IPC 2018: Un vistazo a las herramientas de planificación utilizadas en la competencia

D. Toscano, E. Giraldo, J. Orellana, R. D. Aponte*

Universidad Internacional de La Rioja, Logroño (España)

Resumen

La *International Planning Competition* (*IPC*) es una competición en donde diferentes equipos compiten para evaluar empíricamente los sistemas de planificación o planificadores más avanzados (estado del arte) en una serie de problemas de referencia. En su versión 2018, con soporte para problemas en lenguaje PDDL 3.1, 35 planificadores participaron en la competición clásica en 4 categorías: *Optimal Track, Satisficing Track, Agile Track* y *Cost-Bounded Track.* En este trabajo se revisaron los algoritmos de búsqueda de planificadores desarrollados por 4 equipos participantes. También, se detalla el procedimiento para el uso de cada uno de estos planificadores, desde el proceso de descarga hasta la instalación y ejecución en un entorno de máquina virtual para contenedores o *dockers* (Singularity) desde un sistema operativo nativo (*Linux*) o desde el uso de un subsistema como *Windows Subsystem for Linux* (*WSL*); todo esto para la resolución de un problema descrito en lenguaje *PDDL*, comparando los resultados para los diferentes planificadores.

Palabras Clave

Algoritmos, búsqueda, contenedores, máquinas, planificadores, Singularity, virtuales.

I. Introducción

Los objetivos principales de la Conferencia internacional sobre planificación y programación automatizadas (*ICAPS*, por sus siglas en inglés) son promover el campo de la planificación y programación automatizadas a través de la organización de reuniones técnicas, incluida la conferencia anual, concursos y otros medios para avanzar y evaluar el estado del arte en el campo, mediante la promoción y difusión de publicaciones, sistemas de planificación y programación, dominios, simuladores, herramientas de software y material técnico.

Dentro de las actividades de promoción está el Concurso Internacional de Planificación (*IPC*, por sus siglas en inglés) que se realiza de manera periódica en donde diferentes equipos compiten para evaluar empíricamente los sistemas de planificación o planificadores más avanzados (estado del arte) en una serie de problemas de referencia. Existen 3 ramas de *tracks* o pistas de concurso: determinísticas (clásicas), probabilísticas y Temporales.

En este trabajo se revisaron los trabajos o planificadores desarrollados por 4 equipos en el concurso determinístico (clásico), el cual cuenta a su vez, con 4 objetivos o categorías: óptimos (cobertura), de costo acotado (cobertura), satisfactorios (calidad del plan) y ágiles (tiempo de resolución). También, se detalla el procedimiento para el uso de cada uno de los planificadores, desde el proceso de descarga hasta la instalación y ejecución de estos en un entorno de máquina virtual para contenedores o *dockers* (*Singularity)* desde un sistema operativo nativo (*Linux*) o desde el uso de un subsistema como *Windows Subsystem for Linux* (*WSL*), todo esto para la resolución de un problema descrito en lenguaje *PDDL*.

* Correspondencia de los autores:
david.toscano240@comunidadunir.net (D. Toscano),
edisonjose.giraldo261@comunidadunir.net (E. Giraldo),
joel.orellana102@comunidadunir.net (J. Orellana),
rubendario.aponte552@comunidadunir.net (R. D. Aponte)

II. Métodos

Los 4 planificadores escogidos para análisis fueron principalmente los mejores en cada una de las categorías (ver Tabla I). A continuación, se describirá brevemente el funcionamiento interno de cada uno de ellos.

Tabla I. Equipos planificadores analizados en el Classical IPC 2018.

Nombre	Número	Categoría
Fast Downward Stone Soup 2018	45	*Satisficing Track*
Delfi 1	23	*Optimal Track*
LAPKT-BFWS-Preference	47	*Agile Track*
Complementary1	9	*Optimal Track*

A. Fast Downward Stone Soup 2018

El planificador *Fast Downward Stone Soup 2018* (Seipp & Röger, 2018) es una colección de planificadores basados en el planificador *Fast Downward Planning System* (Helmert, 2006).

Fast Downward es un sistema de planificación clásico basado en las ideas de búsqueda heurística hacia adelante y descomposición jerárquica de problemas. Puede tratar con la gama completa de PDDL2.2 proposicionales, es decir, además de STRIPS, admite fórmulas arbitrarias en las precondiciones y condiciones en la meta, y puede tratar con efectos condicionales y universalmente cuantificados y predicados derivados (axiomas). El *Fast Downward* resuelve tareas de planificación en 3 etapas: *traducción:* Traducir el PDDL para poder manejarlo mejor en estructuras jerárquicas,

recopilación de conocimiento: preparar el escenario para los algoritmos de búsqueda al compilar la información crítica sobre la tarea de planificación en una serie de estructuras de datos para un acceso eficiente; finalmente, *búsqueda:* implementa tres algoritmos de búsqueda diferentes para ejecutar la planificación (*Greedy Best-First Search, Multi-heuristic Best-First Search* y *Focused Iterative-broadening Search*).

Fast Downward Stone Soup 2018 es esencialmente el mismo planificador presentado en los IPC 2011 y 2014 (Helmert et al., 2011). Para el IPC 2018 se presentaron dos variantes, una para el *Optimal Track* y otra para el *Satisficing Track,* donde la segunda fue ganadora.

La variante *Satisficing Track* puede mejorar la calidad de su solución generada con el tiempo. Aquí, la única información que se comunica entre los solucionadores de componentes es la calidad de la mejor solución encontrada hasta el momento, de modo que los solucionadores posteriores de la secuencia puedan eliminar los estados cuyo "costo hasta ahora" ya sea tan grande o mayor que el costo de la mejor solución que se generó previamente.

El término "*Stone Soup"* viene de la analogía de una historia que en resumen, se prepara una "sopa de piedras" y cada vez que se le agregan ingredientes, esta va mejorando su sabor hasta que al final se le retiran las piedras y produce una "deliciosa" sopa. En el caso del planificador, un conjunto de planificadores sirven como "ingredientes" de la colección o portafolio de planificadores (un total 144 de 3 equipos diferentes y otras variantes que usaron algoritmos basados en *Fast Downward* de la IPC 2014 para *Satisficing Track*); como instancias de entrenamiento, se utilizó una muestra de 2115 instancias de las IPC anteriores que eran soportadas por la colección de planificadores y finalmente se contaba con una evaluación de resultados (tiempo de ejecución, costo del plan) de cada planificador para cada instancia.

B. Delfi 1

Delfi de *DEap Learning PortFollos* (Katz et al., 2018) es una colección de planificadores sometido a las pistas optimas (*Optimal Tracks*) del IPC 2018. Consiste en: (a) una colección de planificadores de coste óptimo basados en *Fast Downward Planning System* (Helmert, 2006), y (b) un módulo que, dada una tarea de planificación, selecciona al planificador de la colección para la cual la confianza de que resuelve la tarea de planificación dada es más alta.

La selección online del planificador *Delfi* se basa en un modelo que predice para todos los planificadores de la colección si resuelven una tarea de planificación determinada dentro de los límites fijos de recursos y tiempo de la competencia IPC o no. Para aprender un modelo de este tipo, se crea una colección de tareas para servir como conjunto de entrenamiento, y se ejecutan todos los planificadores de una colección para encontrar si resuelven la tarea o no, y usamos los datos resultantes para entrenar una red neuronal profunda (Redes Neuronales Convolucionales).

En particular, para representar las tareas de planificación, se utilizan dos representaciones gráficas para planificar tareas, se convierten en imágenes y se utiliza una red neuronal convolucional para aprender un modelo que predice si un planificador resuelve una tarea determinada o no. El desempeño de *Delfi 1*, ocupando el primer lugar en la competencia, mostró que el modelo aprendido se generalizó bien a los nuevos puntos de referencia utilizados en la competencia. Mientras que *Delfi 2* terminó 7 de las 16 presentaciones, *Delfi 1* ocupó el primer lugar.

C. LAPKT-BFWS-Preference

Los algoritmos *Width-based Search* (*WS*) son algoritmos de búsqueda que se basan en la *novedad* de un estado. Esta se mide por el tamaño del conjunto más pequeño de *átomos* proposicionales que lo hacen único en la búsqueda. Este algoritmo es eficiente cuando el objetivo es una solo átomo, pero no funciona bien cuando el objetivo es una conjunción de células. En este caso, se requieren estrategias más sofisticadas, como la búsqueda *Best-First Width Search* (*BFWS*).

(Lipovetzky & Geffner, 2017) han demostrado recientemente que es posible lograr un rendimiento de vanguardia en los estándares de planificación clásica utilizando una combinación de la exploración proporcionada por medidas estructurales de *anchura* y la explotación ofrecida por los métodos de búsqueda heurística tradicionales. *BFWS* es una búsqueda en primer lugar que utiliza una definición extendida de la *novedad* de un nodo de búsqueda como criterio principal para priorizar los nodos en la lista abierta. La novedad de un estado se define como el tamaño del conjunto más pequeño de átomos para los cuales el estado es el primero en ser encontrado en la búsqueda con todos los átomos verdaderos al mismo tiempo. El mejor planificador BFWS descrito en (Lipovetzky & Geffner, 2017) es BFWS(f_5), que utiliza una medida de novedad basada en la cantidad de objetivos atómicos no verdaderos en el estado y en una aproximación dependiente del camino hacia el objetivo.

Para el IPC 2018, un equipo presentó 6 algoritmos basados en *BFWS*(f_5) (Frances et al., 2018) haciendo uso de la plataforma *LAPKT* (*LAPKT*, 2017), donde el ultimo algoritmo fue el ganador en la categoría Ágil (*Agile Track*), denominado *BFWS-preference*.

El planificador *BFWS-preference* es escecialmente un *BFWS*(f_5) pero con una diferencia, una vez se encuentra una solución, el costo del plan se da como un límite superior para una búsqueda ponderada *A** (*WA**), que luego se ejecuta para optimizar la calidad de la solución hasta que se alcanza el tiempo de ejecución máximo.

D. Complementary1

Este planificador es una versión actualizada y significativamente modificada de la heurística (CPC) presentada por (Franco et al., 2017).

El planificador *Complementary1* utiliza bases de datos de patrones (*PDB,* por sus siglas en inglés). Un PDB es una función heurística en forma de tabla de búsqueda que contiene los costos de solución óptimos de una versión simplificada de una tarea. En este planificador se usa un método que crea dinámicamente múltiples PDB que luego se combinan en una sola función heurística. En una iteración dada, este método usa estimaciones del tamaño del espacio de búsqueda de A* para crear un PDB que complementa las fortalezas de los PDB creados en iteraciones anteriores.

La mayor diferencia con (Franco et al., 2017) es que el método original siempre comenzaba con PDB más pequeños y usaba límites de tiempo a priori para aumentar secuencialmente el límite de tamaño del PDB, mientras que el nuevo método no tiene ese sesgo inicial. *Complementary1* utiliza el algoritmo UCB1 para saber qué intervalo de tamaño de PDB se ajusta mejor al problema actual dados los PDB seleccionados previamente.

El *Complementary1* terminó finalista en la categoría *Optimal Track* del IPC 2018.

III. Materiales

Para probar los planificadores anteriormente explicados, se debe contar previamente con:

- Sistema operativo *Linux* o un Subsistema *Linux* para *Windows* (*WSL*) instalado.
- *Software Singularity* (creador de contenedores) instalado.
- Código fuente para la creación de los contenedores de los planificadores en *Singularity.*

La instalación y configuración de cada uno, se describe a continuación.

A. Sistema operativo Linux y WSL

Se recomienda el uso del sistema operativo *Linux Ubuntu* para una instalación nativa de este o de su instalación en *WSL*. La elección radica básicamente en su popularidad y en las recomendaciones realizadas en la IPC 2018. La instalación nativa de *Ubuntu* no se explica en este trabajo. Por otro lado, la instalación de *WSL* en *Windows* se considera en este trabajo dado que *Windows* es el sistema operativo más usado.

Brevemente, *WSL* es una función de *Windows* que permite a los usuarios ejecutar un entorno *Linux* sin necesidad de una máquina virtual separada o arranque dual. Los pasos para su instalación fueron tomados de (Microsoft, 2022) y se describen a continuación:

Paso 1: Habilitación del Subsistema de Windows para Linux

Ejecutar en la terminal PowerShell:

```
dism.exe /online /enable-feature
/featurename:Microsoft-Windows-Subsystem-Linux
/all /norestart
```

Paso 2: comprobación de los requisitos para ejecutar WSL 2

Para sistemas x64 debe tenerse *Windows* 10 versión 1903 y compilación 18362 o superiores. En caso contrario, se debe actualizar.

Paso 3: Habilitación de la característica Máquina virtual

Ejecutar en la terminal PowerShell y posteriormente reiniciar:

```
dism.exe /online /enable-feature
/featurename:VirtualMachinePlatform /all
/norestart
```

Paso 4: Descarga del paquete de actualización del kernel de Linux

Descargar e instalar el siguiente archivo e instalarlo:

https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

Paso 5: Definición de WSL 2 como versión predeterminada

Ejecutar en la terminal PowerShell:

```
wsl --set-default-version 2
```

Paso 6: Instalación de la distribución de Linux que quiera

Descargar e instalar de la tienda de *Windows* una distribución de *Linux* (en este caso, *Ubuntu*) y de acuerdo a la versión que se prefiera:

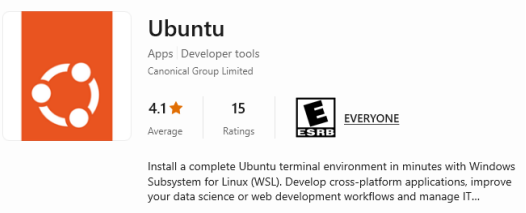

Figura 1. Ubuntu de la tienda de Microsoft.

Paso 7: Abrir WSL con la versión instalada de Ubuntu

Iniciar WSL con Ubuntu. La primera vez, solicitará crear usuario y contraseña.

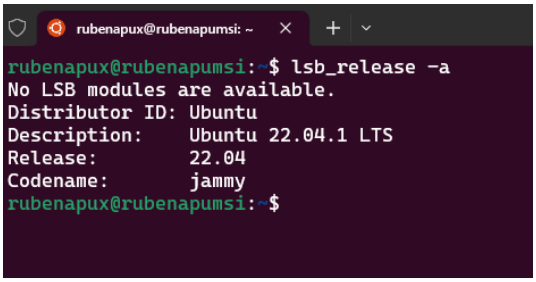

Figura 2. Ubuntu 22.04.1 LTS instalado en WSL.

B. Instalación de Singularity

Singularity (Sylabs Inc., 2017) es una plataforma que permite crear y correr contenedores. Un contenedor es una unidad estándar de software que empaqueta código y todas sus dependencias para que cualquier aplicación dentro de este, se ejecute de manera rápida y confiable de un entorno informático a otro. El proceso de instalación fue tomado de (Boyles, 2021) y se describe a continuación:

Paso 1: Instalación de dependencias

En una terminal de *Linux* (nativo o *WSL*), Actualizar repositorios:

```
sudo apt-get update
```

Instalación de paquetes dependientes:

```
sudo apt-get install -y build-essential
libssl-dev uuid-dev libgpgme11-dev squashfs-
tools libseccomp-dev pkg-config
```

Paso 2: Instalación de Go

Singularity está escrito en *Go*, de manera que, se debe instalar este. Actualmente la versión más reciente de *Go* es la 1.20:

```
export VERSION=1.20 OS=linux ARCH=amd64
wget https://dl.google.com/go/go$VERSION.$OS-
$ARCH.tar.gz
sudo tar -C /usr/local -xzvf go$VERSION.$OS-
$ARCH.tar.gz
rm go$VERSION.$OS-$ARCH.tar.gz
```

Añadir *Go* al *PATH*:

```
echo 'export PATH=/usr/local/go/bin:$PATH'
>> ~/.bashrc && source ~/.bashrc
```

Se puede comprobar la instalación mediante el comando:

```
go version
```

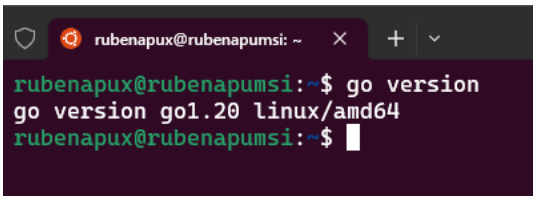

Figura 3. Go instalado.

Paso 3: Instalar Singularity

Descargar *Singularity*. Actualmente la versión más reciente de *Singularity* (*old version*) es la 3.7.3:

```
export VERSION=3.7.3
wget
https://github.com/sylabs/singularity/releases
/download/v${VERSION}/singularity-
${VERSION}.tar.gz
tar -xzf singularity-${VERSION}.tar.gz
cd singularity
```

Instalar *Singularity*:

```
  ./mconfig
make -C builddir
sudo make -C builddir install
```

Se puede comprobar la instalación mediante el comando:

```
  singularity version
```

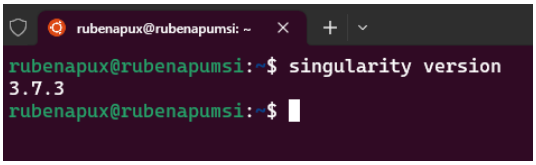

Figura 4. Singularity instalado.

Paso 4: Probar Singularity

Una forma de comprobar la correcta instalación de *Singularity* es crear un contenedor de prueba *"lolcow"*:

```
  singularity pull docker://godlovedc/lolcow
```

y posteriormente correr el contenedor "*lolcow*":

```
  singularity run lolcow_latest.sif
```

Si se obtiene en el *prompt* una vaca hablando, se instaló y configuró bien el contenedor:

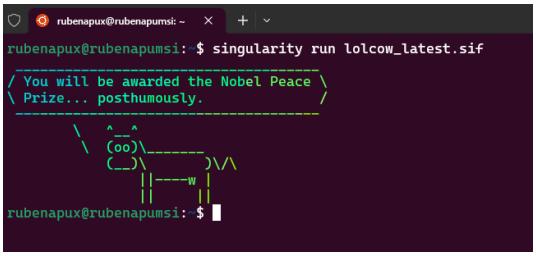

Figura 5. Prueba de instalación de Singularity con lolcow.

C. Instalación de un planificador en Singularity

Para probar la instalación de los planificadores descritos anteriormente, se deben seguir los siguientes pasos:

Paso 1: Ingresar al IPC 2018

Se debe ingresar al *International Planning Competition* (*IPC*) 2018 *Classical Tracks*:

https://ipc2018-classical.bitbucket.io/

Figura 6. Portal web del IPC 2018.

Paso 2: Escoger un planificador

En el apartado *Planners* Escoger un planificador entre los diferentes *Tracks*: *Optimal Tracks, Satisficing Tracks, Agile Tracks* y *Cost-Bounded Tracks*. Como ejemplo, se toma el planificador "*Fast Downward Stone Soup 2018*":

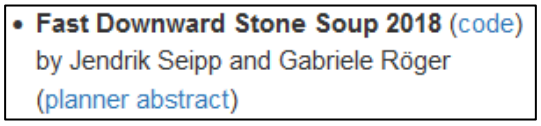

Figura 7. Vínculos para obtener el resumen y código fuente del planificador Fast Downward Stone Soup 2018.

Artículo o resumen del planificador escogido:

https://ipc2018-classical.bitbucket.io/planner-abstracts/team45.pdf

El código fuente del planificador se puede obtener en:

https://bitbucket.org/ipc2018-classical/team45/src/ipc-2018-seq-sat/

Paso 3: Descargar código fuente para crear el contenedor en Singularity

En una terminal de *Linux* (nativo o en WSL) se baja el código fuente del planificador. Cabe resaltar que para distintos planificadores, se debe cambiar en la ruta del enlace:

- En "teamXX" las "XX" por el número del equipo.
- "src" por "raw".
- En "ipc-2018-seq-xxx" las "xxx" por "sat", "opt" o "cbo" de acuerdo con la categoría.

Para el caso de "Fast Downward Stone Soup 2018":

```
  wget https://bitbucket.org/ipc2018-classical/team45/raw/ipc-2018-seq-sat/Singularity
```

Paso 4: Crear el contenedor con el planificador

Se crea un contenedor en una imagen *planner.img* con el planificador seleccionado a partir del código fuente nombrado *Singularity*:

```
  sudo singularity build planner.img Singularity
```

Esperar el empaquetado de la imagen. Dependiendo del planificador y de la conexión a internet, esto puede tomar algunos minutos. Una vez terminado el proceso, se crea la imagen *planner.img* y ya no se requiere volver a ejecutar el anterior comando.

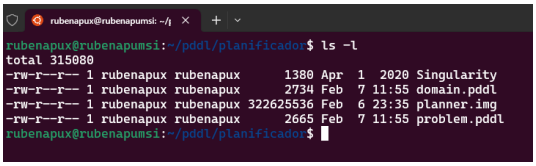

Figura 8. Creación del planificador en el contenedor planner.img.

Paso 5: Ejecutar el planificador

Para este ejercicio, se tomó el problema PDDL de (Yarox, 2012) de manera que, se pudiera comprobar el correcto funcionamiento del planificador en el contenedor. Inspirado en los problemas de los robots exploradores planetarios, este dominio requiere que una colección de robots navegue por la superficie de un planeta, encuentre muestras, tome imágenes y las recopile en un módulo de aterrizaje. En la versión STRIPS, los robots pueden transportar una muestra a la vez. Deben dejarlas en un lugar seguro antes de poder tomar la siguiente. Se optó por este dominio y problema, ya que no requería de definición de requerimientos en el dominio, lo que lo hace más compatible con las diferentes versiones de *PDDL*.

Para correr el planificador del contenedor, se deben ejecutar las siguientes líneas de comando en la terminal, contando previamente

con los siguientes archivos en la misma carpeta:

- *domain.pddl*
- *problem.pddl*
- *Singularity*
- *planner.img*

```
  RUNDIR="$(pwd)"
DOMAIN="$RUNDIR/domain.pddl"
PROBLEM="$RUNDIR/problem.pddl"
PLANFILE="$RUNDIR/sas_plan"
COSTBOUND=42 # Pars cost-bounded track
ulimit -t 1800 # Por defecto es 1800 (30 min).
ulimit -v 8388608
singularity run -C -H $RUNDIR planner.img
$DOMAIN $PROBLEM $PLANFILE $COSTBOUND
```

Los resultados se mostrarán en consola y se guardarán en los archivos "*sas_plan.x*" donde *x* es el plan *x* generado.

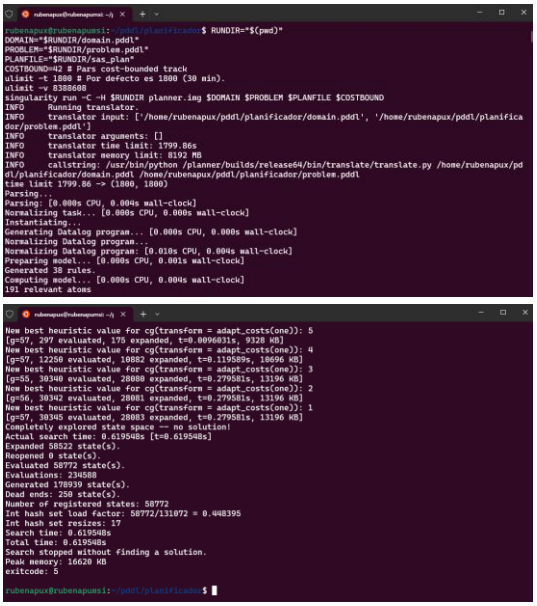

Figura 9. Proceso de búsqueda del planificador en el contenedor para un dominio y problema PDDL dado.

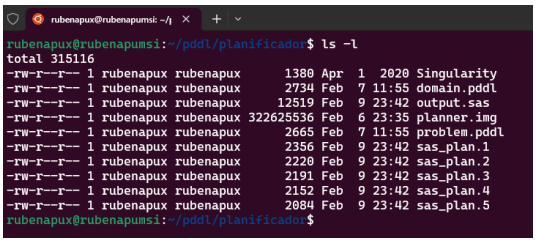

Figura 10. Planes creados por el planificador (5 en este caso).

IV. RESULTADOS

Después de aplicar el procedimiento de descarga, instalación y ejecución para cada uno de los 4 planificadores, siguiendo los procedimientos anteriores, se pudieron obtener resultados satisfactorios para cada uno de ellos.

En la Se puede observar que el planificador *Fast Downward Stone Soup 2018* creó 5 planes, donde en cada iteración, el costo iba disminuyendo hasta llegar al mejor costo de 58. Sin embargo, el tiempo de búsqueda y los nodos expandidos no necesariamente disminuyeron.

Tabla II se muestran los resultados obtenidos para los 4 planificadores, donde se evaluaron las siguientes métricas: costo del plan, tiempo de búsqueda y nodos expandidos. También, se evaluó el tiempo de ejecución total de cada uno de los planificadores (ver Tabla III). Cabe destacar que, los planificadores generaron distintos números de planes.

Se puede observar que el planificador *Fast Downward Stone Soup 2018* creó 5 planes, donde en cada iteración, el costo iba disminuyendo hasta llegar al mejor costo de 58. Sin embargo, el tiempo de búsqueda y los nodos expandidos no necesariamente disminuyeron.

Tabla II. Resumen de estadísticas de los diferentes planificadores.

Planificador	Plan	Costo	Tiempo de búsqueda (ms)	Nodos expandidos
Fast Downward Stone Soup 2018	1	66	9,45	107
	2	62	90,01	4099
	3	61	89,95	7554
	4	60	9,99	185
	5	58	80,01	7510
Delfi 1	1	58	8,31	59
LAPKT-BFWS-Preference	1	68	5,92	222
	2	62	14,07	152
	3	60	550,00	520
	4	58	560,05	27312
Complementary1	1	58	0,00	59

Para el planificador *Delfi 1* solo se generó un solo plan y tiene en general buenos resultados, con costo de 58.

Con respecto al planificador *LAPKT-BFWS-Preference* se generaron 4 planes, donde el costo iba disminuyendo, pero el tiempo de búsqueda y la cantidad de nodos expandidos en general, aumentaron. El mejor plan tuvo un costo de 58.

El planificador *Complementary1* solo generó un plan pero con las mejores métricas entre los 4 planificadores, con costo de 58, tiempo de búsqueda casi 0 ms y nodos expandidos 59.

Finalmente, con respecto a los tiempos de ejecución de cada uno de los planificadores, se observa que el *Complementary1* fue el más rápido, con un tiempo de ejecución inferior a 1 segundo.

Tabla III. Tiempo de ejecución de los planificadores.

Planficador	Tiempo de ejecución (s)
Fast Downward Stone Soup 2018	41,75
Delfi 1	6,98
LAPKT-BFWS-Preference	1,74
Complementary1	0,73

V. CONCLUSIONES

Se pudo revisar el funcionamiento interno de 4 planificadores del estado del arte para la competición *IPC* 2018. Cabe destacar que de los escogidos, la mayoría se basaban en el algoritmo *Fast Downward*.

El planificador *Fast Downward Stone Soup 2018* aparentemente utiliza meta-heurísticas para la extensión de las tareas y acciones aplicado a la división jerárquica de tareas, esto a su vez coincide con los tiempos de ejecución mostrados en resultados, ya que al parecer mantiene una filosofía de cambiar de heurística antes que realizar una búsqueda más y más profunda/amplia come el caso de *LAPKT* cuyo costo computacional crece exponencialmente.

Complementary1 es una muestra de una implementación creativa y disruptiva, evidentemente crear una versión simplificada del problema para iniciar gradualmente la elección de una heurística es computacionalmente eficiente, sin embargo esto no garantiza la

optimalidad y posiblemente si la gama de heurísticas propuestas no son las adecuadas podría generar resultados inferiores a otros algoritmos.

Está claro que crear un planificador de orden general es una tarea difícil, ya que la heurística, o meta-heurística, puede ser mejorada cuanto más específica sea una tarea, sin embargo, *Complementary1* realizó en general exitosamente las diferentes pruebas del concurso demostrando su versatilidad, además no solo se puede tener una "colección de heurísticas" sino también una "colección de planificadores" y una respectiva función de "confianza" que recibe el problema y el planificador y evalúa la fiabilidad con la que podría resolver el problema como es el caso de *Delfi1*, lo que señala que la planificación es una tarea que se beneficia enormemente de la meta-modelización siempre y cuando se contemplen varios casos de planificación.

Se logró seguir satisfactoriamente el procedimiento para instalar el *WSL* en el sistema operativo *Windows* 10. Sobre este, se pudo instalar *Singularity* también de manera satisfactoria.

Finalmente, la instalación, configuración y ejecución de los 4 planificadores desde *WSL* y *Singularity* se logró llevar a cabo de manera satisfactoria y se pudo probar un dominio de *PDDL* arrojando resultados para cada uno de estos y poder compararlos.

El planificador *Complementary1* fue el que presentó mayor rendimiento general para el dominio *PDDL* de prueba.

Referencias

(*ICAPS – International Conference on Automated Planning and Scheduling*, s. f.)(*IPC 2018*, s. f.)

Boyles, F. (2021). *Using Singularity on Windows with WSL2 | Oxford Protein Informatics Group*. https://www.blopig.com/blog/2021/09/using-singularity-on-windows-with-wsl2/

Frances, G., Geffner, H., Lipovetzky, N., & Ramirez, M. (2018). Best-first width search in the IPC 2018: Complete, simulated, and polynomial variants. *IPC2018–Classical Tracks*, 22-26.

Franco, S., Torralba, A., Lelis, L. H. S., & Barley, M. (2017). On creating complementary pattern databases. *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 4302-4309.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191-246.

Helmert, M., Röger, G., & Karpas, E. (2011). Fast downward stone soup: A baseline for building planner portfolios. *ICAPS 2011 Workshop on Planning and Learning*, *2835*.

ICAPS – International Conference on Automated Planning and Scheduling. (s. f.). Recuperado 7 de febrero de 2023, de https://www.icaps-conference.org/

IPC 2018. (s. f.). Recuperado 7 de febrero de 2023, de https://ipc2018-classical.bitbucket.io/

Katz, M., Sohrabi, S., Samulowitz, H., & Sievers, S. (2018). Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts*, 57-64.

LAPKT. (2017). https://lapkt-dev.github.io/docs/

Lipovetzky, N., & Geffner, H. (2017). Best-first width search: Exploration and exploitation in classical planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *31*(1).

Microsoft. (2022). *Pasos de instalación manual para versiones anteriores de WSL | Microsoft Learn*. https://learn.microsoft.com/es-es/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package

Seipp, J., & Röger, G. (2018). *Fast Downward Stone Soup 2018 (planner abstract)*. https://edoc.unibas.ch/68759/

Sylabs Inc. (2017). *User Guide — Singularity container 3.5 documentation*. https://docs.sylabs.io/guides/3.5/user-guide/

Yarox. (2012). *GitHub - yarox/pddl-examples: pddl examples including strips, numeric, and time domains.* https://github.com/yarox/pddl-examples