated 86 state(s).  
Evaluations: 86  
Generated 132 state(s).  
Dead ends: 0 state(s).  
Expanded until last jump: 50 state(s).  
Reopened until last jump: 0 state(s).  
Evaluated until last jump: 79 state(s).  
Generated until last jump: 121 state(s).  
Number of registered states: 86  
total successors before partial order reduction: 132
```

Figura 1.1

En la figura 1.1 podemos comprobar que se corrió el planificador de manera correcta, en 0.0 seg. (posiblemente en milésimas de segundos).

### Planificador Delfi1

El planificador Delfi es un planificador de **portafolio online**. Delfi explota técnicas de **Deep learning** (en contraste con otras técnicas existentes) para aprender un modelo que predice cuál de los planificadores en el portafolio puede resolver una tarea planificada determinada, dentro de los límites de tiempo y memoria dada. Delfi usa representaciones gráficas de una tarea planificada, la cual permite explotar herramientas existentes para la convolución de la imagen.

```
ald@ald: ~  
N Rhythmicbox uristic value for cpdbs_symbolic(genetic_ss(use_uct = true, num_episodes = 1000000, n  
um_collections = 1, pdb_factory = symbolic, genetic_time_limit = 900, time_limit = 1.0, create_pe  
rimeter = true, use_first_goal_vars = true, use_norm_dist = true)): 0  
[g=14, 25 evaluated, 14 expanded, t=1.42007s, 520732 KB]  
Solution found!  
Actual search time: 0s [t=1.42007s]  
mover a l1 l2 (1)  
mover a l2 l4 (1)  
mover a l4 l3 (1)  
subir p1 l3 a (1)  
mover a l3 l4 (1)  
mover a l4 l2 (1)  
mover a l2 l1 (1)  
bajar p1 l1 a (1)  
mover a l1 l2 (1)  
mover a l2 l4 (1)  
subir p2 l4 a (1)  
mover a l4 l2 (1)  
mover a l2 l1 (1)  
bajar p2 l1 a (1)  
Plan length: 14 step(s).  
Plan cost: 14  
Expanded 15 state(s).  
Reopened 0 state(s).  
Evaluated 25 state(s).  
Evaluations: 25  
Generated 33 state(s).  
Dead ends: 0 state(s).  
Expanded until last jump: 0 state(s).  
Reopened until last jump: 0 state(s).  
Evaluated until last jump: 1 state(s).  
Generated until last jump: 0 state(s).  
Number of registered states: 25  
Search time: 0.035826s  
Total time: 1.42007s  
Solution found.  
Peak memory: 987376 KB  
ald@ald: ~
```

Figura 2.1

En la figura 2.1 podemos comprobar que se corrió el planificador de manera correcta con 14 pasos, en 1.42 seg.

### MSP

(Meta-Search-Planner) es un sistema de meta-razonamiento que busca, a través del espacio de planificadores, representaciones y heurísticas problema por problema.

Dado un problema de planificación, MSP se realiza en dos fases: fase de meta-búsqueda y fase de resolución de problemas. La fase de meta-búsqueda es un proceso de búsqueda que genera una combinación adecuada para resolver de manera óptima el problema en específico. La fase de resolución de problema, es básicamente una llamada al planificador seleccionado con otros elementos de la combinación seleccionada por el algoritmo.

Figura 4.1

En la figura 2.1 podemos comprobar que se corrió el planificador de manera correcta con 14 pasos, en 200.505 seg (el que mayor tiempo ha tardado).

```
aldo@aldo: ~  
aldo@aldo:~$ RUNDIR="$(pwd)/rundir"  
aldo@aldo:~$ DOMAIN="$RUNDIR/domain.pddl"  
aldo@aldo:~$ PROBLEM="$RUNDIR/p01.pddl"  
aldo@aldo:~$ PLANFILE="$RUNDIR/sas_plan"  
aldo@aldo:~$ COSTBOUND=42 # only in cost-bounded track  
aldo@aldo:~$ ulimit -t 1800  
aldo@aldo:~$ ulimit -v 8388608  
aldo@aldo:~$ singularity run -C -H $RUNDIR planner.img $DOMAIN $PROBLEM $PLANFILE  
INFO Running translator.  
INFO translator input: ['/home/aldo/rundir/domain.pddl', '/home/aldo/rundir/  
p01.pddl']  
INFO translator arguments: []  
INFO translator time limit: 1799.91s  
INFO translator memory limit: 8192 MB  
INFO callstring: /usr/bin/python /planner/builds/release64/bin/translate/tra  
nslate.py /home/aldo/rundir/domain.pddl /home/aldo/rundir/p01.pddl  
time limit 1799.91 -> (1800, 1800)  
Parsing...  
Undeclared predicate: delivered_package1_l1_t2  
Command '['/usr/bin/python', '/planner/builds/release64/bin/translate/translate.  
py', '/home/aldo/rundir/domain.pddl', '/home/aldo/rundir/p01.pddl']' returned no  
n-zero exit status 1  
aldo@aldo:~$
```

## Modificación del problema

Se decidió hacer 2 variantes ligeras del problema para no influenciar su tiempo de ejecución. Analizando la ejecución podremos ver que los 2 mejores planificadores que obtuvieron mejores resultados son **DecStar** y **MSP**.

Aunque Delfi1 tuvo un tiempo de ejecución muy corto (milisegundos), analizando los resultados en consola podremos observar que la utilización de memoria fue demasiada (987376 KB) a comparación de los demás.

Para el nuevo problema se crearon 2 localizaciones extras, además de movimiento en diagonal, para aumentar la dificultad del problema.

Se movieron de localización a los pacientes de L4 y L3 a L5 y L6, mientras que la ambulancia y el hospital se dejaron como estaban.

```
aldo@aldo:~$ singularity run -C -H $RUNDIR planner.img $DOMAIN $PROBLEM $PLANFILE  
INFO Running translator.  
INFO translator input: ['/home/aldo/rundir/domain.pddl', '/home/aldo/rundir/  
p01.pddl']  
INFO translator arguments: []  
INFO translator time limit: 1799.91s  
INFO translator memory limit: 8192 MB  
INFO callstring: /usr/bin/python /planner/builds/release64/bin/translate/tra  
nslate.py /home/aldo/rundir/domain.pddl /home/aldo/rundir/p01.pddl  
time limit 1799.91 -> (1800, 1800)  
Parsing...  
Undeclared predicate: delivered_package1_l1_t2  
Command '['/usr/bin/python', '/planner/builds/release64/bin/translate/translate.  
py', '/home/aldo/rundir/domain.pddl', '/home/aldo/rundir/p01.pddl']' returned no  
n-zero exit status 1  
aldo@aldo:~$
```

Figura 3.1

En la figura 3.1 podemos comprobar que se corrió el planificador de manera correcta con 14 pasos, en 1.4 seg.

## Scorpion

El planificador utiliza A\* con la heurística admisible para crear planes óptimos. La heurística en general esta basada en componentes abstractos heurísticos que son combinados mediante la partición de costos saturados

```
aldo@aldo:~$ singularity run -C -H $RUNDIR planner.img $DOMAIN $PROBLEM $PLANFILE  
INFO Running translator.  
INFO translator input: ['/home/aldo/rundir/domain.pddl', '/home/aldo/rundir/  
p01.pddl']  
INFO translator arguments: []  
INFO translator time limit: 1799.91s  
INFO translator memory limit: 8192 MB  
INFO callstring: /usr/bin/python /planner/builds/release64/bin/translate/tra  
nslate.py /home/aldo/rundir/domain.pddl /home/aldo/rundir/p01.pddl  
time limit 1799.91 -> (1800, 1800)  
Parsing...  
Undeclared predicate: delivered_package1_l1_t2  
Command '['/usr/bin/python', '/planner/builds/release64/bin/translate/translate.  
py', '/home/aldo/rundir/domain.pddl', '/home/aldo/rundir/p01.pddl']' returned no  
n-zero exit status 1  
aldo@aldo:~$
```

## Modificación 1

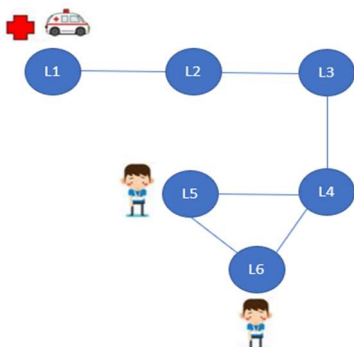


Figura 5.1

En la figura 5.1 podemos ver el problema a resolver con la primera modificación, con 2 nuevas localizaciones (L5 y L6), además de la reubicación de los pacientes (L6 y L5, ahora).

A continuación, se mostrarán los resultados de los planificadores DecStar y MSP con el archivo Problem.pddl, cambiado acorde a la primera modificación.

### DecStar

Fue más rápido (0.05 seg) que MSP, la utilización de su **memoria** (5204 KB) para ejecutar el problema en el planificador fue una gran sorpresa, haciéndolo a este el gran ganador. **20 pasos** fueron suficientes para resolver el problema.

```
alido@alido: ~$ cat Problem.pddl
...
(mover a 14 13 (1))
(mover a 13 12 (1))
(mover a 12 11 (1))
(bajar p1 14 a (1))
(mover a 11 12 (1))
(mover a 12 13 (1))
(mover a 13 14 (1))
(mover a 14 15 (1))
(subir p2 16 a (1))
(mover a 16 14 (1))
(mover a 14 13 (1))
(mover a 13 12 (1))
(mover a 12 11 (1))
(bajar p2 11 a (1))
(plan length: 20 step(s).)
(plan cost: 20)
Initial state h value: 9.
Expanded 331 state(s).
Reopened 9 state(s).
Evaluated 683 state(s).
Evaluations: 683
Generated 1119 state(s).
Dead ends: 0 state(s).
Expanded until last jump: 302 state(s).
Reopened until last jump: 9 state(s).
Evaluated until last jump: 619 state(s).
Generated until last jump: 1041 state(s).
Number of registered states: 683
Number of registered leaf states in factor 0: 57
Number of registered leaf states in factor 1: 740
Decoupled state space size [1] 20607
Decoupled state space size [2] 5247
Decoupled state space size [3] 54783
Decoupled state space size [4] 14526
Maximum duplicate counter 100
Total successors before partial-order reduction: 1122
Total successors after partial-order reduction: 1122
Search time: 0.05s
Total time: 0.05s
Solution found.
Peak memory: 5204 KB
...
alido@alido: ~$
```

Figura 5.2

En la figura 6.1 se muestra el planificador DecStar ejecutando el problema modificado.

### MSP

Este planificador **no fue el más rápido** (0.16 seg) y además fue el que **más memoria usó** (238928 KB), al igual que DecStar, utilizó 20 pasos para la ejecución del problema en el planificador.

```
alido@alido: ~$ cat Problem.pddl
...
(mover a 11 12 (1))
(mover a 12 13 (1))
(mover a 13 14 (1))
(mover a 14 15 (1))
(subir p2 16 a (1))
(mover a 16 14 (1))
(mover a 14 13 (1))
(mover a 13 12 (1))
(mover a 12 11 (1))
(bajar p2 11 a (1))
(mover a 11 12 (1))
(mover a 12 13 (1))
(mover a 13 14 (1))
(mover a 14 15 (1))
(subir p1 15 a (1))
(mover a 15 14 (1))
(mover a 14 13 (1))
(mover a 13 12 (1))
(mover a 12 11 (1))
(bajar p1 11 a (1))
(plan length: 20 step(s).)
(plan cost: 20)
Expanded 0 state(s).
Reopened 0 state(s).
Evaluated 0 state(s).
Evaluations: 0
Generated 0 state(s).
Dead ends: 0 state(s).
Search space hash size: 0
Search space hash bucket count: 193
Search time: 0.02s
Total time: 0.16s
Solution found.
Peak memory: 238928 KB
...
alido@alido: ~$
```

Figura 5.3

En la figura 6.1 se muestra el planificador MSP ejecutando el problema modificado.

## Modificación 2

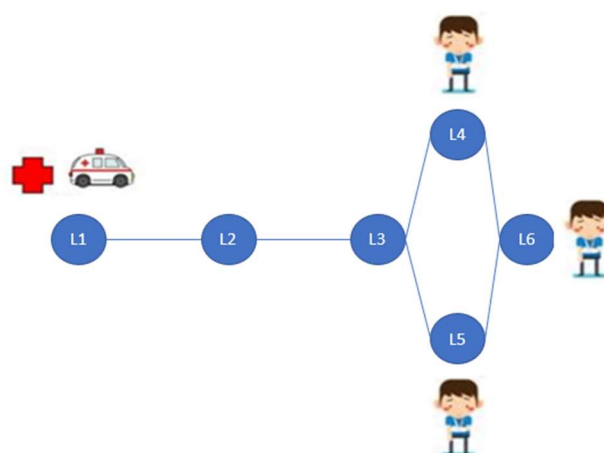


Figura 6.1

En la figura 6.1 podemos ver el problema a resolver con la segunda modificación, con 2 nuevas localizaciones (L5 y L6), se agrega un nuevo paciente (L4) y se reubican los 2 restantes (L6 y L5).

A continuación, se mostrarán los resultados de los planificadores DecStar y MSP con el archivo Problem.pddl, cambiado acorde a la segunda modificación.

DecStar

A diferencia de la primera modificación, DecStar resulto ser el menos viable. Con un uso de **memoria de 30908 KB**, un tiempo de ejecución de 32.19 seg y un costo de **28 pasos**, queda muy por debajo de MSP.

```

nover a 12 11 (1)
bajzar p3 11 a (1)
nover a 11 12 (1)
nover a 12 12 (1)
nover a 13 14 (1)
nover a 13 14 (1)
nover p1 14 a (1)
nover a 14 13 (1)
nover a 13 12 (1)
nover a 12 12 (1)
bajzar p1 11 a (1)
Plan length: 28 steps(s).
Plan cost: 28
Initial state h value: 11.
Expanded 13663 state(s).
Reopened 138 state(s).
Evaluated 25938 state(s).
Evaluation time: 23978
Generated 58928 state(s).
Load memory: 0 state(s).
Expanded until last jump: 13646 state(s).
Reopened until last jump: 138 state(s).
Evaluated until last jump: 25866 state(s).
Generated until last jump: 58804 state(s).
Number of registered states: 25938
Number of registered leaf states in factor 0: 209
Number of registered leaf states in factor 1: 5768
decoupled state space size [1] 1277260
decoupled state space size [2] 1994260
decoupled state space size [3] 155261558
decoupled state space size [4] 287821
newton duplicate counter: 449
Total successors before partial-order reduction: 3633
Total successors after partial-order reduction: 3603
Search time: 32.35s
Total time: 32.19s
Solution found.
Peak memory: 30908 kb
returncode: 0

Exit codes: [5, 0]

```

MSP

Este algoritmo resulto la mejor opción para la segunda modificación, ya que solamente tardo **.500 seg** en ejecutarse, sorprendentemente el uso de memoria de **11180 KB** es mucho más bajo que DecStar , y tardo **28 pasos** el resolver el problema.

```

bajzer p2 14 a (1)
hover a 11 12 (1)
hover a 12 13 (1)
hover a 13 14 (1)
hover a 14 15 (1)
subL p1 16 a (1)
hover a 16 14 (1)
hover a 14 13 (1)
hover a 13 12 (1)
hover a 12 11 (1)
bajzer p2 14 a (1)
hover a 11 12 (1)
hover a 12 13 (1)
hover a 13 14 (1)
subL p1 14 a (1)
hover a 14 13 (1)
hover a 13 12 (1)
hover a 12 11 (1)
bajzer p1 11 a (1)
Plan length: 28 step(s).
Plan cost: 28
Expanded 8865 state(s).
Reopened 9 state(s).
Evaluated 12072 state(s).
Evaluation cost: 12672
Generated 21994 state(s).
Dead ends: 0 state(s).
Expanded until last jump: 7575 state(s).
Reopened until last jump: 0 state(s).
Evaluated until last jump: 11891 state(s).
Generated until last jump: 20572 state(s).
Number of registered states: 12072
Search time: 0.85489s
Total time: 0.379807s
Sollition found.
Peak memory: 11189 MB
Overall time: (0.506s CPU, 0.579s wall-clock)

==== Done running the selected planner. ====

aldagaldin:~$

```

## Conclusión

Se ejecutaron 2 planificadores con 2 modificaciones en la problemática (2 localizaciones extra , cambio de ubicación en los pacientes y se agregó un nuevo paciente) de manera rápida en Ubuntu 20.04.4 LTS y VirtualBox 6.1.

Cabe resaltar que, aunque el resultado es el mismo (20 pasos y 28 pasos, dependiendo el problema), el algoritmo que usa cada planificador es diferente, DecStar usa topología estrella mientras que MSP usa un proceso de búsqueda que genera una combinación adecuada para resolver de manera óptima el problema en específico.

Como se dijo en el trabajo presentado, **DecStar** fue el que mejor obtuvo resultados para la modificación 1, mientras que **MSP** fue el mejor que obtuvo resultados para la modificación 2, estos resultados fueron determinados por el uso de memoria y el tiempo de ejecución de los 2 algoritmos por separado



## **Referencias**

Tema: Planificación en Inteligencia Artificial.  
Agentes Planificadores. (s. f.). Facultad: Ingeniería  
Escuela: Computación Asignatura: Sistemas  
Expertos e Inteligencia Artificial. Recuperado 13  
de marzo de 2022, de  
[https://docplayer.es/91429912-Tema-  
planificacion-en-inteligencia-artificial-agentes-  
planificadores.html](https://docplayer.es/91429912-Tema-planificacion-en-inteligencia-artificial-agentes-planificadores.html)

<https://planning.wiki/ref>

<http://editor.planning.domains>