Facultad de Informática de Barcelona, UPC

Planificación de menús de restaurantes con PDDL

Autores:
Eulàlia Peiret, Marc Teixidó y Eudald Pizarro

Profesor: Carles Fenollosa

Q2 2024-25

Índice

1 Identificación del problema

Se nos pide hacer un programa con el que se pueda planificar el menú semanal de un restaurante. El menú tiene que estar compuesto por los platos que puede cocinar el chef del restaurante que nos lo solicita, y debe respetar incompatibilidades entre platos del mismo día o de días consecutivos. Todo esto debe hacerse manteniendo una dieta saludable y equilibrada para los clientes del restaurante.

1.1 Justificación del uso de PDDL

En este problema lo que queríamos es que, con la información de entrada que le proporcionáramos, el programa fuera capaz de planificar un menú para alcanzar el objetivo final, que es tener una dieta equilibrada y variada para los clientes del restaurante. Es decir, lo que nosotros queremos no es el estado final, sino los pasos intermedios: qué tenemos que servir cada día para alcanzar nuestro objetivo.

El problema se puede resolver con planificación, ya que este entorno cumple con las siguientes características:

- **Completamente observable**: el planificador es consciente de todo lo que pasa en el entorno, tiene toda la información sobre los diferentes elementos y las consecuencias que puede tener cada acción.
- **Determinista**: podemos definir todas las acciones. Podemos definir todos los platos que se pueden cocinar y todas sus características.
- **Finito**: no hay infinitos platos en el restaurante.
- **Estático**: los platos, sus características y los menús no cambian a menos que el agente planificador actúe sobre ellos.

2 Partes del Dominio

El dominio define los elementos generales de nuestro problema, es decir, el conjunto de tipos, predicados y operadores que utilizaremos para describir cualquier instancia del problema. Esta información se encuentra en el fichero de dominio (`domain6.pddl`).

2.1 Requerimientos

Se especifican las características del lenguaje PDDL que vamos a usar:

- `:strips` — Permite usar precondiciones y efectos simples.
- `:adl` — Permite precondiciones complejas como conjunciones, disyunciones y negaciones.
- `:typing` — Permite definir tipos de objetos.
- `:equality` — Permite usar comparaciones de igualdad.
- `:fluents` — Permite utilizar funciones numéricas que cambian a lo largo del plan (como calorías y precios).

2.2 Tipos

Definimos los tipos de objetos que manejamos:

- `dia` — Representa los días laborables.
- `plato` — Supertipo de los tipos:
  - `primero`
  - `segundo`

2.3 Predicados

Representan propiedades lógicas del estado del mundo y se usan para establecer precondiciones y efectos. A continuación se listan los predicados utilizados:

- `(hayprimero ?d - dia)` — Indica que el primer plato ha sido asignado al día `?d`.
- `(haysegundo ?d - dia)` — Indica que el segundo plato ha sido asignado al día `?d`.
- `(assignado-primero ?d - dia ?p - primero)` — El primer plato `?p` ha sido asignado al día `?d`.
- `(assignado-segundo ?d - dia ?s - segundo)` — El segundo plato `?s` ha sido asignado al día `?d`.
- `(assignado ?p - plato)` — El plato `?p` ya ha sido usado durante la semana.
- `(mismoTipo ?p1 ?p2 - plato)` — Los platos `?p1` y `?p2` son del mismo tipo (por ejemplo, arroces o sopas frías).
- `(diaConsecutivo ?d1 ?d2 - dia)` — El día `?d2` sigue inmediatamente después de `?d1`.
- `(diaInicial ?d - dia)` — Marca el primer día de planificación (lunes).
- `(incompatible ?p - primero ?s - segundo)` — El primer plato `?p` no es compatible con el segundo `?s`.
- `(impres ?p - primero ?s - segundo ?d - dia)` — Predicado para printar el primero y segundo asignado a un dia.

En la sección 4 (Completado de los Niveles) se explica cómo hemos creado la práctica fase a fase y por qué hemos añadido cada predicado a medida que incorporábamos nuevas funcionalidades.

2.4 Funciones (Fluentes)

Funciones numéricas que permiten el uso de valores que se pueden cambiar durante la ejecución del plan:

- `(calorias ?p - plato)` — Calorías de un plato.
- `(calorias-dia ?d - dia)` — Calorías totales consumidas en un día.
- `(preu ?p - plato)` — Precio de un plato.
- `(preuTotal)` — Acumulado del coste total de todos los platos asignados.

2.5 Acciones

Las acciones son las acciones de planificación que permiten cambiar el estado del mundo. Cada una tiene parámetros, precondiciones y efectos.

- **assignar-primero-inicial**: Asigna el primer plato en el día inicial. Asegura que el día no tiene aún primer plato y que no se repite el plato.
- **assignar-primero**: Asigna un primer plato en días no iniciales. Evita repetir platos y también evita repetir el mismo tipo de plato del día anterior.
- **assignar-segundo-inicial**: Asigna el segundo plato en el primer día, verificando que no es incompatible con el primero.
- **assignar-segundo**: Asigna el segundo plato en días posteriores. Verifica que no haya incompatibilidad con el primero, que el plato no se haya usado y que no se repita tipo respecto al día anterior.
- **printar**: Comprueba si un día tiene primero y segundo, y si es así se imprime.

Justificación de las acciones La diferenciación entre las acciones `assignar-primero-inicial` y `assignar-primero`, así como entre `assignar-segundo-inicial` y `assignar-segundo`, es esencial para manejar correctamente la secuencia temporal en la planificación. En particular, las acciones para días no iniciales dependen de la existencia de asignaciones en el día anterior para poder evaluar restricciones de tipo consecutivo. Si no existieran estas acciones iniciales específicas, no sería posible iniciar la asignación de platos, ya que las precondiciones exigirían que el día anterior tuviese platos asignados, generando un ciclo circular imposible de resolver.

Además, hemos optado por mantener las precondiciones y efectos de las acciones lo más simples y minimalistas posible. Esto evita la complejidad excesiva en las fórmulas, facilita el proceso de planificación y mejora la eficiencia computacional. A su vez, este diseño modular y claro facilita la extensión futura del dominio y la incorporación de nuevas restricciones o funcionalidades.

3 El Problema

Esta parte se define en un fichero separado del dominio (`problem6.pddl`) y contiene una instancia concreta del problema a resolver. Incluye los siguientes elementos:

3.1 Dominio

`GestMenu` — Especifica el dominio al que pertenece el problema.

3.2 Objetos

Se definen los objetos (instancias de tipos) como los días y los platos concretos que pueden aparecer durante la semana.

3.3 Estado Inicial

Contiene la información inicial del problema:

- Calorías y precios de cada plato.
- Valor inicial de calorías por día y del precio total (ambas inicializadas a 0).
- Predicados como incompatibilidades entre platos y relación entre platos del mismo tipo.
- Definición de los días consecutivos.

3.4 Objetivo

Queremos encontrar un menú semanal con las siguientes condiciones:

- Cada día (lunes a viernes) debe tener asignado un primer y un segundo plato.
- Los menús deben respetar las restricciones de calorías diarias (entre 1000 y 1500).
- Los platos no deben repetirse ni coincidir en tipo con el día anterior.
- Se deben evitar las incompatibilidades entre primeros y segundos.
- El menú debe ser impreso al finalizar la asignación de cada día.
- El menú puede tener platos obligatorios.

4 Completado de los niveles

Nuestro sistema es una búsqueda hacia adelante (*planificaión progresiva*). El estado inicial de la búsqueda es el estado inicial del problema y a través de las acciones, cambiamos la descripción del estado.

4.1 Nivel básico

En la primera versión se definen los diferentes días de la semana, así como diversas opciones de platos principales (primeros) y secundarios (segundos). También se especifican las incompatibilidades entre determinados primeros y segundos. El sistema es capaz de asignar un primer y un segundo plato a cada día, respetando dichas incompatibilidades.

4.2 Primera versión

En esta versión se introduce un nuevo predicado `asignado`, con el objetivo de evitar que un mismo plato se asigne más de una vez durante la semana. Para ello, se añade en la precondición de las acciones `assignarPrimero` y `assignarSegundo` que el plato a asignar no haya sido previamente utilizado.

4.3 Segunda versión

Para controlar que no se repitan platos del mismo tipo en días consecutivos, se definen los predicados `mismoTipo` y `diaConsecutivo`. En la inicialización del problema se indica qué platos son del mismo tipo (por ejemplo, `gazpacho` y `salmorejo`), y también se establece la relación de días consecutivos (por ejemplo, `lunes` es consecutivo a `martes`). Además, se define el primer día de la semana (por ejemplo, `lunes`) para poder aplicar las restricciones. Las acciones se modifican para incluir en su precondición que no se sirvan platos del mismo tipo en días consecutivos.

4.4 Tercera versión

Para esta versión, simplemente hay que añadir en la descripción del problema, en el goal, predicados tipo `asignado-primero( ?d - dia ?p - primero)`, o bien, `asignado-segundo( ?d - dia ?p - segundo)`, que nos permite obligar que haya estos platos en el menú sin modificar el dominio.

4.5 Cuarta versión

Para controlar las calorías de cada menú diario, se introducen fluentes. Se definen funciones que asignan un valor calórico a cada plato, así como una función que acumula las calorías totales por día. En cada acción, uno de los efectos es incrementar las calorías del día correspondiente, sumando las calorías del plato asignado. Las calorías diarias se inicializan a cero y, en la condición objetivo (`:goal`), se especifica que el total de calorías por día debe estar dentro de un rango razonable (entre 1000 y 1500 calorías).

4.6 Quinta versión

Finalmente, se añade la funcionalidad de calcular el precio de los platos y de los menús diarios. De manera análoga a las calorías, en cada acción se incrementa el precio total mediante las funciones `preu` (precio del plato) y `preuTotal` (precio acumulado). En la inicialización del problema se definen los precios individuales de todos los platos, y se establece `preuTotal` con un valor inicial de cero. Como objetivo adicional, se utiliza la instrucción `:metric` para minimizar el coste total del menú semanal:

```
(:metric minimize (preuTotal))
```

4.7 Generador de Juegos de Prueba

Para probar el sistema, tenemos un generador automático de problemas en Python. Utiliza la biblioteca `Faker`, concretamente el módulo `faker_food`, que permite generar nombres aleatorios de platos y alimentos. El generador crea automáticamente ficheros `.pddl` que definen instancias del problema para el dominio `GestMenu`. Cada problema incluye la inicialización de platos, calorías, precios, incompatibilidades y relaciones de tipo.

Se generan tres niveles de dificultad:

- **Easy**: Menú con 50 platos disponibles (25 primeros y 25 segundos).
- **Medium**: Menú con 40 platos en total.

- **Hard**: Menú con solo 30 platos (15 primeros y 15 segundos), lo que incrementa la probabilidad de conflictos e imposibilidad de planificación.

Este generador nos permite poner a prueba la robustez y capacidad de resolución de nuestro sistema en diferentes contextos.

5 Juegos de prueba

5.1 Juego de dificultad fácil

El fichero para reproducir esta prueba se llama `gestmenu-easy.pddl`, generado automáticamente por nuestro generador de pruebas. Contiene 25 primeros y 25 segundos, con las siguientes incompatibilidades entre primeros y segundos:

Primero	**Segundo incompatible**
tuna-sashimi	ebiten-maki
califlower-penne	ebiten-maki
tuna-sashimi	mushroom-risotto
tuna-sashimi	chicken-wings

Table 1: Incompatibilidades entre primeros y segundos (easy)

Se definen también 4 pares de platos del mismo tipo que no pueden servirse en días consecutivos:

Plato 1	**Plato 2 (mismo tipo)**
mushroom-risotto	salmon-nigiri
barbecue-ribs	lasagne
chilli-con-carne	pasta-carbonara
french-fries-with-sausages	teriyaki-chicken-donburi

Table 2: Restricciones de mismo tipo (easy)

Menú asignado:

Día	**Primero asignado**	**Segundo asignado**
Lunes	pork-belly-buns	katsu-curry
Martes	chicken-fajitas	pasta-carbonara
Miércoles	mushroom-risotto	french-fries-with-sausages
Jueves	vegetable-soup	ebiten-maki
Viernes	tuna-sashimi	meatballs-with-sauce

Table 3: Menú obtenido (easy)

Justificación: Este juego de prueba fácil permite comprobar que el sistema puede gestionar un volumen considerable de opciones manteniendo el cumplimiento de incompatibilidades y restric-

ciones de consecutividad. La alta variedad de platos asegura soluciones flexibles y sin repeticiones. Vemos que lo resuelve sin ningún problema.

5.2 Juego de dificultad media

El fichero para reproducir esta prueba se llama `gestmenu-medium.pddl` . Incluye 20 primeros y 20 segundos, con 5 incompatibilidades:

Primero	**Segundo incompatible**
fettuccine-alfredo	salmon-nigiri
califlower-penne	chilli-con-carne
caprese-salad	barbecue-ribs
chicken-milanese	tiramisù
pappardelle-alla-bolognese	salmon-nigiri

Table 4: Incompatibilidades (medio)

4 pares con restricción de mismo tipo:

Plato 1	**Plato 2 (mismo tipo)**
fish-and-chips	fettuccine-alfredo
pappardelle-alla-bolognese	pizza
salmon-nigiri	mushroom-risotto
mushroom-risotto	cheeseburger

Table 5: Restricciones de mismo tipo (medio)

Menú asignado:

Día	**Primero asignado**	**Segundo asignado**
Lunes	califlower-penne	mushroom-risotto
Martes	fish-and-chips	barbecue-ribs
Miércoles	philadelphia-maki	chilli-con-carne
Jueves	chicken-milanese	lasagne
Viernes	pork-belly-buns	cheeseburger

Table 6: Menú obtenido (medio)

Justificación: Este juego intermedio presenta más restricciones, reduciendo combinaciones y poniendo a prueba la capacidad del sistema para generar menús válidos con un catálogo más limitado. Como se puede ver en la última tabla, es capaz de generar un menu que cumpla con todas las restricciones.

5.3 Juego de dificultad alta

El fichero para reproducir esta prueba se llama `gestmenu-hard.pddl` . Con 15 primeros y 15 segundos, se añaden 5 incompatibilidades y 4 pares de platos con restricciones estrictas:

Primero	**Segundo incompatible**
ebiten-maki	fish-and-chips
katsu-curry	teriyaki-chicken-donburi
fish-and-chips	bruschette-with-tomato
meatballs-with-sauce	caesar-salad
salmon-nigiri	caesar-salad

Table 7: Incompatibilidades (difícil)

Plato 1	**Plato 2 (mismo tipo)**
tiramisù	chicken-milanese
fish-and-chips	chicken-fajitas
fettuccine-alfredo	fish-and-chips
bruschette-with-tomato	scotch-eggs

Table 8: Restricciones de mismo tipo (difícil)

Menú asignado:

Día	**Primero asignado**	**Segundo asignado**
Lunes	fish-and-chips	french-fries-with-sausages
Martes	philadelphia-maki	tiramis
Miércoles	scotch-eggs	california-maki
Jueves	salmon-nigiri	califlower-penne
Viernes	chicken-milanese	souvlaki

Table 9: Menú obtenido (difícil)

Justificación: El juego difícil simula un entorno con pocas opciones y múltiples restricciones, demostrando que el sistema es capaz de planificar menús válidos aún en condiciones complejas y limitantes. A pesar de las pocas opciones, se consigue un menú válido.