Desarrollo de un problema de planificación con PDDL y el planner ganador de la IPC2018

Gianmarco Doza Caballero

[Nombre de la institución]

Razonamiento y Planificación Automática

[Nombre del docente]

29 de agosto de 2026

1. Introducción

La **planificación automática** es el área de la inteligencia artificial que estudia cómo generar secuencias de acciones (planes) que permiten alcanzar un objetivo a partir de un estado inicial, dado un modelo del dominio. Su lenguaje estándar es **PDDL** (Planning Domain Definition Language) (McDermott et al., 1998), que separa el **dominio** (tipos, predicados y acciones del mundo) del **problema** (objetos, estado inicial y meta).

La **International Conference on Automated Planning and Scheduling (ICAPS)** organiza la International Planning Competition (**IPC**), la competencia más importante de la disciplina (ICAPS, s. f.). En este trabajo se revisa la edición **IPC2018 (Classical Tracks)**, se identifica al planner ganador del optimal track, se ejecuta con él la tarea **Snake (problema 1)** usando **Singularity**, se transcribe a PDDL un problema de un **robot rover** y se propone un **escenario diferente** con más minerales, localidades, restricciones y laboratorios.

2. Revisión de la IPC2018 — Classical Tracks

La página oficial de la competencia (IPC2018, s. f.) organiza la información en las siguientes secciones de interés:

- **Tracks.** Cuatro modalidades: **optimal** (un núcleo, 8 GB de memoria, 30 min por tarea; los planes deben ser óptimos y la puntuación es el número de tareas resueltas), **bounded-cost** (planes dentro de un límite de coste), **satisficing** (calidad del plan: ratio C*/C) y **agile** (tiempo de resolución). El track multi-core fue cancelado.

- **PDDL Fragment.** Se usó un subconjunto de **PDDL 3.1**: STRIPS, costes de acción, precondiciones negativas y efectos condicionales (Fox & Long, 2003; compatible con IPC 2011/2014).
- **Domains.** Once dominios: Agricola, Caldera, Data Network, Nurikabe, Organic Synthesis, Petri Net Alignment, Settlers, **Snake**, Spider y Termes, entre otros. Los archivos PDDL, instancias y planes de referencia están en el repositorio oficial `ipc2018-classical/domains`.
- **Planners (optimal track).** Participaron 35 planners (73 entradas en total). Todos enviaron un archivo **Singularity** que define un contenedor reproducible con su planner compilado.
- **Results (presentation slides).** Según las diapositivas oficiales de resultados (IPC2018, 2018a) y la tabla de puntuaciones (IPC2018, 2018b), el **ganador del optimal track fue "Delfi 1"** con **126 tareas resueltas** (de 200), seguido de Complementary1/ Complementary2 (124) y Planning-PDBs (122). Delfi1 fue desarrollado por **Michael Katz, Shirin Sohrabi, Horst Samulowitz y Silvan Sievers** (Katz et al., 2018) y usa un selector de planners aprendido con una red neuronal que elige el mejor algoritmo para cada tarea a partir del grafo abstracto del problema.

Tabla 1. Resultados del optimal track (cobertura sobre 200 tareas)

Planner	Agricola	Caldera	Data Net.	Nurikabe	Org. Syn.	P.N. Align.	Settlers	Snake	Spider	Te
Delfi1	12	13	13	12	13	20	9	11	11	12
Complementary2	6	12	12	12	13	18	9	14	12	10
Complementary1	10	11	14	13	13	17	8	11	11	10

Planner	Agricola	Caldera	Data Net.	Nurikabe	Org. Syn.	P.N. Align.	Settlers	Snake	Spider	Te
Planning-PDBs	6	12	14	11	13	18	8	13	11	1(

Fuente: https://ipc2018-classical.bitbucket.io/scores.html

3. Instalación de Singularity

Singularity (hoy Apptainer / SingularityCE) es un sistema de contenedores diseñado para cómputo científico y HPC (Sylabs, s. f.). La IPC2018 lo usó para garantizar la reproducibilidad de los planners: cada equipo entregó un archivo `Singularity` que compila su planner dentro de un contenedor, de modo que la ejecución no depende de la máquina anfitriona.

En este trabajo se instaló en **Ubuntu 24.04 LTS** mediante el paquete oficial del repositorio (`singularity-container` v4.1.1, compatible con los archivos Singularity de la IPC2018, que fueron actualizados para la versión 3.5):

```
sudo apt update
sudo apt install -y singularity-container
singularity --version
# singularity-ce version 4.1.1
```

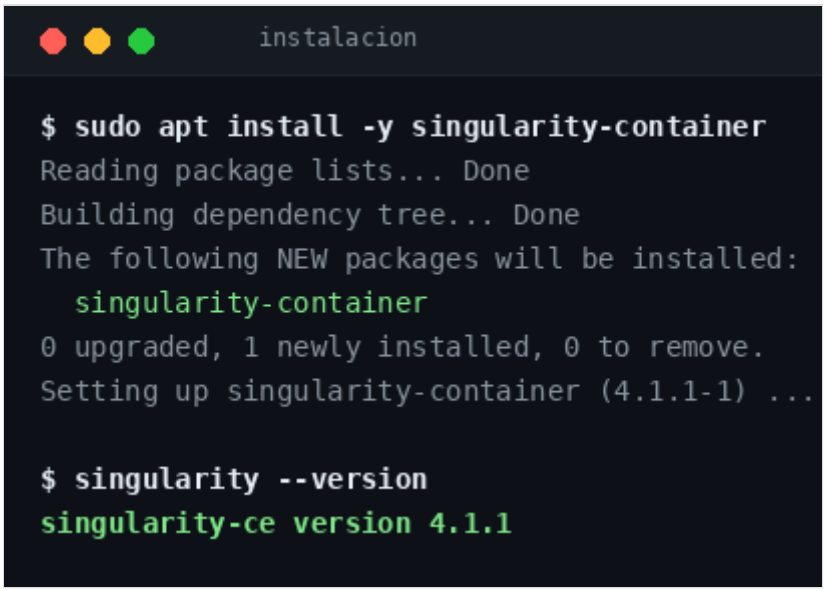

Figura 1. Instalación de Singularity y verificación de la versión (4.1.1).

4. Construcción del contenedor del planner ganador (Delfi1)

Con Singularity instalado, se descargó el archivo `Singularity` del repositorio oficial del equipo ganador y se construyó la imagen:

```
        wget https://bitbucket.org/ipc2018-classical/team23/raw/ipc-2018-seq-opt/
Singularity
        sudo singularity build planner.img Singularity
```

La receta de Delfi1 (`Bootstrap: docker`, base `ubuntu:xenial`) instala las dependencias de Python (TensorFlow 1.8, Keras 2.1.6), compila **Fast Downward** y **SymBA*** (los motores de búsqueda que el selector neuronal combina) y define como runscript la invocación `/planner/plan-ipc.py <domain> <problem> <planfile> --image-from-lifted-task`. El resultado fue la imagen `planner.img` (584 MB).

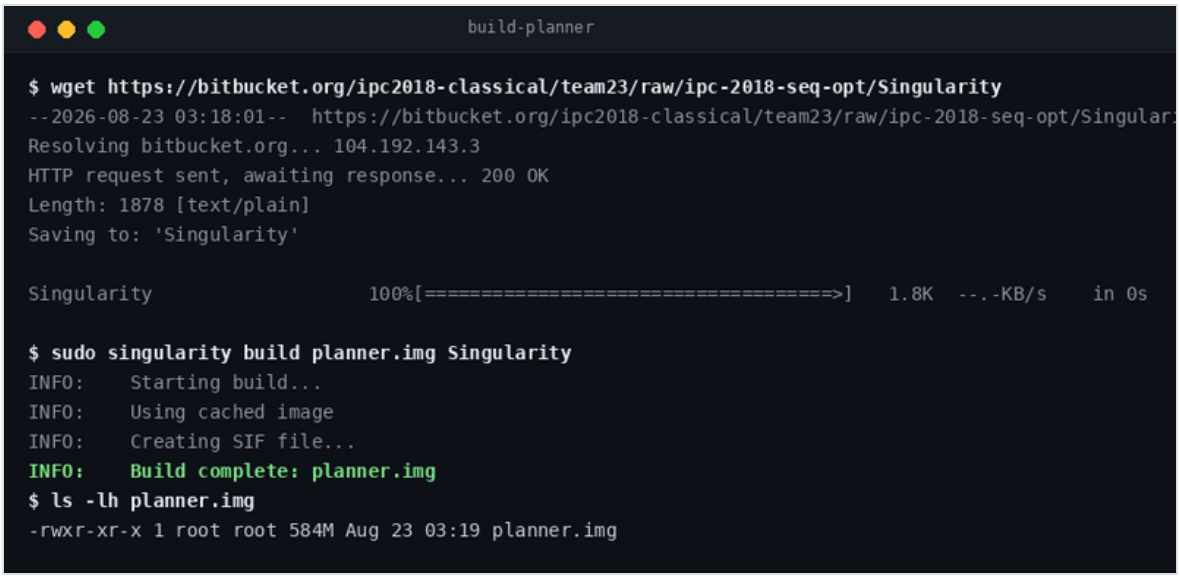

Figura 2. Descarga de la receta Singularity y construcción de la imagen *planner.img*.

5. Ejecución de la tarea Snake (problema 1) con el script oficial

Siguiendo la sección **“How can I test my containers?”** de DETAILS ON SINGULARITY (adaptada al optimal track, que recibe 3 argumentos), se ejecutó el problema 1 de Snake:

```
        mkdir rundir
        cp domain.pddl rundir
        cp p01.pddl rundir/problem.pddl
        RUNDIR="$(pwd)/rundir"
        DOMAIN="$RUNDIR/domain.pddl"
        PROBLEM="$RUNDIR/problem.pddl"
        PLANFILE="$RUNDIR/sas_plan"
        ulimit -t 1800        # límite de 30 min (regla del optimal track)

        ulimit -v 8388608     # límite de 8 GB (regla del optimal track)
        singularity run -C -H $RUNDIR planner.img $DOMAIN $PROBLEM $PLANFILE
```

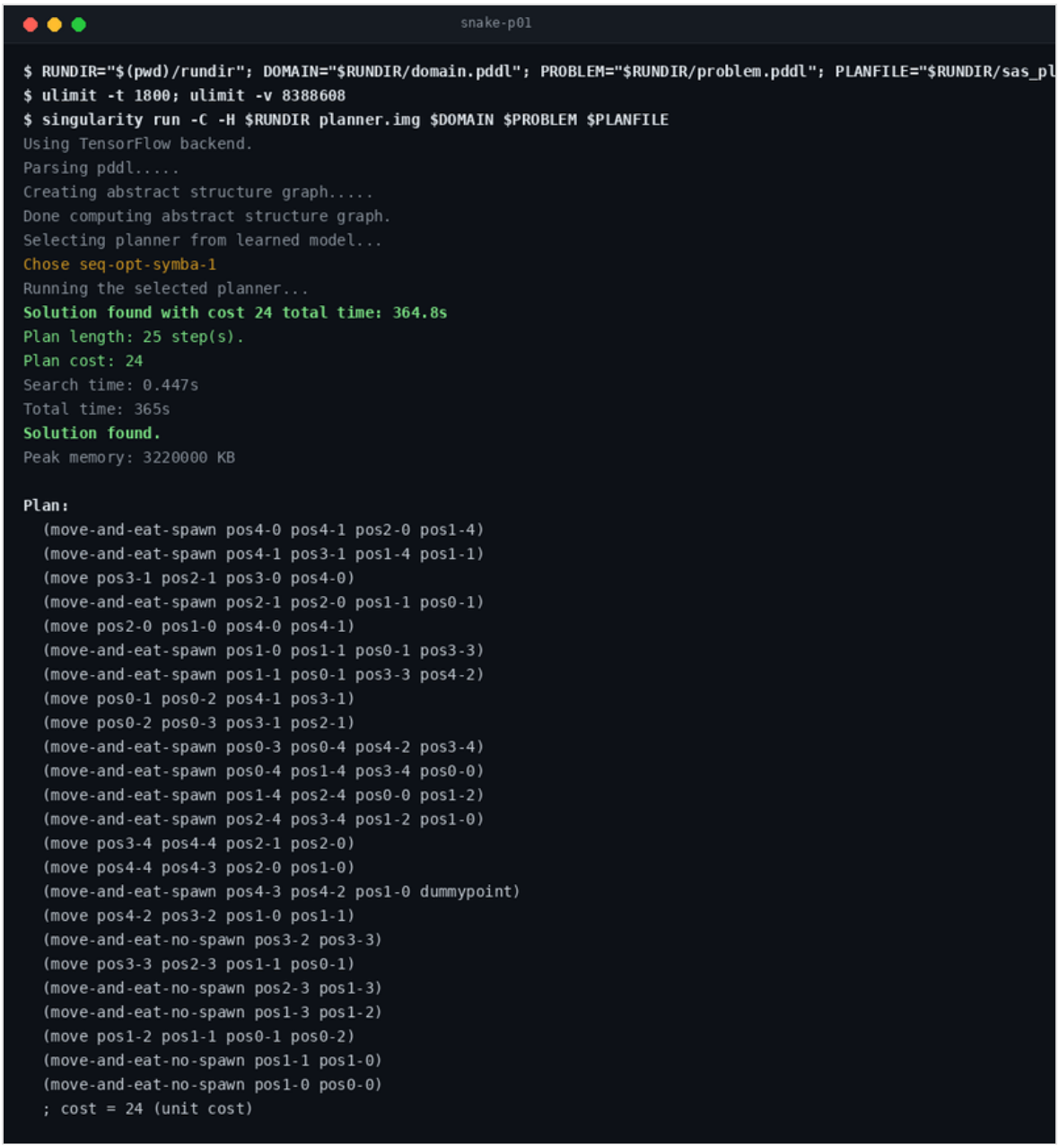

Figura 3. Ejecución de la tarea Snake (problema 1) con Delfi1: plan óptimo de 24 pasos.

Resultado obtenido:

```
Plan length: 24 step(s).
Plan cost: 24
Search time: 0.447s
Total time: 365s
Solution found.
```

Delfi1 resolvió la tarea con un **plan óptimo de 24 pasos** (costo 24). El tiempo total (≈6 min) corresponde principalmente a la traducción del

problema y a la generación del grafo abstracto y la predicción del selector neuronal; la búsqueda propiamente dicha tomó menos de medio segundo. El plan completo quedó guardado en `sas_plan`:

```
(move-and-eat-spawn pos4-0 pos4-1 pos2-0 pos1-4)   ; la serpiente avanza
y come el punto de pos4-1
(move-and-eat-spawn pos4-1 pos3-1 pos1-4 pos1-1)   ; ... (24 acciones en
total)
...
Plan length: 24 step(s).
```

6. Análisis de los archivos de la tarea Snake

`domain.pddl` (92 líneas). Declara `(:requirements :strips :negative-preconditions)` y define:

- **Predicados:** `ISADJACENT` (adyacencia entre casillas), `headsnake`/`tailsnake` (cabeza/cola), `nextsnake` (conexión de segmentos), `blocked` (casilla ocupada), `spawn`/`NEXTSPAWN` (cadena de aparición de puntos), `ispoint` (casilla con punto), y la constante `dummypoint` (centinela para indicar que ya no quedan puntos por aparecer).
- **Acciones (3):**
  1. `move`: avanza la serpiente a una casilla adyacente libre **sin** punto; la cola se libera.
  2. `move-and-eat-spawn`: avanza a una casilla **con punto**; la serpiente crece (no se libera cola) y se activa el siguiente punto de la cadena `NEXTSPAWN`.
  3. `move-and-eat-no-spawn`: igual que la anterior pero cuando ya no quedan puntos por aparecer (`spawn dummypoint`).
- **Meta:** comer todos los puntos (`not (ispoint ...)` para cada punto inicial y cada punto de la cadena de spawn).

`p01.pddl` (127 líneas). Instancia `snake-empty-5x5-1-5-10-11170`: tablero **5×5** (25 casillas `posX-Y`), serpiente inicial de 2 segmentos (`headsnake`

`pos4-0`, `tailsnake pos3-0`), **5 puntos iniciales** (`ispoint`) y una **cadena de 10 puntos adicionales** definida con `spawn pos2-0` y `NEXTSPAWN`. El objetivo es consumir los 15 puntos (IPC2018, s. f.).

7. Transcripción a PDDL del problema del Rover

Enunciado: un robot tipo rover excavó dos rocas en las **localidades 1 y 2**; por mal tiempo no pudo trasladarlas. Se debe generar el plan para llevar los minerales al **laboratorio de análisis**, con las siguientes restricciones de trayectoria (Figura 1):

- Localidad 3 ↔ Localidad 1: camino **bidireccional**.
- Localidad 3 → Localidad 2: **una sola dirección**.
- Localidad 2 → Localidad 4: **una sola dirección**.
- Localidad 3 ↔ Localidad 4 y Localidad 4 ↔ Localidad 5: **bidireccionales**.

El laboratorio se ubica en la **localidad 5** (extremo del grafo según el esquema).

7.1 Modelado del dominio (`domain.pddl`)

Se diseñó el dominio `rover-minerales` con el fragmento PDDL 3.1 de la IPC2018 (`:strips :typing :negative-preconditions :action-costs`):

- **Tipos:** `rover`, `localidad`, `mineral`.
- **Predicados:** `(en ?r ?l)` posición del rover; `(camino ?a ?b)` camino directo y **direccional** de `a` a `b` (los caminos bidireccionales se declaran en ambos sentidos; los de una vía, en un solo sentido); `(mineral-en ?m ?l)` ubicación de cada mineral; `(cargando ?r ?m)` mineral transportado; `(brazo-libre ?r)` rover sin carga.
- **Acciones:** `mover` (desplazarse por un camino válido), `recoger` (tomar un mineral de la localidad actual; requiere brazo libre) y `dejar`

(depositar el mineral transportado en la localidad actual). Cada acción tiene coste 1; la métrica minimiza `(total-cost)`.

Decisión de modelado: el rover tiene **capacidad para un mineral a la vez** (predicado `brazo-libre`), lo que refleja la física de transportar rocas y obliga al planner a generar múltiples viajes, haciendo el plan más interesante y verificable.

7.2 Problema base (`problem1.pddl`)

Declara 5 localidades (`l1`–`l5`), el rover en `l3` (nodo central), `mineral1` en `l1` y `mineral2` en `l2`, los caminos según el enunciado y la meta de que **ambos minerales estén en `l5`** (laboratorio).

7.3 Ejecución con el planner ganador

Se ejecutó con el contenedor Delfi1 usando el mismo script oficial:

```
        singularity run -C -H rundir planner.img domain.pddl problem1.pddl
sas_plan
```

Figura 4. Ejecución del escenario base del rover con Delfi1: plan óptimo de 13 pasos.

Plan óptimo obtenido (13 pasos, costo 13):

```
(mover rover1 l3 l1)      ; va a la localidad 1
(recoger rover1 mineral1 l1)
(mover rover1 l1 l3)      ; l1<->l3 es bidireccional
(mover rover1 l3 l4)
(mover rover1 l4 l5)      ; deja mineral1 en el laboratorio
(dejar rover1 mineral1 l5)
(mover rover1 l5 l4)
(mover rover1 l4 l3)
(mover rover1 l3 l2)      ; l3->l2 es de una sola dirección
(recoger rover1 mineral2 l2)
(mover rover1 l2 l4)      ; l2->l4 es de una sola dirección
(mover rover1 l4 l5)
(dejar rover1 mineral2 l5)
```

Verificación: el plan es óptimo y respeta todas las restricciones: el rover solo usa `l3→l2` y `l2→l4` en el sentido permitido, y aprovecha la bidireccionalidad de `l3↔l1` y `l4↔l5`. Como solo puede llevar un mineral, debe hacer dos viajes completos, lo que explica los 13 pasos (cada viaje: ida, recogida, traslado y entrega).

8. Escenario diferente

Se propusieron **dos variantes** del escenario (archivos `problem2.pddl` y `problem3.pddl`), todas resueltas con el mismo `domain.pddl` y el planner ganador.

8.1 Variante 1 (`problem2.pddl`) — laboratorio extra y mineral nuevo

- **Localidad extra `l6`**, con un **segundo laboratorio** (`l5` y `l6`).
- **Mineral extra `mineral3`** excavado en `l4`.
- **Caminos nuevos:** `l5↔l6` bidireccional y `l4→l6` de una sola dirección (atajo hacia el laboratorio nuevo).
- **Meta:** `mineral1` y `mineral2` → `l5`; `mineral3` → `l6`.

Plan obtenido (17 pasos, costo 17): el rover entrega `mineral2` (l3→l2→l4→l5), luego `mineral1` (vía l1↔l3) y finalmente usa el atajo `l4→l6` para entregar `mineral3` en el laboratorio extra sin pasar por l5.

8.2 Variante 2 (`problem3.pddl`) — red ampliada con ciclo

- **Localidades extra `l6` y `l7`**; dos laboratorios (`l5` y `l7`).
- **Dos minerales extra:** `mineral3` en `l4` y `mineral4` en `l6`.
- **Restricciones nuevas:** `l5↔l6` bidireccional; `l6→l7`, `l4→l6` y `l7→l3` de una sola dirección. La vía `l7→l3` **cierra el ciclo**, obligando al rover a volver por l3 para reabastecerse.
- **Meta:** `mineral1` y `mineral2` → `l5`; `mineral3` y `mineral4` → `l7`.

Plan obtenido (21 pasos, costo 21): `mineral1` a l5; `mineral3` a l7 (l4→l6→l7); retorno por el ciclo `l7→l3`; `mineral2` a l5; y `mineral4` a l7. El plan demuestra que el planner explota correctamente el ciclo para volver a la zona de excavación.

rover-p2-p3

```
$ singularity run -C -H rundir-rover-p2 planner.img domain.pddl problem.pddl sas_pla
Using TensorFlow backend.
Chose seq-opt-symba-1
Solution found with cost 17 total time: 1.42s
Plan length: 17 step(s).
Plan cost: 17
Total time: 1.42s
Peak memory: 248648 KB

  (mover rover1 l3 l1)
  (recoger rover1 mineral1 l1)
  (mover rover1 l1 l3)
  (mover rover1 l3 l4)
  (mover rover1 l4 l5)
  (dejar rover1 mineral1 l5)
  (mover rover1 l5 l4)
  (mover rover1 l4 l3)
  (mover rover1 l3 l2)
  (recoger rover1 mineral2 l2)
  (mover rover1 l2 l4)
  (mover rover1 l4 l5)
  (dejar rover1 mineral2 l5)
  (mover rover1 l5 l4)
  (recoger rover1 mineral3 l4)
  (mover rover1 l4 l6)
  (dejar rover1 mineral3 l6)

$ singularity run -C -H rundir-rover-p3 planner.img domain.pddl problem.pddl sas_pla
Using TensorFlow backend.
Chose seq-opt-symba-1
Solution found with cost 21 total time: 0.5s
Plan length: 21 step(s).
Plan cost: 21
Total time: 0.5s
Peak memory: 206076 KB

  (mover rover1 l3 l1)
  (recoger rover1 mineral1 l1)
  (mover rover1 l1 l3)
  (mover rover1 l3 l4)
  (mover rover1 l4 l5)
  (dejar rover1 mineral1 l5)
  (mover rover1 l5 l4)
  (recoger rover1 mineral3 l4)
  (mover rover1 l4 l6)
  (mover rover1 l6 l7)
  (dejar rover1 mineral3 l7)
  (mover rover1 l7 l3)
  (mover rover1 l3 l2)
  (recoger rover1 mineral2 l2)
  (mover rover1 l2 l4)
  (mover rover1 l4 l5)
  (dejar rover1 mineral2 l5)
  (mover rover1 l5 l6)
  (recoger rover1 mineral4 l6)
```

Figura 5. Ejecución de los escenarios alternativos del rover: planes óptimos de 17 y 21 pasos.

9. Conclusiones

1. Se revisó la **IPC2018 Classical Tracks** (tracks, fragmento PDDL, dominios, planners y resultados) y se identificó al **ganador del optimal track: Delfi1** (126/200 tareas), cuyo contenedor Singularity se construyó y ejecutó exitosamente.
2. Se instaló **Singularity** y se siguió el **script oficial de "How can I test my containers?"**, resolviendo la **tarea Snake (problema 1)** con un **plan óptimo de 24 pasos**.
3. Se analizó la estructura del dominio de Snake (3 acciones: `move`, `move-and-eat-spawn`, `move-and-eat-no-spawn`; modelado de crecimiento de la serpiente mediante `headsnake`/`tailsnake`/`nextsnake` y de la aparición de puntos mediante `spawn`/`NEXTSPAWN`).
4. Se **transcribió a PDDL el problema del rover** (dominio + problema base) y se ejecutó con el planner ganador, obteniendo un **plan óptimo de 13 pasos** que respeta todas las restricciones unidireccionales del terreno.
5. Se propusieron **dos escenarios diferentes** (laboratorios extra, minerales y localidades adicionales, atajos y ciclos de una vía), resueltos con planes óptimos de 17 y 21 pasos, validando la expresividad del dominio.
6. La combinación **PDDL + contenedores Singularity** garantiza reproducibilidad: cualquier máquina con Singularity puede reconstruir y ejecutar exactamente el planner ganador, como exige el espíritu de la IPC.

10. Referencias

International Conference on Automated Planning and Scheduling. (s. f.). Competitions. https://www.icaps-conference.org/competitions/

IPC2018. (s. f.). International Planning Competition 2018. Classical Tracks. https://ipc2018-classical.bitbucket.io/

IPC2018. (2018a). Results [diapositivas]. https://ipc2018-classical.bitbucket.io/results/presentation.pdf

IPC2018. (2018b). Scores. https://ipc2018-classical.bitbucket.io/scores.html

Sylabs. (s. f.). Singularity Admin Guide — Installing Singularity. https://docs.sylabs.io/guides/3.5/admin-guide/installation.html

Katz, M., Sohrabi, S., Samulowitz, H., & Sievers, S. (2018). Delfi: Online planner selection for cost-optimal planning [resumen de planner, IPC2018]. https://ipc2018-classical.bitbucket.io/planner-abstracts/team23.pdf

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL — The Planning Domain Definition Language (Technical Report CVC TR-98-003). Yale Center for Computational Vision and Control.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research, 20, 61–124.

11. Anexos — Archivos PDDL

A continuación se transcriben en texto los cuatro archivos PDDL desarrollados en el trabajo: el dominio `domain.pddl` y los tres problemas

`problem1.pddl`, `problem2.pddl` y `problem3.pddl`. Pueden copiarse directamente y ejecutarse con el planner ganador Delfi1 siguiendo el manual de ejecución incluido en el material de entrega.

A.1 `domain.pddl` — Dominio del rover (acciones mover, recoger y dejar)

```
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        ;; DOMINIO: rover-minerales.pddl
        ;;
        ;; Problema: un robot tipo rover debe trasladar los minerales que
        ;; excavó (en las localidades 1 y 2) hasta el laboratorio de análisis,

        ;; respetando las restricciones de direccionalidad de los caminos.

        ;;
        ;; Modelado:
        ;;   - El rover se mueve entre localidades conectadas por caminos.

        ;;   - Un camino se representa de forma DIRECCIONAL con el predicado

        ;;     (camino ?a ?b): "se puede ir directamente de ?a a ?b".
        ;;       * Camino bidireccional  -> se declaran ambos sentidos.
        ;;       * Camino de una vía    -> se declara solo el sentido permitido.

        ;;   - El rover tiene capacidad para llevar UN mineral a la vez
        ;;     (predicado (brazo-libre ?r)), lo que obliga a planificar
        ;;     múltiples viajes entre la zona de excavación y el laboratorio.

        ;;
        ;; Fragmento PDDL 3.1 (compatible IPC 2018): STRIPS + typing +
        ;; precondiciones negativas + costes de acción.
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        (define (domain rover-minerales)

          (:requirements :strips :typing :negative-preconditions :action-
 costs)

          (:types
            rover     ; el robot
            localidad ; lugares del terreno (incluye el laboratorio)
            mineral   ; rocas/minerales excavados
          )

          (:predicates
            ; El rover r se encuentra en la localidad l
            (en ?r - rover ?l - localidad)

            ; Existe camino directo desde la localidad a hasta la b
            (camino ?a - localidad ?b - localidad)
```

```
    ; El mineral m se encuentra en la localidad l
    (mineral-en ?m - mineral ?l - localidad)

    ; El rover r está transportando el mineral m
    (cargando ?r - rover ?m - mineral)

    ; El rover r tiene el brazo libre (no transporta ningún mineral)

    (brazo-libre ?r - rover)
  )

  (:functions
    ; Coste total del plan (número de acciones ejecutadas)
    (total-cost) - number
  )

  ;; MOVER: desplaza el rover de la localidad ?a a la ?b.
  ;; Solo se puede usar si existe camino en ese sentido.
  (:action mover
    :parameters (?r - rover ?a - localidad ?b - localidad)
    :precondition
      (and
        (en ?r ?a)
        (camino ?a ?b)
      )
    :effect
      (and
        (not (en ?r ?a))
        (en ?r ?b)
        (increase (total-cost) 1)
      )
  )

  ;; RECOGER: el rover toma el mineral ?m que está en su misma
  ;; localidad. Requiere tener el brazo libre (capacidad 1).
  (:action recoger
    :parameters (?r - rover ?m - mineral ?l - localidad)
    :precondition
      (and
        (en ?r ?l)
        (mineral-en ?m ?l)
        (brazo-libre ?r)
      )
    :effect
      (and
        (cargando ?r ?m)
        (not (mineral-en ?m ?l))
        (not (brazo-libre ?r))
        (increase (total-cost) 1)
      )
  )

  ;; DEJAR: el rover deposita el mineral ?m que transporta en la
  ;; localidad ?l donde se encuentra (p. ej., el laboratorio).
  (:action dejar
    :parameters (?r - rover ?m - mineral ?l - localidad)
    :precondition
```

```
    (and
      (en ?r ?l)
      (cargando ?r ?m)
    )
  :effect
    (and
      (mineral-en ?m ?l)
      (not (cargando ?r ?m))
      (brazo-libre ?r)
      (increase (total-cost) 1)
    )
)

)
```

A.2 `problem1.pddl` — Problema 1 — escenario base del enunciado

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; PROBLEMA 1: Escenario base (transcripción del enunciado)
;;
;; - 5 localidades (l1..l5). El laboratorio de análisis está en l5.

;; - El rover parte de la localidad l3 (nodo central del grafo).
;; - mineral1 fue excavado en l1; mineral2 fue excavado en l2.
;; - Caminos (según el enunciado y la Figura 1):
;;       * l3 <-> l1 : bidireccional
;;       * l3  -> l2 : una sola dirección
;;       * l2  -> l4 : una sola dirección
;;       * l3 <-> l4 : bidireccional
;;       * l4 <-> l5 : bidireccional
;;
;; Objetivo: ambos minerales en el laboratorio (l5).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem rover-base)
  (:domain rover-minerales)

  (:objects
    rover1 - rover
    l1 l2 l3 l4 l5 - localidad
    mineral1 mineral2 - mineral
  )

  (:init
    ;; Posición inicial del rover (nodo central)
    (en rover1 l3)

    ;; Caminos bidireccionales: se declaran ambos sentidos
    (camino l1 l3) (camino l3 l1)   ; l3 <-> l1
    (camino l3 l4) (camino l4 l3)   ; l3 <-> l4
    (camino l4 l5) (camino l5 l4)   ; l4 <-> l5

    ;; Caminos de una sola dirección: se declara solo el sentido válido

    (camino l3 l2)                  ; l3  -> l2
```

```
    (camino l2 l4)                  ; l2  -> l4

    ;; Minerales excavados en sus localidades
    (mineral-en mineral1 l1)
    (mineral-en mineral2 l2)

    ;; El rover comienza sin carga
    (brazo-libre rover1)

    ;; Métrica de coste (inicia en 0)
    (= (total-cost) 0)
  )

  ;; Meta: los dos minerales deben estar en el laboratorio (l5)
  (:goal
    (and
      (mineral-en mineral1 l5)
      (mineral-en mineral2 l5)
    )
  )

  (:metric minimize (total-cost))
)
```

A.3 `problem2.pddl` — Problema 2 — escenario con laboratorio extra y mineral adicional

```
(define (problem rover-escenario-dos-labs)
  (:domain rover-minerales)

  (:objects
    rover1 - rover
    l1 l2 l3 l4 l5 l6 - localidad
    mineral1 mineral2 mineral3 - mineral
  )

  (:init
    (en rover1 l3)

    ;; Caminos bidireccionales
    (camino l1 l3) (camino l3 l1)   ; l3 <-> l1
    (camino l3 l4) (camino l4 l3)   ; l3 <-> l4
    (camino l4 l5) (camino l5 l4)   ; l4 <-> l5
    (camino l5 l6) (camino l6 l5)   ; l5 <-> l6

    ;; Caminos de una sola dirección
    (camino l3 l2)                  ; l3  -> l2
    (camino l2 l4)                  ; l2  -> l4
    (camino l4 l6)                  ; l4  -> l6 (atajo al lab2)

    ;; Minerales: los dos originales + mineral3 en l4
    (mineral-en mineral1 l1)
    (mineral-en mineral2 l2)
    (mineral-en mineral3 l4)
```

```
    (brazo-libre rover1)
    (= (total-cost) 0)
  )

  (:goal
    (and
      (mineral-en mineral1 l5)
      (mineral-en mineral2 l5)
      (mineral-en mineral3 l6)
    )
  )

  (:metric minimize (total-cost))
)
```

A.4 `problem3.pddl` — Problema 3 — escenario con red ampliada y ciclo de una vía

```
(define (problem rover-escenario-grande)
  (:domain rover-minerales)

  (:objects
    rover1 - rover
    l1 l2 l3 l4 l5 l6 l7 - localidad
    mineral1 mineral2 mineral3 mineral4 - mineral
  )

  (:init
    (en rover1 l3)

    ;; Caminos bidireccionales
    (camino l1 l3) (camino l3 l1)   ; l3 <-> l1
    (camino l3 l4) (camino l4 l3)   ; l3 <-> l4
    (camino l4 l5) (camino l5 l4)   ; l4 <-> l5
    (camino l5 l6) (camino l6 l5)   ; l5 <-> l6

    ;; Caminos de una sola dirección
    (camino l3 l2)                  ; l3  -> l2
    (camino l2 l4)                  ; l2  -> l4
    (camino l4 l6)                  ; l4  -> l6 (atajo)
    (camino l6 l7)                  ; l6  -> l7 (lab secundario)
    (camino l7 l3)                  ; l7  -> l3 (cierra el ciclo)

    ;; Cuatro minerales distribuidos
    (mineral-en mineral1 l1)
    (mineral-en mineral2 l2)
    (mineral-en mineral3 l4)
    (mineral-en mineral4 l6)

    (brazo-libre rover1)
    (= (total-cost) 0)
  )

  (:goal
    (and
```

```
            (mineral-en mineral1 l5)
            (mineral-en mineral2 l5)
            (mineral-en mineral3 l7)
            (mineral-en mineral4 l7)
          )
        )

        (:metric minimize (total-cost))
      )
```