UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Planificación Inteligente

TRABAJO SIN

Master en Ingeniería Informática

Autor: Brian Valiente Rodenas

Tutor: Jaume Magí Jordán Prunera

Curso 2024-2025

Índice general

Índice general III
Índice de figuras III
Índice de tablas III
1 Dominio Transporte 1
1.1 Descripción del Problema . . . 1
1.2 Especificación del Contexto . . . 1
1.2.1 Estado Inicial . . . 1
1.2.2 Objetivo a Resolver . . . 1
1.3 Modelado del Conocimiento Experto . . . 2
1.3.1 Definición del Problema (problem.py) . . . 2
1.3.2 Definición del Dominio (domain.py) . . . 5
1.4 Resolución del Problema . . . 6
1.5 Expansión del Problema . . . 7
1.5.1 Instancia 1: Expansión de Recursos . . . 7
1.5.2 Instancia 2: Restricciones del Mismo Tipo Modificadas de Orden Excepto la Primera . . . 8
1.5.3 Instancia 3: Líneas de Transporte Adicional y Bus Gratis . . . 9
1.6 Conclusiones Generales . . . 10

Índice de figuras

1.1 Ejemplo del problema en cuestión. . . . 2
1.2 Modelización basada en tareas, métodos y operadores. . . . 2
1.3 Secuencia lógica de tareas: establecer límite, entregar paquetes, mover camiones, mover conductores. . . . 4

Índice de tablas

1.1 Clasificación de variables: información estática y dinámica. . . . 3

CAPÍTULO 1
Dominio Transporte

La planificación inteligente es un área fundamental en los sistemas inteligentes. En este documento, se abordará la planificación en un dominio de transporte, en el que se deben movilizar paquetes, camiones y conductores entre diferentes ciudades cumpliendo con ciertas restricciones.

1.1 Descripción del Problema

El problema se basa en un entorno con tres ciudades: C0, C1 y C2, además de dos puntos intermedios (P01, P12). Se tienen camiones, conductores y paquetes que deben ser transportados a sus destinos finales siguiendo ciertas reglas:

- Los camiones pueden moverse solo por carreteras y requieren de un conductor para desplazarse.
- Los conductores pueden moverse caminando o utilizando el autobús público, con un coste asociado con las variables `COST_BUS` y `COST_WALK`, usando los puntos intermedios.
- Los paquetes deben ser transportados a su destino dentro de un camión. Un camión solo puede transportar un paquete.

1.2 Especificación del Contexto

1.2.1. Estado Inicial

- El conductor D1 inicia en el punto P01 y D2 inicia en C1.
- El camión T1 inicia en la ciudad C1 y T2 inicia en C0.
- El paquete P1 está en la ciudad C0 y P2 está en la ciudad C0.

1.2.2. Objetivo a Resolver

- (Añadido) El coste límite final de los movimientos de caminar y bus es 5.
- El conductor D1 debe finalizar en C0.
- El camión T1 debe finalizar en C0.

- El paquete P1 debe llegar a C1 y P2 debe llegar a C2.

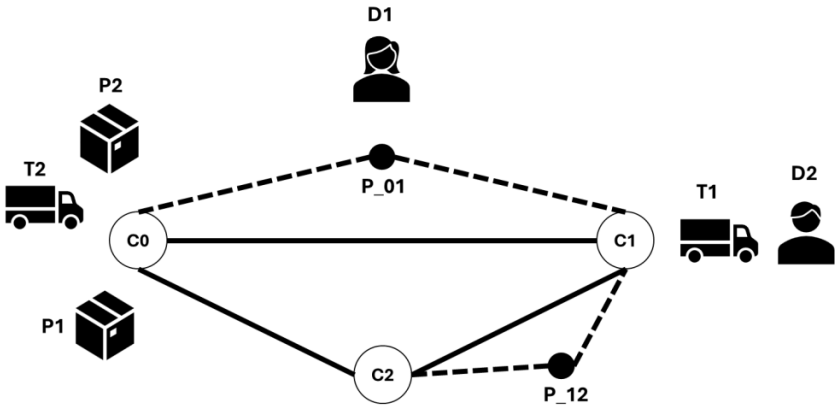

Figura 1.1: Ejemplo del problema en cuestión.
Fuente: elaboración propia.

1.3 Modelado del Conocimiento Experto

Para resolver el problema se utilizará una modelización basada en tareas, métodos y operadores. A continuación, se presenta una representación gráfica del modelo de planificación:

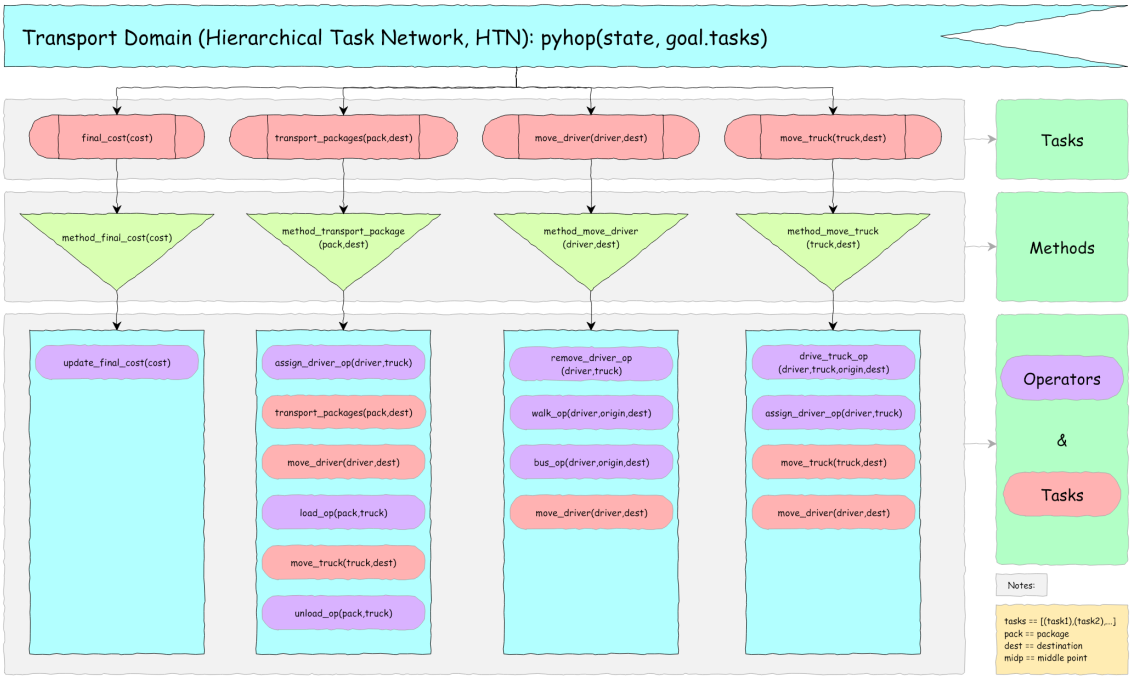

Figura 1.2: Modelización basada en tareas, métodos y operadores.
Fuente: elaboración propia.

Para esta implementación se han desarrollado dos archivos principales: `problem.py` y `domain.py`.

1.3.1. Definición del Problema (problem.py)

Este archivo define el estado inicial y los objetivos del problema. Las principales variables incluidas son:

- **Ciudades:** 'C0', 'C1', 'C2' listados en `state.cities`.
- **Conductores:** 'D1', 'D2' listados en `state.drivers`.
- **Camiones:** 'T1', 'T2' listados en `state.trucks`.
- **Paquetes:** 'P1', 'P2' listados en `state.packages`.
- **Ubicaciones iniciales:** diccionario que define la ubicación inicial de cada objeto `state.loc[obj][pos]`.
- **Relación de conductores y camiones:** diccionario que indica qué conductor tiene asignado cada camión `state.driver_of[truck][driver]`.
- **Ubicación de paquetes:** diccionario que define en qué camión se encuentra cada paquete `state.pack_in[package][truck]`.
- **Mapas de carreteras y caminos:**
  - `state.roadmap[city][list of cities]`: Define conexiones entre ciudades para los camiones.
  - `state.footmap[initial point][list of end points]`: Define caminos y rutas de autobús para los conductores.
- **Costo inicial:** Se define un costo inicial de `state.cost = 0`, que se incrementa a medida que se ejecutan los operadores de `walk_op` o `bus_op` y se actualiza con el coste `COST_WALK` o `COST_BUS`.

Variable	Tipo	Descripción
`state.cities`	Estática	Lista de ciudades del dominio
`state.drivers`	Estática	Lista de conductores
`state.trucks`	Estática	Lista de camiones
`state.packages`	Estática	Lista de paquetes
`state.loc`	Dinámica	Ubicación actual de cada objeto
`state.driver_of`	Dinámica	Asignación de conductores a camiones
`state.pack_in`	Dinámica	Asignación de paquetes a camiones
`state.roadmap`	Estática	Conexiones de carreteras entre ciudades
`state.footmap`	Estática	Rutas de caminos y autobús para conductores
`state.cost`	Dinámica	Coste acumulado de las acciones
`state.limit_cost`	Estática	Límite máximo de coste permitido
`COST_WALK`	Estática	Variable global del coste de caminar
`COST_BUS`	Estática	Variable global del coste de ir en bus

Tabla 1.1: Clasificación de variables: información estática y dinámica.
Fuente: elaboración propia.

La elección de representar el estado mediante diccionarios se justifica por la eficiencia y flexibilidad que ofrece para acceder y actualizar la información de cada objeto.

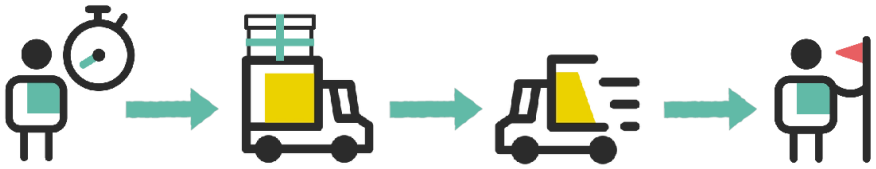

Figura 1.3: Secuencia lógica de tareas: establecer límite, entregar paquetes, mover camiones, mover conductores.
Fuente: elaboración propia.

- **Tareas objetivo:** Se ha definido el objetivo de alto nivel mediante una secuencia lógica de tareas (la lista `goal.tasks`). Antes de empezar, establecemos el límite de coste de desplazamiento de los conductores. A continuación, se planifica el transporte de los paquetes a sus destinos. Seguidamente, se aseguran los movimientos finales de los camiones. Finalmente, los conductores regresan a su ubicación final (por ejemplo, a su casa).

  Estas tareas son:

  - `final_cost (X)`: Garantiza que el costo total del plan no supere el valor 'X'. En este caso, cuando se ejecute esta tarea, su respectivo método y el operador asociado, se creará de forma dinámica (en tiempo de ejecución) una nueva variable en el estado `state.limit_cost`, sirve para comprobar que no se superen los límites objetivos.
  - `transport_package (P, destination)`: Transportar el paquete 'P' a la ciudad destino, asegurando que se cargue y descargue correctamente de un camión adecuado.
  - `move_truck (T, destination)`: Mover el camión 'T' a la ciudad destino, dejándolo en su posición final después de haber entregado los paquetes.
  - `move_driver (D, destination)`: Mover el conductor 'D' a la ciudad destino, permitiéndole regresar a su ubicación final después de completar las entregas.

  Este orden secuencial, en el que se inicia con la tarea `final_cost` para establecer el límite de coste, garantiza que la restricción fundamental se defina antes de ejecutar otras acciones que puedan afectar el coste acumulado. De esta forma, se asegura que los recursos se asignen y utilicen de manera coherente a lo largo de la ejecución del plan.

  Es importante destacar que el orden de las tareas es obligatorio para que el plan tenga sentido y éxito. Si deseamos que D3 finalice en C0, si antes debemos transportar paquetes, primero irán estas tareas, luego el movimiento de los camiones, y finalmente los conductores bajaran de los camiones para volver a casa. Entre las tareas del mismo tipo, se pueden especificar en el orden que se considere más conveniente, siempre y cuando se respeten las dependencias y requisitos entre el resto.

1.3.2. Definición del Dominio (domain.py)

Este archivo define las tareas, métodos y operadores del problema, además de una función auxiliar.

Operadores

Los operadores son acciones atómicas que modifican el estado del sistema. Cada uno de ellos verifica condiciones específicas antes de efectuar cambios. Se definen los siguientes:

- `assign_driver_op(state, driver, truck)`: Asigna al conductor `driver` al camión `truck` si ambos se encuentran en la misma ubicación y el camión no tiene conductor asignado.
- `remove_driver_op(state, driver, truck)`: Remueve al conductor `driver` del camión `truck` si se encuentran juntos.
- `drive_truck_op(state, truck, city_from, city_to)`: Mueve el camión `truck` de la ciudad `city_from` a `city_to` siempre que el camión tenga asignado un conductor y exista conexión en la `roadmap`.
- `walk_op(state, driver, city_from, city_to)`: Permite que el conductor `driver` camine de `city_from` a `city_to`, incrementando el coste en `COST_WALK` sin superar `state.limit_cost`.
- `bus_op(state, driver, city_from, city_to)`: Realiza el desplazamiento en autobús, incrementando el coste en `COST_BUS` sin superar la restricción de coste.
- `load_op(state, package, truck)`: Carga el paquete `package` en el camión `truck` si ambos se encuentran en la misma ubicación y el paquete no está cargado.
- `unload_op(state, package, truck)`: Descarga el paquete `package` del camión `truck` y actualiza su ubicación a la ciudad del camión.
- `update_final_cost(state, cost)`: Establece el límite de coste `state.limit_cost` para el plan, garantizando que el coste acumulado no supere el valor especificado.

Métodos

Los métodos descomponen tareas complejas en secuencias de acciones (sub-tareas) que, ejecutadas en orden, permiten alcanzar los objetivos. Se destacan los siguientes:

- `method_final_cost(state, cost)`: Implementa la tarea `final_cost` retornando la acción para actualizar el límite de coste mediante el operador `update_final_cost`.
- `method_move_truck(state, truck, city_dest)`: Define la estrategia para mover el camión `truck` a la ciudad destino `city_dest`. Verifica si el camión ya cuenta con un conductor y si el destino es alcanzable directamente o requiere pasos intermedios.
- `method_move_driver(state, driver, city_dest)`: Determina cómo mover al conductor `driver` hasta `city_dest`. Si el conductor está asignado a un camión, primero lo libera. Utiliza la función `find_path_with_modes` para obtener la ruta óptima, eligiendo entre caminar o tomar autobús según convenga.

- `method_transport_package(state, package, city_dest)`: Implementa la tarea de transportar el paquete `package` a su destino `city_dest`. Se encarga de gestionar la asignación y el movimiento de camiones, y de asegurar que el paquete se cargue y descargue correctamente.

Función Auxiliar

Se incluye una función auxiliar:

- **find_path_with_modes(state, graph, start, goal)**: Encuentra la ruta más eficiente desde un nodo `start` hasta un nodo `goal` en el grafo (`state.footmap`).Para cada conexión se evalúan dos modos de desplazamiento:
  - **Caminar:** Incrementa el coste en 1 `COST_WALK`.
  - **Autobús:** Incrementa el coste en 3 `COST_BUS`.

  Usa búsqueda en anchura (BFS) con una cola de prioridad para minimizar costos sin exceder `state.limit_cost`. Devuelve una lista de pasos, donde cada paso es una tupla `(nodo_actual, nodo_siguiente, modo)`. Si no se encuentra una ruta válida, devuelve `None`.

1.4 Resolución del Problema

Se ejecuta el código implementado y se obtiene un resultado correcto de acuerdo con el estado inicial y el objetivo planteado. El estado final obtenido es el siguiente:

```
final state =
initial_state.cities = ['C0', 'C1', 'C2']
initial_state.drivers = ['D1', 'D2']
initial_state.trucks = ['T1', 'T2']
initial_state.packages = ['P1', 'P2']
initial_state.loc = {'D1': 'C0', 'D2': 'C0', 'T1': 'C0',
'T2': 'C1', 'P1': 'C1', 'P2': 'C2'}
initial_state.driver_of = {'T1': 'D2', 'T2': None}
initial_state.pack_in = {'P1': None, 'P2': None}
initial_state.roadmap = {'C0': ['C1', 'C2'], 'C1': ['C0', 'C2'],
'C2': ['C1', 'C0']}
initial_state.footmap = {'C0': ['P_01'], 'P_01': ['C0', 'C1'],
'C1': ['P_01', 'P_12'], 'P_12': ['C1', 'C2'], 'C2': ['P_12']}
initial_state.cost = 3
initial_state.limit_cost = 5
```

El plan de ejecución generado por el planificador PyHop es el siguiente:

```
Planning operator by PyHop:
('update_final_cost', 5)
('walk_op', 'D1', 'P_01', 'C0')
('assign_driver_op', 'D1', 'T2')
('load_op', 'P1', 'T2')
('drive_truck_op', 'T2', 'C0', 'C1')
('unload_op', 'P1', 'T2')
```

```
('assign_driver_op', 'D2', 'T1')
('drive_truck_op', 'T1', 'C1', 'C0')
('load_op', 'P2', 'T1')
('drive_truck_op', 'T1', 'C0', 'C2')
('unload_op', 'P2', 'T1')
('drive_truck_op', 'T1', 'C2', 'C0')
('remove_driver_op', 'D1', 'T2')
('walk_op', 'D1', 'C1', 'P_01')
('walk_op', 'D1', 'P_01', 'C0')
```

Este resultado muestra que las acciones ejecutadas por el planificador cumplen con los requisitos del problema, logrando trasladar los paquetes a sus destinos sin exceder el costo límite establecido. Cada operador ejecutado sigue una secuencia lógica que permite alcanzar el estado objetivo de manera eficiente. Por lo tanto, podemos concluir que la solución obtenida es correcta y cumple con las especificaciones del problema planteado.

1.5 Expansión del Problema

Lo que se muestra aquí es solo un ejemplo. El problema se puede ampliar añadiendo más conductores, camiones, paquetes y cambiado las tareas objetivo, haciendo una modelización más compleja. A continuación, se realizarán 3 instancias distintas del problema:

1.5.1. Instancia 1: Expansión de Recursos

En esta instancia se aumenta el número de elementos en el dominio:

- **Conductores:** Se incluye `D3` en C2.
- **Camiones:** Se incorpora `T3` en C0.
- **Paquetes:** Se añade `P3` en C1.

Los objetivos se modifican para distribuir la carga y definir destinos diferenciados, por ejemplo:

- **Coste Final:** Se cambia el coste final a 10, ya que al añadir complejidad no encontraba un plan con tan pocos movimientos.
- `transport_package (P1, C1)`,`transport_package (P2, C2)` y `transport_package (P3, C0)`.
- Los camiones deben terminar en ciudades distintas: `move_truck (T1, C0)`,`move_truck (T2, C1)` y `move_truck (T3, C2)`.
- Se asignan tareas de desplazamiento de conductores, por ejemplo: `move_driver (D1, C0)`,`move_driver (D3, C0)`.

Para la primera instancia del problema, donde se aumentó el número de conductores, camiones y paquetes, se obtuvo el siguiente estado final:

```
final state =
initial_state.cities = ['C0', 'C1', 'C2']
initial_state.drivers = ['D1', 'D2', 'D3']
initial_state.trucks = ['T1', 'T2', 'T3']
initial_state.packages = ['P1', 'P2', 'P3']
initial_state.loc = {'D1': 'C0', 'D2': 'C2', 'T1': 'C0', 'T2': 'C1',
'P1': 'C1', 'P2': 'C2', 'D3': 'C0', 'T3': 'C2', 'P3': 'C0'}
initial_state.driver_of = {'T1': 'D3', 'T2': None, 'T3': 'D2'}
initial_state.pack_in = {'P1': None, 'P2': None, 'P3': None}
initial_state.roadmap = {'C0': ['C1', 'C2'], 'C1': ['C0', 'C2'],
'C2': ['C1', 'C0']}
initial_state.footmap = {'C0': ['P_01'], 'P_01': ['C0', 'C1'],
'C1': ['P_01', 'P_12'],
'P_12': ['C1', 'C2'], 'C2': ['P_12']}
initial_state.cost = 7
initial_state.limit_cost = 10
```

El resultado obtenido cumple con todos los objetivos planteados. Se logra una correcta asignación de conductores a camiones y una distribución eficiente de los paquetes a sus destinos correspondientes. Además, el costo total de ejecución es de 7 unidades, lo cual se encuentra por debajo del límite establecido de 10.

Este resultado indica que el planificador fue capaz de encontrar una solución óptima dentro de los recursos y restricciones especificadas, maximizando la eficiencia de transporte sin superar los límites de costos definidos.

1.5.2. Instancia 2: Restricciones del Mismo Tipo Modificadas de Orden Excepto la Primera

En esta segunda instancia, partiendo de las tareas de la instancia anterior, se modificó el orden de las tareas del mismo tipo. La nueva secuencia de tareas ejecutadas fue la siguiente:

```
tasks = [
('final_cost', 10),
('transport_package', 'P1', 'C1'),
('transport_package', 'P2', 'C2'),
('transport_package', 'P3', 'C0'),
('move_truck', 'T2', 'C1'),
('move_truck', 'T3', 'C2'),
('move_truck', 'T1', 'C0'),
('move_driver', 'D3', 'C0'),
('move_driver', 'D1', 'C0'),
]
```

El estado final obtenido tras la ejecución fue el siguiente:

```
final state =
initial_state.cities = ['C0', 'C1', 'C2']
initial_state.drivers = ['D1', 'D2', 'D3']
initial_state.trucks = ['T1', 'T2', 'T3']
initial_state.packages = ['P1', 'P2', 'P3']
```

```
initial_state.loc = {'D1': 'C0', 'D2': 'C2', 'T1': 'C0', 'T2': 'C1',
'P1': 'C1', 'P2': 'C2', 'D3': 'C0', 'T3': 'C2', 'P3': 'C0'}
initial_state.driver_of = {'T1': 'D3', 'T2': None, 'T3': 'D2'}
initial_state.pack_in = {'P1': None, 'P2': None, 'P3': None}
initial_state.roadmap = {'C0': ['C1', 'C2'], 'C1': ['C0', 'C2'], 'C2': ['C1', 'C0']}
initial_state.footmap = {'C0': ['P_01'], 'P_01': ['C0', 'C1'], 'C1': ['P_01', 'P_12'],
'P_12': ['C1', 'C2'], 'C2': ['P_12']}
initial_state.cost = 7
initial_state.limit_cost = 10
```

A pesar del cambio en el orden de ejecución de las tareas, el resultado final ha sido el mismo que en la primera instancia. Todos los paquetes han sido transportados a sus ubicaciones objetivo, los camiones se han desplazado correctamente y los conductores han finalizado en los lugares previstos.

El costo total de la ejecución sigue siendo de 7 unidades, dentro del límite máximo de 10. Esto sugiere que el orden de ciertas tareas no afecta el desempeño del planificador en este caso, lo que indica que la estrategia de planificación mantiene su robustez ante variaciones en la secuencia de operaciones.

1.5.3. Instancia 3: Líneas de Transporte Adicional y Bus Gratis

En esta tercera instancia, partiendo de las tareas de la instancia anterior, se han añadido dos puntos intermedios, `PX` y `PY`, a la red de caminos a pie, permitiendo nuevas rutas de transporte para los conductores. Además, se ha introducido la posibilidad de utilizar un servicio de autobús gratuito (`COST_BUS = 0`), lo que puede afectar el coste total del plan de transporte.

La nueva configuración de la red peatonal es la siguiente:

```
state0.footmap = {
'PX': ['C1', 'PY'],
'PY': ['C0', 'PX'],
'C0': ['P_01', 'PY'],
'P_01': ['C0', 'C1'],
'C1': ['P_01', 'P_12', 'PX'],
'P_12': ['C1', 'C2'],
'C2': ['P_12']
}
```

Tras ejecutar el código con esta nueva configuración:

```
final state =
initial_state.cities = ['C0', 'C1', 'C2']
initial_state.drivers = ['D1', 'D2', 'D3']
initial_state.trucks = ['T1', 'T2', 'T3']
initial_state.packages = ['P1', 'P2', 'P3']
initial_state.loc = {'D1': 'C0', 'D2': 'C2', 'T1': 'C0', 'T2': 'C1',
'P1': 'C1', 'P2': 'C2', 'D3': 'C0', 'T3': 'C2', 'P3': 'C0'}
initial_state.driver_of = {'T1': 'D3', 'T2': None, 'T3': 'D2'}
initial_state.pack_in = {'P1': None, 'P2': None, 'P3': None}
initial_state.roadmap = {'C0': ['C1', 'C2'], 'C1': ['C0', 'C2'],
'C2': ['C1', 'C0']}
```

```
initial_state.footmap = {'PX': ['C1', 'PY'], 'PY': ['C0', 'PX'],
'C0': ['P_01', 'PY'], 'P_01': ['C0', 'C1'],
'C1': ['P_01', 'P_12', 'PX'], 'P_12': ['C1', 'C2'],
'C2': ['P_12']}
initial_state.cost = 0
initial_state.limit_cost = 10
```

El resultado muestra que el sistema ha logrado alcanzar el estado objetivo sin ningún coste adicional, gracias a la utilización del servicio de autobús gratuito. Esto demuestra que la inclusión de nuevas vías de transporte y la reducción de costes pueden tener un impacto positivo en la optimización del plan de transporte, logrando un resultado eficiente sin consumir recursos adicionales. En comparación con las instancias anteriores, esta variante permite que los conductores se desplacen entre las ciudades sin penalización de coste, lo que podría ser beneficioso en escenarios donde la movilidad de los conductores sea un factor crítico en la planificación del transporte.

1.6 Conclusiones Generales

En este trabajo se ha abordado el problema del transporte en un entorno con múltiples elementos (ciudades, conductores, camiones y paquetes) utilizando una metodología de planificación jerárquica (HTN) basada en el framework PyHop. El enfoque adoptado permite descomponer tareas complejas en sub-tareas más simples, facilitando la integración de restricciones, como el límite de costo.

Uno de los principales retos fue la representación del estado del sistema. Se optó por utilizar diccionarios para gestionar la ubicación de cada objeto (definida en `state.loc`), la relación entre conductores y camiones (`state.driver_of`) y la localización de los paquetes (`state.pack_in`). Esta representación se demostró más eficiente que otras alternativas, como agrupar objetos por ciudad, permitiendo así un manejo más directo y flexible de la información.

Otra dificultad importante fue la incorporación del límite de costo en la planificación. Para ello, se implementó la función `find_path_with_modes`, que permite buscar rutas óptimas combinando dos modos de desplazamiento (caminar y autobús) y asegurando que el coste acumulado no exceda el valor establecido en `state.limit_cost`. Esta solución, basada en una cola de prioridad y una variante de búsqueda en anchura, ayuda a evitar bucles infinitos y optimiza el proceso de toma de decisiones en el desplazamiento de los conductores.

Además, la coordinación entre conductores y camiones se gestionó mediante operadores y métodos específicos que garantizan la correcta asignación y movimiento de estos recursos. Por ejemplo, se asegura que un camión se mueva únicamente si tiene asignado un conductor y que los conductores puedan ser liberados para desplazarse a nuevas ubicaciones cuando sea necesario (siendo libres de poder ir a una ubicación diferente tras dejar el camión en la ubicación objetivo).

Se optó por definir el `goal1.tasks` como una secuencia de tareas jerárquicas (primero la restricción de coste, luego el transporte de paquetes, seguido del movimiento de camiones y finalmente de conductores). La implementación desarrollada es modular y escalable, lo que permite ampliar el dominio agregando más conductores, camiones o paquetes, y ajustar el límite de costo según se requiera.