UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Práctica de planificadores

Inteligencia artificial

Algorismes Bàsics per la Intel·Ligència Artificial

Autors

Artur Aubach Altes

Daniel López García

Grup 11

Profesor: Sergio Alvarez Napagao

Quadrimestre Primavera 2023/2024

TABLA DE CONTENIDOS

1. IDENTIFICACIÓN DEL PROBLEMA

1.1 Contextualización

En la actualidad la gran oferta existente de literatura hace difícil elegir qué leer. Muchas tiendas por internet poseen sistemas de recomendación que permiten obtener listas de libros que nos pueden interesar a partir de los libros que hemos adquirido o los que buscamos en la tienda. Otros sistemas de recomendación de libros usan características más o menos intuitivas que nos permiten describir el tipo de libro que buscamos y nos recomiendan libros que tienen esas características. Sin embargo en el caso de las novelas de ficción, a menudo hay complejas sagas de libros donde, para seguir correctamente la historia, se han de leer en un orden determinado. Y esto es algo que los sistemas de recomendación online no suelen hacer.

El plan de lectura ha de tener en cuenta que a veces hay dependencias entre libros:

- **Libros predecesores:** es típico de sagas o de historias divididas en varios libros. Llamaremos libros predecesores a aquellos libros que se han de leer antes de un libro para poder disfrutarlo mejor. Por ejemplo, si el usuario quiere leer la 4a entrega de Harry Potter y solo ha leído el primer libro de la saga, el planificador ha de incluir en el plan de lectura el segundo y tercer libro, que preceden al cuarto.
- **Libros paralelos:** esto ocurre en aquellos casos en que los libros pertenecen a universos ficticios donde las historias no solo pasan una después de la otra, sino también en paralelo, por lo que es bueno leer las historias más o menos en el orden en el que ocurren. Este fenómeno es muy típico del cómic americano desde los años 60 (crossovers), pero también pasa en los libros de universos como Star Wars o Star Trek.

1.2 Descripción de los Problema

Al sistema se le ha de dar conocimiento sobre:

- los libros del catálogo
- los libros predecesores a un libro
- los libros paralelos a un libro
- los libros que el usuario ya ha leído
- los libros que el usuario quiere leer

El resultado es un plan de lectura que refleja:

- los libros mínimos que el usuario ha de leer
- para cada libro, indica en qué mes ha de leerlo el usuario
- para todos los libros del plan se cumple en todo momento que sus predecesores se leen en meses anteriores
- para todos los libros del plan se cumple en todo momento que sus paralelos se leen en el mismo mes o en el mes anterior (o en el mes siguiente, para la relación simétrica).

Además, tendremos cuatro variantes del problema, cuatro niveles:

- **Nivel básico:** En el plan de lectura todos los libros tienen 0 o 1 predecesores y ningún paralelo. El planner es capaz de encontrar un plan para poder llegar a leer los libros objetivo encadenando libros, donde cada libro tiene solo uno o ningún predecesor.
- **Extensión 1:** Los libros pueden tener de 0 a N predecesores pero ningún paralelo. El planner es capaz de construir un plan para poder llegar a leer los libros objetivo, donde para todo libro que pertenece al plan, todos sus libros predecesores pertenecen al plan y están en meses anteriores.
- **Extensión 2:** Aumentamos, hacemos más compleja, la extensión uno añadiendo que los libros pueden tener de 0 a M libros paralelos. El planner es capaz de construir un plan para poder llegar a leer los libros objetivo, donde para todo libro que pertenece al plan, todos sus libros paralelos pertenecen al plan y están en el mismo mes o en meses anteriores(además de la extensión uno)..
- **Extensión 3:** Los libros tienen además un número de páginas. El planificador controla que en el plan generado no se superen las 800 páginas al mes.

1.3 Elementos de los Problema

El problema consta, por las particularidades de pddl, de dos partes(dos ficheros de código):

- **La parte del problema:** Aquí declaramos que objetos forman parte de nuestro problema, declaramos las variables (en nuestro caso los libros y meses), declaramos el estado inicial de nuestro problema a base de predicados atómicos(que libro son predecesores y de quién, qué libros son objetivos…), especificamos nuestro objetivo y, en caso de que usemos fluentes, que métrica hay que minimizar(o maximizar).

- **La parte del dominio:** Aquí declaramos los tipos(las etiquetas con las que clasificamos variables nuestras para guiar al planificador), nuestros predicados con el formato que deben seguir y el tipo de variable que pueden unificar, los flotantes que usamos(si usamos) y los operadores, las acciones, que puede realizar el planificador junto con sus correspondientes parámetros, precondiciones y efectos.

1.4 Objetivo

En esta práctica nuestra meta es desarrollar una herramienta sencilla que, basándose en los libros que el usuario ya ha leído y los libros que quiere leer durante el próximo año, haga un plan de lectura mensual en el que se le recomiende al usuario en qué orden ha de leer los libros, intentando balancear un poco el número de páginas que le tocaría leer cada mes(como mucho 800 páginas).

1.5 Estrategia Propuesta y planificadores

Nuestra estrategia consiste en la asignación de libros a meses, comenzando desde Enero, que cumplan todas las restricciones(son un objetivo con todo predecesor o paralelo leído o son uno de estos predecesores y/o paralelos que deben leerse). Nuestra propuesta final(extensión 3) tiene en cuenta, además, la cantidad de páginas máxima que puede haber en un mes y la cantidad de páginas de los libros para proporcionar un plan de lectura mensual con la cantidad de páginas a leer mejor distribuida.

Los planificadores son herramientas algorítmicas ideales para abordar el problema de diseñar un plan de lectura, ya que se centran en resolver problemas mediante la generación de una secuencia de pasos, en nuestro caso asignaciones de libros a meses, que permitan pasar de un estado inicial a un estado objetivo, utilizando operadores definidos en un dominio específico.

En el contexto de diseñar un plan de lectura, nosotros queremos encontrar la manera de asignar libros a meses y, como antes hemos explicado, los planificadores son expertos en encontrar soluciones a este tipo de desafíos.
Al aplicar un planificador a este problema, el algoritmo buscará una secuencia de pasos que, cuando se ejecuten, logren la transición del estado inicial al estado objetivo. En este caso, esos pasos representarán la asignación de libros a meses que

debemos hacer de manera que se satisfagan todos los requisitos y restricciones establecidas.

2. IMPLEMENTACIÓN

Aquí explicaremos la versión final de nuestra solución al problema, la cual se corresponde a la extensión tres.

2.1 Representación del Dominio

Aquí explicaremos las diferentes secciones de nuestro dominio:

2.2.1 Tipos

Disponemos de dos tipos: libros y mes acorde a las dos variables de que nuestro problema dispone, las cuales son libros y los meses del año.

2.1.2 Predicados

Disponemos de *9 predicados*, cada uno con su razón y función:

- ***objectiu ?l*** - *libro:* nos permite definir cuales de nuestros libros son objetivos a leer y, en las precondiciones, comprobarlo con facilidad siempre y cuando consideremos necesario comprobarlo.
- ***predecesor ?l1 ?l2*** - *libro:* este predicado nos permite definir que un libro 1 sea predecesor de un libro 2 cosa crucial para el problema ya que mediante el encadenamiento de este predicado definiremos sagas literarias o otros conjuntos de libros. Además, también es necesario para algunas de las precondiciones de nuestras acciones ya que unificando libros a sus variables podremos saber si un libro es predecesor de otro o no.
- ***es_sucesor ?l*** - *libro:* este predicado nos permite simplificar el dilucidar si un libro TIENE sucesores, aquellos libros con sucesores serán definidos con este predicado y de esta manera en las precondiciones de nuestros operadores podemos fácilmente comprobar si un libro tiene predecesores que deben ser leídos o no sin necesidad de un exists o un for all que, sin este predicado, sería necesario. Pongamos un sencillo ejemplo: tenemos la secuencia libro1-libro2, y hemos declarado 'predecesor libro1 libro2'(libro1 es predecesor de libro2), esto implica que debemos declarar también 'es_sucesor libro2' ya que libro2 TIENE un sucesor(libro1) pero libro1 no tiene sucesores y, por tanto, no lo tenemos que declarar como es_sucesor.
- ***paralelo ?l1 ?l2*** - *libro:* similar a predecesor, indica si un libro1 es paralelo a un libro 2 lo cual es crucial para el problema. Cabe mencionar que no es

bidireccional, es decir, si tenemos un libro 1 paralelo a un libro 2 en nuestro estado inicial no definiremos 'paralelo libro1 libro2' y 'paralelo libro2 libro1', sinó que únicamente definiremos 'paralelo libro1 libro2'

- ***es_paralelo ?l*** - *libro:* cumple exactamente la misma función que su homólogo es_sucesor: su presencia nos permite comprobar fácilmente en una precondición si un libro ES o TIENE paralelos sin necesidad de usar un for all o exists. En el estado inicial todo libro que forme parte de un predicado 'paralelo libro1 libro2' deberá ser declarado como 'es_paralelo'. (en nuestro caso 'es_paralelo libro1' y 'es_paralelo libro2'(en este respecto SI difiere de es_sucesor).
- ***leido_en_mes ?l*** - *libro* ***?m*** - *mes:* este es un predicado esencial ya que es el predicado que usaremos para asignar un libro a un mes.
- ***leido ?l*** - *libro:* este predicado nos permite saber si el usuario ya leyó el libro en el estado inicial y en los efectos de los operadores, tras asignar un libro a un mes con el estado anterior, también lo declaramos con este predicado como leído. Esto puede parecer redundante, pero hacerlo de esta manera nos ahorramos for alls y exists cada vez que queramos comprobar que un libro haya sido leído o no. Por ejemplo, tenemos dos libros, libro1 y libro2. Tenemos que en el estado inicial declaramos libro1 como leido por el usuario, así que solo buscaremos asignar a libro2 a un mes. Cuando finalmente hacemos la asignación en el effect de un operador no solo declaramos 'leido_en_mes libro2 mes', también declararemos 'leido libro1'.
- ***siguiente_mes ?m1 ?m2*** - *mes:* este predicado nos permite ordenar los meses de comenzando por enero y acabando en diciembre de manera que podamos asegurarnos de que los predecesores se leen en un mes anterior al de los objetivos que preceden.
- ***mes_actual ?m*** - *mes:* este predicado hace la función de cursor con el que vamos apuntando a los meses de uno en uno comenzando por enero. En los operadores miramos que el mes al que estemos asignando un libro sea el actual y entonces para este mes miramos que haya espacio(que las páginas del libro que queremos asignar + la de las que ya están asignados no supere el límite de 800), miramos que el predecesor lo hayamos leído en un mes anterior(si tiene)... Es un operador crucial para el funcionamiento, eficiente, de nuestro código y nos ahorra for alls y exists. Existe una acción

específicamente para cambiar el mes de mes_actual al siguiente. Usar este cursor nos permite, entre otras ventajas, poder usar un único flotante para contar la cantidad de páginas asignadas a cada mes con un único flotante que reseteamos a 0 cada vez que cambiemos de mes.

2.1.3 Operadores

En nuestro dominio disponemos de tres operadores:

- **leer_no_sucesor_no_paralel:** esta acción se encarga de asignar a un mes aquellos libros que no tienen sucesores ni son paralelos. Para ello necesitamos dos parámetros: libro y month, un libro y un mes al que asignar nuestro libro, y en las precondiciones comprobamos que el mes sea aquel al que nuestro cursor está apuntando(mes_actual), que el libro que queremos asignar no haya sido leído(not (leido ?libro)), y que no sea ni tenga sucesores ni sea paralelo((not (es_sucesor ?libro)) y (not (es_paralelo ?libro))) además, finalmente, también debemos comprobar que la suma de las páginas de los libros que han sido asignados a este mes y la de este nuevo libro que queremos asignar no superen el límite(800). Finalmente en effects asignamos el libro a un mes con leido_en_mes y lo ponemos como leído con el predicado leido. Además, sumamos el número de páginas del libro al de las asignadas al mes. Finalmente, si el libro leído era un objetivo(ya que puede ser el primer libro de una secuencia de libros (libro1-libro2-libro3 estoy hablando de libro1) restamos el número que tenemos de objetivos por leer en 1(un flotante).

```
Unset
  (:action leer_no_sucesor_no_paralel
    :parameters (?libro - libro ?month - mes)
    :precondition (and
      (mes_actual ?month)
      (not (leido ?libro)) ;el libro no ha estado leido
      (not (es_sucesor ?libro))
      (not (es_paralelo ?libro))
      (<= (+ (paginas_totales) (num_paginas ?libro)) 800)
    )
```

```
    :effect (and
      (leido_en_mes ?libro ?month)
      (leido ?libro)
      (increase (paginas_totales) (num_paginas ?libro))
      (when
        (objectiu ?libro)
        (decrease (libros_restantes) 1))
    )
  )
```

- **leer_sucesor_y_paralelo1:** este operador se encarga de asignar a un mes aquellos libros que o tienen sucesores o son paralelos. Para ello dispone de tres parámetros: libro, month y pas_month(el libro, el mes donde quiero asignarlo y el mes pasado a este). En las precondiciones comprobamos que el mes que quiero asignar(month) sea aquel al que el cursor(mes_actual) apunta, compruebo que el libro que quiero leer no haya sido ya leído y que pas_month sea anterior a month(siguiente_mes ?pas_month ?month), comprobamos que el libro sea o bién tenga un sucesor o bién sea un paralelo y que, para todo libro l1 si este es un predecesor de libro pues este l1 debe haber sido leído, y debe estar asignado en pas_month(en el mes anterior) y también comprobamos que si este l1 es un paralelo de libro que este haya sido leído en el mes en que estamos o el anterior(es decir, que comprobamos que el predecesor este en un mes anterior y que el paralelo este en un mes anterior o el actual). Como puede observarse, decidimos simplificar nuestro problema haciendo que los libros paralelos puedan asignarse en el mes anteior o el actual no teniendo en cuenta que puedan asignarse a un mes futuro ya que esto nos permite obtener un código más óptimo y rápido que si tenemos esto en cuenta y seguimos obteniendo soluciones, planes, que respetan las restricciones establecidas en el enunciado. Los efectos son los mismos que los de la solución anterior

Unset

```
(:action leer_sucesor_y_paralelo1
    :parameters (?libro - libro ?month - mes ?pas_month - mes)
    :precondition (and
      (mes_actual ?month)
      (not (leido ?libro)) ; el libro no ha sido leído
      (siguiente_mes ?pas_month ?month) ; Identificar el mes pasado de ?month
      (or(es_sucesor ?libro) ; el libro es sucesor
      (es_paralelo ?libro))
      (<= (+ (paginas_totales) (num_paginas ?libro)) 800)
      (forall
        (?l1 - libro)
        (and
          (imply
            (paralelo ?l1 ?libro)
            (and (leido ?l1)
            (or (leido_en_mes ?l1 ?month)
              (leido_en_mes ?l1 ?pas_month)
            ))
          )

          (imply
            (predecesor ?l1 ?libro)
            (and (leido ?l1)
              (not (leido_en_mes ?l1 ?month))
               ) ; que se lea en un mes diferente a aquellos en los que se leyeron sus
libros predecesores.
          )
          )
        )
    )
    :effect (and
      (leido_en_mes ?libro ?month)
      (leido ?libro)
      (increase (paginas_totales) (num_paginas ?libro))
      (when
        (objectiu ?libro)
        (decrease (libros_restantes) 1))
```

```
    )
  )
```

- **canviar_mes:** esta acción la usamos para cambiar el mes de nuestro cursor(mes_actual) al siguiente mes. Para ello requerimos dos parametros: acual_month y next_month y en las precondiciones comprobamos que acual_month sea el mes al que mi cursor apunta y que next_month sea el mes a continuación, el mes siguiente, al actual. Como efecto negamos el antiguo mes al que el cursor apuntaba((not (mes_actual ?acual_month)), declaramos el nuevo mes_actual como next_month ((mes_actual ?next_month)) y reseteamos la cantidad de páginas ya que el nuevo mes al que apuntamos no tiene ningún libro asignado.

```
Unset
  (:action canviar_mes
    :parameters (?acual_month ?next_month - mes)
    :precondition (and
      (mes_actual ?acual_month)
      (siguiente_mes ?acual_month ?next_month))

    :effect (and
      (not (mes_actual ?acual_month)) ; Elimina el valor actual de mes_actual
      (mes_actual ?next_month) ; Añade el nuevo valor a mes_actual
      (assign (paginas_totales) 0)) ; Reinicia paginas_totales a 0
  )
```

2.1.4 Fluentes

Disponemos de tres fluentes que nos permiten saber la cantidad de páginas que he asignado a un mes, la cantidad de páginas de un libro y la cantidad de objetivos que tenemos pendientes por leer:

- **(num_paginas ?l - libro):** establecemos en el estado inicial la cantidad de páginas de cada libro y después la usaremos en las precondiciones y efectos para sumar a la cantidad de páginas asignadas a un mes.
- **(libros_restantes):** lo declaramos en el estado inicial como la cantidad de objetivos que quiero leer y es, además, la métrica a minimizar. Esto nos permite no tener que poner en la acción leer_no_sucesor_no_paralel explícitamente que el libro que lea sea un objetivo ya que si no lo es preferirá no leerlo a no ser que no tenga remedio(es el inicio de una secuencia que acaba en un objetivo) brindándonos flexibilidad. Además, nos permite guiar al algoritmo proporcionandonos mejor rendimiento. En los efectos de las acciones, cada vez que leamos un objetivo, lo hacemos decrecer en 1.
- **(paginas_totales):** este flotante, que inicializamos en el estado inicial como 0, guarda la cantidad de páginas que hemos asignado al mes al que estamos apuntando con nuestro cursor(mes_actual). En las precondiciones de las acciones comprobemos que no sumado a la cantidad de páginas del libro que queremos asignar no se supere el límite(800) y en los efectos le sumamos el numero de páginas del libro. Además, cada vez que utilizamos el operador canviar_mes reseteamos su valor a 0 porque apuntamos a un nuevo mes sin libros todavía asignados.

2.2 Representación del Problema

Además de declarar en objects los libros que tenemos y los 12 meses del año, en esta código representamos el estado inicial del problema y el objetivo a cumplir.

```
Unset
  (:objects
    ll_1 ll_1_2 ll_2 ll_2_1 ll_99 ll_3 ll_4 ll_5 - libro
    enero febrero marzo abril mayo junio julio agosto septiembre octubre noviembre
diciembre - mes
  )
```

2.2.1 Representación del estado inicial

Podemos dividir el estado inicial(:init) en siete partes:

- Declaramos la cantidad de páginas que tiene cada uno de los libros declarados en el apartado de object.((=(num_paginas libro) X)) donde X es el número de páginas del libro

```
Unset
(=(num_paginas ll_1) 200)
(=(num_paginas ll_1_2) 150)
(=(num_paginas ll_2) 300)
(=(num_paginas ll_2_1) 800)
(=(num_paginas ll_99) 100)
(=(num_paginas ll_3) 150)
(=(num_paginas ll_4) 100)
(=(num_paginas ll_5) 250)
```

- Declaramos el orden de los meses (siguiente_mes enero febrero… siguiente_mes noviembre diciembre).

```
Unset
(siguiente_mes enero febrero)
(siguiente_mes febrero marzo)
(siguiente_mes marzo abril)
(siguiente_mes abril mayo)
(siguiente_mes mayo junio)
(siguiente_mes junio julio)
(siguiente_mes julio agosto)
(siguiente_mes agosto septiembre)
(siguiente_mes septiembre octubre)
(siguiente_mes octubre noviembre)
```

```
(siguiente_mes noviembre diciembre)
```

- Asignamos el cursor(mes_actual) al primero de nuestros meses del año: mes_actual enero e inicializamos el flotante encargado de llevar la cuenta de la cantidad de páginas asignadas a cada mes como 0.

```
Unset
(mes_actual enero)
(= (paginas_totales) 0)
```

- Declaramos cuales de nuestros libros serán objetivos e inicializamos el flotante que cuenta la cantidad de objetivos pendientes con el número de objetivos previamente establecidos.

```
Unset
(objectiu ll_5)
(objectiu ll_99)
(objectiu ll_2_1)

(= (libros_restantes) 3)
```

- Establecemos los predecesores y declaramos que libros tenemos que tienen sucesores.

```
Unset
(predecesor ll_1 ll_1_2)
(predecesor ll_2 ll_2_1)
(predecesor ll_2_1 ll_99)
(predecesor ll_1_2 ll_99)
```

```
    (es_sucesor ll_1_2)
    (es_sucesor ll_2_1)
    (es_sucesor ll_99)
```

- Establecemos los paralelos y declaramos todos los libros involucrados como es_paralelo

```
Unset
    (paralelo ll_4 ll_1_2)
    (es_paralelo ll_4)
    (es_paralelo ll_1_2)
```

- Establecemos qué libros ha leído ya el usuario.

```
Unset
(leido ll_3)
```

2.2.1 Representación del goal

Gracias al flotante que controla cuantos objetivos tenemos pendiente por leer, nuestro objetivo será que no tengamos ninguno pendiente, es decir:

```
Unset
  (:goal
    (= (libros_restantes) 0)
  )
```

2.2.2 Que optimizamos

Tenemos un objetivo a minimizar: el número de objetivos pendientes ya que cuanto menor sea mejor, queremos priorizar aquellos operadores que nos permitan reducir este número:

```
Unset
  (:metric minimize
    (libros_restantes)
  )
```

2.2.3 Representación de la solución

Para observar cómo son las soluciones que proporciona nuestro problema, observaremos un ejemplo:

step 0: LEER_NO_SUCESOR_NO_PARALEL LL_2 ENERO
1: CANVIAR_MES ENERO FEBRERO
2: LEER_NO_SUCESOR_NO_PARALEL LL_5 FEBRERO
3: CANVIAR_MES FEBRERO MARZO
4: LEER_SUCESOR_Y_PARALELO1 LL_2_1 MARZO FEBRERO
5: CANVIAR_MES MARZO ABRIL
6: LEER_SUCESOR_Y_PARALELO1 LL_99 ABRIL MARZO

Las acciones 'LEER_NO_SUCESOR_NO_PARALEL son acciones de asignación de unos determinados tipos de libros a un mes, en concreto asignará aquellos libros que no tienen predecesores ni son(ni tienen) paralelos, es decir, libros objetivos sin predecesores e inicios de secuencias de predecesores.

La acción CANVIAR_MES no nos importa en lo más mínimo en nuestra solución ya que simplemente estamos moviendo el cursor.

La acción LEER_SUCESOR_Y_PARALELO1, en cambio, sí debemos aclararla. Podemos observar que en esta acción tenemos dos meses, debemos fijarnos solo en el primero ya que es en el primero de los meses donde estamos asignando el libro, el segundo mes es el mes previo al que nos interesa y está para facilitar precondiciones más eficientes en la acción. En cuanto a qué libros lee esta acción,

esta acción leerá todo libro que TIENE un predecesor o que es paralelo(o tiene un paralelo).

2.3 Optimización

Para conseguir un código, unos operadores, que permitan al planificador encontrar una solución más rápida y eficientemente decidimos tomar una serie de decisiones en cuanto a optimización:

- *Uso de tipos*, esto nos permite guiar al algoritmo del planificador y ahorrarnos que este intente unificar meses a predicados que pertocan a libros y viceversa.
- *Añadimos predicados que filtran los elementos* que pueden entrar a una acción reduciendo la cantidad de unificaciones y pruebas necesarias. Estos predicados de los que hablamos son es_sucesor y es_paralelo que, además. enlazan con la optimización a continuación.
- *Evasión del abuso de foralls* y exists mediante los predicados mencionados anteriormente. Ahorrar en foralls implica ayudar al planificador enormemente ya que ahorra en gran medida una elevada cantidad de unificaciones y pruebas.
- *Adición de un fluente* que guía al planificador. Acá incluiríamos al fluente que queremos minimizar y nos indica la cantidad de objetivos que nos quedan además de permitirnos comprobar rápida y fácilmente nuestro objetivo(que libros objetivos que nos quedan por asignar sea 0).
- Finalmente cabe destacar nuestro *'cursor'* que nos ayuda enormemente al ahorro antes mencionado de foralls y exists, aunque nos fuerza a añadir una nueva acción que, sin embargo, sabemos que pocas veces se llamará gracias a la métrica a minimizar.

3. EXTENSIONES

3.1 Nivel Básico

En esta primera versión del código nuestro objetivo era asignar libros a meses donde un libro podía tener unicamente un predecesor y no podía tener paralelos. Rápidamente nos fijamos que si no se tenía en cuenta un límite de páginas asignables a cada mes, podíamos asignar todo en uno o dos meses así que decidimos realizar otra simplificación: no tendríamos en cuenta los meses, considerando que todos los libros están asignados a un mismo mes, y en cambio los ordenaríamos según cual leer primero siguiendo las restricciones pertinentes.
Para esta primera y muy simplificada versión dispondríamos de tres predicados:

- **objectiu ?x:** predicado crucial para saber que libro es un objetivo que queremos leer y cuál no y que usamos para guiar al planificador
- **predecesor ?x ?y:** predicado que usamos para saber si un libro y tiene un predecesor x y que es crucial para las precondiciones(ya que debemos saber si debemos leer un libro x antes que el y si queremos hacer una asignación adecuada).
- **leido ?x:** con este predicado podemos saber qué libros ha leído ya el usuario o hemos asignado ya en una posición. Es un predicado crucial para las Precondiciones y también se usa en los efectos.

En cuanto a acciones, en este primer nivel por el momento contamos con solamente dos acciones:

- **leer_objectiu:** esta acción tiene como objetivo leer un libro objetivo que no tenga, o si tiene hallan sido leídos, libros predecesores. Para esto dispone únicamente un parámetro, el libro que queremos asignar a una posición, y en sus precondiciones comprobamos que sea un objetivo, que no lo hayamos leído ya y que no exista ningún libro predecesor suyo o, en caso contrario lo hayamos leído. Como efecto, indicamos que el libro ha sido leído(ha sido asignado).

```
Unset
(:action leer_objectiu
```

```
    :parameters (?libro)
    :precondition (and
        (objectiu ?libro)
        (not (leido ?libro))
        (forall
            (?x)
            (or (not (predecesor ?x ?libro)) (leido ?x)))
    )
    :effect (leido ?libro)
)
```

- **leer_predecesor:** esta acción tiene como objetivo leer libros predecesores de libros objetivos. Para ello dispone de dos parámetros: ?libro(el libro que deseamos asignar) y ?obj(el libro del cual es predecesor). En las precondiciones, comprobaremos una serie de cosas:
    - que ?libro sea predecesor de ?obj
    - que ?obj sea un objetivo a leer
    - que no hayamos leído ?libro

  Como efecto, este es el mismo que en la acción anterior: ?libro pasa a estar como leído.

Unset

```
(:action leer_predecesor
    :parameters (?libro ?obj)
    :precondition (and
        (predecesor ?libro ?obj)
        (objectiu ?obj)
        (not (leido ?libro))
    )
    :effect (leido ?libro)
)
```

3.1.1 Juegos de pruebas

3.1.1.1 Objetivo leído implica que su predecesor no se lee

3.1.1.1.1 ¿Qué queremos comprobar?

Queremos comprobar que si el objetivo que queremos leer ya ha sido leído por el usuario(en el estado inicial) no leamos sus predecesores(ya que no es necesario). Disponemos de dos objetivos,LL_1_2 que tiene de predecesor a LL_1 y LL_2, y uno de estos objetivos, LL_1_2, ha sido leído ya por el usuario, esta como leído en el estado inicial.

3.1.1.1.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step 0: LEER_OBJECTIU LL_2

time spent: 0.00 seconds instantiating 1 easy, 1 hard action templates
0.00 seconds reachability analysis, yielding 11 facts and 2 actions
0.00 seconds creating final representation with 4 relevant facts, 0 relevant fluents
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 2 states, to a max depth of 1
0.00 seconds total time

El resultado es correcto, unicamente leemos LL_2 un objetivo sin ningún predecesor e ignoramos a LL_1_2 ya que lo hemos leído y, por tanto, también a su predecesor. Podemos observar, además, que no recorremos demasiados nodos ni evaluamos estados antes de llegar a la solución cosa que encaja con un problema de el tamaño y sencillez que estamos evaluando.

3.1.1.1.3 Conclusión

El código permite al planificador manejar esta situación correctamente.

3.1.1.2 El predecesor de un libro que es un objetivo se lee correctamente

3.1.1.2.1 ¿Qué queremos comprobar?

Queremos comprobar que si el objetivo que queremos leer tiene un predecesor se lea primero su predecesor y a continuación el libro objetivo en cuestión.
Disponemos de dos objetivos,LL_1_2 que tiene de predecesor a LL_1 y LL_2.

3.1.1.2.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step 0: LEER_PREDECESOR LL_1 LL_1_2
1: LEER_OBJECTIU LL_2
2: LEER_OBJECTIU LL_1_2

time spent: 0.00 seconds instantiating 1 easy, 2 hard action templates
0.00 seconds reachability analysis, yielding 12 facts and 3 actions
0.00 seconds creating final representation with 6 relevant facts, 0 relevant fluents
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 4 states, to a max depth of 1
0.00 seconds total time

El resultado es correcto, leemos primero el predecesor de nuestro objetivo y depués nuestros objetivos. Podemos observar, además, que no recorremos demasiados nodos ni evaluamos estados antes de llegar a la solución cosa que encaja con un problema de el tamaño y sencillez que estamos evaluando.

3.1.1.2.3 Conclusión

El código permite al planificador manejar esta situación correctamente.

3.2 Extensión 1

En esta extensión, donde ya no tenemos únicamente un único predecesor sinó que podemos tener múltiples predecesores, cadenas de predecesores, ya consideramos más necesario asignar libros a meses y así hicimos. Este no fué el único cambio, también decidimos añadir un flotante que controlase la cantidad de libros objetivos pendientes por asignar y esta es la métrica que queremos minimizar. También decidimos que asignariamos meses moviéndonos a través de ellos con un predicado que haría de cursor. Finalmente, introducimos los tipos ya que ahora no solo tenemos libros, sinó que también tenemos meses.

En cuanto a predicados, ahora tenemos:

- objectiu ?l - libro: predicado crucial para saber que libro es un objetivo que queremos leer y cuál no y que usamos para guiar al planificador
- predecesor ?l1 ?l2 - libro: predicado que usamos para saber si un libro l2 tiene un predecesor l1, lo cual es crucial para las precondiciones(ya que debemos saber si debemos leer un libro l1 antes que un libro l2 si queremos hacer una asignación adecuada).
- leido ?l - libro: con este predicado podemos saber qué libros ha leído ya el usuario o hemos asignado ya en una posición. Es un predicado crucial para las Precondiciones y también se usa en los efectos.
- es_sucesor ?l - libro: este predicado ya hemos explicado anteriormente la razón de su existencia, pero para resumir, su existencia nos permite comprobar más fácil y eficientemente si un libro ?l tiene predecesores o no.
- leido_en_mes ?l - libro ?m - mes: este predicado es crucial ya que es el predicado que usamos para asignar libros a un mes, a diferencia de leido ?l que únicamente comprueba si un libro ha sido leído(o asignado ya que así podemos comprobar más fácilmente en la precondición si hemos asignado o no un libro)
- siguiente_mes ?m1 ?m2 - mes: este predicado lo usamos para, en el estado inicial, indicar el orden de los meses(y en la precondición comprobar si estamos respetando dicho orden)
- mes_actual ?m - mes: nuestro cursor, cuya función y existencia hemos explicado en un apartado anterior.

En cuanto a acciones, disponemos de tres:

- leer_no_sucesor_: dispone dos parámetros, un libro ?libro y un mes ?month y comprobamos que el mes sea el mes al que estamos apuntando, que no hayamos leído el libro y que dicho libro no tenga predecesores. Como efectos asignamos el libro a un mes, lo consideramos como leido y, si el libro era un objetivo, decrecemos en 1 la cantidad de objetivos por leer.

```
Unset
  (:action leer_no_sucesor_
    :parameters (?libro - libro ?month - mes)
    :precondition (and
      (mes_actual ?month)
      (not (leido ?libro)) ;el libro no ha estado leido
      (not (es_sucesor ?libro))
    )

    :effect (and
      (leido_en_mes ?libro ?month)
      (leido ?libro)
      (when
        (objectiu ?libro)
        (decrease (libros_restantes) 1))
    )
  )
```

- leer_sucesor: dispone dos parámetros, un libro ?libro y un mes ?month. Comprobamos que el mes de nuestro parámetro sea el mes al que apuntamos, que el libro que pretendemos leer no haya sido asignado/leído ya, que dicho libro tenga predecesores y que estos predecesores los hayamos leído/asignado y que esta asignación haya sido en un mes anterior. Los efectos son los mismos que en la acción anterior.

```
Unset
  (:action leer_sucesor
    :parameters (?libro - libro ?month - mes)
    :precondition (and
```

```
      (mes_actual ?month)
      (not (leido ?libro)) ; el libro no ha sido leído
      (es_sucesor ?libro) ; el libro es sucesor
      (forall
        (?l1 - libro)
          (imply
            (predecesor ?l1 ?libro)
            (and (leido ?l1)
              (not (leido_en_mes ?l1 ?month))
               ) ; que se lea en un mes diferente a aquellos en los que se leyeron sus
libros predecesores.
          )
        )
      )
    :effect (and
      (leido_en_mes ?libro ?month)
      (leido ?libro)
      (when
        (objectiu ?libro)
        (decrease (libros_restantes) 1))
    )
  )
```

- canviar_mes: tenemos dos parámetros, ?acual_month y ?next_month y comprobamos que ?acual_month sea el mes actual y que ?next_month sea el mes siguiente, a continuación del actual. Como efecto, negamos que el cursor este en acual_month y lo movemos a next_month, es decir, esta es la acción encargada de mover de mes en mes nuestro cursor.

Unset

```
  (:action canviar_mes
      :parameters (?acual_month ?next_month - mes)
      :precondition (and
          (mes_actual ?acual_month)
```

```
            (siguiente_mes ?acual_month ?next_month))

        :effect (and
                (not (mes_actual ?acual_month)) ; Elimina el valor
actual de mes_actual
                (mes_actual ?next_month) ; Añade el nuevo valor a
mes_actual
        )
```

Además, tenemos el fluente libros_restantes que en el estado inicial declararemos como igual al número de objetivos que tenemos y que en cada acción de lectura decrementaremos en 1 si el libro que hemos asignado era uno de los objetivos. Además, indicamos al algoritmo que este flotante es una métrica a minimizar(lo cual nos brinda múltiples ventajas).

3.2.1 Juegos de pruebas

3.2.1.1 Objetivo en un punto intermedio de una secuencia

3.2.1.1.1 ¿Qué queremos comprobar?

Queremos comprobar que si tenemos una secuencia de, como mínimo, tres libros, si el objetivo a leer es libro 2 (de una secuencia de libros 1, libro 2 y libro 3) se lea correctamente libro 2 y sus predecesores (en el caso del ejemplo libro 1).
Tenemos dos objetivos: LL_4 y LL_1_2 nuestro objetivo en un punto intermedio de una secuencia(LL_1,LL_1_2,LL_1_3).

3.2.1.1.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step 0: LEER_NO_SUCESOR_ LL_4 ENERO
1: LEER_NO_SUCESOR_ LL_1 ENERO
2: CANVIAR_MES ENERO FEBRERO
3: LEER_SUCESOR LL_1_2 FEBRERO

time spent: 0.00 seconds instantiating 107 easy, 24 hard action templates
0.00 seconds reachability analysis, yielding 272 facts and 131 actions
0.00 seconds creating final representation with 272 relevant facts, 2 relevant fluents
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 8 states, to a max depth of 2
0.00 seconds total time

Como podemos observar, hemos asignado a nuestro primer mes de manera correcta al predecesor de nuestro libro objetivo y a nuestro objetivo sin predecesores para, tras cambiar de mes, asignar nuestro objetivo de una secuencia intermedia no leyendo, de manera correcta, el tercer elemento de la sucesión. En cuanto a nodos recorridos, tenemos una cantidad responsable de 102 pero hemos evaluado únicamente 8 además de que la profundidad máxima a la que se ha visto forzado ir es de 2, lo cual es todavía razonable.

3.2.1.1.3 Conclusión

El código permite al planificador manejar correctamente esta circunstancia.

3.2.1.2 Objetivo elemento final de secuencia de 5 libros:

3.2.1.2.1 ¿Qué queremos comprobar?

Queremos comprobar que si nuestro objetivo es el quinto libro de una secuencia de 5 libros, asignemos todos sus predecesores correctamente en meses anteriores.
Tenemos 1 objetivo, LL_1_5 el objetivo final de una secuencia. (LL_1,LL_1_2,LL_1_3,LL_1_4, LL_1_5).

3.2.1.2.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step 0: LEER_NO_SUCESOR_ LL_1 ENERO
1: CANVIAR_MES ENERO FEBRERO

2: LEER_SUCESOR LL_1_2 FEBRERO

3: CANVIAR_MES FEBRERO MARZO

4: LEER_SUCESOR LL_1_3 MARZO

5: CANVIAR_MES MARZO ABRIL

6: LEER_SUCESOR LL_1_4 ABRIL

7: CANVIAR_MES ABRIL MAYO

8: LEER_SUCESOR LL_1_5 MAYO

time spent: 0.00 seconds instantiating 23 easy, 48 hard action templates

0.00 seconds reachability analysis, yielding 142 facts and 71 actions

0.00 seconds creating final representation with 142 relevant facts, 2 relevant fluents

0.00 seconds computing LNF

0.00 seconds building connectivity graph

0.00 seconds searching, evaluating 10 states, to a max depth of 1

0.00 seconds total time

Como podemos observar, hemos asignado correctamente nuestros predecesores, asignando el primer elemento de nuesta secuencia al primer mes, el segundo al segundo… hasta llegar a nuestro objetivo final garantizando, así, que todos los predecesores esten en meses anteriores al libro al que preceden.

En cuanto a nodos recorridos, hemos recorrido muchos menos(23) pero hemos evaluado más estados, 10, sin embargo la profundidad máxima ha sido de solamente .

3.2.1.2.3 Conclusión

El código permite al planificador manejar correctamente esta circunstancia.

3.3 Extensión 2

Esta versión es muy parecida a la extensión 1 con algunas modificaciones para añadir los libros paralelos. Hemos añadido dos nuevos predicados:

- paralelo ?l1 ?l2 - libro: tiene la misma función, e importancia, que su homólogo predecesor ?l1 ?l2. Este predicado nos permite establecer que ?l1 es paralelo a ?l2. Como se explica en apartados anteriores, no es bidireccional.
- es_paralelo ?l -libro: tiene la misma función y objetivos que su homólogo es_sucesor ?l con una pequeña diferencia: mientras sucesor solo nos indicaba que ?l TIENE predecesores es_paralelo nos indica que el libro es paralelo. Un sencillo ejemplo nos permite ver mejor la diferencia: si tenemos una secuencia de l1-l2, donde declaramos predecesor l1 l2, debemos declarar también es_sucesor l2. Sin embargo, si tuvieramos que paralelo l1 l2, deberíamos declarar es_paralelo l1 y es_paralelo l2.

En cuanto a acciones, tenemos:

- leer_no_sucesor_no_paralel: exactamente igual que la acción explicada anteriormente leer_no_sucesor_ con la única diferencia de que(además del nombre) añadimos como precondición que el libro que queremos asignar ?libro no debe ser un paralelo: (not (es_paralelo ?libro))

Unset

```
(:action leer_no_sucesor_no_paralel
    :parameters (?libro - libro ?month - mes)
    :precondition (and
        (mes_actual ?month)
        (not (leido ?libro)) ;el libro no ha estado leido
        (not (es_sucesor ?libro))
        (not (es_paralelo ?libro))
    )

    :effect (and
        (leido_en_mes ?libro ?month)
        (leido ?libro)
        (when
            (objectiu ?libro)
            (decrease (libros_restantes) 1))
    )
```

```
    )
```

- leer_sucesor_y_paralelo1: muy similar a la acción leer_sucesor con un par de diferencias: añadimos que ?libro debe ser o sucesor o debe ser paralelo (or(es_sucesor ?libro) (es_paralelo ?libro)) y añadiendo un imply al for all que comprueba que el libro que es paralelo al nuestro haya sido leído o antes o ahora(como comentamos anteriormente, simplificamos el libro paralelo a que lo leamos antes o en el mismo mes pero no en el siguiente).

Unset

```
      (:action leer_sucesor_y_paralelo1
       :parameters (?libro - libro ?month - mes ?pas_month - mes)
       :precondition (and
           (mes_actual ?month)
           (not (leido ?libro)) ; el libro no ha sido leído
            (siguiente_mes ?pas_month ?month) ; Identificar el mes
pasado de ?month
           (or(es_sucesor ?libro)
               (es_paralelo ?libro))
           (forall
               (?l1 - libro)
               (and
                   (imply
                       (paralelo ?l1 ?libro)
                       (and (leido ?l1)
                           (or (leido_en_mes ?l1 ?month)
                               (leido_en_mes ?l1 ?pas_month)
                           ))
                   )

                   (imply
```

```
                        (predecesor ?l1 ?libro)
                        (and (leido ?l1)
                            (not (leido_en_mes ?l1 ?month))
                             ) ; que se lea en un mes diferente a
aquellos en los que se leyeron sus libros predecesores.
                    )
                )
            )
        )
        :effect (and
            (leido_en_mes ?libro ?month)
            (leido ?libro)
            (when
                (objectiu ?libro)
                (decrease (libros_restantes) 1))
        )
    )
```

- canviar_mes no padece ninguna modificación.

El flotante que usamos tampoco sufre ningún cambio.

3.3.1 Juegos de pruebas

3.3.1.1 Predecesores y paralelos de un libro no objetivo

3.3.1.1.1 ¿Qué queremos comprobar?

Queremos comprobar que los predecesores, o paralelos, de un libro que NO es objetivo NO se lean.

En el estado inicial creado para la comprobación tenemos como objetivos dos libros y hemos comentado el tercer objetivo que correspondería al libro con predecesores y paralelos Los dos objetivos del test son LL_5 y LL_2_1.

Los libros que no deben aparecer son LL_1, LL_4,LL_1_2, LL_1_3 y LL_99.

3.3.1.1.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step 0: LEER_NO_SUCESOR_NO_PARALEL LL_5 ENERO
1: LEER_NO_SUCESOR_NO_PARALEL LL_2_1 ENERO

time spent: 0.00 seconds instantiating 95 easy, 44 hard action templates
0.00 seconds reachability analysis, yielding 269 facts and 138 actions
0.00 seconds creating final representation with 266 relevant facts, 2 relevant fluents
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 3 states, to a max depth of 1
0.00 seconds total time

El resultado es correcto ya que leemos nuestros dos objetivos(LL_2_1 y LL_5), correctamente y los libros predecesores y paralelos de los no objetivos los hemos ignorado correctamente. Podemos observar que recorremos bastantes nodos pero evaluamos pocos estados(3) y, además la profundidad máxima a la que nos vemos forzados a ir es de 1, es decir, un buen resultado.

3.3.1.1.3 Conclusión

El código permite al planificador manejar esta situación correctamente.

3.3.2 Libro predecesor y paralelo al mismo tiempo

3.3.2.1 ¿Qué queremos comprobar?

Queremos comprobar que si tenemos un libro que es predecesor de un libro y paralelo de otro libro, distinto, y ambos(el libro del que es predecesor y del que es paralelo) son objetivos, leamos este libro predecesor y paralelo correctamente(una única vez y cumpliendo las restricciones).
En la prueba que hago hay 2 objetivos: LL_4 y LL_99 que tiene predecesores LL_1 y LL_1_2 donde este es a la vez un predecesor de LL_99 y paralelo a LL_4.

3.3.2.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step 0: LEER_NO_SUCESOR_NO_PARALEL LL_1 ENERO
1: CANVIAR_MES ENERO FEBRERO
2: LEER_SUCESOR_Y_PARALELO1 LL_1_2 FEBRERO ENERO
3: CANVIAR_MES FEBRERO MARZO
4: LEER_SUCESOR_Y_PARALELO1 LL_4 MARZO FEBRERO
5: LEER_SUCESOR_Y_PARALELO1 LL_99 MARZO FEBRERO

time spent: 0.00 seconds instantiating 95 easy, 44 hard action templates
0.00 seconds reachability analysis, yielding 269 facts and 138 actions
0.00 seconds creating final representation with 266 relevant facts, 2 relevant fluents
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 8 states, to a max depth of 1
0.00 seconds total time

Como puede observarse, el resultado es adecuado y se ajusta a las restricciones del problema: asignamos al primer mes el primero de los libros de la secuencia de predecesores que tenemos, cambiamos de mes y a continuación ponemos el libro que es a la vez predecesor y paralelo para, en el siguiente mes, asignar el libro del que es paralelo y el libro del que es predecesor, es decir, asignamos este libro predecesor y paralelo una única vez correctamente cumpliendo con todas las restricciones. En cuanto a nodos recorridos, hemos recorrido 95 y evaluado unos pocos estados(8) además de que la profundidad máxima a la que ha tenido que ir es de solamente 1, es decir, un buen resultado que tiene una profundidad máxima excelente y que ha evaluado muy pocos estados.

3.3.2.3 Conclusión

El código permite al planificador manejar correctamente esta circunstancia.

4. JUEGOS DE PRUEBAS FINAL (EXTENSIÓN 3)

En esta fase del proyecto, hemos implementado dos enfoques distintos de pruebas para evaluar nuestro código, cada uno enfocado en un aspecto específico del rendimiento y la fiabilidad del mismo:

Juegos de Pruebas Manuales: Estas pruebas son diseñadas y ejecutadas manualmente con el objetivo de verificar el dominio de nuestro trabajo. Al seleccionar y aplicar casos de prueba específicos, buscamos evaluar la precisión del código en escenarios controlados. Este enfoque nos permite una comprensión profunda de cómo se comporta el código en condiciones conocidas y predecibles, asegurando que cada función cumple con su propósito de manera efectiva.

Juegos de Pruebas Aleatorios: Por otro lado, las pruebas aleatorias están diseñadas para evaluar la eficacia global del código. Mediante la generación de casos de prueba aleatorios, ponemos a prueba la capacidad del código de manejar una variedad de situaciones cada vez más complejas.

4.1. Comprobaciones con el Juego de Pruebas a Mano

4.1.1 Solución vacía

4.1.1.1 ¿Qué queremos comprobar?

Queremos comprobar que si no tenemos ningún libro que queramos leer, ningún objetivo, el planificador devuelva una solución acorde: la solución vacía. Tendremos definidos una serie de libros, predecesores y paralelos pero ningún objetivo para realizar esta comprobación.

4.1.1.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step

time spent: 0.00 seconds instantiating 59 easy, 55 hard action templates
0.00 seconds reachability analysis, yielding 216 facts and 113 actions

0.00 seconds creating final representation with 212 relevant facts, 4 relevant fluents

0.00 seconds computing LNF

0.00 seconds building connectivity graph

0.00 seconds searching, evaluating 1 states, to a max depth of 0

0.00 seconds total time

El resultado es correcto, una solución vacía. Podemos observar, además, que no recorremos demasiados nodos ni evaluamos estados antes de llegar a la solución.

4.1.1.3 Conclusión

Un estado inicial careciente de objetivos a leer no nos causa ningún problema, simplemente retornamos una solución vacía, sin pasos a seguir.

4.1.2 Objetivo al inicio de una secuencia

4.1.2.1 ¿Qué queremos comprobar?

Queremos comprobar que si tenemos una secuencia de libros y queremos leer el primero, podamos leer este, y únicamente este, correctamente.
Tenemos tres objetivos, LL_5, LL_2_1 que tiene un predecesor LL_2 y LL_1 que es el inicio de una secuencia de tres libros(LL_1,LL_1_2,LL_99).

4.1.2.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step 0: LEER_NO_SUCESOR_NO_PARALEL LL_5 ENERO

1: LEER_NO_SUCESOR_NO_PARALEL LL_1 ENERO

2: LEER_NO_SUCESOR_NO_PARALEL LL_2 ENERO

3: CANVIAR_MES ENERO FEBRERO

4: LEER_SUCESOR_Y_PARALELO1 LL_2_1 FEBRERO ENERO

time spent: 0.00 seconds instantiating 59 easy, 55 hard action templates

0.00 seconds reachability analysis, yielding 216 facts and 113 actions

0.00 seconds creating final representation with 212 relevant facts, 4 relevant fluents

0.00 seconds computing LNF

0.00 seconds building connectivity graph

0.00 seconds searching, evaluating 10 states, to a max depth of 2

0.00 seconds total time

Como vemos se han asignado nuestros tres objetivos(LL_5, LL_1 y LL_2_1) y sus predecesores(solo LL_2_1 tiene un predecesor: LL_2 y lo hemos asignado correctamente en un mes anterior) y además nuestro inicio de secuencia, LL_1, ha sido asignado correctamente sin ninguna clase de problema y sin haber leído ningún libro de más. En cuanto a nodos recorridos, podemos apreciar que es prácticamente la misma cantidad con un pequeño aumento en estados evaluados que pasa de 8 a 10, además, hay que vigilar la profundidad máxima a la que nos vemos forzados a ir(2).

4.1.2.3 Conclusión

El código permite al planificador manejar correctamente esta circunstancia.

4.1.3 Libro intermedio de una secuencia ya leída en el estado inicial

4.1.3.1 ¿Qué queremos comprobar?

Queremos comprobar que, teniendo como mínimo una secuencia de tres libros(por ejemplo libro 1, libro 2 y libro 3) si hemos leído un libro intermedio de la secuencia(no es ni el primero ni el último) en el estado inicial, leamos únicamente los libros siguientes a este hasta llegar al objetivo a leer(si tenemos que en el estado inicial leímos libro 2 y nuestro objetivo a leer es libro 3, asignamos a libro 3 a un mes y listo).

Tenemos tres objetivos: LL_5, LL_99 y LL_2_1(con predecesor LL_2) donde LL_99 es el objetivo de una secuencia de libros LL_1, LL_1_2, LL_99 donde LL_1_2 lo tenemos como leído en el estado inicial. Debería leerse LL_99 sin tener que leer ningún predecesor y tampoco debería leerse LL_4, libro paralelo a LL_1_2.

4.1.3.2 ¿Es el resultado obtenido correcto?

ff: found legal plan as follows

step 0: LEER_NO_SUCESOR_NO_PARALEL LL_2 ENERO
1: CANVIAR_MES ENERO FEBRERO
2: LEER_NO_SUCESOR_NO_PARALEL LL_5 FEBRERO
3: CANVIAR_MES FEBRERO MARZO
4: LEER_SUCESOR_Y_PARALELO1 LL_2_1 MARZO FEBRERO
5: CANVIAR_MES MARZO ABRIL
6: LEER_SUCESOR_Y_PARALELO1 LL_99 ABRIL MARZO

time spent: 0.00 seconds instantiating 59 easy, 33 hard action templates
0.00 seconds reachability analysis, yielding 204 facts and 92 actions
0.00 seconds creating final representation with 188 relevant facts, 4 relevant fluents
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 10 states, to a max depth of 2
0.00 seconds total time

La solución es correcta, he leído los predecesores en meses anteriores a los objetivos y he leído LL_99 directamente como corresponde ya que su predecesor(LL_1_2) lo había leído ya el usuario(leído en el estado inicial). Esta vez, además, he recorrido menos nodos y únicamente he evaluado 10 estados con una profundidad máxima de 2 lo que es todavía un buen resultado.

4.1.3.3 Conclusión

El código permite al planificador manejar correctamente esta circunstancia.

4.1.4 Libro objetivo con más de 800 páginas

4.1.4.1 ¿Qué queremos comprobar?

Queremos comprobar que si uno de nuestros objetivos es un libro con más de 800 páginas, teniendo en cuenta que un mes solo puede tener 800 páginas como mucho, no halle ninguna solución.
Tenemos 1 objetivo: LL_1, que es un libro con 900 páginas, el resultado que debemos obtener es que es imposible encontrar un resultado.

4.1.4.2 ¿Es el resultado obtenido correcto?

ff: search configuration is EHC, if that fails then best-first on 1*g(s) + 5*h(s) where
   metric is plan length

Enforced Hill-climbing failed !
switching to Best-first Search now.

best first search space empty! problem proven unsolvable.

time spent: 0.00 seconds instantiating 23 easy, 0 hard action templates
0.00 seconds reachability analysis, yielding 26 facts and 23 actions
0.00 seconds creating final representation with 26 relevant facts, 4 relevant fluents
0.00 seconds computing LNF
0.00 seconds building connectivity graph
0.00 seconds searching, evaluating 2 states, to a max depth of 0
0.00 seconds total time

De manera acorde al estado inicial dado, no encuentra ninguna solución y, de hecho, apenas recorre nodos o evalúa estados.

4.1.4.3 Conclusión

Las restricciones de los flotantes de cantidades de páginas de libros y mensuales funcionan correctamente.

4.2. Juego de Pruebas aleatorios

En el marco de nuestro proyecto, hemos desarrollado una serie de herramientas automatizadas para generar y evaluar pruebas aleatorias. Todo el código relacionado con esta tarea se encuentra en el archivo generador.ipynb, ubicado dentro de la carpeta “juegos_prueba”. A continuación, detallamos las funciones clave implementadas para este propósito:

- **Función ejecutar_programa:** Esta función es fundamental para la automatización de nuestras pruebas. Al recibir el nombre de un archivo .pddl, ejecuta el programa correspondiente directamente en una celda de Jupyter Notebook en el entorno de Windows.
- **Clase llibre:** Esta clase ha sido diseñada para definir y manejar las propiedades de los libros. Gracias a esta abstracción, podemos gestionar de manera más eficiente y clara las características individuales de cada libro en nuestras pruebas.
- **Función dividir_lista_aleatoriamente:** Esta función toma una lista, en nuestro caso, una lista de libros, y la divide en subconjuntas más pequeñas. Estas subconjuntas no exceden de 12 elementos, representando así la secuencia de libros que se leerán en los meses. Este enfoque permite una simulación más realista de la distribución temporal de la lectura.
- **Función asignar_paginas_equitativamente:** Esta función asigna un número de páginas a cada libro de manera que se respeten las restricciones de máximo de páginas leídas por mes, asegurando así que todas las soluciones generadas sean válidas.
- **Función generar_caso:** Esta función crea un archivo .pddl de manera aleatoria para nuestros experimentos. Mediante este método, podemos generar una amplia gama de escenarios de prueba.

- **Función realizar_estudio:** Con esta función, generamos y ejecutamos ficheros .pddl para observar la evolución del tiempo de ejecución a medida que incrementamos el número de libros.

4.2.1 Experimento predecesor

En este experimento, nuestro enfoque se centra en analizar la variación del tiempo de computación de nuestro programa a medida que incrementamos la cantidad de libros, bajo la premisa específica de que estos libros forman una secuencia exclusiva de predecesores.

4.2.1.1 Método

Para garantizar la rigurosidad científica, hemos adoptado un enfoque metodológico que enfatiza la replicabilidad y la consistencia. Nuestro protocolo está diseñado para asegurar que cada experimento se ejecute bajo condiciones uniformes y controladas.

4.2.1.1.1 Preparación del entorno experimental

Consistencia: Hemos establecido un entorno experimental controlado y estandarizado para cada réplica del experimento. Este enfoque minimiza las variaciones externas que podrían influir en los resultados, garantizando así la fiabilidad de los datos recopilados.

4.2.1.1.2 Selección de semillas

Para cada conjunto de pruebas, hemos seleccionado 15 semillas aleatorias. Esta diversidad en las semillas asegura una variabilidad adecuada en las inicializaciones de los experimentos, proporcionando así una perspectiva amplia y representativa del rendimiento del sistema.

4.2.1.1.3 Ejecución de experimentos

En la fase de ejecución, llevaremos a cabo los siguientes pasos:

- **Inicialización y Progresión:** Comenzaremos con un solo libro y aumentaremos la cantidad de manera incremental, de dos en dos, hasta alcanzar un total de 40 libros

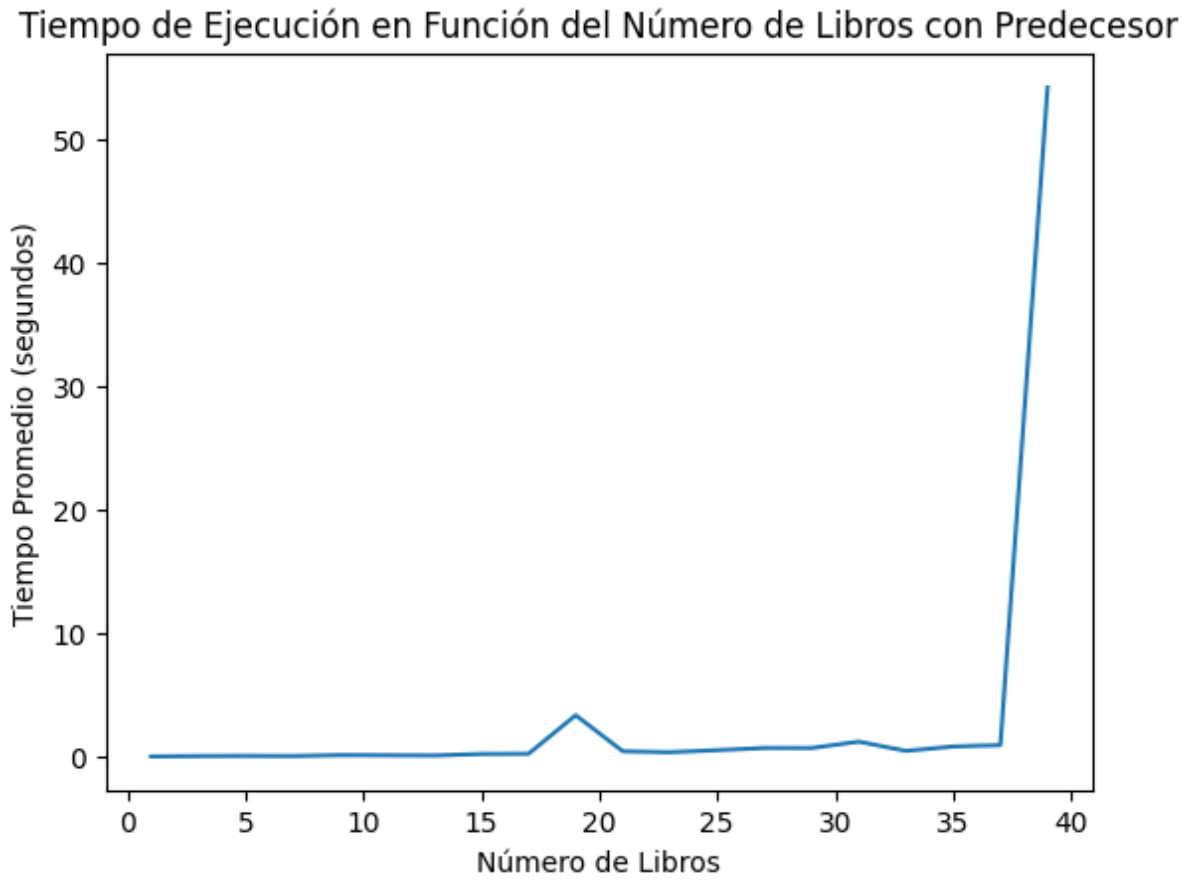
- **Generación de Problemas:** Crearemos 15 conjuntos de problemas (archivos .pddl) para cada cantidad de libros.
- **Especificación de Libros:** Cada libro tendrá un número específico de páginas asignadas.
- **Tipo de Libros:** Utilizaremos exclusivamente libros que sigan una secuencia de predecesores.
- **Objetivos:** Todos los libros serán tratados como objetivos en el contexto del experimento.

4.2.1.1.4 Parámetros de medición

Los siguientes parámetros serán esenciales para la evaluación y análisis de los resultados:

- **Número de Libros:** Se documentará el número de libros utilizados en cada conjunto de problemas.
- **Tiempo de Ejecución:** Registraremos el tiempo de ejecución para cada conjunto de problemas, lo que nos permitirá observar cómo varía el rendimiento con el incremento en el número de libros.

4.2.1.2 Resultados del experimento



Los resultados obtenidos del experimento Predecesor se presentan en la figura adjunta, donde se ilustra la relación entre el número de libros y el tiempo de ejecución promedio necesario para completar el procesamiento de los archivos .pddl correspondientes. Como puede observarse en el gráfico, el tiempo de ejecución permanece relativamente estable y bajo para un número de libros que oscila entre 1 y 30. Sin embargo, se identifica un aumento significativo y abrupto en el tiempo de ejecución promedio al llegar a un umbral cercano a los 40 libros.

Este pico en el tiempo de ejecución sugiere un punto de inflexión en la complejidad computacional relacionada con la secuencia de predecesores. Es importante destacar que hasta el número de libros mencionado, el sistema demuestra una eficiencia notable, manteniendo tiempos de respuesta dentro de un rango manejable.

4.2.2 Experimento Paralelo

El Experimento Paralelo tiene como objetivo investigar cómo se comporta el tiempo de computación a medida que incrementamos el número de libros, bajo la condición de que dichos libros se procesan en paralelo.

4.2.2.1 Método

La metodología empleada en el Experimento Paralelo sigue un procedimiento riguroso y sistemático para garantizar la validez y la replicabilidad de los resultados. Nuestro enfoque metodológico es detallado a continuación.

4.2.2.1.1 Preparación del entorno experimental

Consistencia: Se ha establecido un entorno de prueba controlado, asegurando que todas las variables externas se mantengan constantes a lo largo de las distintas ejecuciones. Esta estandarización es vital para asegurar que cualquier variación en el tiempo de ejecución sea atribuible únicamente a la manipulación controlada del número de libros.

4.2.2.1.2 Selección de semillas

Hemos seleccionado 15 semillas de manera aleatoria para fomentar la variabilidad y la imparcialidad en los resultados. Esta diversidad en las semillas de inicialización es crucial para obtener una evaluación general del rendimiento y evitar sesgos en los resultados del experimento.

4.2.2.1.3 Ejecución de experimentos

Durante la ejecución del experimento, se llevarán a cabo las siguientes acciones:

- **Inicialización y Progresión:** Comenzaremos con un solo libro y aumentaremos la cantidad de manera incremental, de tres en tres, hasta alcanzar un total de 29 libros
- **Conjuntos de Problemas:** Se generarán y ejecutarán 15 conjuntos de problemas (archivos .pddl) para cada punto de datos correspondiente al número de libros.
- **Asignación de Páginas:** A cada libro se le asignará un número específico de páginas, manteniendo la consistencia en las variables de control.

- **Modo Paralelo:** Se utilizarán únicamente libros que serán procesados en modo paralelo.
- **Objetivos:** Todos los libros serán considerados como objetivos dentro de la configuración del experimento.

4.2.2.1.4 Parámetros de medición

Los parámetros que se medirán en el Experimento Paralelo son los siguientes:

- **Número de Libros:** Documentaremos el número de libros presentes en cada conjunto de problemas para correlacionarlos con el tiempo de ejecución.
- **Tiempo de Ejecución:** El tiempo de ejecución será registrado meticulosamente para cada uno de los conjuntos de problemas, proporcionando así un análisis detallado del impacto del procesamiento paralelo en el rendimiento.

4.2.2.2 Resultados del experimento

Los resultados del Experimento Paralelo se presentan en la gráfica, la cual muestra la curva del tiempo de ejecución promedio en relación con el incremento progresivo

del número de libros. La secuencia de datos comienza con un solo libro y aumenta en incrementos de dos, culminando en 28 libros.

El gráfico revela un patrón de comportamiento interesante; el tiempo de ejecución se mantiene en niveles bajos y estables para pequeños conjuntos de libros. Sin embargo, observamos ciertas fluctuaciones en el rendimiento que requieren un análisis adicional para comprender las causas. Al acercarnos a la marca de 20 libros, se percibe una disminución notable en el tiempo de ejecución, lo cual podría indicar una optimización efectiva o un umbral de eficiencia en el procesamiento paralelo.

A medida que continuamos incrementando el número de libros, se registra un aumento drástico en el tiempo de ejecución alrededor del punto de 25 libros, lo que sugiere un límite en la capacidad de procesamiento paralelo del sistema o posibles cuellos de botella en la gestión de recursos. Este súbito incremento se convierte en un punto de inflexión que refleja una probable saturación en la capacidad de procesamiento concurrente, lo que resulta en tiempos de ejecución prolongados.

5. REFLEXIONES Y RESPUESTAS

En base a la exhaustiva evaluación de nuestra solución al problema planteado podemos afirmar, con confianza, que hemos creado una solución eficiente y rápida. Esto lo afirmamos apoyado en la rapidez con la que hemos podido observar que nuestro código permite al planificador abordar cantidades significativas de libros, lo que necesariamente demuestra, también, la capacidad de nuestro enfoque para escalar de manera efectiva, cosa crucial en posibles escenarios con muchos libros.

La gran eficiencia lograda se atribuye en gran medida a las optimizaciones implementadas a medida que construíamos nuestra solución final. Estas optimizaciones han mejorado la velocidad de ejecución a base de conseguir precondiciones más eficientes que permiten al algoritmo tener que hacer menos comprobaciones y, también, permiten ver al algoritmo rápidamente si una unificación hecha no es válida.

Además, al realizar una revisión de posibles fuentes de error, no hemos identificado que el código cause que el planificador retorne errores ni soluciones incorrectas. Esta robustez ante posibles errores subraya la solidez y confiabilidad de nuestra solución lo que nos permite afirmar que, en conjunto, consideramos que nuestro dominio definido, junto a la estructura de problema definido en pddl, constituye una buena respuesta al problema planteado. Asimismo, las restricciones y simplificaciones que hemos aplicado han demostrado ser apropiadas y necesarias para abordar el problema de manera práctica, sin comprometer la integridad ni la precisión de la solución final.

En resumen, nuestra solución no solo cumple con los requisitos específicos del problema, sino que también obtiene un buen rendimiento, que se constata en un escalado notable, y su resistencia frente a posibles fuentes de error.