Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

PLANIFICACIÓN

1.1 Introducción.

El campo de la **planificación en IA** tiene como objetivo construir algoritmos de control que permitan a un agente sintetizar una secuencia de acciones que le lleve a alcanzar sus objetivos. Un problema de planificación en IA es un problema de búsqueda que requiere encontrar una secuencia eficiente de acciones para conducir a un sistema desde un estado inicial hasta un estado objetivo.

Consideremos un escenario formado por un conjunto de ciudades donde el objetivo de un agente es transportar un paquete de una ciudad origen `cO` a una destino `cD`. El algoritmo de búsqueda tendría que examinar todas las posibles rutas (planes) que se pueden formar para ir de `cO` a `cD`. Para un caso concreto de $n = 7$ ciudades con cinco ciudades intermedias, el espacio de búsqueda estaría formado por 380 rutas diferentes, y para tamaños mayores de problemas este espacio podría resultar inabordable ($(\sum_{j=1}^{n} \prod_{i=j}^{n} i) + 1$ rutas).

A priori, y dado el planteamiento inicial de este problema, podrían aplicarse técnicas clásicas de resolución de problemas para encontrar una solución, como por ejemplo la aplicación del algoritmo del viajante de comercio. Sin embargo, la complejidad en problemas reales de planificación puede llegar a ser tan elevada que se hace necesario un razonamiento inteligente sobre las acciones y sus consecuencias para afrontar eficientemente la resolución de estos problemas. Supongamos que el agente dispone de tres camiones de diferente tamaño para transportar el paquete; en este caso, el espacio de búsqueda del agente se incrementaría hasta 3*380= 1140 posibles rutas. Adicionalmente, si en el estado inicial hay más de un paquete a transportar de `cO` a `cD` habría que considerar todas las posibles formas de transportar los paquetes, bien individualmente o bien en grupos, utilizando uno, dos o todos los camiones a través de todas las posibles rutas. Al crecer el espacio de búsqueda de forma exponencial, los algoritmos tradicionales de búsqueda no resultan viables para resolver estos problemas. Si se introduce un factor adicional en nuestro escenario, como la existencia de más paquetes en una ciudad intermedia cuyo destino es igualmente `cD`, un planificador inteligente deberá razonar sobre qué camiones utilizar para transportar los paquetes y qué ruta(s) realizar para cumplir el objetivo. Existen múltiples alternativas, como: 1) utilizar un único camión para todos los paquetes del problema y planificar una única ruta que pase por la ciudad intermedia, 2) utilizar dos camiones, uno para los paquetes que están en `cO` y otro para los paquetes de la ciudad intermedia (en este caso habría que planificar rutas independientes, una para cada camión), 3) utilizar todos los camiones disponibles, etc.

Como en todo problema de búsqueda inteligente, un elemento clave en planificación es disponer

de buenas **funciones heurísticas** para guiar eficientemente la búsqueda del planificador (véase Capítulo **??**). En nuestro problema, se daría mayor prioridad a un estado en el que existe un camión en la ciudad intermedia porque en dicho estado se satisfacen parte de las condiciones necesarias para transportar los paquetes de esa ciudad. Esto indica que dicho estado estará más próximo a un estado final donde se satisfagan todos los objetivos del problema y, por tanto, el proceso de búsqueda a partir de ese estado necesitará menos acciones o pasos de ejecución para alcanzar el objetivo. Uno de los propósitos de la planificación es el desarrollo de funciones heurísticas *independientes del dominio*, esto es funciones genéricas y reutilizables que pueden aplicarse al espacio de búsqueda de cualquier problema, con independencia del tipo de escenario en el que se desarrolla dicho problema.

Los problemas de planificación se suelen plantear en sistemas dinámicos donde, dado el estado actual y los objetivos, deducir la siguiente acción a aplicar no es una tarea obvia. La planificación es una tarea compleja y ésta es la razón por la cual la mayoría de planificadores trabajan sobre un modelo restringido del entorno (**planificación clásica**), siendo dicho modelo determinista, estático y totalmente observable. Concretamente, la planificación clásica fija las siguientes asunciones:

A1) **Modelo observable**. El mundo es totalmente observable y no existe información desconocida para el planificador.

A2) **Modelo estático**. Se puede predecir correctamente la evolución de las secuencias de acciones que se aplican sobre un estado inicial completamente conocido, ya que no hay influencias externas que afecten al entorno. Los objetivos son conocidos antes de comenzar la planificación y no cambian durante el transcurso del proceso de planificación.

A3) **Modelo determinista**. Los efectos de la aplicación de una acción en un estado son totalmente predecibles, lo que conduce determinísticamente a un único estado.

A4) **Enfoque proposicional**. Todas las variables del modelo de planificación (sentencias atómicas) pertenecen al dominio lógico, tomando por tanto los valores *cierto* o *falso*.

A5) **Acciones instantáneas**. Todas las acciones del modelo de planificación tienen la misma duración y se consideran atómicas e instantáneas.

A6) **Acciones no descomponibles**. Las acciones del modelo son directamente ejecutables en el entorno y no pueden descomponerse o dividirse en subacciones.

A7) **Planificación *offline***. La tarea de planificación consiste en construir un plan completo que satisface el objetivo antes de la ejecución de cualquier parte del mismo.

A pesar de estas simplificaciones, la resolución de un problema de planificación es PSPACE-completo. Por ello no resulta factible el empleo de técnicas clásicas de resolución de problemas o demostración de teoremas. Este capítulo está dedicado a presentar los principios fundamentales de la planificación como una actividad centrada en técnicas de búsqueda, representación y tratamiento de un problema de planificación, algoritmos y técnicas de control para hacer la búsqueda más eficiente.

1.2 Problema de planificación.

Un problema de planificación se describe mediante un conjunto de acciones, un estado inicial del entorno y una descripción del objetivo a conseguir. Este problema se resuelve obteniendo el conjunto de acciones que transforman el estado inicial en un estado que satisface el objetivo. Esta

Estado inicial ($\mathcal{I}$)	`pos(p1,cA)` $\wedge$ `pos(c1,cA)`
Objetivo ($\mathcal{G}$)	`pos(p1,cB)`
Operadores ($\mathcal{O}$)	`mv(?cam,?ori,?des)`
	Pre: `pos(?cam,?ori)`
	Efe: `pos(?cam,?des)` $\wedge \neg$ `pos(?cam,?ori)`
	`cg(?paq,?cam,?ciu)`
	Pre: `pos(?paq,?ciu)` $\wedge$ `pos(?cam,?ciu)`
	Efe: `en(?paq,?cam)` $\wedge \neg$ `pos(?paq,?ciu)`
	`dcg(?paq,?cam,?ciu)`
	Pre: `pos(?cam,?ciu)` $\wedge$ `en(?paq,?cam)`
	Efe: `pos(?paq,?ciu)` $\wedge \neg$ `en(?paq,?cam)`

Tabla 1.1: Definición de un problema simple de transporte que se empleará a lo largo del capítulo utilizando STRIPS.

visión del problema de planificación, totalmente centrada en el concepto de acción, hereda muchos aspectos del cálculo de situaciones desarrollado por McCarthy en [McCarthy and P.J., 1969], en donde se especifica cómo las situaciones, descritas en un lenguaje de primer orden, se ven afectadas por las acciones ejecutadas por un agente. Para lograr un proceso de planificación eficiente es tan importante contar con buenos algoritmos como con buenos lenguajes de modelado y representación. El sistema STRIPS ha condicionado la gran mayoría de trabajos sobre planificación desde comienzos de los años 70, gracias a la definición de una sencilla sintaxis para la especificación de problemas de planificación.

Un problema de planificación STRIPS se define como una tripleta $\mathcal{P} = \langle \mathcal{I}, \mathcal{G}, \mathcal{O} \rangle$ donde $\mathcal{I}$ es el estado inicial del problema, $\mathcal{G}$ es el objetivo (*goal*) a conseguir y $\mathcal{O}$ es el conjunto de operadores de planificación. Un **estado** de un problema de planificación es una representación del conjunto de propiedades o características de los objetos que intervienen en el problema junto con los valores de dichas propiedades. En STRIPS se trabaja con literales de primer orden y las descripciones de estados se componen de literales positivos totalmente instanciados y sin dependencias funcionales. En un dominio de transporte, como el indicado en la sección anterior, donde existen paquetes, camiones y ciudades, un estado del problema representaría la ubicación de paquetes y camiones en ciudades. Un predicado binario de primer orden como `pos(?obj,?ciu)` se utiliza para indicar la posición de cualquier objeto del problema (paquete o camión) en una ciudad determinada. En la Tabla 1.2 se representa el estado inicial $\mathcal{I}$ de un problema de transporte en el que existe un paquete `p1` y un camión `c1`, ambos localizados en la ciudad `cA`. Se asume la hipótesis de *mundo cerrado*, lo que significa que sólo las condiciones que se listan explícitamente tienen valor *cierto* y que cualquier condición que no se menciona en un estado se considera *falsa*.

Para describir el **objetivo** de un problema de planificación hay que especificar el valor que se desea que tengan las propiedades de los objetos en el estado meta del problema. A diferencia del estado inicial, un objetivo de planificación no es una descripción completa de un estado sino que describe simplemente el valor de aquellas propiedades de objetos que constituyen el objetivo del problema. En la Tabla 1.2 se representa el objetivo $\mathcal{G}$ para el problema de transporte, que consiste en tener el paquete `p1` en la ciudad `cB`. Se puede observar que no se especifica la posición final del camión `c1`, indicando así que éste puede finalizar en cualquier ciudad.

Un **operador** es una generalización de una función de transición que toma como entrada un estado del problema y determina cuál será el siguiente estado; una **acción** es una instanciación de un operador. En la Tabla 1.2 se muestran los tres operadores $\mathcal{O}$ para el problema de transporte. Un

operador consta de tres partes:

- Nombre y lista de argumentos. Por ejemplo, el primer operador es `mv` (mover un camión de una ciudad a otra) y requiere tres argumentos: el camión `?cam` a mover, la ciudad origen `?ori` del camión y la de destino `?des`.
- Precondiciones (Pre). Conjunto de literales positivos que determinan las condiciones que deben satisfacerse en el estado anterior a la ejecución del operador y mantenerse durante toda la ejecución del mismo. Las variables que aparecen en las precondiciones deben aparecer también en la lista de argumentos. Por ejemplo, las precondiciones del operador `cg` (cargar un paquete en un camión) son que el paquete y el camión se encuentren ambos en la ciudad `?ciu`.
- Efectos (Efe). Conjunción de literales que describe cómo cambia el estado cuando se ejecuta el operador. Un literal positivo es un efecto que se añade en el estado resultante (literal con valor *cierto* en el nuevo estado), y un literal negativo ($\neg$) es un efecto que se elimina (literal con valor *falso* en el nuevo estado). La lista de variables de los efectos también debe aparecer en la lista de argumentos. Por ejemplo, los efectos del operador `dcg` (descargar un paquete de un camión en una ciudad) son que el paquete se encontrará en la ciudad `?ciu` y dejará de estar dentro del camión ($\neg$`en(?paq,?cam)`).

Una acción, como por ejemplo `mv(c1,cA,cB)`, se dice que es **aplicable** en un estado si se satisfacen las precondiciones de la acción en dicho estado. Las precondiciones de la acción anterior (`pos(c1,cA)`) se satisfacen en el estado inicial de la Tabla 1.2, por lo que la acción es aplicable en dicho estado, dando como resultado un nuevo estado en el que se añade el literal `pos(c1,cB)` y se elimina el literal `pos(c1,cA)`. Sin embargo, esta misma acción no es aplicable en el estado $\mathcal{E}$={`pos(c1,cB),pos(p1,cA)`}. Si una acción no es aplicable no produce ningún efecto. En conclusión, la aplicación de una acción en un estado $\mathcal{E}$ produce un nuevo estado $\mathcal{E}' = \mathcal{E} - \text{Efe}^- \cup \text{Efe}^+$, donde Efe^+ son los efectos positivos que se añaden y Efe^- son los efectos negativos que se borran, respectivamente. Una de las grandes aportaciones de STRIPS es la asunción para evitar la complejidad del *problema marco* introducida en [McCarthy and P.J., 1969], la cual consiste en asumir que los únicos cambios que se producen como resultado de la aplicación de una acción son aquellos literales que explícitamente se mencionan como efectos de la misma; es decir, el resto de literales se satisface también en el nuevo estado.

Por último, la solución a un problema de planificación se denomina **plan**. En su forma más simple, un plan es una secuencia de acciones que, cuando se ejecuta desde el estado inicial, produce un estado que satisface el objetivo. El plan solución al problema de la Tabla 1.2 sería ⟨`cg(p1,c1,cA), mv(c1,cA,cB), dcg(p1,c1,cB)`⟩.

1.3 Lenguaje de planificación PDDL.

El lenguaje STRIPS se planteó con el deseo de diseñar algoritmos simples y eficientes, sin complicar excesivamente la definición de problemas reales. Con el objetivo de enriquecer la expresividad permitida por STRIPS, en los últimos años se han desarrollado otros lenguajes de planificación entre los que destaca PDDL, que últimamente se ha convertido en un estándar en planificación. Actualmente existen varias versiones de PDDL, siendo la más reciente PDDL3 que engloba todas las características de sus predecesoras e incluye nuevas funcionalidades. En esta sección se apuntan sólo aquellas características de PDDL que introducen un cambio en el modelo semántico de las acciones de STRIPS.

- Acciones con duración (acciones *durativas*), donde *pre*condiciones y efectos se asocian a los puntos de inicio y final de la duración de la acción; además, las precondiciones también se pueden asociar a todo el intervalo de duración. De esta forma se puede modelar mejor la *física* del problema. Por ejemplo, en el caso del operador `mv`, el camión deja de estar en `?ori` en cuanto inicia su recorrido, y no llega a `?des` hasta el final del mismo. Por tanto, el efecto negativo ¬`pos(?cam,?ori)` se produciría al iniciar el operador, y el efecto positivo `pos(?cam,?des)` al final del mismo. En la Sección 1.8.1 se estudiará este modelo con más detalle dentro de la planificación temporal.

- Expresiones y variables numéricas. Las expresiones numéricas se construyen mediante operadores aritméticos y funciones numéricas, las cuales asocian valores numéricos a tuplas de objetos del problema. Las condiciones numéricas en las acciones son siempre comparaciones entre pares de expresiones numéricas, mientras que los efectos permiten modificar los valores de las funciones numéricas. Gracias al uso de variables numéricas se puede, por ejemplo, controlar el consumo de combustible (en litros) de un camión, comprobando que el camión dispone de cantidad suficiente para realizar un recorrido determinado (precondición numérica) y actualizando dicho valor al final de la ejecución de la acción (efecto numérico). En la Sección 1.8.1 se estudiará este modelo con más detalle dentro de la planificación numérica.

- Métricas del problema. Permiten definir funciones de optimización como una combinación lineal de uno o varios factores del problema. Por ejemplo, se puede definir una métrica para minimizar el consumo total de combustible de todos los camiones de un problema, o una función para minimizar el tiempo total del plan y el consumo de energía de un robot, ponderando cada factor con un porcentaje determinado (véase Sección 1.8.1).

- Ventanas temporales. Se utilizan como una forma restrictiva de expresar eventos exógenos que no se ven afectados por las acciones del plan; es decir, hechos que serán *ciertos* o *falsos* en puntos de tiempo conocidos por el planificador. Por ejemplo, mediante el uso de ventanas temporales se puede expresar que el almacén donde se guardan los paquetes está abierto desde las 8h hasta las 20h. Para ello, se informa al planificador de la existencia de un hecho *cierto* (apertura del almacén) en el instante 8, y que dicho literal será *falso* a partir del instante 20. Estos hechos suceden en el plan independientemente de las acciones del mismo.

- Restricciones duras y blandas sobre el plan. Definen restricciones sobre la estructura de los planes que deben satisfacerse en los estados intermedios del plan. Estas restricciones se expresan mediante operadores modales como `always`, `sometime`, `at-most-once`, `at-end`, `always-within`, `hold-after`, etc. para indicar cómo y cuándo deben satisfacerse. Una restricción dura es aquélla que *obligatoriamente* debe cumplirse en el plan, como por ejemplo que todos los camiones deben visitar cada ciudad a lo sumo una vez. Una restricción blanda o preferencia es una restricción que debe satisfacerse en la medida de lo posible pero no es condición imprescindible para la obtención del plan. De este modo, algunas preferencias podrían no satisfacerse bien porque ello supone un coste excesivo que afecta a la función de optimización del problema o bien porque dicha preferencia entra en conflicto con otras restricciones u objetivos. Un ejemplo de preferencia sería: todos los camiones del problema deben finalizar, *preferiblemente*, en la ciudad `cA`.

Soportar estas características permite que PDDL sea un lenguaje de modelado útil para la definición de problemas más cercanos a la realidad, haciendo hincapié no sólo en los elementos clave

para conseguir un plan, sino también en los necesarios para especificar los criterios de calidad del plan.

1.4 Planificación en un espacio de estados.

El enfoque más sencillo en planificación consiste en realizar una búsqueda en un espacio de estados (véase Capítulo **??**), donde se genera un árbol de búsqueda cuyos nodos denotan estados, entendiendo estado como un conjunto de literales. Dado que las descripciones de acciones especifican tanto las precondiciones como los efectos, es posible realizar una búsqueda en ambas direcciones: búsqueda hacia delante desde el estado inicial o búsqueda hacia atrás desde los objetivos.

1.4.1 Búsqueda hacia delante.

En la búsqueda hacia delante o **búsqueda progresiva** los nodos del árbol representan **estados** del problema de planificación, siendo el nodo raíz del árbol el estado inicial del problema. Un nodo se expande a partir de todas las acciones del problema que son **aplicables** en dicho estado, generando así el nuevo conjunto de nodos (estados) sucesores, los cuales formarán parte de la lista *ABIERTA* del árbol. Sea $\mathcal{I}$ el estado inicial y $\{\mathtt{a}_1, \mathtt{a}_2, \ldots \mathtt{a}_n\}$ el conjunto de todas las posibles acciones del problema. El subconjunto de acciones aplicables en $\mathcal{I}$ será $\{\mathtt{a_i}\} \mid \mathrm{Pre}(\mathtt{a_i}) \in \mathcal{I}$. Los estados resultantes de aplicar las acciones correspondientes serán aquellos nodos que se forman eliminando del estado padre los efectos negativos y añadiendo los efectos positivos de la acción correspondiente; para el caso del estado $\mathcal{I}$, los sucesores que se forman serán: $\{\mathcal{E}_i\} \mid \mathcal{E}_i = \mathcal{I} - \mathrm{Efe}^-(\mathtt{a_i}) \cup \mathrm{Efe}^+(\mathtt{a_i})$.

En un planificador progresivo la aplicación de cada acción genera un paso, y un plan consiste en una simple lista de pasos; se comienza desde la descripción del estado inicial y se busca el objetivo en el espacio de estados del problema. El plan se construye añadiendo pasos (acciones) totalmente instanciados al final del plan. Así, cuando se alcanzan los objetivos se devuelve una solución (plan) totalmente ordenada. Por este motivo, a los planificadores basados en este tipo de búsqueda también se les denomina planificadores de **orden total**, es decir sólo se exploran secuencias lineales de acciones que conectan al estado inicial del problema con el objetivo.

Durante la expansión del árbol, los nodos sucesores que se generan se introducen en la lista *ABIERTA* del árbol, los cuales constituyen el conjunto de estados candidatos a ser expandidos en la siguiente iteración. Dependiendo de la estrategia de búsqueda utilizada (anchura, profundidad, profundización iterativa, coste uniforme, búsqueda heurística, etc.) los nodos se ordenarán en la lista de distinto modo, siendo siempre el primer nodo de la lista el siguiente nodo a expandir (véase Algoritmo **??** del Capítulo **??**). En el caso que nos atañe no emplearemos ninguna estrategia de búsqueda concreta sino que el nodo a expandir se seleccionará no determinísticamente y se extraerá de la lista *ABIERTA* del árbol.

Ejemplo. En la Figura 1.1 se muestra el árbol de búsqueda que se genera para el problema de transporte de ejemplo. Las dos acciones aplicables en el estado inicial $\mathcal{I}$ son aquéllas cuyas precondiciones se satisfacen en dicho estado (`mv(c1,cA,cB)` y `cg(p1,c1,cA)`). Los estados resultantes de la aplicación de estas dos acciones se generan añadiendo los efectos positivos y borrando los efectos negativos de las mismas, y dichos nodos se introducen en la lista *ABIERTA*. El número asociado a cada nodo indica un orden de expansión arbitrario de los nodos de la lista *ABIERTA*; de este modo, cuando se expande el nodo 2, la única acción aplicable es la **acción inversa** a la que ha generado el nodo 2 y, por consiguiente, el sucesor que se generaría sería un estado repetido que coincidiría con el estado $\mathcal{I}$. Lo mismo sucede en la expansión del nodo 3 con la aplicación de la

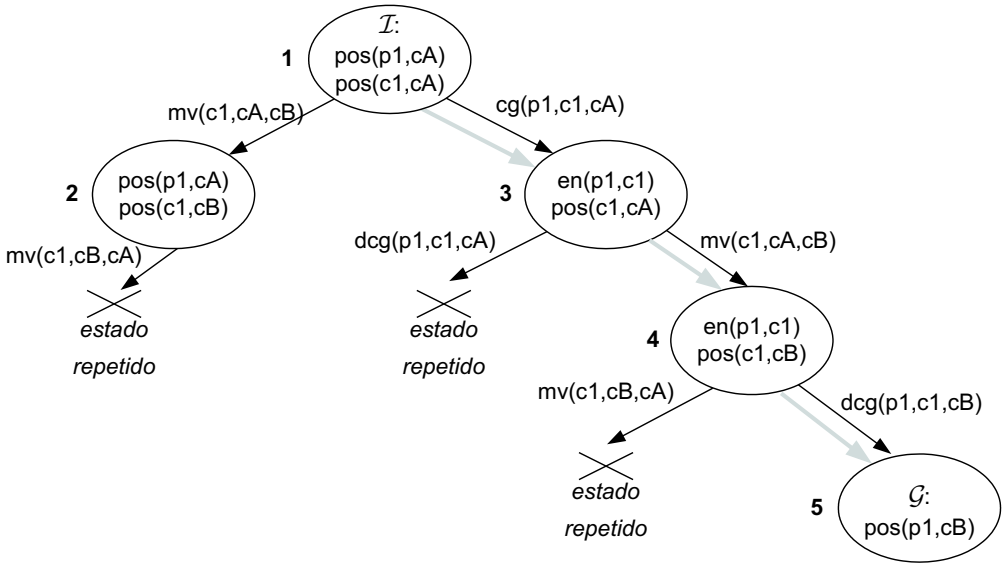

Figura 1.1: Ejemplo del árbol de búsqueda hacia delante que se generara para el problema de transporte (véase Tabla 1.2). En trazo gris se muestra la rama que constituye el plan.

acción `dcg(p1,c1,cA)`, que daría de nuevo un estado idéntico al estado $\mathcal{I}$, o con la aplicación de la acción `mv(c1,cB,cA)` en la expansión del nodo 4. El objetivo $\mathcal{G} = \{\texttt{pos(p1,cB)}\}$ se satisface en el momento en que se selecciona el nodo 5 para ser expandido.

La búsqueda hacia delante fue la primera aproximación que se utilizó para resolver problemas de planificación. Este tipo de búsqueda plantea los siguientes inconvenientes:

- Se consideran todas las acciones aplicables en un estado sin desestimar aquellas acciones que son irrelevantes para la consecución de los objetivos. Por ejemplo, supongamos un problema con tres paquetes {`p1,p2,p3`} y cuyo objetivo es simplemente llevar `p1` a la ciudad `cB`. Una estrategia de búsqueda hacia delante considerará todas las acciones aplicables, incluyendo las de cargar, mover y descargar los paquetes `p2` y `p3`, aunque éstas no influyan en la consecución del objetivo

- Tal y como se indica en la Figura 1.1, en dominios *reversibles* donde existen acciones inversas que conducen de nuevo al estado antecesor, se producen infinidad de ciclos en el árbol de búsqueda. Este es un factor que debe controlarse convenientemente para evitar un proceso de búsqueda infructuoso. Por este motivo, y porque el mismo estado puede alcanzarse por diferentes caminos (diferente secuencias de acciones), en general se suele decir que se tiene un grafo de búsqueda en lugar de un árbol de búsqueda.

- El factor de ramificación para problemas de gran tamaño puede resultar en una explosión del espacio de búsqueda. Considérese un problema en el que hay 10 ciudades, 50 camiones y 20 paquetes a transportar. El objetivo es mover los 20 paquetes de la ciudad `cA` a la ciudad `cB`. Una solución sencilla es cargar los 20 paquetes en un camión, mover el camión de `cA` a `cB` y descargar los paquetes. Pero encontrar una solución puede resultar complejo porque el factor de ramificación es excesivamente elevado: cada uno de los 50 camiones puede ir a

las 9 ciudades restantes, y cada uno de los 20 paquetes puede cargarse en cualquier camión en la ciudad donde éste se encuentre (si no está cargado) o descargarse (si está cargado). En general, el factor de ramificación de un planificador progresivo viene determinado por el número de acciones aplicables en cada estado. Si llamamos a este factor b y, teniendo en cuenta que la profundidad de la búsqueda, n, viene determinada por el mínimo número de acciones necesarias para alcanzar el objetivo desde el estado inicial, tenemos que el coste del árbol de búsqueda o número de nodos generados sería $O(b^n)$.

Por las razones expuestas arriba, se consideró que la búsqueda hacia delante en un espacio de estados era una estrategia muy poco eficiente para la resolución de problemas de planificación. Sin embargo, esta consideración ha cambiado recientemente porque la investigación en planificación de los últimos años ha dado luz a planificadores de búsqueda hacia delante muy eficientes gracias al empleo de excelentes funciones heurísticas. Evidentemente, sin el uso de buenas funciones heurísticas esta estrategia resulta inadecuada para resolver problemas de gran tamaño.

1.4.2 Búsqueda hacia atrás.

Este tipo de búsqueda, también conocida como **búsqueda regresiva**, parte del objetivo $\mathcal{G}$ del problema, el cual está constituido por el conjunto de literales a conseguir $\mathcal{G} = \{\mathtt{g}_1, \mathtt{g}_2, \ldots \mathtt{g}_n\}$ y aplica inversamente los operadores del problema, generando así estados subobjetivos. El proceso terminará si se produce un conjunto de subobjetivos que se satisface en $\mathcal{I}$. El esquema de funcionamiento de la búsqueda hacia atrás se muestra en el Algoritmo 1.1. Inicialmente se parte de un plan vacío Π que se va construyendo sucesivamente en cada iteración del algoritmo. El objetivo de la búsqueda hacia atrás consiste en ir actualizando sucesivamente el conjunto $\mathcal{G}$ (que inicialmente se compone de todos los objetivos del problema) hasta que se llega a un conjunto que se satisface en $\mathcal{I}$. Para ello, en cada iteración del bucle, se obtiene el conjunto de acciones relevantes, que son las que consiguen algún objetivo de $\mathcal{G}$. Si el conjunto *Relevante* resulta vacío entonces es que no existen acciones para conseguir los objetivos de $\mathcal{G}$, y el algoritmo devuelve fallo. En caso contrario, se selecciona aleatoriamente una acción y se añade al plan Π. El último paso actualiza el conjunto $\mathcal{G}$ eliminando los efectos ya conseguidos por la acción seleccionada y añadiendo al nuevo conjunto las precondiciones de la misma.

Algoritmo 1.1 Esquema básico de la búsqueda hacia atrás.

1: $\Pi \leftarrow$ Plan vacío
2: **bucle**
3: **si** $\mathcal{G} \in \mathcal{I}$ **entonces**
4: Devuelve Π *{Éxito}*
5: **fin si**
6: *Relevante* $\leftarrow \{\mathtt{a} \mid \mathtt{a}$ es una acción relevante para $\mathcal{G}\}$
7: **si** *Relevante* $= \emptyset$ **entonces**
8: Devuelve fallo *{No existe plan}*
9: **fin si**
10: Seleccionar no determinísticamente y extraer $\mathtt{a} \in$ *Relevante*
11: $\Pi \leftarrow \mathtt{a} \cdot \Pi$
12: $\mathcal{G} = \mathcal{G} - \mathrm{Efe}^+(\mathtt{a}) \bigcup \mathrm{Pre}(\mathtt{a})$
13: **fin bucle**

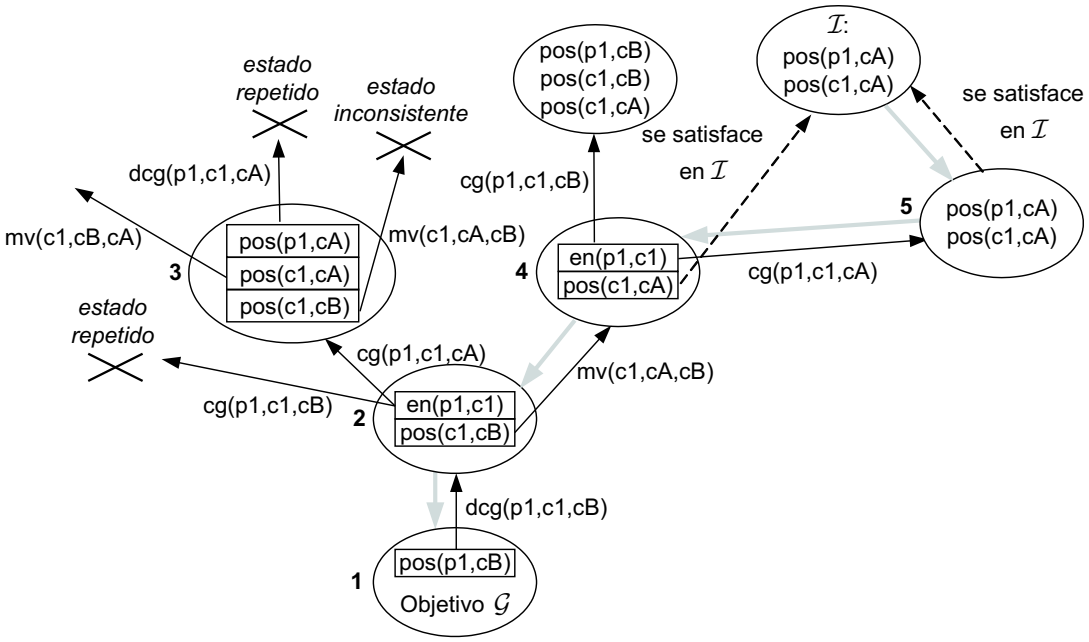

Figura 1.2: Ejemplo del árbol de búsqueda hacia atrás que se generara para el problema de transporte (véase Tabla 1.2). En trazo gris se muestra la rama que constituye el plan.

Los nodos del árbol que se generan en una búsqueda hacia atrás representan el conjunto de literales que restan por satisfacer para obtener una solución al problema, es decir, no representan estados del problema sino *estados objetivo*. Estableciendo una equivalencia con el Algoritmo 1.1, los nodos del árbol representarían los diferentes conjuntos $\mathcal{G}$ que se generan en las distintas iteraciones de la búsqueda.

El proceso de expansión de un árbol en una búsqueda hacia atrás es muy similar al de la búsqueda hacia delante. La búsqueda de acciones relevantes durante la expansión de un estado objetivo (paso 6 del Algoritmo 1.1) supone la generación de sus nodos predecesores que a su vez se introducirán en la lista *ABIERTA* del árbol. Al igual que con la búsqueda hacia delante, la aplicación de una estrategia de búsqueda concreta determinaría el orden en el que los nodos son expandidos; sin embargo, en nuestro caso no utilizaremos ninguna estrategia en particular y en su lugar seleccionaremos no determinísticamente el nodo a expandir (paso 10 del Algoritmo 1.1).

Ejemplo. La Figura 1.2 muestra el árbol de búsqueda que se genera para el problema de transporte de ejemplo. El número asociado a cada nodo indica el orden de expansión del mismo. El primer estado objetivo del problema está constituido por el único literal del objetivo del problema, $\mathcal{G} = \{$`pos(p1,cB)`$\}$ y ése será el primer nodo a expandir. La única acción relevante para dicho literal es `dcg(p1,c1,cB)`, por lo que el nuevo estado objetivo (nodo 2) se compone de las dos precondiciones de dicha acción. Para obtener los siguientes estados predecesores se aplican todas las posibles acciones relevantes en el nodo 2, es decir acciones que consigan alguno de los dos literales de dicho nodo, y se actualiza el conjunto $\mathcal{G}$ (tal y como se muestra en el último paso del Algoritmo 1.1). En el árbol de la Figura se pueden observar las siguientes situaciones:

1. La búsqueda desde el nodo 2 cuando se aplica la acción `cg(p1,c1,cB)` generaría un estado

repetido ya que se trata de la acción inversa que ha dado lugar al nodo 2 y por tanto se generaría un nodo predecesor que contendría como objetivos a conseguir el literal `pos(p1,cB)`, que es el mismo literal del nodo inicial $\mathcal{G}$. En este caso se produciría un bucle al intentar satisfacer nuevamente el objetivo $\mathcal{G}$. Esta misma situación se produce al intentar satisfacer el literal `pos(p1,cA)` del nodo 3, ya que la acción que consigue dicho literal es la inversa de la acción `cg(p1,c1,cA)`.

2. Puede observarse cómo aparecen varios nodos marcados como *estado inconsistente*. Tomemos como ejemplo el literal `pos(c1,cA)` del nodo 3. La acción que satisface dicho literal es `mv(c1,cB,cA)`, la cual contiene entre sus efectos negativos el literal `pos(c1,cB)` que a su vez es un literal a satisfacer en el nodo 3; en este caso, se dice que el estado que se alcanzaría es inconsistente porque la acción que consigue un literal borra otro literal del conjunto $\mathcal{G}$. Además de que las acciones alcancen algunos literales de $\mathcal{G}$ (acciones relevantes), las acciones no deben *deshacer* ningún literal de ese conjunto. Una acción que satisfaga esta propiedad se dice que es una *acción consistente*. Por tanto, la aplicación de una acción relevante `a` lleva a un estado inconsistente si se cumple $(\exists \mathtt{g} \in \mathcal{G} \mid \mathtt{g} \in \mathrm{Efe}^{-}(\mathtt{a}))$. Esta misma situación se produciría al intentar satisfacer los literales `pos(c1,cB)` y `pos(c1,cA)` del estado que se genera a partir del nodo 4 con la aplicación de la acción `cg(p1,c1,cB)`.

3. Una de las ramas que se generan en la expansión del nodo 4 lleva a un estado objetivo (nodo 5) cuyos literales se satisfacen en el estado inicial. El algoritmo devolvería, por tanto, el plan que se indica con las líneas de trazo gris, que es el plan que se ha formado concatenando sucesivamente las acciones relevantes encontradas.

Nótese que en caso de generar un estado repetido o un estado inconsistente el algoritmo continuaría la expansión por cualquiera de los nodos de la lista *ABIERTA* del árbol (selección arbitraria). También debe observarse que, dado que se trata de una búsqueda regresiva, el orden en el que se encuentran las acciones del plan es el orden inverso en el que se encuentran las acciones en un proceso de búsqueda hacia delante; esto es, la primera acción encontrada será la última acción del plan.

Al igual que en la búsqueda hacia delante, un planificador regresivo genera un plan añadiendo pasos (acciones) totalmente instanciados al final del plan y se devuelve una solución totalmente ordenada cuando se alcanza el objetivo del problema. Por tanto, los planificadores regresivos son también planificadores de **orden total**. Sin embargo, una importante ventaja que presenta la búsqueda regresiva frente a la búsqueda progresiva es que permite considerar sólo acciones relevantes para la consecución de los objetivos, reduciendo de este modo el factor de ramificación del árbol de búsqueda en varios órdenes de magnitud. Por ejemplo, si existen tres paquetes y el objetivo es transportar sólo uno de ellos, las acciones que involucran a los otros dos no se considerarán. Es por esto que, en general, la búsqueda regresiva es más eficiente que la búsqueda progresiva para la resolución de problemas de planificación.

El algoritmo STRIPS.

El algoritmo de planificación del sistema STRIPS está basado en un proceso de búsqueda hacia atrás. Se trata, en realidad, de un proceso recursivo que difiere del esquema general del Algoritmo 1.1 en los siguientes aspectos:

- En cada llamada recursiva del algoritmo STRIPS, los estados objetivos predecesores se forman

únicamente con las precondiciones de la última acción, permitiendo así reducir sustancialmente el factor de ramificación.

- Si en el estado actual del problema se satisfacen todas las precondiciones de una acción, STRIPS ejecuta dicha acción en el estado actual del problema y actualiza el estado de planificación. Dado que la ejecución de una acción es una decisión irrevocable, puede suceder que alguno de los objetivos ya conseguidos se eliminen como consecuencia de dicha operación, es decir, al ejecutar una acción para conseguir un objetivo g_i se puede deshacer otro ya conseguido g_j. En este caso, se introduce de nuevo g_j como objetivo, o bien se realiza una vuelta atrás para explorar otro camino que no deshaga g_j. Con este tipo de funcionamiento se consigue igualmente reducir una gran parte del espacio de búsqueda.

1.5 Planificación de orden parcial.

En la planificación de orden total las acciones del plan se obtienen en el mismo orden en que éstas se ejecutarían; esto es, el orden de planificación coincide con el orden de ejecución. De este modo, si un planificador selecciona una acción *equivocada*, deberá incorporar una acción adicional para *deshacer* los efectos de la acción errónea y volver a un estado de planificación correcto. En problemas complejos se suceden múltiples interacciones entre los objetivos del problema porque las acciones que se utilizan para resolver un objetivo pueden interferir en la solución de otro objetivo. Un planificador que tenga en cuenta esta característica generará un plan entrelazado en el que se trabaja simultáneamente con múltiples objetivos del problema. De este modo, el planificador tendrá que tomar decisiones sobre cómo las acciones que resuelven una parte del problema afectan a la resolución del resto del problema. A este tipo de aproximación se le denomina **planificación de orden parcial** (POP).

La ventaja de la aproximación POP es la flexibilidad a la hora de establecer un orden entre las acciones que componen un plan ya que los planificadores pueden primero tomar decisiones sobre las partes importantes del plan, en lugar de forzar un orden cronológico entre las acciones del mismo. En POP se trabaja sobre los objetivos del problema simultáneamente y se mantiene un orden parcial entre las acciones sin tener que comprometer un orden concreto entre las mismas, hasta que la propia estructura del plan determina cuál debe ser este orden. A la estrategia general de retrasar una decisión durante el proceso de búsqueda se le conoce como **menor compromiso**. Por ejemplo, si se dispone de dos paquetes {`p1`,`p2`} en la ciudad `cA` que deben ser cargados en el camión `c1`, el orden en el que se realice las acciones `cg(p1,c1,cA)` y `cg(p2,c1,cA)` no es relevante para la consecución del objetivo del problema. Un POP deja abierto el orden entre ambas acciones, es decir, establece un orden parcial entre las mismas hasta que las condiciones del problema determinen cuál de los dos paquetes debe cargarse antes. Si la estructura o condiciones del problema no impone un orden explícito entre los dos paquetes, entonces un POP puede devolver cualquier orden entre las dos acciones (cargar `p1` primero y luego `p2`, o viceversa) o bien devolver las acciones en paralelo si el agente de ejecución es capaz de ejecutar acciones concurrentemente.

1.5.1 Estructura de un plan de orden parcial.

La Figura 1.3 muestra un plan de orden parcial para el problema de transporte de ejemplo, correspondiente al problema de la Tabla 1.2, en tres etapas a lo largo de su construcción. Los elementos que definen un plan de orden parcial son los siguientes:

- **Pasos del plan**. Un paso es una versión total o parcialmente instanciada de un operador del problema. Por consiguiente, cada paso de un plan se compone de un conjunto de precondiciones y efectos que están, a priori, total o parcialmente instanciados. Cuando finalmente se obtiene el plan que resuelve el problema, los pasos del plan serán instancias completas de los operadores del plan (acciones). En la Figura 1.3-a) se muestra el plan vacío inicial, el cual se compone de dos pasos ficticios, el paso inicial `I` y el paso final `F`. El paso `I` no tiene precondiciones y sus efectos se corresponden con los literales del estado inicial. El paso `F` tiene como precondiciones el objetivo del problema, en este caso {`pos(p1,cB)`}, y no tiene efectos. En la Figura 1.3-b) se puede observar que aparecen tres nuevos pasos: los pasos `dcg(p1,c1,cB)` y `mv(c1,cA,cB)` se corresponden con instanciaciones totales de los operadores descargar y mover, respectivamente, mientras que el paso `cg(p1,c1,?ciu)` es una versión parcialmente instanciada del operador cargar ya que el parámetro `?ciu` podría tomar, a priori, los valores `cA` o `cB`.

- **Restricciones de orden**. Una restricción de orden (**precedencia**) entre dos pasos del plan P_i, P_j se representa como $P_i < P_j$ y denota que el paso P_i debe ejecutarse antes que el paso P_j, aunque no necesariamente inmediatamente antes. Las restricciones de orden establecen una relación de orden parcial entre sus elementos. Esta relación de orden parcial es una relación binaria irreflexiva, antisimétrica (una restricción de orden no se puede añadir si genera un ciclo) y transitiva. En la Figura 1.3-a) se establece una restricción de orden entre los pasos `I` y `F` para indicar que el paso `I` siempre se ejecutará antes que el paso `F`, mientras que en la Figura 1.3-c) se establece una restricción de orden entre los pasos `cg(p1,c1,?ciu)` y `mv(c1,cA,cB)` para indicar la precedencia de ejecución del primer paso.

- **Restricciones entre variables**. Un POP también puede contener un conjunto de restricciones sobre variables como restricciones de desigualdad del tipo $?x \neq ?y$, donde $?x$ es una variable de un paso del plan e $?y$ es una constante u otra variable. Otro tipo de restricciones que puede contener son las denominadas restricciones de *codesignación* o *unificación*. Éstas son restricciones del tipo $?x = ?y$ donde se fuerza que la variable $?x$ tenga el mismo valor que $?y$. En la Figura 1.3-c) se establece la restricción de unificación `?ciu=cA` del paso `cg(p1,c1,?ciu)`. De este modo, las precondiciones del paso quedarán totalmente instanciadas y se podrán establecer sendos enlaces causales con los efectos del paso `I`. Nótese que el hecho de no ligar un valor a una variable en el momento de insertar el paso es otra variante de menor compromiso. Si en el momento de insertar un paso se desconoce el valor exacto de uno de sus parámetros, éste puede dejarse sin instanciar y en su defecto se le asigna todos los posibles valores del dominio de la variable. La restricción de unificación para dicha variable se establecerá en el momento que se disponga de suficiente información en el plan para poder asignar un valor irrevocablemente. La ventaja de aplicar una estrategia de menor compromiso en las restricciones sobre variables es que, de ese modo, se evita generar tantos planes parciales como valores pueda tomar la variable, y en su lugar se genera un único plan parcial donde la variable contiene inicialmente todos los posibles valores de su dominio.

- **Enlaces causales y amenazas**. Un enlace causal entre dos pasos de un plan P_i, P_j se escribe como $\langle P_i, l, P_j \rangle$ y denota que P_i tiene un efecto que consigue el literal l para una precondición de P_j. En las Figuras 1.3-b),c) se pueden encontrar varios enlaces causales; por ejemplo, $\langle$`dcg(p1,c1,cB)`, `pos(p1,cB)`, `F`$\rangle$ o bien $\langle$`mv(p1,cA,cB)`, `pos(c1,cB)`, `dcg(p1,c1,cB)`$\rangle$. Todo enlace causal establece por defecto una restricción de orden entre los dos pasos. En un enlace causal el literal que el paso productor consigue para el paso consumidor debe mantenerse cierto desde el punto de tiempo del primer paso hasta el punto de tiempo del segundo. Es decir,

no debe existir acción alguna que entre en conflicto con el enlace causal. Un paso P_k entra en conflicto o **amenaza** un enlace causal $\langle P_i, l, P_j \rangle$ si P_k tiene un efecto $\neg l$ y P_k puede situarse entre P_i y P_j. Para resolver la amenaza el paso P_k debería situarse antes de P_i (método *promoción*) o después de P_j (método *democión*). En la Figura 1.3-c), una vez establecida la restricción de unificación del parámetro `?ciu`, el paso `mv(c1,cA,cB)` amenaza el enlace causal $\langle$`I`, `pos(c1,cA)`, `cg(p1,c1,cA)`$\rangle$ porque dicho paso borra el efecto `pos(c1,cA)`. Como el paso `mv(c1,cA,cB)` no puede situarse antes del paso `I`, la única posibilidad de resolver la amenaza es aplicando democión para que el paso `mv(c1,cA,cB)` se sitúe detrás de `cg(p1,c1,cA)`. Es interesante destacar que se pueden generar amenazas *potenciales* cuando los literales en conflicto involucran variables, en cuyo caso se pueden resolver a través de restricciones de desigualdad. Supongamos que el parámetro `?ciu` del paso `cg(p1,c1,?ciu)` no se ha instanciado aún. Dado que el paso `mv(c1,cA,cB)` tiene el efecto $\neg$`pos(c1,cA)` se produce una amenaza potencial porque los valores de la variable `?ciu` serían todos los posibles valores de su dominio, es decir las ciudades {`cA`,`cB`}. Una posible forma de resolver una potencial amenaza es eliminando del dominio de la variable el valor que provocaría la amenaza, es decir, estableciendo la restricción `?ciu` $\neq$ `cA` (método *separación*).

1.5.2 Búsqueda en un espacio de planes para POP.

Un proceso de búsqueda en un espacio de planes realiza una exploración en un árbol donde cada nodo representa un plan parcial. El Algoritmo 1.2 muestra el esquema básico de búsqueda en un espacio de planes. El nodo raíz del árbol será un plan vacío consistente únicamente en los pasos `I` y `F`. Dicho plan se introduce en la lista *Lista_planes*, donde se irán añadiendo sucesivamente los nodos frontera del árbol y de donde se escogerán aleatoriamente los nodos, es decir los planes Π (inicialmente se escogerá el plan vacío ya que es el único plan de la lista). Una vez seleccionado el plan Π con el que se va a trabajar en la actual iteración del algoritmo, se obtienen las precondiciones no resueltas y amenazas del plan y se introducen en la lista *Pendiente*. Si no hay nada pendiente de resolución entonces eso indica que el plan es correcto (resuelve el problema) y se devuelve dicho plan. En caso contrario, se selecciona uno de los problemas pendientes de resolución, se extrae de la lista y se buscan todas las formas de resolver dicho problema. Si se trata de una precondición no resuelta de un paso, existen dos formas genéricas de resolución:

- **Reutilizar un paso existente**. Por cada paso existente en el plan que resuelve dicha precondición se genera un nodo sucesor o plan parcial que incluye el enlace causal correspondiente.
- **Insertar un nuevo paso**. Por cada acción relevante para la precondición se genera un nodo sucesor o plan parcial, incluyendo un nuevo paso asociado a la acción y el enlace causal correspondiente.

Si el problema pendiente de resolución es una amenaza, entonces se genera un nodo sucesor o plan parcial por cada una de las métodos de resolución (promoción, democión o separación) que sean aplicables. Por ejemplo, en la amenaza de la Figura 1.3-c), no es posible aplicar el método promoción porque el paso `mv(c1,cA,cB)` no puede situarse delante del paso `I`.

A tenor de lo explicado se puede ver que las acciones aplicables en esta búsqueda no son acciones ejecutables en el mundo real sino acciones sobre los planes como añadir un paso, establecer un orden entre acciones, etc. El algoritmo continúa seleccionando el siguiente plan parcial a expandir y repitiendo las mismas operaciones. El proceso termina cuando: 1) se selecciona un plan parcial que

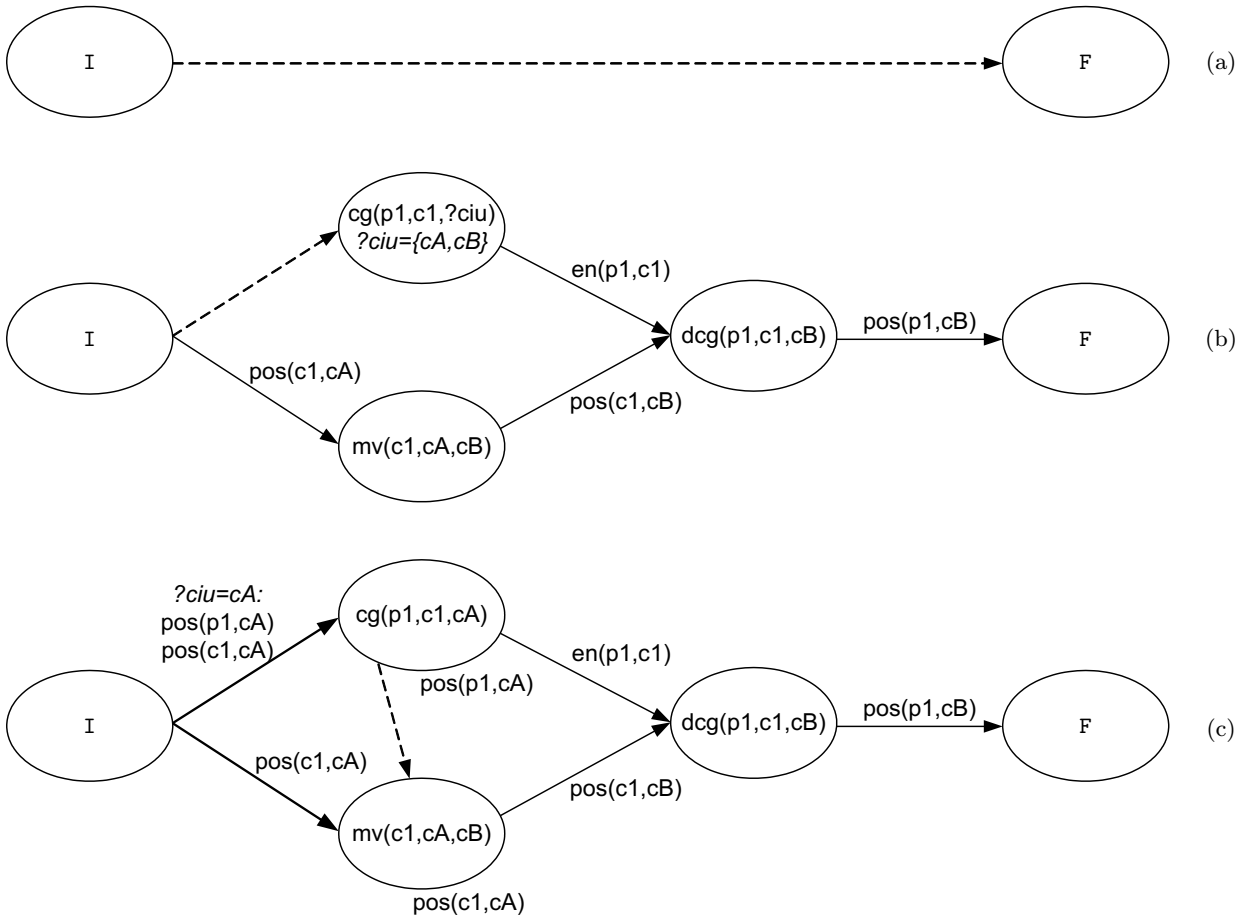

Figura 1.3: Ejemplo de un plan de orden parcial para el problema de transporte (véase Tabla 1.2). Las flechas continuas representan enlaces causales y las punteadas restricciones de orden.

es correcto, es decir, que resuelve el problema, o 2) la lista de nodos a expandir del árbol se queda vacía, lo que indica que el problema no tiene solución. En el caso de nuestro ejemplo se seleccionaría un nodo con el plan parcial de la Figura 1.3-c), que contiene justamente el plan que resuelve el problema.

Algoritmo 1.2 Esquema básico de búsqueda en un espacio de planes para POP.

1: $Lista\_planes \leftarrow$ {Plan vacío}
2: **repetir**
3: Seleccionar no determinísticamente y extraer $\Pi \in Lista\_planes$
4: $Pendiente \leftarrow prec\_no\_resueltas(\Pi) \bigcup amenazas(\Pi)$
5: **si** $Pendiente = \emptyset$ **entonces**
6: Devuelve Π *{Éxito}*
7: **fin si**
8: Seleccionar y extraer $\Phi \in Pendiente$
9: $Relevante \leftarrow \{\Pi_r\} \forall\ \Pi_r$ que resuelve Φ *{Cada r es una forma (plan parcial) de resolver* Φ*}*
10: **si** $Relevante \neq \emptyset$ **entonces**
11: $Lista\_planes \leftarrow Lista\_planes \bigcup Relevante$
12: **fin si**
13: **hasta** $lista\_planes = \emptyset$
14: Devuelve fallo *{No existe plan}*

Un POP que utilice el algoritmo de la Figura 1.2 generará todas las posibles formas de resolver tanto las precondiciones pendientes como las amenazas existentes. Por otro lado, la selección que se realiza en el paso 8 del algoritmo es un paso determinístico ya que se deben resolver **todas** las precondiciones y amenazas antes de alcanzar un plan solución. Esto indica que un algoritmo POP generará todos los posibles planes parciales para resolver un problema. Sin embargo, el procedimiento POP será completo sólo si garantiza que se explorarán todos los planes parciales hasta un determinado nivel de profundidad del árbol. Para ello sería necesario emplear una estrategia de búsqueda de profundización iterativa, como las mostradas en la Sección **??** del Capítulo **??**, donde se va incrementando progresivamente el límite de la solución buscada. En caso contrario, la búsqueda podría seleccionar un camino del espacio de búsqueda y continuar la exploración de dicho camino a niveles cada vez más profundos, añadiendo indefinidamente nuevas acciones al plan actual sin realizar ninguna vuelta atrás.

A pesar de las ventajas que las aproximaciones POP aportan sobre la búsqueda basada en estados, la complejidad de la resolución de un problema sigue siendo muy elevada. El factor de ramificación de un árbol de planes se puede expresar con la siguiente fórmula: $(B_e + B_n) * m^a$, donde B_e y B_n son las formas de resolver una precondición con pasos existentes en el plan o introduciendo pasos nuevos, respectivamente; m son las formas de resolver una amenaza y a es el número de amenazas de un plan. Hay que tener en cuenta que por cada alternativa de resolver una precondición se genera un plan parcial, y para cada uno de ellos hay que estudiar y resolver las amenazas que puedan existir (m^a). Considerando que un plan contiene p precondiciones, tendríamos que el factor efectivo de ramificación sería $((B_e + B_n) * m^a)^p$, y asumiendo que el plan solución para un problema tiene n pasos, tendríamos que el número de nodos que se pueden generar en un proceso de búsqueda en un espacio de planes es $((B_e + B_n) * m^a)^{pn}$. En términos efectivos este factor puede resultar en una explosión combinatoria para resolver instancias de problemas de una cierta complejidad. Aunque es innegable el avance que supusieron las aproximaciones POP a la planificación, éstas resultan aún demasiado costosas.

Ejemplo. Retomando el ejemplo de la Figura 1.3, veremos cómo se desarrolla el proceso de búsqueda del Algoritmo 1.2 para transformar el plan de la Figura 1.3-a) en el plan de la Figura 1.3-c). Inicialmente, el plan vacío se corresponde con el que aparece en la Figura 1.3-a). La única precondición pendiente de resolución es `pos(p1,cB)` del paso `F` por lo que la lista *Pendiente* solo contendrá este objetivo. Para resolver dicha precondición sólo existe la alternativa de añadir el paso `dcg(p1,c1,cB)` que proporciona el enlace causal correspondiente; por consiguiente, sólo se generará un nodo sucesor que contendrá el plan parcial donde se incluye el nuevo paso para descargar el paquete. En la siguiente iteración se escoge el único plan de *Lista_Planes*, y se analiza los dos objetivos de la lista *Pendiente*, que serán las dos precondiciones {`en(c1,p1)`,`pos(c1,cB)`} del paso `dcg(p1,c1,cB)`. Supongamos que se escoge la precondición `en(p1,c1)`. Un análisis de la resolución de dicha precondición revela los siguientes aspectos:

- No existe forma de resolver dicha precondición mediante reutilización de un paso existente, ya que el único paso anterior a `dcg(p1,c1,cB)` es `I`, y éste no contiene un efecto que pueda resolver la precondición.
- Una forma de resolver la precondición es añadiendo el paso `cg(p1,c1,cA)` o bien el paso `cg(p1,c1,cB)`. En principio esta doble posibilidad generaría dos nodos sucesores pero dado que se trata del mismo operador cuyo tercer argumento puede asumir dos valores diferentes, se genera un único plan donde la variable `?ciu` toma provisionalmente todos los valores de su dominio, `?ciu`={`cA`,`cB`}.

De nuevo se tiene que la lista *Lista_Planes* contiene un único plan, el cual consiste momentáneamente en los pasos {`I, cg(p1,c1,?ciu), dcg(p1,c1,cB), F`}. La lista *Pendiente* estaría ahora formada por la precondición pendiente del paso descargar y las dos precondiciones del paso cargar. Asumiendo que se selecciona `pos(c1,cB)` del paso descargar, la única posibilidad de resolverla es insertando un nuevo paso, `mv(c1,cA,cB)`. En este punto existen tres precondiciones pendientes de resolución, la del paso `mv(c1,cA,cB)` y las dos del paso `cg(p1,c1,?ciu)`. Si se selecciona la precondición `pos(c1,cA)` del paso mover, se puede reutilizar el paso existente `I` y establecer el enlace $\langle$`I`, `pos`(`c1`, `cA`), `mv`(`c1`, `cA`, `cB`)$\rangle$ o bien insertar un nuevo paso `mv(c1,cB,cA)` cuyo objetivo sería llevar el camión a `cA`. En este punto del árbol se generaría una ramificación con las dos posibles soluciones.

Supongamos que el algoritmo escoge el plan donde se establece el enlace causal con el paso `I`, situación que se corresponde con la Figura 1.3-b). En este punto sólo quedarían las dos precondiciones del paso `cg(p1,c1,?ciu)` en la lista *Pendiente*. Para resolver la precondición `pos(p1,?ciu)` existen dos opciones, al igual que con la precondición del paso mover: establecer un enlace causal con el paso existente `I`, lo que ligaría la variable `?ciu` al valor `cA`, o bien añadir un nuevo paso `dcg(p1,c1,?ciu)`. Asumiendo que el proceso de búsqueda escoge el plan donde se reutiliza el paso `I`, la resolución de la siguiente precondición `pos(c1,cA)` produciría las mismas alternativas que anteriormente con la precondición del paso mover. Si el algoritmo de búsqueda escoge el plan que reutiliza el paso `I`, todas las precondiciones estarán resueltas pero entonces aparece la amenaza del paso `mv(c1,cA,cB)` al enlace causal $\langle$`I`, `pos`(`c1`, `cA`), `cg`(`p1`, `c1`, `cA`)$\rangle$. De los dos métodos posibles para resolver la amenaza, promoción y democión, sólo es aplicable el segundo ya que ningún paso puede situarse antes del paso `I`. Se establece por tanto la la restricción de orden que aparece en la Figura 1.3-c), la cual determina que la operación de cargar debe realizarse antes que mover el camión.

En la siguiente iteración, el algoritmo puede seleccionar el nodo que representa el plan de la Figura 1.3-c) o bien cualquiera de los nodos abiertos correspondientes a las ramificaciones producidas

durante el proceso de búsqueda. Si se selecciona el nodo de la Figura 1.3-c), como no existen precondiciones ni amenazas que resolver, el algoritmo termina con éxito devolviendo dicho plan.

1.5.3 Heurísticas para planificación de orden parcial.

Los dos puntos claves en una búsqueda en un espacio de planes son la selección del plan de *Lista_Planes* y la selección, dentro de dicho plan, del objetivo pendiente a estudiar. Estos dos puntos son críticos, ya que de las elecciones tomadas dependerá el coste de resolución del problema. Existen diferentes estrategias o heurísticas enfocadas a tomar la mejor decisión posible y enfocar así la búsqueda hacia el camino donde se encuentra la solución.

Heurísticas para selección de planes.

La estrategia habitual es ordenar los nodos frontera del árbol por el coste total de los operadores del plan (número de pasos) y seleccionar el plan con el mínimo coste. Se puede diseñar un algoritmo $A^\star$ con función $f(\Pi) = g(\Pi) + h(\Pi)$, donde $g(\Pi)$ representa el coste de la solución desde el nodo raíz hasta plan Π, y $h(\Pi)$ representa la estimación del coste de la solución desde el plan Π hasta el objetivo. Para diseñar una buena estrategia heurística hay que determinar cómo afectan cada uno de los componentes de un plan a los factores $g(\Pi)$ y $h(\Pi)$:

- **Número de pasos** N: se utiliza habitualmente como una medida del coste del camino ya que la complejidad de un plan viene dado por el número de pasos que contiene.
- **Precondiciones sin resolver** OP: dado que para cada precondición pendiente de resolver se debe introducir un paso, OP se puede incluir como medida heurística en $h(\Pi)$. En principio, podría deducirse que OP sobreestima $h^\star(\Pi)$ ya que no se tiene en cuenta la reutilización de pasos existentes para resolver una precondición, es decir, no siempre es necesario añadir un paso para resolver una precondición. Sin embargo, la posibilidad de que OP sobreestime el coste restante del plan no es habitual porque, típicamente, se necesitarán pasos futuros adicionales para conseguir algunos objetivos y estos pasos, a su vez, requerirán nuevos pasos que no se contemplan en el valor devuelto por OP. Por tanto, OP se puede incluir en $h(\Pi)$ con una alta garantía de que en la mayoría de situaciones no sobreestimará $h^\star(\Pi)$.
- **Amenazas** A: son un factor circunstancial de un plan que aparecen en un momento concreto y que pueden resolverse, o bien desaparecer, a medida que el plan se va construyendo. Por dicha razón, A no se incluye en $h(\Pi)$ pero puede considerarse como una medida del coste de encontrar una solución (a mayor número de amenazas más costoso será, en principio, encontrar la solución).

La combinación que se ha utilizado habitualmente en las aproximaciones POP es $f(\Pi) = N(\Pi) + OP(\Pi)$, donde existen altas garantías de que $OP(\Pi) < h^\star(\Pi)$ y por tanto se pueda encontrar el plan óptimo para el problema, es decir el plan de menor número de pasos o acciones.

Heurísticas para selección de objetivo pendiente.

Existen múltiples estrategias que se han aplicado en las aproximaciones POP. Básicamente, las estrategias existentes priorizan entre escoger una amenaza o una precondición pendiente de resolución y entre ellas se establece otra prioridad para determinar el objetivo candidato a ser estudiado:

- **LIFO**. Esta estrategia aplica una estrategia LIFO para seleccionar el siguiente objetivo pendiente a estudiar.
- **Retrasar las amenazas potenciales**. Esta estrategia determina que es preferible ignorar cualquier amenaza potencial que pueda haber en un plan y que los objetivos pendientes deben resolverse en el siguiente orden: 1) amenazas no resolubles, 2) amenazas que puedan resolverse con un solo método, 3) precondiciones sin resolver, y 4) amenazas que puedan resolverse con más de un método.
- **Objetivo de menor coste**. Esta estrategia selecciona la amenaza o precondición pendiente de menor coste de reparación, es decir, aquel objetivo pendiente con menos alternativas de solución. En el caso de precondiciones, el número de alternativas vendrá dado por el número de pasos existentes (B_e) y pasos nuevos (B_n) que pueden crearse para resolver la precondición.
- **ZLIFO** (*Zero LIFO*). El objetivo de esta estrategia es dar mayor prioridad a aquellas precondiciones que puedan resolverse determinísticamente (0 o 1 solución). Para ello, establece el siguiente nivel de prioridades: 1) escoger una amenaza (si hay varias, aplicar una estrategia LIFO), 2) escoger una precondición pendiente con 0 formas de resolverse (precondición irresoluble), 3) escoger una precondición para la que sólo exista una alternativa de solución, y 4) escoger una precondición pendiente aplicando una estrategia LIFO. A diferencia de la heurística objetivo de menor coste, *ZLIFO* no calcula el coste de reparación de las precondiciones ni de las amenazas, sino que simplemente determinan si existen 0, 1 o varias formas de resolverlas. En el último caso se selecciona una precondición aplicando la estrategia LIFO.

Aunque no existe una estrategia heurística universal para toda la tipología de problemas, la estrategia *ZLIFO* ha sido la heurística que mejores resultados ha obtenido en términos generales.

Ejemplo. Consideremos de nuevo el ejemplo utilizado para mostrar el proceso de búsqueda en el espacio de planes de la Figura 1.3, situándose en el punto de resolución de la precondición `pos(c1,cA)` del paso `mv(c1,cA,cB)`. Existen dos alternativas: reutilizar el paso `I`, o introducir un nuevo paso `mv(c1,cB,cA)`. La aplicación de la función $f(\Pi) = N(\Pi) + OP(\Pi)$ para cada una de estas dos alternativas daría los siguientes resultados:

- **Reutilizar paso** `I`. El número de pasos del plan si se reutiliza el paso `I` sería 5, y el número de precondiciones por resolver sería 2 (las dos precondiciones del paso cargar). Por tanto: $f(\Pi) = N(\Pi) + OP(\Pi) = 5 + 2 = 7$.
- **Insertar paso nuevo**. En este caso el número de pasos del plan sería 6 y el número de precondiciones por resolver sería 3 (las dos precondiciones del paso cargar y la precondición del paso `mv(c1,cB,cA)`). Por tanto: $f(\Pi) = N(\Pi) + OP(\Pi) = 6 + 3 = 9$.

De acuerdo a esta información, la alternativa seleccionada sería la de reutilizar el paso `I` pues es menos costosa que la de insertar un paso nuevo.

1.6 Planificación basada en grafos de planificación.

La planificación basada en grafos de planificación consiste en la construcción, análisis y explotación de una estructura bidimensional, que representa de forma compacta las proposiciones, acciones y ciertas restricciones entre ambas que aparecen durante la resolución de un problema de planificación.

1.6.1 Grafos de planificación.

Un grafo de planificación es una estructura de tamaño polinómico, similar a las empleadas en programación dinámica, que modela la parte estática (proposiciones y acciones) y dinámica (relaciones de orden). La única restricción que impone el grafo de planificación es la de manejar un enfoque **proposicional** (véase Asunción A4 en Sección 1.1) basado en STRIPS, es decir con literales positivos. Tanto las acciones como proposiciones (literales positivos) deben estar totalmente instanciadas, esto es sin variables. Más precisamente, un grafo de planificación es un grafo dirigido multinivel con dos tipos de nodos (proposición y acción) y tres tipos de aristas (aristas-Pre, aristas-Efe$^+$ y aristas-Efe$^-$). Cada nivel t del grafo contiene, a su vez, un nivel de acción $A_{[t]}$ y uno de proposición $P_{[t]}$ con las acciones y proposiciones presentes, respectivamente, en dicho nivel.

Ejemplo. Si consideramos el problema de transporte definido en la Tabla 1.2, el grafo de planificación resultante es el que se muestra en la Figura 1.4. Este grafo se construye incrementalmente a partir de la información de la situación inicial $\mathcal{I}$, que constituye el nivel $P_{[0]}$. Las aristas del grafo sólo pueden unir nodos de niveles adyacentes; una acción del nivel t está conectada a todas sus precondiciones en el nivel $t-1$ mediante aristas-Pre y a sus efectos en el mismo nivel t, tanto positivos como negativos mediante aristas-Efe$^+$ y aristas-Efe$^-$, respectivamente. Así por ejemplo, la acción `cg(p1,c1,cA)` en $A_{[1]}$ tiene como precondiciones en $P_{[0]}$ el conjunto {`pos(c1,cA)`,`pos(p1,cA)`}, como efecto positivo `en(p1,c1)` y negativo `pos(p1,cA)`, ambos en $P_{[1]}$. Como se puede observar, en los niveles de acción existen unas acciones artificiales denominadas `no-op` (*no-operación*). Estas acciones son habituales en los grafos de planificación y, aunque no aportan nueva información al problema (su único efecto positivo es la precondición que requiere), permiten propagar información a lo largo del grafo, simplificando el *problema marco* y garantizando la persistencia de las proposiciones. En general, existen dos condiciones para detener la construcción del grafo de planificación: 1) cuando en un nivel de proposición se alcanzan todos los objetivos del problema, o 2) cuando se alcanza un nivel que es idéntico al anterior, tanto a nivel de proposiciones como de acciones. En el grafo de la Figura 1.4, la primera condición se cumple en $P_{[2]}$ al alcanzarse el objetivo `pos(p1,cB)`. La segunda condición se cumpliría tras generar el nivel 4 que coincidiría plenamente con el nivel 3 ($A_{[4]} = A_{[3]}$ y $P_{[4]} = P_{[3]}$), al haberse generado todas las posibles proposiciones y acciones (véase Ejercicio 7).

La construcción de un grafo de planificación no conlleva restricciones muy estrictas. En un nivel de acción $A_{[t]}$ aparecen todas las acciones (incluidas las acciones `no-op`) cuyas precondiciones están presentes en el nivel $P_{[t-1]}$, con independencia de si dichas precondiciones pueden coexistir simultáneamente, o si las acciones pueden ejecutarse concurrentemente en el mundo real. Análogamente, en un nivel de proposición $P_{[t]}$ aparecen todas las proposiciones que se han alcanzado a lo largo del grafo de planificación, aunque haya acciones que las eliminen como parte de sus efectos Efe$^-$. Por lo tanto, es importante darse cuenta de que un nivel de un grafo de planificación no representa un estado individual válido, sino la unión de varios de ellos que no siempre pueden satisfacerse simultáneamente en el mundo real. Por esta razón, a un grafo de planificación que no contempla los efectos negativos (Efe$^-$) de las acciones se le conoce habitualmente como **grafo de planificación relajado**. Por ejemplo, en el nivel $P_{[1]}$ del grafo de la Figura 1.4 aparecen simultáneamente las proposiciones {`pos(c1,cA)`,`pos(c1,cB)`} como resultado de una relajación del problema original, a pesar de ser imposible para cualquier camión. Por el contrario, una información importante que se deduce de este grafo es que si una proposición o acción no aparece en un nivel t es porque no existe ninguna combinación factible que permita alcanzar dicha proposición o acción en t pasos de planificación. Por ejemplo, la acción `mv(c1,cB,cA)` no aparece hasta el nivel $A_{[2]}$, lo que indica que

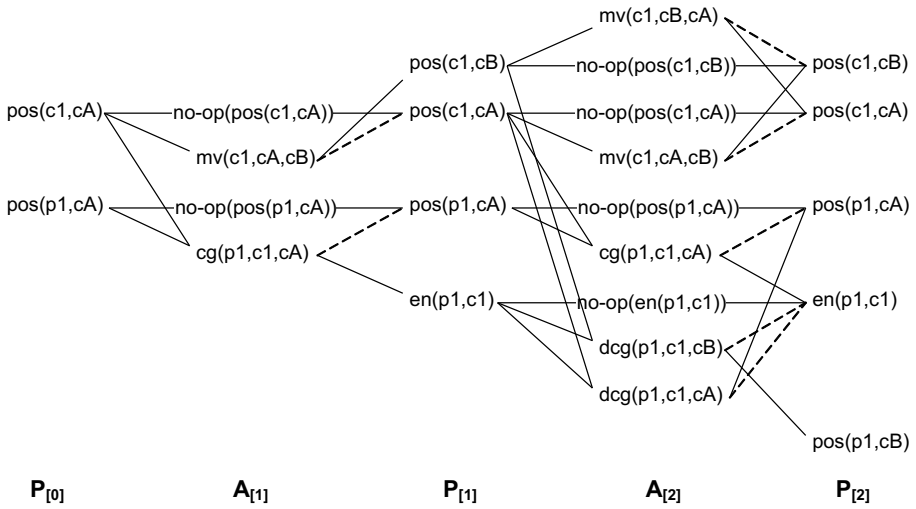

Figura 1.4: Ejemplo del grafo de planificación para el problema de transporte. Las aristas-Pre y aristas-Efe$^+$ se representan mediante líneas continuas, mientras que las aristas-Efe$^-$ se representan por líneas punteadas.

no existe ninguna forma de ejecutar esa acción en el nivel $A_{[1]}$ (obviamente, para mover el camión `c1` desde la ciudad `cB` primero hay que llegar a `cB`, lo que requiere al menos un paso de planificación). No obstante, una vez una acción o proposición aparece en un nivel, se mantiene en todos los niveles siguientes. Esto es consecuencia directa de la aplicación de las acciones `no-op`, que mantienen la proposición invariable a lo largo de los siguientes niveles, permitiendo la ejecución de las acciones que las utilizan como precondiciones. Por esta razón se dice que en un grafo de planificación el número de proposiciones y acciones es monótonamente creciente, pues nunca disminuye.

Los grafos de planificación cumplen varias propiedades que los hacen muy interesantes como fuente básica de información para estimar y mejorar el proceso de planificación:

1. Un grafo de planificación mantiene una noción interna del tiempo mediante los niveles o pasos de planificación, y representa implícitamente muchas de las restricciones entre acciones y proposiciones. Mediante las aristas-Pre y aristas-Efe$^+$ se pueden representar restricciones cualitativas entre acciones, de forma similar a como se hace en planificación de orden parcial con la utilización de enlaces causales.

2. Un grafo de planificación codifica un plan que se encuentra *a mitad camino* entre un plan de orden parcial y uno de orden total. En cada nivel del grafo las acciones se ejecutan concurrentemente (sin imponer un orden de ejecución, tal y como ocurre en un enfoque de orden parcial), pero los niveles deben ejecutarse de forma ordenada (tal y como ocurre en un enfoque de orden total). De esta forma, las acciones que se ejecutan en el nivel t se podrán ejecutar en cualquier orden, pero siempre antes que las del nivel $t+1$.

3. Un grafo de planificación se construye con complejidad temporal y espacial polinómica con respecto al tamaño del problema. La prueba de esta afirmación es sencilla ya que en un dominio proposicional las acciones no pueden crear nuevas proposiciones. Esto obliga a que las acciones y proposiciones sean conjuntos finitos y que el número máximo de nodos acción/proposición

en un nivel de acción/proposición también lo sea. En el ejemplo puede comprobarse como el grafo no crece indefinidamente; por un lado, las proposiciones y acciones son fijas y constituyen un conjunto finito y, por el otro, los pasos de planificación también son finitos pues llega un punto en el que un nivel es idéntico al anterior.

4. Si existe un plan válido de t pasos para un problema de planificación, dicho plan existe como un subgrafo del grafo de planificación correspondiente de t niveles. Inversamente, si no existe ningún plan válido de t pasos para el problema, no existe ningún subgrafo en el grafo de planificación de t niveles que resuelva el problema.

1.6.2 Extracción de planes: Graphplan.

Graphplan es, sin duda, el planificador basado en grafos de planificación más conocido. En lugar de comenzar directamente con un proceso de búsqueda hacia delante o hacia atrás desde la situación inicial o final, respectivamente, actúa en dos etapas. En la primera, Graphplan construye un grafo de planificación hasta un nivel t y, en la segunda, realiza una búsqueda regresiva para extraer un plan de longitud t de dicho grafo.

Relaciones de exclusión mutua.

Un grafo de planificación relajado resulta demasiado permisivo en cuanto a las acciones o proposiciones que pueden aparecer en un determinado nivel, al representar la unión de muchos estados. Si observamos el grafo de la Figura 1.4 puede observarse como en el nivel $A_{[1]}$ podrían ejecutarse simultáneamente las acciones `cg(p1,c1,cA)` y `mv(c1,cA,cB)`. Esto hace que en el nivel $P_{[1]}$ se disponga simultáneamente de las proposiciones {`pos(c1,cB)`,`en(p1,c1)`}, junto con el efecto `pos(c1,cA)`, resultante de aplicar la acción `no-op(pos(c1,cA)`. Obviamente, ninguna de estas dos situaciones es factible en un plan válido. Por lo tanto, resulta necesario contar con un mecanismo que permita conocer cuáles son las relaciones de exclusión en un nivel, es decir, qué elementos (proposiciones/acciones) no pueden coexistir simultáneamente. Adicionalmente, supongamos un problema cuyo plan solución requiere tres pasos de planificación[1]. En tal caso, comenzar la búsqueda desde el nivel 1 o 2 conducirá inevitablemente al fracaso, y será necesario alcanzar el nivel 3 antes de comenzar la etapa de búsqueda. De nuevo, contar con un mecanismo de propagación de relaciones de exclusión puede evitar esta búsqueda infructuosa.

Graphplan amplía la construcción del grafo de planificación con un razonamiento sobre relaciones de exclusión mutua entre pares de nodos de un mismo nivel, bien sea acción-acción o proposición-proposición. En general, se dice que dos acciones o proposiciones de un mismo nivel son mutuamente excluyentes (*mutex*) si ningún plan válido puede contenerlas simultáneamente, es decir, no pueden coexistir en el mundo real. El cálculo de esta información de exclusión mutua produce un grafo de planificación más informado que permite detectar con mayor precisión el nivel desde el que comenzar la búsqueda y restringir el espacio de búsqueda. En general, identificar todas las relaciones de exclusión puede resultar muy costoso, por lo que Graphplan calcula y propaga estas relaciones mediante la aplicación de unas reglas sencillas. En el caso de las acciones, dos acciones `a` y `b` son mutex cuando: 1) `a` borra una precondición o efecto positivo de `b` (*interferencia*), o 2) una precondición de `a` y una de `b` se han marcado como mutuamente excluyentes en el nivel de proposición anterior (*necesidades competitivas*). En el caso de las proposiciones, dos proposiciones `p` y `q` son mutex si todas las formas (acciones) de alcanzar `p` son excluyentes con todas las formas de alcanzar

[1] El lector debe recordar que un grafo de planificación de tres pasos (niveles) de planificación se traducirá en un plan de, al menos, tres acciones (una por nivel).

Nivel	Relaciones mutex
$A_{[1]}$	{mv(c1,cA,CB)} × {cg(p1,c1,cA)}
$P_{[1]}$	{pos(c1,cB)} × {pos(c1,cA),en(p1,c1)}
	{pos(p1,cA)} × {en(p1,c1)}
$A_{[2]}$	{mv(c1,cB,cA)} × {mv(c1,cA,cB),cg(p1,c1,cA),dcg(p1,c1,cA)}
	{mv(c1,cA,cB)} × {cg(p1,c1,cA),dcg(p1,c1,cA)}
	{cg(p1,c1,cA)} × {dcg(p1,c1,cA)}
$P_{[2]}$	{pos(c1,cB)} × {pos(c1,cA)}
	{pos(p1,cA)} × {en(p1,c1)}
$A_{[3]}$	{mv(c1,cB,cA)} × {mv(c1,cA,cB),cg(p1,c1,cA),dcg(p1,c1,cB),dcg(p1,c1,cA)}
	{mv(c1,cA,cB)} × {cg(p1,c1,cA),dcg(p1,c1,cB),dcg(p1,c1,cA)}
	{cg(p1,c1,cA)} × {dcg(p1,c1,cB),dcg(p1,c1,cA)}
	{dcg(p1,c1,cB)} × {dcg(p1,c1,cA))}
$P_{[3]}$	{pos(c1,cB)} × {pos(c1,cA)}
	{pos(c1,cA)} × {pos(p1,cB)}
	{pos(p1,cA)} × {en(p1,c1),pos(p1,cB)}
	{en(p1,c1)} × {pos(p1,cB)}

Tabla 1.2: Mutex resultantes (complementarios al grafo de planificación) para el ejemplo de transporte. Por simplicidad, los mutex en los que se involucran acciones `no-op` no se muestran.

q. Es importante notar que los mutex entre proposiciones son consecuencia directa de la propagación de los mutex entre acciones a lo largo del grafo de planificación. De esta forma cuando dos acciones dejan de ser mutex, el mutex entre sus efectos también desaparece (existe al menos una forma de generar los efectos que no es mutex). Obviamente, el grafo de planificación resultante proporciona más información que el correspondiente grafo relajado, al considerarse la interacción entre los efectos negativos de las acciones.

Ejemplo. En el problema de transporte definido en la Tabla 1.2, los mutex que aparecen en cada nivel son los que aparecen en la Tabla 1.6.2. Por ejemplo, en el nivel $A_{[1]}$ tenemos la pareja de mutex {mv(c1,cA,CB),cg(p1,c1,cA)}, al eliminar mv(c1,cA,CB) el camión c1 de cA, algo que requiere la acción de carga. La propagación de los mutex del nivel $A_{[1]}$ hace que en $P_{[1]}$ la proposición pos(c1,cB) sea mutex, entre otras, con en(p1,c1), situación que desaparece en el nivel $P_{[2]}$, permitiendo la ejecución de dcg(p1,c1,cB) en $A_{[3]}$. Además, también se detectan mutex que, a priori, no son directamente observables como la pareja {mv(c1,cA,CB),dcg(p1,c1,cB)} en $A_{[3]}$ (claramente, en el problema real estas dos acciones no se pueden ejecutar a la vez). Es justamente esta propagación de mutex la que permite generar un grafo de planificación más informado y preciso. En Graphplan, si las precondiciones de una acción son mutex, ésta no se genera. En la Figura 1.4, la acción dcg(p1,c1,cB) y el objetivo pos(c1,cB) aparecen en el nivel 2, mientras que con la propagación de los mutex no aparecen hasta el nivel 3, resultando más acorde con el problema real. Por tanto, el número de niveles del grafo de planificación de Graphplan nunca será menor que el de uno relajado.

Aunque las reglas que aplica Graphplan no garantizan encontrar todas las relaciones de exclusión, sí son lo suficientemente potentes para determinar cuándo dos acciones o dos proposiciones no pueden darse simultáneamente. De hecho, la condición de terminación para la construcción del grafo de planificación en Graphplan es la de alcanzar un nivel en el que los objetivos del problema

dejan de ser mutex dos a dos[2].

Algoritmo de Graphplan.

El esquema de funcionamiento de Graphplan se muestra en el Algoritmo 1.3. Tras construir el grafo de planificación, se realiza la etapa de extracción de un plan de igual longitud a la del grafo, que es donde se encuentra la verdadera complejidad temporal de la planificación. Si no se encuentra un plan en el grafo actual, puede ocurrir que no sea posible encontrar solución y, por tanto el problema no tenga solución, o sea necesario extender un nuevo nivel del grafo y repetir la etapa de búsqueda. Básicamente, $Extraer\_Plan$ realiza una búsqueda regresiva basada en backtracking cronológico extrayendo un plan como un *flujo de acciones* del grafo de planificación. Para ello selecciona un conjunto de acciones que resuelven los objetivos en un nivel t, luego trata de resolver las precondiciones de dichas acciones en el nivel $t-1$, y así sucesivamente hasta llegar al nivel 0. Por otro lado, $No\_Es\_Posible\_Encontrar\_Plan$ consiste en comprobar si se ha alcanzado un nivel que no aporta nada respecto al anterior, porque: 1) las proposiciones, acciones y mutex son idénticos, y 2) el mismo conjunto de objetivos sigue siendo irresoluble. Con este esquema y para el problema de ejemplo que nos ocupa, Graphplan encontraría el plan de longitud 3 $\langle$`{cg(p1,c1,cA)}`$_{[1]}$, `{mv(c1,cA,cB)}`$_{[2]}$, `{dcg(p1,c1,cB)}`$_{[3]}\rangle$.

Algoritmo 1.3 Esquema básico de Graphplan.

```
1:  GP ← Construir_Grafo_Planificacion(I) {Construcción del grafo de planificación}
2:  bucle
3:    Π ← Extraer_Plan(GP, Longitud(GP), G) {Búsqueda}
4:    si Π es solución entonces
5:      Devuelve Π {Éxito}
6:    si no
7:      si No_Es_Posible_Encontrar_Plan(GP, Longitud(GP)) entonces
8:        Devuelve fallo {Garantía de terminación si no existe plan}
9:      si no
10:       GP ← Extender_Un_Nivel(GP) {Construcción del grafo de planificación}
11:     fin si
12:   fin si
13: fin bucle
```

Graphplan propone un algoritmo de planificación correcto, completo y óptimo. La correctitud y completitud son consecuencia de la construcción del grafo de planificación y de la completitud de la búsqueda basada en backtracking. La optimalidad se garantiza en términos de pasos de planificación (niveles), pero no en número de acciones ya que múltiples acciones se pueden ejecutar en paralelo en cada paso.

Graphplan *vs.* resolución de un CSP.

El proceso de búsqueda llevado a cabo por Graphplan se puede considerar equivalente al de resolución de un CSP. El primer paso consiste en codificar el grafo de planificación como un CSP

[2]Esta condición de terminación es necesaria para poder encontrar un plan, pero no suficiente. Por lo tanto, no existe garantía de que un plan válido se pueda encontrar desde el nivel actual (los mutex se calculan por parejas, pero puede existir un mutex que involucre a más de dos proposiciones/acciones que no se haya detectado).

Variables	pos(c1,cB)={mv(c1,cA,cB)}, pos(c1,cA)={no-op(pos(c1,cA))}, pos(p1,cA)={no-op(pos(p1,cA))}, en(p1,c1)={cg(p1,c1,cA)}
R. Mutex	pos(c1,cB)=mv(c1,cA,cB) ⇒ en(p1,c1)≠cg(p1,c1,cA) ∧ pos(c1,cA)≠no-op(pos(c1,cA))
	pos(c1,cA)=no-op(pos(c1,cA)) ⇒ pos(c1,cB)≠mv(c1,cA,cB)
	pos(p1,cA)=no-op(pos(p1,cA)) ⇒ en(p1,c1)≠cg(p1,c1,cA)
	en(p1,c1)=cg(p1,c1,cA) ⇒ pos(c1,cB)≠mv(c1,cA,cB) ∧ pos(p1,cA)≠no-op(pos(p1,cA))
R. Activación	pos(c1,cB)=mv(c1,cA,cB) ⇒ Activa(pos(c1,cA))
	pos(c1,cA)=no-op(pos(c1,cA)) ⇒ Activa(pos(c1,cA))
	pos(p1,cA)=no-op(pos(p1,cA)) ⇒ Activa(pos(p1,cA))
	en(p1,c1)=cg(p1,c1,cA) ⇒ Activa(pos(c1,cA) ∧ pos(p1,cA))

Tabla 1.3: Ejemplo de CSP dinámico resultante para un fragmento de un grafo de planificación de Graphplan.

dinámico, y el segundo en utilizar cualquiera de las técnicas estudiadas en el Capítulo **??** para resolverlo.

Para convertir un grafo de planificación en un CSP dinámico basta con aplicar las siguientes reglas sencillas. Las proposiciones (incluyendo objetivos) constituyen las variables, y las acciones que las generan (relevantes) constituyen los dominios de dichas variables. Las relaciones de exclusión entre acciones se representan mediante *restricciones de mutex*: si las proposiciones `p` y `q` son generadas por las acciones `a` y `b`, respectivamente, que son mutex, entonces se genera `p=a` ⇒ `q`≠ `b`. La asignación de valores a las variables es un proceso dinámico porque cada asignación en un determinado nivel activa otras variables en un nivel previo (*restricción de activación*): utilizar la acción `a` para satisfacer `p` en un nivel hace que las precondiciones de `a` se activen en el nivel previo, $\forall$`p` $\in$ Efe$^+$(`a`): `p=a` ⇒ Activa(Pre(`a`)). Inicialmente, sólo las variables correspondientes a los objetivos del problema están activas. En la Tabla 1.6.2 se muestra el CSP dinámico correspondiente a los niveles $P_{[0]}$, $P_{[1]}$ y $A_{[1]}$ del grafo de planificación de la Figura 1.4, teniendo en cuenta los mutex de la Tabla 1.6.2. Esta simple conversión en un CSP facilita su resolución mediante técnicas y criterios heurísticos eficientes ampliamente estudiados.

1.6.3 Heurísticas basadas en grafos de planificación.

Los grafos de planificación proporcionan una estructura eficiente que permite calcular estimaciones heurísticas útiles para guiar la búsqueda en planificadores progresivos basados en estados. La utilización de estos grafos conlleva la relajación de los efectos negativos de las acciones para que el conjunto de proposiciones alcanzables sea siempre creciente[3]. La idea consiste en construir desde cada estado un grafo de planificación hasta el objetivo. De esta forma, el coste h(`p`) de alcanzar una proposición `p` desde un estado inicial se puede estimar como el primer nivel en el que `p` aparece en el grafo de planificación. Por ejemplo según la Figura 1.4, h(pos(c1, cA)) = 0 y h(pos(p1, cB)) = 2. Aunque esta heurística es admisible, también es poco informada pues los problemas de planificación implican generalmente varios objetivos. No obstante, extender esta estimación para aplicarla a un conjunto de objetivos, como por ejemplo $\mathcal{G}$ = {pos(c1, cA), pos(p1, cB)}, resulta bastante intuitivo, apareciendo la siguiente familia de heurísticas:

[3]Realizar una estimación heurística teniendo en cuenta los efectos negativos (las proposiciones ya generadas pueden eliminarse) sería tan costoso como resolver el problema original, y la estimación dejaría de tener utilidad práctica.

- h_{sum}, que estima el coste como la suma de los costes individuales de cada una de las proposiciones: $h_{sum}(\mathcal{G}) = \sum_{\mathtt{g}\in\mathcal{G}} h(\mathtt{g}) = 0 + 2 = 2$. h_{sum} es no admisible porque no tiene en cuenta que una acción genere varios objetivos simultáneamente, pero funciona muy bien cuando los objetivos son independientes entre sí y el problema se puede descomponer fácilmente.

- $h_{\text{máx}}$, que estima el coste como el máximo de los costes individuales: $h_{\text{máx}}(\mathcal{G}) = \text{máx}_{\mathtt{g}\in\mathcal{G}}\, h(\mathtt{g}) = \text{máx}(0, 2) = 2$. $h_{\text{máx}}$ es admisible pero cuando existen muchos objetivos no siempre es suficientemente informada al descartarse el resto de los costes individuales.

- $h_{\text{máx}\,2}$, que estima el coste como el máximo nivel en el que coexisten cada par de objetivos: $h_{\text{máx}\,2}(\mathcal{G}) = \text{máx}_{\{\mathtt{g}_1,\mathtt{g}_2\}\in\mathcal{G}}\, h(\mathtt{g}_1 \wedge \mathtt{g}_2) = \text{máx}(2) = 2$. $h_{\text{máx}\,2}$ sigue siendo admisible y, en general, es mucho más informada que $h_{\text{máx}}$ al tener en cuenta el coste máximo de alcanzar cada par de objetivos.

- $h_{\text{máx}\,k}$, que generaliza $h_{\text{máx}\,2}$, y estima el coste como el máximo nivel en el que coexiste cada tupla de k-objetivos: $h_{\text{máx}\,k}(\mathcal{G}) = \text{máx}_{\{\mathtt{g}_1,\mathtt{g}_2\ldots\mathtt{g}_k\}\in\mathcal{G}}\, h(\mathtt{g}_1 \wedge \mathtt{g}_2 \wedge \cdots \wedge \mathtt{g}_k)$. A pesar de que $h_{\text{máx}\,k}$ es admisible y muy informada, es muy costosa de calcular y resulta poco práctica, por lo que la mayoría de las estimaciones utiliza valores $k \leq 2$.

El razonamiento sobre mutex que lleva a cabo Graphplan en el grafo de planificación, aunque ligeramente más costoso, también resulta beneficioso para la estimación de las heurísticas anteriores. Si se considera la información sobre mutex de la Tabla 1.6.2, el nuevo valor para las heurísticas anteriores es: $h_{sum}(\mathcal{G}) = 0 + 3 = 3$, $h_{\text{máx}}(\mathcal{G}) = \text{máx}(0, 3) = 3$ y $h_{\text{máx}\,2}(\mathcal{G}) = \text{máx}(4) = 4$. Puede observarse como el cálculo de los mutex proporciona información más precisa en los tres casos al tener en cuenta sólo las combinaciones factibles que permiten alcanzar los objetivos. Por ejemplo, la estimación de $h_{\text{máx}\,2}$ coincide con el valor real; según la Tabla 1.6.2 dicho par de objetivos es mutex en el nivel $P_{[3]}$, y no deja de serlo hasta $P_{[4]}$ (tal y como ocurre en el plan solución).

Esta familia de heurísticas es de gran ayuda en planificadores progresivos, aunque son costosas al tener que reconstruir un grafo de planificación desde cada estado hasta el objetivo del problema. No obstante, es fácil adaptar dichas heurísticas para su utilización en planificadores regresivos simplemente construyendo el grafo de planificación desde el estado inicial una **única** vez, estimando el coste a cada una de las proposiciones del problema, y utilizando dichas estimaciones tantas veces como sea necesaria. Así, cada vez que la búsqueda regresiva deba decidirse por uno u otro estado hará uso de este valor y elegirá el que presente un menor coste hasta la situación inicial.

La calidad de las heurísticas anteriores se puede mejorar todavía más, calculando la estimación no sobre un grafo de planificación sino sobre un plan. De nuevo, para calcular el plan se mantiene la misma relajación de eliminación de efectos negativos, por lo que el plan es realmente un **plan relajado** sin interacciones negativas que se puede calcular en tiempo polinómico (véase Ejercicio 6). La información que proporciona este plan es mucho más precisa y variada. En lugar de estimar el coste como número de pasos de planificación, al disponerse de un plan se puede calcular el número de acciones del mismo, o se puede asignar un coste (temporal, económico, etc.) a cada acción y calcular el coste real del plan. Estas estimaciones, aunque bien informadas, suelen ser no admisibles, pues encontrar un plan relajado óptimo resulta tan costoso como encontrar el propio plan. Aún así, los resultados que se obtienen ayudan notablemente a los planificadores y les permiten abordar problemas de elevada complejidad.

1.7 Planificación basada en satisfactibilidad.

La idea de abordar un problema de planificación como un problema basado en satisfactibilidad surge con el objetivo de reutilizar técnicas basadas en satisfacción de fórmulas proposicionales para su aplicación a planificación. El funcionamiento básico se muestra en el Algoritmo 1.4. Un sistema de traducción toma como entrada un problema de planificación con una longitud estimada y genera un conjunto de fórmulas lógicas o axiomas. A continuación, una serie de algoritmos muy eficientes busca una asignación factible (*modelo*) que satisfaga las fórmulas anteriores y, de ser así, ésta se traduce en un plan solución.

Algoritmo 1.4 Esquema básico de planificación basada en satisfactibilidad.

1: $formulas \leftarrow Traducir\_a\_SAT(\mathcal{I} + Acciones + \mathcal{G}, longitud)$
2: $modelo \leftarrow Algoritmo\_Resolucion\_SAT(formulas)$
3: **si** $\exists$ *modelo* **entonces**
4: Devuelve $Extraer\_Plan(modelo)$ *{Éxito, se ha encontrado un plan con la longitud dada}*
5: **si no**
6: Devuelve fallo *{No existe solución con la longitud dada}*
7: **fin si**

La traducción del problema de planificación en fórmulas lógicas es un paso clave, pues la complejidad de los algoritmos de resolución depende en gran medida del número de fórmulas y su longitud. Básicamente, esta traducción debe codificar los estados y las transiciones entre ellos producidas por las acciones. En la Tabla 1.7 se muestra dicha codificación para nuestro problema ejemplo. Un estado se codifica como una fórmula lógica con conjunciones y/o disyunciones de predicados, y debe estar completamente especificado, indicando los predicados que son ciertos y los que no (véase estado $\mathcal{I}$), excepto cuando hay predicados cuyo valor es desconocido o irrelevante (en $\mathcal{G}$ la posición del camión `c1` es indiferente). Además, los estados evolucionan en el tiempo, por lo que es necesario extender cada predicado para indicar que se cumple en un instante de tiempo pero no en otro (por ejemplo el predicado $\texttt{pos}(\texttt{c1}, \texttt{cB})_{[0]}$ debe ser falso en el instante 0 en $\mathcal{I}$, pero $\texttt{pos}(\texttt{p1}, \texttt{cB})_{[2]}$ debe ser cierto en el instante 2 en $\mathcal{G}$). La principal dificultad que aparece aquí es que también los objetivos deben estar asociados con un instante de tiempo que viene marcado por el parámetro *longitud* del plan (véase Algoritmo 1.4). En un esquema de resolución general, se comienza desde una *longitud* mínima y se va incrementando iterativamente hasta encontrar un plan o hasta alcanzar una *longitud* máxima[4]. En el ejemplo de la Tabla 1.7, los objetivos se han asociado con el instante 2, tal y como marca el grafo de planificación de la Figura 1.4. Las acciones se codifican como fórmulas que causan transiciones y deben asociarse con todos los instantes de tiempo en los que cada acción puede ejecutarse[5]; es decir, si existen dos instantes de tiempo posibles existirá una fórmula para cada instante. No obstante, la codificación de las acciones es un poco más elaborada pues requiere tres tipos de fórmulas por acción (que aparecen etiquetadas como F1, F2 y F3 en la Tabla 1.7): F1) las que codifican la propia transición, indicando la acción, precondiciones y efectos ($\texttt{a}_\texttt{i} \Rightarrow \text{Pre}(\texttt{a}_{\texttt{i}-1}) \wedge \text{Efe}(\texttt{a}_\texttt{i})$), F2) las que garantizan que una acción sólo cambia lo que está en sus efectos (*problema marco*) o, dicho de otra forma, que si un efecto cambia lo hace por la ejecución de la acción que lo tiene como efecto $\left(\left(\neg \texttt{p}_\texttt{i} \wedge \texttt{p}_{\texttt{i+1}} \Rightarrow \left(\bigvee_{\texttt{a}|\texttt{p}\in \text{Efe}^+(\texttt{a})} \texttt{a}_{\texttt{i+1}}\right)\right) \wedge \left(\texttt{p}_\texttt{i} \wedge \neg \texttt{p}_{\texttt{i+1}} \Rightarrow \left(\bigvee_{\texttt{a}|\texttt{p}\in \text{Efe}^-(\texttt{a})} \texttt{a}_{\texttt{i+1}}\right)\right)\right)$, y F3) las que impiden

[4] Algunos enfoques utilizan el último nivel del grafo de planificación correspondiente para inicializar el valor de la *longitud* mínima, evitando iteraciones innecesarias.

[5] De nuevo, el grafo de planificación correspondiente proporciona información acerca de todos los posibles instantes de tiempo en los que las acciones pueden ejecutarse.

Estado $\mathcal{I}$	$\texttt{pos(c1,cA)}_{[0]} \wedge \texttt{pos(p1,cA)}_{[0]} \wedge \neg\texttt{pos(c1,cB)}_{[0]} \wedge \neg\texttt{en(p1,c1)}_{[0]} \wedge \neg\texttt{pos(p1,cB)}_{[0]}$
Estado $\mathcal{G}$	$\texttt{pos(p1,cB)}_{[2]} \wedge \neg\texttt{pos(p1,cA)}_{[2]} \wedge \neg\texttt{en(p1,c1)}_{[2]}$
Acciones	F1. $\texttt{mv(c1,cA,cB)}_{[1]} \Rightarrow \texttt{pos(c1,cA)}_{[0]} \wedge \neg\texttt{pos(c1,cB)}_{[0]} \wedge \texttt{pos(c1,cB)}_{[1]} \wedge \neg\texttt{pos(c1,cA)}_{[1]}$
	F2. $\neg\texttt{pos(c1,cB)}_{[0]} \wedge \texttt{pos(c1,cB)}_{[1]} \Rightarrow \texttt{mv(c1,cA,cB)}_{[1]}$
	F2. $\texttt{pos(c1,cA)}_{[0]} \wedge \neg\texttt{pos(c1,cA)}_{[1]} \Rightarrow \texttt{mv(c1,cA,cB)}_{[1]}$
	F3. $\neg\texttt{mv(c1,cA,cB)}_{[1]} \vee \neg\texttt{cg(p1,c1,cA)}_{[1]}$
	F1. $\texttt{cg(p1,c1,cA)}_{[1]} \Rightarrow \texttt{pos(c1,cA)}_{[0]} \wedge \texttt{pos(p1,cA)}_{[0]} \wedge \texttt{en(p1,c1)}_{[1]} \wedge \neg\texttt{pos(p1,cA)}_{[1]}$
	F2. $\texttt{pos(p1,cA)}_{[0]} \wedge \neg\texttt{pos(p1,cA)}_{[1]} \Rightarrow \texttt{cg(p1,c1,cA)}_{[1]}$
	F2. $\neg\texttt{en(p1,c1)}_{[0]} \wedge \texttt{en(p1,c1)}_{[1]} \Rightarrow \texttt{cg(p1,c1,cA)}_{[1]}$
	F3. $\neg\texttt{cg(p1,c1,cA)}_{[1]} \vee \neg\texttt{mv(c1,cA,cB)}_{[1]}$
	F1. $\texttt{dcg(p1,c1,cB)}_{[2]} \Rightarrow \texttt{pos(c1,cB)}_{[1]} \wedge \texttt{en(p1,c1)}_{[1]} \wedge \neg\texttt{en(p1,c1)}_{[2]} \wedge \texttt{pos(p1,cB)}_{[2]}$
	F2. $\texttt{en(p1,c1)}_{[1]} \wedge \neg\texttt{en(p1,c1)}_{[2]} \Rightarrow \texttt{dcg(p1,c1,cB)}_{[2]} \vee \texttt{dcg(p1,c1,cA)}_{[2]}$
	F2. $\neg\texttt{pos(p1,cB)}_{[1]} \wedge \texttt{pos(p1,cB)}_{[2]} \Rightarrow \texttt{dcg(p1,c1,cB)}_{[2]}$
	F3. $(\neg\texttt{dcg(p1,c1,cB)}_{[2]} \vee \neg\texttt{mv(c1,cB,cA)}_{[2]}) \wedge (\neg\texttt{dcg(p1,c1,cB)}_{[2]} \vee \neg\texttt{mv(c1,cA,cB)}_{[2]}) \wedge$ $(\neg\texttt{dcg(p1,c1,cB)}_{[2]} \vee \neg\texttt{cg(p1,c1,cA)}_{[2]}) \wedge (\neg\texttt{dcg(p1,c1,cB)}_{[2]} \vee \neg\texttt{dcg(p1,c1,cA)}_{[2]})$

Tabla 1.4: Ejemplo de codificación de un problema de planificación con las acciones `mv(c1,cA,cB)`, `cg(p1,c1,cA)` y `dcg(p1,c1,cB)` mediante fórmulas lógicas. Por simplicidad, sólo se han representado las fórmulas asociadas a un instante de tiempo.

que dos acciones mutuamente excluyentes se ejecuten simultáneamente ($\neg\texttt{a}_\texttt{i} \vee \neg\texttt{b}_\texttt{i}$).

Una vez finalizada la conversión del problema en fórmulas lógicas se ejecutaría el algoritmo de resolución basado en satisfactibilidad. Con el valor actual de *longitud*=2 no se encontraría una solución, siendo necesario extenderlo hasta el instante 3, donde se encontraría la asignación factible $\{\texttt{cg(p1,c1,cA)}_{[1]}, \texttt{mv(c1,cA,cB)}_{[2]}, \texttt{dcg(p1,c1,cB)}_{[3]}\}$ de la que se puede extraer un plan solución.

1.8 Planificación para el mundo real.

La aplicación de las técnicas de planificación a problemas del mundo real está aumentando a un ritmo considerable. Actualmente existen decenas de escenarios reales en los que la planificación resulta una herramienta muy útil:

- **Control y navegación de robots autónomos**. Este campo es uno de los que motivó la investigación inicial en planificación de trayectorias y exploración del entorno para identificar objetos. En la actualidad, la planificación también se está usando con éxito en entornos portuarios, fábricas de automóviles e incluso en misiones aeroespaciales de la NASA y la ESA (*Agencia Espacial Europea*) donde la mayoría de las tareas a realizar se planifican automáticamente sin intervención humana [Jónsson *et al.*, 2000].

- **Planificación de rutas**. La generación de rutas logísticas para empresas de transporte o incluso la elaboración de los planes de rutas de los servicios de transporte público utilizan técnicas de planificación para realizar la asignación de vehículos y personal de forma automatizada [Ghallab *et al.*, 2004].

- **Entornos simulados**. Las técnicas de planificación inteligentes se han aplicado en el diseño de agentes para programas de entrenamiento e incluso en juegos comerciales de acción, cartas, etc. [Nareyek, 2004].

- **Entornos de red**. Con el auge de las redes la planificación se ha comenzado a utilizar en muchos de los buscadores para realizar consultas y en la construcción de servicios Web distribuidos [Knoblock, 2003]. Otra aplicación importante es la organización de flujos de tareas (*workflows*) en redes computacionales [Gil *et al.*, 2004].

- **Situaciones catastróficas y manejo de crisis**. Desde hace unos años las técnicas de planificación están dando soporte a los planes de actuación ante situaciones desastrosas, como por ejemplo vertidos de petróleo, incendios forestales, evacuaciones urbanas y rescates, etc. [Gervasio *et al.*, 1998, RoboCup Rescue Technical Committee, 2000, Asunción *et al.*, 2005].

- **Aplicaciones militares**. Las técnicas de planificación han tomado parte en el diseño de importantes campañas militares y en la toma de decisiones logísticas y estratégicas de posicionamiento de tropas [Chun, 1999].

Tal y como puede observarse, son muchos los escenarios que se benefician de las técnicas de planificación pero, para ello, es necesario ir más allá de la planificación clásica. Los problemas del mundo real requieren características que exceden del modelo clásico y de sus asunciones (véase Sección 1.1), como gestión de duraciones y recursos, optimización multicriterio, abstracción de acciones o trabajo con incertidumbre, entre otras. El resto de esta sección proporciona una breve introducción a nuevas técnicas de planificación que solventan las principales limitaciones de la planificación clásica.

1.8.1 Planificación numérica: tiempo + recursos.

Planificación temporal.

En los problemas reales de planificación no es aceptable que todas las acciones tengan la misma duración. Obviamente, la duración de una acción `mv(c1,cA,cB)` no tiene porqué ser igual a la de `mv(c2,cA,cC)` pues la velocidad del camión o distancia entre las ciudades puede ser distinta. Por lo tanto, la primera asunción de la planificación clásica que relajaremos es la A5: las acciones ahora tienen distinta duración y es el planificador quien debe gestionar las características temporales de forma explícita (**planificación temporal**). La resolución de un problema de planificación temporal incrementa la complejidad del problema porque no sólo estriba en la correcta elección de la secuencia de acciones que alcanza el objetivo, sino también en la correcta elección del instante de ejecución apropiado para las mismas. Adicionalmente, esta doble elección debe realizarse con el objetivo de generar planes de mínima duración. Es decir, en un contexto de planificación temporal el criterio de optimización de los planes se centra en minimizar la duración del plan, en lugar del número de pasos o acciones del mismo.

En la planificación temporal las acciones dejan de ser instantáneas y su duración se puede extender a lo largo del tiempo. Esto hace que el modelo simple (conocido como modelo *conservativo*) de acción en el que las precondiciones deben mantenerse durante toda la ejecución de la acción y los efectos se generan sólo al final de la misma resulte un poco limitado. Para amoldarse a la duración de la acción (Dur) y hacer frente a esta limitación, el modelo de acción se extiende teniendo en cuenta condiciones locales: *pre*condiciones (Pre), *post*condiciones (Post) y condiciones *invariantes* (Inv) que deben satisfacerse antes, después y durante la ejecución de la acción, respectivamente. Adicionalmente, los efectos de las acción también se dividen en efectos *iniciales* (EfeI) y *finales* (EfeF) que se generan al principio y final de la acción, respectivamente[6]. Esto permite disponer de

[6]Éste es el modelo de acción propuesto en las últimas versiones de PDDL. Además de este modelo existen otros

`mv(?cam,?ori,?des)` Dur: `dur-mv(?cam,?ori,?des)` Pre: `pos(?cam,?ori)` Post: `almacen(?des)` Inv: `combustible(?cam)` EfeI: ¬`pos(?cam,?ori)` EfeF: `pos(?cam,?des)` ¬`combustible(?cam)`	`rep(?cam,?ciu)` Dur: `dur-rep(?cam,?ciu)` Pre: `operario(?ciu)` Post: `operario(?ciu)` Inv: `pos(?cam,?ciu)` EfeI: ∅ EfeF: `combustible(?cam)`
`cg(?paq,?cam,?ciu)` Dur: `dur-cg(?paq,?cam,?ciu)` Pre: `pos(?paq,?ciu)` Post: ∅ Inv: `pos(?cam,?ciu)` EfeI: ¬`pos(?paq,?ciu)` EfeF: `en(?paq,?cam)`	`dcg(?paq,?cam,?ciu)` Dur: `dur-dcg(?paq,?cam,?ciu)` Pre: `en(?paq,?cam)` Post: ∅ Inv: `pos(?cam,?ciu)` EfeI: ¬`en(?paq,?cam)` EfeF: `pos(?paq,?ciu)`

Tabla 1.5: Ejemplo de operadores utilizando un modelo de acciones extendido para manejar duraciones y condiciones/efectos locales.

acciones más expresivas como las que se muestran en la Tabla 1.8.1, y modelar con mayor precisión los cambios que las mismas provocan en el mundo, dando lugar a planes con más oportunidades de concurrencia entre acciones. Por ejemplo, en la Figura 1.8.1 el operador `mv` tiene como *post*condición que exista un almacén en la ciudad de destino y como condición invariante que el camión tenga combustible. Para obtener este combustible se añade un nuevo operador de repostar (`rep`) que requiere la presencia de un operario sólo al inicio (*pre*condición) y final (*post*condición) del mismo y, lógicamente, que el camión permanezca en esa posición de forma invariante. Análogamente, los operadores `cg` y `dcg` también modelan condiciones y efectos locales, acercándose más al problema real.

Los planificadores temporales gestionan la información temporal mediante una representación explícita del tiempo y algoritmos de búsqueda para asignar las acciones en el tiempo. Inicialmente, la planificación temporal se abordó bajo una aproximación de orden parcial, basándose principalmente en las técnicas de menor compromiso; la idea es retrasar la instanciación temporal de las acciones tanto como sea posible para reducir la complejidad del problema. En otras aproximaciones, el manejo de tiempo se consigue gracias a la inclusión de una red temporal que registra las restricciones temporales entre acciones y permite la definición de restricciones sobre la duración de las acciones y del plan. Gracias a esta representación temporal, el planificador puede manejar relaciones cualitativas (restricciones de orden y enlaces causales) y cuantitativas (instantes de comienzo/fin y duraciones). La aproximación basada en grafos también se ha aplicado a la planificación temporal, extendiendo el razonamiento sobre mutex, que es ahora más elaborado, y generalizando el grafo de planificación para contemplar el modelo más expresivo de las acciones con duración. En este caso, los niveles del grafo no representan pasos de planificación, sino verdaderos instantes de tiempo, lo que incrementa el número de niveles del grafo y hace la búsqueda más costosa (el espacio de búsqueda es ahora mayor). Por último, la aproximación que posiblemente ha permitido un mayor avance en el campo de la planificación temporal ha sido la de planificación heurística, pero ésta ha abordado conjuntamente el problema de la planificación con recursos.

menos comunes, pero más expresivos, que permiten que tanto las condiciones como los efectos se definan como ventanas temporales a lo largo de la ejecución de la acción.

Planificación con recursos.

Habitualmente, las acciones de un plan requieren un conjunto de recursos para llevar a cabo sus tareas. Se puede decir que un recurso es cualquier elemento (generalmente limitado y que hay que compartir a lo largo del tiempo) necesario para poder ejecutar las acciones. Un recurso puede representar a una entidad material (camión, máquina, combustible, etc.) o personal (conductor, operario, tripulación, etc.) Aunque tradicionalmente la gestión de recursos se ha relegado al proceso de **scheduling**, en planificación el uso no simultáneo de recursos se resuelve mediante métodos *artificiales* que consisten, básicamente, en introducir nuevas proposiciones para representar la disponibilidad y uso exclusivo de recursos, como por ejemplo `camion-libre`, `conductor-disponible`, etc. Sin embargo, existen recursos que requieren una representación basada en valores numéricos continuos (por ejemplo, combustible, energía, tiempo, etc.) que no se pueden representar bajo el modelo proposicional impuesto por la planificación clásica. Por lo tanto, para abordar problemas reales con este tipo de recursos es necesario relajar la asunción A4 (enfoque proposicional): las acciones ahora pueden trabajar con variables numéricas cuyo dominio es continuo. Esto permite modelar restricciones más complejas sobre las variables y acciones: 1) precondiciones como `(nivel-combustible(c1)`$\geq$`100.5)` o `(espacio-ocupado(c1)<50)`, y 2) efectos como `(decrementar nivel-combustible(c1) 12.25)` o `(incrementar beneficio 1.5)`. A este tipo de planificación se le conoce como **planificación numérica** y conlleva dos claras ventajas:

- Esta planificación subsume a la planificación temporal, pues basta considerar al tiempo como una variable numérica más. Así, al igual que una acción puede incrementar el total de combustible consumido, también puede incrementar el tiempo (duración) total del plan en función de su propia duración.

- Esta planificación permite llevar a cabo una **optimización multicriterio**. Ahora es posible expresar una función objetivo del estilo `MINIMIZAR ((2 * duracion-plan) + (0.005 * combustible-consumido(c1)))` en el que se tienen en cuenta varios factores ponderados para evaluar y minimizar el coste del plan, obteniendo planes de mejor calidad.

La planificación numérica (incluyendo tiempo y recursos) resulta bastante compleja por lo que es necesario abordarla bajo una aproximación heurística. En muchos casos las estimaciones heurísticas se calculan a partir de grafos de planificación y planes relajados. Estas estimaciones suelen representar cotas optimistas y pesimistas que se combinan con técnicas de **búsqueda local** y **descomposición de problemas** para realizar una exploración inteligente del espacio de búsqueda del problema.

A pesar del intento de incluir características propias de scheduling, como la gestión del tiempo y recursos, dentro del proceso de planificación todavía existen algunas limitaciones. La tipología de restricciones temporales y recursos de un problema real es muy variada e incluir por completo su razonamiento en la planificación resulta excesivamente complejo. Por ejemplo, supongamos una planta de robots industriales. En función del robot a utilizar (recurso) y de las acciones a realizar (ensamblar, pintar, soldar, etc.) se deberán satisfacer unas restricciones u otras, con distintos tiempos de preparación del recurso y de utilización (la preparación del robot requiere distintos tiempos y acciones para ensamblar piezas que para pintarlas, o los tiempos de espera entre dos acciones de soldar serán distintos que entre dos acciones de pintar). Por tanto, uno de los campos de trabajo más activos dentro de la planificación es su **integración** con scheduling. La idea consiste en dirigir los esfuerzos de planificación en la elaboración de los planes, mientras que los esfuerzos de scheduling se enfocan en la validación y satisfacción de las restricciones complejas del problema. El objetivo

Objetivos del problema	
Tarea	`transportar-paquete(?paq,?ori,?des)`
Descomposición jerárquica	
Método	`transportar(?paq,?cam,?ori,?des)`
Pre	`pos(?paq,?ori),pos(?cam,?ori)`
Subtareas	$\langle$`cg(?paq,?cam,?ori)`, `mv(?cam,?ori,?des)`, `dcg(?paq,?cam,?des)`$\rangle$

Tabla 1.6: Ejemplo de descomposición del método `transportar(?paq,?cam,?ori,?des)` que se aplica para la tarea de transportar un paquete.

es aunar esfuerzos para resolver problemas reales complejos de una forma integrada, compartiendo información y criterios heurísticos de optimización.

1.8.2 Planificación jerárquica.

En muchos problemas reales, la *granularidad* que se espera de los planes no tiene que ser muy alta, al menos en un primer momento. Es decir, a menudo resulta más interesante generar un plan inicial compuesto por tareas de alto nivel para luego ir desglosándolo en acciones más simples. Esto es especialmente útil en problemas muy complejos, donde planificar desde cero todas las acciones de bajo nivel puede resultar una tarea muy costosa. Por ejemplo, en un escenario de transporte con decenas de paquetes es preferible contar, en un primer nivel, con una planificación más abstracta sobre las rutas para transportar los paquetes. Posteriormente, en un segundo nivel se podrá descomponer cada una de estas rutas en las acciones correspondientes de carga, movimiento, repostaje y descarga sobre camiones determinados. De ser necesario, este proceso de descomposición puede continuar hasta llegar a las acciones de más bajo nivel. La idea subyacente es la de establecer una jerarquía de acciones que permita descomponer un plan desde las acciones más genéricas hasta las más primitivas. Para ser capaces de manejar esta jerarquía es necesario relajar la asunción A6 de la planificación clásica que obliga a trabajar con acciones primitivas, es decir que no se pueden descomponer. Al modelo de planificación resultante se le conoce como **planificación jerárquica** o HTN (*Hierarchical Task Network*).

En la planificación jerárquica existen dos tipos de acciones: unas de más alto nivel denominadas **métodos** y unas de bajo nivel denominadas **acciones primitivas**. Al contrario que las acciones primitivas, los métodos se descomponen en subtareas que pueden ser otros métodos o acciones primitivas. Esta descomposición se define mediante una red de tareas, que establece la ordenación total o parcial de las subtareas de los métodos. Por ejemplo, en el problema de transporte podemos definir una tarea para transportar un paquete de una ciudad a otra (véase Tabla 1.8.2). Esta tarea requerirá la ejecución de un método que se descompondrá, siguiendo una ordenación total, en las subtareas $\langle cg, mv, dcg\rangle$ que, en este caso se corresponden con acciones primitivas.

El funcionamiento de un planificador HTN es muy similar al de un planificador clásico. El objetivo del problema consiste en aplicar una serie de tareas mediante la descomposición y planificación de una secuencia de métodos que, finalmente, se desglosará en las acciones primitivas. Por lo tanto, se trata de un procedimiento recursivo que va descomponiendo tareas no primitivas hasta llegar a un plan formado sólo por acciones primitivas, que son las que se pueden ejecutar directamente en el entorno. Este tipo de planificación presenta dos ventajas: 1) el dominio del problema se describe en términos de acciones estructuradas jerárquicamente, resultando más intuitivo para el experto que modela el problema, y 2) la función del planificador consiste en refinar estas estructuras, generando las expansiones necesarias y simplificando la resolución del problema original.

1.8.3 Planificación con incertidumbre.

La planificación clásica asume que la información sobre el mundo es estática y totalmente observable (el planificador tiene un conocimiento total sobre el estado del sistema –asunciones A1 y A2) y que los efectos de las acciones son deterministas (la aplicación de una acción genera siempre los mismos efectos conocidos –asunción A3). Sin embargo, en los problemas reales esto no siempre es cierto, existiendo cierta incertidumbre. Esta incertidumbre puede aparecer en el entorno (mundo) del problema, en la percepción que el planificador tiene de ese entorno y/o en las propias acciones. En primer lugar, el modelo de entorno puede ser dinámico, existiendo agentes externos que lo modifican sin conocimiento del planificador. De hecho, esto también incluye a los objetivos del problema que pueden cambiar o incluso volverse imposibles de satisfacer. En segundo lugar, la percepción que el planificador posee del entorno puede ser incompleta, desconociendo cierta información. Por último, la ejecución de las acciones no siempre es determinista y predecible: en ocasiones la duración de las acciones fluctúa y no siempre tienen el éxito esperado, o incluso sólo existe una cierta probabilidad de éxito. Supongamos, por ejemplo, un escenario de planificación para la lucha contra incendios forestales. Es evidente que este problema tiene un elevado componente dinámico, pues el fuego (agente externo) altera el entorno (aparición y desaparición de nuevos focos que el planificador desconoce) y modifica constantemente los objetivos del problema. Además, las duraciones y efectos de las acciones de extinción son difícilmente predecibles y su probabilidad de éxito depende de la virulencia del fuego. Para gestionar esta incertidumbre se han planteado dos aproximaciones distintas:

- Construir planes suficientemente genéricos que contemplen dicha incertidumbre mediante la inclusión de acciones específicas de *sensorización* para adquirir la información incompleta, o mediante *monitorización* para observar el entorno y actuar en consecuencia. Más concretamente existen las siguientes técnicas: 1) **planificación conforme** para entornos parcialmente observables y sin posibilidad de monitorización, que generan planes para todos los posibles mundos a partir de distintos estados iniciales, 2) **planificación contingente** para generar planes condicionales o probabilísticos preparados para la aparición de *ciertos* imprevistos conocidos, y 3) **planificación reactiva** que, tras monitorizar el estado, recomienda las próximas acciones a ejecutar.

- Modificar y adaptar el plan a medida que se ejecuta. Esta técnica de **planificación y ejecución** alterna las etapas de planificación, monitorización durante la ejecución y replanificación en caso de que el entorno no se encuentre en el estado esperado. A este tipo de planificación también se le conoce como **planificación *on-line***, y relaja la asunción A7 del modelo clásico.

1.9 Notas bibliográficas y lecturas recomendadas.

Desde sus orígenes, la planificación ha consistido en un proceso formal de resolución de problemas mediante búsqueda y técnicas de demostración de problemas, entre los que cabe destacar a GPS [Newell and Simon, 1963] y QA3 [Green, 1969], respectivamente. STRIPS [Fikes and Nilsson, 1971] supuso un importante punto de partida, tanto por ser el primer sistema práctico aplicado a la planificación de trayectorias de un robot, como por fijar las bases de un modelo de acciones que, con algunas extensiones como las que propone ADL [Pednault, 1989], todavía se sigue utilizando. Aunque la idea de STRIPS era la de mantener el espacio de búsqueda próximo al espacio de situaciones, pronto dio paso a nuevos planificadores más complejos, expresivos y con un mejor control sobre la búsqueda, como NOAH [Sacerdoti, 1977] y Nonlin [Tate, 1977]. En estos dos planificadores el orden

de acciones no tiene que ser estrictamente total y lineal, sino que a menudo se mantienen en un orden parcial [Barret and Weld, 1994].

En paralelo a los avances teóricos, la planificación práctica también avanzaba con paso firme, apareciendo sistemas que ayudaban en la construcción de edificios o en las operaciones de vehículos espaciales [Yang, 1997]. Por ejemplo, Deviser [Vere, 1983], SIPE [Wilkins, 1988], ZENO [Penberthy, 1993, Penberthy and Weld, 1994] y parcPLAN [El-Kholy and Richards, 1996] incluyeron criterios de temporalidad en las acciones y planes. Sin embargo, estos sistemas resultan mucho más complejos y no tuvieron el mismo impacto que los basados en planificación clásica como SNLP [McAllester and Rosenblitt, 1991], que integra planificación de orden parcial [Barret and Weld, 1994] y una dependencia entre acciones mediante enlaces causales [McAllester and Rosenblitt, 1991, Weld, 1994]. Las propiedades formales de los POP fueron demostradas por UCPOP [Barret *et al.*, 1996, Penberthy and Weld, 1992], uno de los planificadores de orden parcial más conocidos que, junto con Prodigy [Fink and Veloso, 1994], aplicaba además técnicas de menor compromiso [Weld, 1994]. A medida que la complejidad de los problemas crecía, se hacía más necesario el desarrollo de técnicas de planificación abstracta y jerárquica [Erol *et al.*, 1994], como en ABSTRIPS [Sacerdoti, 1974], ABTWEAK [Yang and Tenenberg, 1990], ALPINE [Knoblock, 1994] y SIADEX [Asunción *et al.*, 2005, Castillo *et al.*, 2005]. Sin embargo, la complejidad no se origina sólo por el tamaño de los problemas, sino también por la expresividad requerida. En la línea de unificar elementos de planificación y scheduling bajo un mismo enfoque cabe destacar a O-Plan [Currie and Tate, 1991] (utilizado en aplicaciones militares), HSTS [Muscettola, 1994] (utilizado en la planificación de las observaciones del telescopio espacial *Hubble*) e IxTeT [Ghallab and Laruelle, 1994, Vidal and Ghallab, 1996].

La planificación basada en grafos de planificación y en satisfactibilidad, con Graphplan [Blum and Furst, 1997] y SATPLAN [Kautz and Selman, 1992, Kautz and Selman, 1996] a la cabeza respectivamente, ha revivido el interés por la planificación, en lo que algunos autores han pasado a denominar *planificación neoclásica* [Ghallab *et al.*, 2004]. Ambas aproximaciones han supuesto un importante punto de partida para el desarrollo de planificadores más modernos, potentes y expresivos como IPP [Köehler *et al.*, 1997, Nebel *et al.*, 1997], BlackBox [Kautz and Selman, 1998], SGP [Anderson *et al.*, 1998, Weld and Smith, 1998], STAN [Fox and Long, 1999], TGP [Smith and Weld, 1999] o TPSYS [Garrido and Onaindía, 200 entre otros. Algunos de estos planificadores participaron en la primera competición internacional de planificación celebrada en 1998 [IPC, 2006]. Esta competición surgió como un punto de encuentro para comparar y mostrar los avances de las distintas técnicas y sistemas de planificación. Para facilitar esta comparación y proporcionar un lenguaje común de descripción de problemas se definió PDDL, que se ha ido extendiendo con nuevas funcionalidades [McDermott, 1998, Fox and Long, 2003, Edelkamp and Hoffmann, 2004, Gerevini and Long, 2006]. PDDL ha sido testigo de lo que probablemente han sido los años de mayores avances en el campo de la planificación, gracias a la planificación heurística revitalizada por HSP [Bonet and Geffner, 1999]. HSP planteó heurísticas como las h_{sum} y $h_{\text{máx}}$ vistas en la Sección 1.6.3 y dio pie a otros planificadores como GRT [Refanidis and Vlahavas, 1999], AltAlt [Kambhampati and Sanchez Nigenda, 2000, Nguyen *et al.*, 2002], TP4 [Haslum and Geffner, 2001, Haslum, 2004] o Sapa [Do and Kambhampati, 2001, Do and Kambhampati, 2003], que proporciona características de optimización multicriterio. FF [Hoffmann, 2000, Hoffmann and Nebel, 2001], junto con su versión numérica metric-FF [Hoffmann, 2003], ha sido uno de los planificadores con más influencia en los últimos años por su gran eficiencia. FF basa sus heurísticas en planes relajados y sus ideas se utilizan en otros planificadores como Mips [Edelkamp and Helmert, 2001] o LPG [Gerevini and Serina, 2002, Gerevini *et al.*, 2003, Gerevini *et al.*, 2004], basado en búsqueda local no determinista. La tendencia en los últimos años ya no es construir un planificador desde cero, sino reutilizar funcionalidades de los ya existentes como FF o LPG. En esta línea se han desarrollado Macro-FF [Botea *et al.*, 2004], Fast (Diagonally) Downward [Helmert and Richter, 2004],

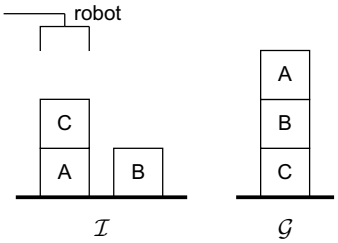

Figura 1.5: Estado inicial y objetivo del problema de la anomalía de *Sussman*.

YAHSP [Vidal, 2004], SGPlan [Chen *et al.*, 2004, Chen *et al.*, 2005, Hsu *et al.*, 2006], MIPS-XXL [Edelkamp *et al.*, 2
SATPLAN04 [Kautz and Selman, 2006], MaxPlan [Xing *et al.*, 2006], YochanPS [Benton and Kambhampati, 2006] y muchos más. De todos ellos, el más rápido en la última competición de planificación de 2006 es SGPlan, que utiliza técnicas de descomposición de problemas para reducir la complejidad del problema original.

En las últimas décadas, el campo de la planificación ha crecido tanto y tan deprisa que es difícil encontrar libros de consulta que cubran en amplitud las principales técnicas y algoritmos. Aunque los libros de Russell y Norvig [Russell and Norvig, 2004] y Nilsson [Nilsson, 2000] dedican capítulos específicos a la planificación, es el libro de Yang [Yang, 1997] el que está dedicado íntegramente a la planificación, introduciendo los algoritmos básicos, técnicas de descomposición y abstracción jerárquica. Actualmente, el libro más actualizado y completo sobre planificación es el de Ghallab *et al.* [Ghallab *et al.*, 2004] que hace un recorrido detallado y muy didáctico de todos los aspectos de planificación. La realización de este capítulo se ha visto influenciada por estos cuatro textos, y más particularmente por el último.

La planificación es un campo muy activo dentro de la IA y los artículos más novedosos se publican en las principales revistas y conferencias sobre IA. Además, los investigadores e interesados en planificación cuentan con una conferencia anual específica sobre planificación y scheduling denominada ICAPS (*International Conference on Automated Planning and Scheduling*) que abarca todos los terrenos de la planificación.

1.10 Ejercicios propuestos.

1.1. El problema del *mundo de bloques* ha sido uno de los problemas más utilizados en el campo de la planificación. Dicho problema consiste en construir torres, apilando y desapilando bloques con la ayuda de un robot. En la Tabla 1 se muestra el dominio, utilizando 4 operadores, para este problema así como el estado inicial y objetivo para un problema conocido como la *anomalía de Sussman*. Para resolver este problema, es necesario deshacer la torre inicial para construir una nueva que se corresponda con el objetivo (véase Figura 1.5). A continuación abordaremos la resolución de este problema mediante las técnicas de búsqueda hacia delante, hacia atrás, basada en POP y en Graphplan.

1. **Resolución mediante una búsqueda hacia delante**.
   El nodo raíz del árbol de la Figura 1.6 representa el estado inicial $\mathcal{I}$. Al expandir dicho nodo se encuentran dos acciones aplicables (`recoger(B)` y `desapilar(C,A)`), ya que son las dos únicas acciones cuyas precondiciones se satisfacen en $\mathcal{I}$. Los estados

Estado inicial ($\mathcal{I}$)	`libre(C)` $\wedge$ `sobre(C,A)` $\wedge$ `en-mesa(A)` $\wedge$ `libre(B)` $\wedge$ `en-mesa(B)` $\wedge$ `rob-libre`
Objetivo ($\mathcal{G}$)	`sobre(A,B)` $\wedge$ `sobre(B,C)`

Operadores ($\mathcal{O}$)	`recoger(?b)`
	Pre: `libre(?b)` $\wedge$ `en-mesa(?b)` $\wedge$ `rob-libre`
	Efe: `sujeto(?b)` $\wedge\neg$`libre(?b)` $\wedge\neg$`en-mesa(?b)` $\wedge\neg$`rob-libre`
	`soltar(?b)`
	Pre: `sujeto(?b)`
	Efe: `libre(?b)` $\wedge$ `en-mesa(?b)` $\wedge$ `rob-libre` $\wedge\neg$`sujeto(?b)`
	`apilar(?b1,?b2)`
	Pre: `sujeto(?b1)` $\wedge$ `libre(?b2)`
	Efe: `sobre(?b1,?b2)` $\wedge$ `libre(?b1)` $\wedge$ `rob-libre` $\wedge$
	$\neg$`sujeto(?b1)` $\wedge\neg$`libre(?b2)`
	`desapilar(?b1,?b2)`
	Pre: `sobre(?b1,?b2)` $\wedge$ `libre(?b1)` $\wedge$ `rob-libre`
	Efe: `sujeto(?b1)` $\wedge$ `libre(?b2)` $\wedge$
	$\neg$`sobre(?b1,?b2)` $\wedge\neg$`rob-libre` $\wedge\neg$`libre(?b1)`

Tabla 1.7: Definición de un problema sencillo con 4 operadores para construir una torre de 3 bloques.

resultantes son el producto de borrar los efectos negativos y añadir los efectos positivos de las acciones correspondientes. A partir del nodo 1 se escoge un nodo a expandir, por ejemplo el nodo 2, y se procede añadiendo los efectos correspondientes. Los estados repetidos aparecen marcados con un aspa, indicando así que la expansión del árbol no debe continuar a partir de dicho nodo. Con líneas punteadas se indican los nodos que deben generarse en el siguiente nivel (ignorando los estados repetidos). Nótese también que si la acción aplicable desde un nodo es la inversa de la acción que ha generado dicho nodo (por ejemplo, a partir de los nodos 2, 3 y 4), la expansión del árbol no continúa por dicho nodo. Como ejercicio, se propone al lector completar el árbol de búsqueda hacia delante de la Figura 1.6.

2. **Resolución mediante una búsqueda hacia atrás**.

   El primer estado objetivo del problema contiene los dos literales de $\mathcal{G}$ y es el primer nodo que se expande. Existen dos acciones relevantes para conseguir cada uno de los literales de $\mathcal{G}$: `apilar(B,C)` y `apilar(A,B)`. En la Figura 1.7 se muestra un fragmento de la expansión del árbol que se origina a partir del sucesor generado con la acción `apilar(A,B)` (nodo 2). El nodo 2 será entonces el siguiente nodo que se expande, existiendo múltiples acciones para conseguir sus objetivos: 1) `apilar(B,C)` para conseguir `sobre(B,C)`, 2) `soltar(B)` y `apilar(B,?y)` para conseguir `libre(B)`, 3) `desapilar(A,?x)`, y 4) `recoger(A)` para conseguir `sujeto(A)`. Si se expande el nodo etiquetado con el número 3 se puede observar que se alcanzan estados inconsistentes; por ejemplo, la aplicación de la acción `recoger(B)` para satisfacer el literal `sujeto(B)` borraría el literal `libre(B)`, que también forma parte del estado objetivo, y por tanto produciría un estado inconsistente. Si a continuación se expande el nodo etiquetado como nodo 4, se producirá igualmente un estado inconsistente. La expansión a partir del nodo 5 produciría tres estados objetivos (sin contabilizar los estados repetidos ni inconsistentes). La solución a este problema se encuentra a través de uno de los nodos generados a partir del nodo 5. Como ejercicio, se propone al lector completar el árbol de búsqueda hacia atrás de la Figura 1.7 y se recomienda que continúe la expansión a

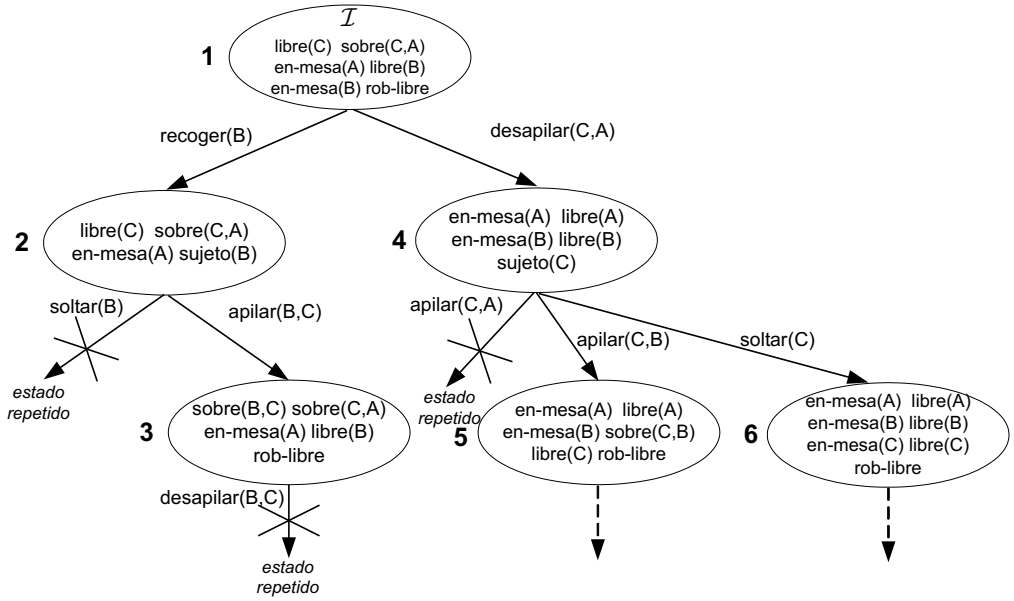

Figura 1.6: Árbol parcial generado para el problema de *Sussman* en una búsqueda hacia delante en un espacio de estados.

partir de los nodos etiquetados como 6, 7 y 8.

3. **Resolución mediante planificación de orden parcial**.

   Inicialmente (véase Figura 1.8-a)), de los dos objetivos a resolver se selecciona uno de ellos, por ejemplo `sobre(A,B)`. Ambos objetivos tienen una única forma de resolución, mediante la introducción de un operador `apilar`, de modo que la aplicación de las heurísticas vistas en la Sección 1.5.3 no permitirían discriminar entre ambos objetivos. En la Figura 1.8-b) del árbol existen tres objetivos pendientes de resolución: `sobre(B,C)` y las dos precondiciones del operador `apilar(A,B)`. Existen dos formas de resolver el objetivo `sujeto(A)`: mediante la introducción del operador `recoger(A)` o `desapilar(A,?x)`. Por otro lado, existen tres formas de resolver `libre(B)`: mediante los operadores `soltar(B)` o `apilar(B,?x)`, o mediante el estado inicial `I`. Sin embargo, sólo existe un modo de resolver `sobre(B,C)` y, por consiguiente, escogemos dicho objetivo como el siguiente a resolver. En el plan mostrado en la Figura 1.8-c) existen cuatro precondiciones pendientes de resolver. Para las dos precondiciones del tipo `sujeto` existen dos formas de resolución, tres para el objetivo `libre(C)` y cuatro para `libre(B)`. Se deja al lector la resolución del resto del ejercicio, partiendo de la Figura 1.8-c) y averiguando cuántas formas diferentes existen de resolver los objetivos pendientes de un nodo y seleccionando aquel objetivo que se considere más apropiado.

4. **Resolución mediante Graphplan**.

   El primer paso de Graphplan consiste en la construcción del grafo de planificación. La Tabla 4 muestra los niveles de acción del grafo de planificación resultante. Aunque el nivel de acción $A_{[6]}$ coincide plenamente con $A_{[5]}$, los mutex entre `sobre(A,B)` y `sobre(B,C)` no desaparecen hasta $P_{[6]}$, nivel desde el que comienza la etapa de extracción del plan. A partir de este nivel se realiza la búsqueda regresiva tal y como se expli-

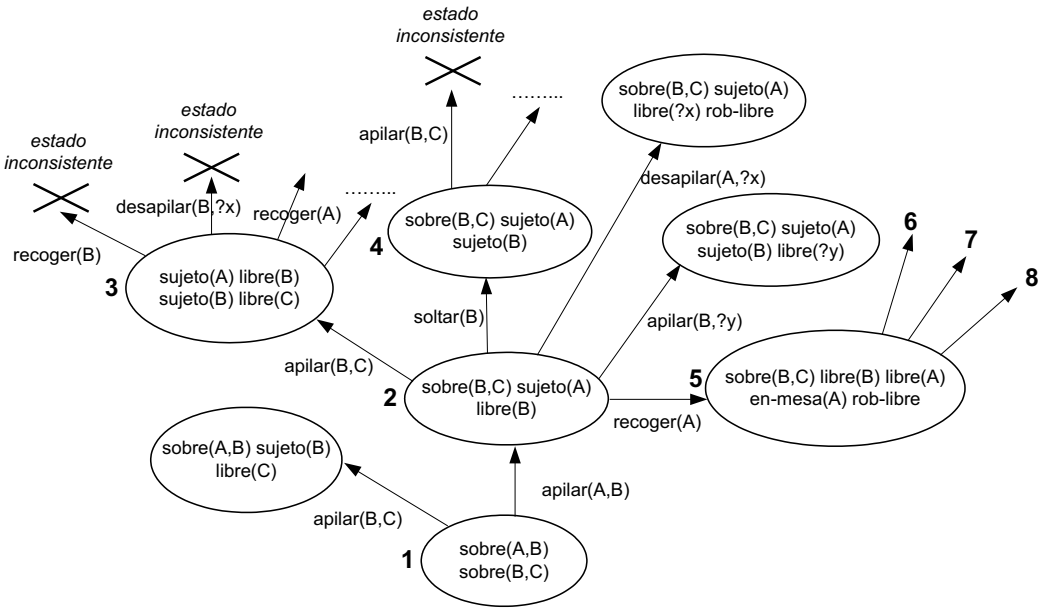

Figura 1.7: Árbol parcial generado para el problema de *Sussman* en una búsqueda hacia atrás.

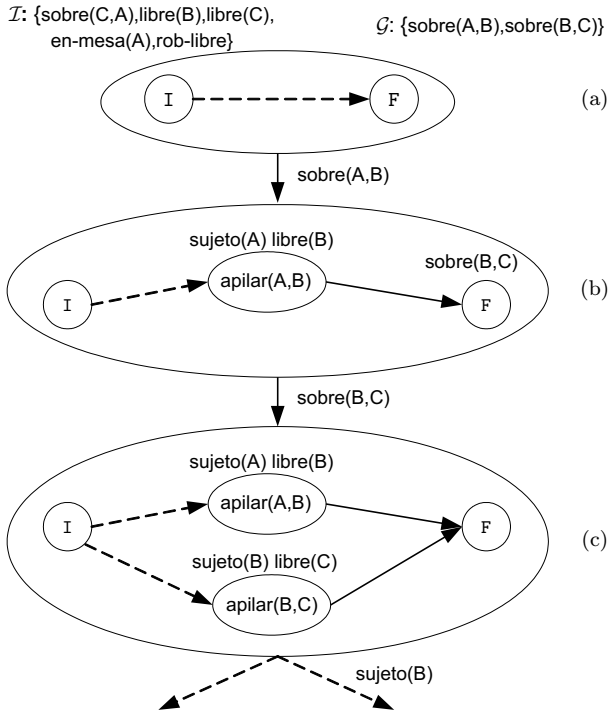

Figura 1.8: Árbol parcial generado para el problema de *Sussman* en una planificación de orden parcial.

$A_{[1]}$	$A_{[2]}$	$A_{[3]}$	$A_{[4]}$	$A_{[5]}$
`recoger(B)`	`recoger(B)`	`recoger(B)`	`recoger(B)`	`recoger(B)`
`desapilar(C,A)`	`desapilar(C,A)`	`desapilar(C,A)`	`desapilar(C,A)`	`desapilar(C,A)`
	`apilar(C,A)`	`apilar(C,A)`	`apilar(C,A)`	`apilar(C,A)`
	`apilar(C,B)`	`apilar(C,B)`	`apilar(C,B)`	`apilar(C,B)`
	`soltar(B)`	`soltar(B)`	`soltar(B)`	`soltar(B)`
	`soltar(C)`	`soltar(C)`	`soltar(C)`	`soltar(C)`
	`apilar(B,C)`	`apilar(B,C)`	`apilar(B,C)`	`apilar(B,C)`
		`recoger(A)`	`recoger(A)`	`recoger(A)`
		`recoger(C)`	`recoger(C)`	`recoger(C)`
		`desapilar(C,B)`	`desapilar(C,B)`	`desapilar(C,B)`
		`desapilar(B,C)`	`desapilar(B,C)`	`desapilar(B,C)`
			`soltar(A)`	`soltar(A)`
			`apilar(B,A)`	`apilar(B,A)`
			`apilar(A,B)`	`apilar(A,B)`
			`apilar(A,C)`	`apilar(A,C)`
				`desapilar(B,A)`
				`desapilar(A,B)`
				`desapilar(A,C)`

Tabla 1.8: Niveles de acción del grafo de planificación de Graphplan para el problema de *Sussman*. Por simplicidad, las acciones `no-op` no se muestran.

ca en el Apartado 1.6.2, encontrándose el plan $\langle\{$`desapilar(C,A)`$\}_{[1]}, \{$`soltar(C)`$\}_{[2]}, \{$`recoger(B)`$\}_{[3]}, \{$`apilar(B,C)`$\}_{[4]}, \{$`recoger(A)`$\}_{[5]}, \{$`apilar(A,B)`$\}_{[6]}\rangle$. Se deja como ejercicio al lector completar el grafo de planificación representado en la Tabla 4 con los niveles de proposición y la etapa de extracción del plan.

1.2. Desarrolla el árbol de una búsqueda hacia delante para el problema de la anomalía de *Sussman* del ejercicio 1 siguiendo una estrategia de anchura y profundidad y calcula el número de nodos que es necesario expandir en cada una de las dos estrategias de búsqueda.

1.3. Si suponemos que en el problema de la anomalía de *Sussman* del ejercicio 1 existen dos robots en lugar de uno, contestar a las siguientes preguntas:

1. ¿Qué modificaciones son necesarias en la definición del problema de planificación?
2. ¿Qué estrategia de planificación habría que utilizar si se quiere encontrar la solución más corta?
3. ¿Qué diferencias habría en la resolución de este problema mediante una aproximación POP con respecto al problema original?
4. ¿Cuál sería la solución que encontraría un proceso de búsqueda hacia delante? ¿Y la encontrada mediante Graphplan?

1.4. Sea el siguiente problema de planificación donde el objetivo es intercambiar el valor de dos variables `v1` y `v2`.

Si se ejecuta una búsqueda hacia delante para este problema, ¿cuántas iteraciones serán necesarias para encontrar la solución más corta? ¿y para la solución más larga?

1.5. Resolver el ejercicio anterior mediante una aproximación basada en POP y calcular:

Estado inicial ($\mathcal{I}$)	`valor(v1,3)` $\wedge$ `valor(v2,5)` $\wedge$ `valor(v3,0)`
Objetivo ($\mathcal{G}$)	`valor(v1,5)` $\wedge$ `valor(v2,3)`
Operadores ($\mathcal{O}$)	`asignar(?v,?w,?x,?y)`
	Pre: `valor(?v,?x)` $\wedge$ `valor(?w,?y)`
	Efe: `valor(?v,?y)` $\wedge\neg$`valor(?v,?x)`

Tabla 1.9: Definición de un sencillo problema de planificación para el intercambio de valor de dos variables.

1. ¿Cuántas amenazas se generan?
2. ¿Cuántos nodos se generan en el árbol POP que se desarrolla para este problema?
3. ¿Cuántos planes solución se pueden encontrar para este problema?
4. Realiza una traza en el árbol de búsqueda para encontrar la solución con el menor número de acciones.

1.6. Partiendo del problema de la anomalía de *Sussman* definido en el ejercicio 1, realizar los siguientes apartados:

1. Construir el grafo de planificación relajado (es decir, sin tener en cuenta los efectos negativos de las acciones y sin calcular las relaciones de exclusión mutua). ¿Cuál es el primer nivel en el que se inicia la etapa de extracción de un plan?
2. Comparar el tamaño (número de niveles y proposiciones/acciones en cada uno de ellos) con el del grafo de planificación generado por Graphplan en el ejercicio 1.
3. Extraer un plan relajado sobre dicho grafo de planificación. Al no contemplarse interacciones negativas, demostrar cómo este plan se puede extraer en tiempo polinómico y sin necesidad de operaciones de backtracking.
4. Construir el CSP resultante para los dos primeros niveles del grafo de planificación relajado.
5. Calcular el valor de las heurísticas h_{sum}, $h_{\max}$ y $h_{\max 2}$ para $\mathcal{G}$=`{sobre(A,B),sobre(B,C)}`. Comparar los valores de dichas heurísticas utilizando como base de estimaciones un grafo de planificación con y sin cálculo de mutex. Comprobar también cómo varía el valor de $h_{\max 2}$ si en $\mathcal{G}$ existen más de dos objetivos.
6. Codificar mediante fórmulas lógicas las acciones de los dos primeros niveles del grafo de planificación relajado.

1.7. Dado el grafo de planificación de la Figura 1.4 definido sobre el problema de transporte de ejemplo, continuar el proceso de extensión del mismo hasta alcanzar un nivel que sea idéntico al anterior ($A_{[t]} = A_{[t-1]}$ y $P_{[t]} = P_{[t-1]}$). ¿Cuál es este nivel y cuántas acciones/proposiciones aparecen en él? ¿Cuántas de estas acciones son del tipo `no-op`?

1.8. Descargar el planificador Graphplan[7] y probarlo en distintos dominios y problemas, incluyendo el problema de transporte utilizado como ejemplo y el de la anomalía de *Sussman*. Analizar la aplicabilidad de este planificador cuando se incrementa el número de camiones y ciudades en el primer problema y el de bloques en el segundo.

[7] `http://www.cs.cmu.edu/~avrim/graphplan.html`

1.9. Visitar la dirección `http://teamcore.usc.edu/koenig/icaps/competitions.htm` para acceder a las distintas competiciones de planificación celebradas en la conferencia ICAPS. Estudiar los dominios definidos en PDDL, descargar los planificadores disponibles públicamente y probar algunos de los problemas existentes.

Bibliografía

[Anderson *et al.*, 1998] C. Anderson, D.E. Smith, and D. Weld. Conditional effects in Graphplan. In *Proc. AIPS-98*. AAAI Press, 1998.

[Asunción *et al.*, 2005] M. Asunción, L. Castillo, J. Fdez.-Olivares, O. García-Pérez, A. González, and F. Palao. SIADEX: an interactive artificial intelligence planner for decision support in forest fire fighting. *AI Communications*, 18:257–268, 2005.

[Barret and Weld, 1994] A. Barret and D.S. Weld. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[Barret *et al.*, 1996] A. Barret, D. Christianson, M. Friedman, K. Golden, J. Penberthy, Y. Sun, and D. Weld. UCPOP v4.0 user's manual. Technical Report TR 93-09-06d, Dept. of Computer Science and Engineering, University of Washington, Seattle, WA., 1996.

[Benton and Kambhampati, 2006] J. Benton and S. Kambhampati. YochanPS: PDDL3 simple preferences as partial satisfaction planning. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pages 23–25, 2006.

[Blum and Furst, 1997] A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[Bonet and Geffner, 1999] B. Bonet and H. Geffner. Planning as heuristic search: New results. In S. Biundo and M. Fox, editors, *Proc. European Conference on Planning (ECP-99)*, pages 360–372. Springer, 1999.

[Botea *et al.*, 2004] A. Botea, M. Enzenberger, M. Muller, and J. Schaeffer. Macro-FF. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pages 15–17, 2004.

[Castillo *et al.*, 2005] L. Castillo, J. Fdez.-Olivares, O. García-Pérez, and F. Palao. SIADEX. un entorno integral de planificación para el diseño de planes de actuación en situaciones de crisis. In *Jornadas de Transferencia Tecnológica de Inteligencia Artificial (TTIA-2005)*, pages 333–342, 2005.

[Chen *et al.*, 2004] Y. Chen, C.W. Hsu, and B.W. Wah. SGPlan: Subgoal partitioning and resolution in planning. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pages 30–32, 2004.

[Chen *et al.*, 2005] Y. Chen, C.W. Hsu, and B.W. Wah. Subgoal partitioning and resolution in SGPlan. In *Proc. System Demonstration Session, Int. Conference on Automated Planning and Scheduling (ICAPS-2005)*, pages 32–35, 2005.

[Chun, 1999] S.B. Chun. *Wargaming.* Air University Library. Maxwell AFB, Al., 1999.

[Currie and Tate, 1991] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[Do and Kambhampati, 2001] M.B. Do and S. Kambhampati. Sapa: a domain-independent heuristic metric temporal planner. In A. Cesta and D. Borrajo, editors, *Proc. European Conference on Planning (ECP-2001)*, pages 109–120, 2001.

[Do and Kambhampati, 2003] M.B. Do and S. Kambhampati. SAPA: a multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20:155–194, 2003.

[Edelkamp and Helmert, 2001] S. Edelkamp and M. Helmert. The model checking integrated planning system MIPS. *AI Magazine*, pages 67–71, 2001.

[Edelkamp and Hoffmann, 2004] S. Edelkamp and J. Hoffmann. PDDL2.2: the language for the classical part of IPC–4. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pages 2–6, 2004.

[Edelkamp *et al.*, 2006] S. Edelkamp, S. Jabbar, and M. Nazih. Large-scale optimal PDDL3 planning with MIPS-XXL. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pages 28–30, 2006.

[El-Kholy and Richards, 1996] A. El-Kholy and B. Richards. Temporal and resource reasoning in planning: the parcPLAN approach. In *Proc. 12th European Conference on AI (ECAI-96)*, pages 614–618, 1996.

[Erol *et al.*, 1994] K. Erol, J. Hendler, and D.S. Nau. HTN planning: Complexity and expressivity. In *Proc. 12th Nat. Conference on AI (AAAI-94)*, pages 1123–1128, Seattle, WA, 1994. AAAI Press.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fink and Veloso, 1994] E. Fink and M. Veloso. Prodigy planning algorithm. Technical Report CMU-94-123, Carnegie Mellon University, 1994.

[Fox and Long, 1999] M. Fox and D. Long. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

[Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Garrido and Onaindía, 2006] A. Garrido and E. Onaindía. Domain-independent temporal planning in a planning-graph-based approach. *AI Communications*, page (to be published), 2006.

[Gerevini and Long, 2006] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pages 7–13, 2006.

[Gerevini and Serina, 2002] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs with action costs. In *Proc. 6th Int. Conference on AI Planning and Scheduling (AIPS-2002)*, pages 281–290. AAAI Press, 2002.

[Gerevini *et al.*, 2003] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

[Gerevini *et al.*, 2004] A. Gerevini, A. Saetti, I. Serina, and P. Toninelli. Planning in PDDL2.2 domains with LPG-TD. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pages 33–34, 2004.

[Gervasio *et al.*, 1998] M. Gervasio, W. Iba, P. Langley, and S. Sage. Interactive adaptation for crisis response. In *Proc. Workshop on Interactive and Collaborative Planning (AIPS-1998)*, pages 29–36, 1998.

[Ghallab and Laruelle, 1994] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proc. 2nd Int. Conference on AI Planning Systems*, pages 61–67. Hammond, 1994.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning. Theory and Practice*. Morgan Kaufmann, 2004.

[Gil *et al.*, 2004] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit. Artificial intelligence and grids: Workflow planning and beyond. *IEEE Intelligent Systems*, pages 26–33, 2004.

[Green, 1969] C. Green. Application of theorem proving to problem solving. In *Proc. Int. Joint Conference on AI (IJCAI-69)*, pages 219–239, San Mateo, CA, 1969. Morgan Kaufmann.

[Haslum and Geffner, 2001] P. Haslum and H. Geffner. Heuristic planning with time and resources. In A. Cesta and D. Borrajo, editors, *Proc. European Conference on Planning (ECP-2001)*, pages 121–132, 2001.

[Haslum, 2004] P. Haslum. TP4'04 and HSP^*_a. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pages 38–40, 2004.

[Helmert and Richter, 2004] M. Helmert and S. Richter. Fast downward making use of causal dependencies in the problem representation. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition*, pages 41–43, 2004.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Hoffmann, 2000] J. Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proc. 12th Int. Symp. on Methodologies for Intelligent Systems*, Charlotte, North Carolina, USA, 2000.

[Hoffmann, 2003] J. Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[Hsu *et al.*, 2006] C-W. Hsu, B.W. Wah, R. Huang, and Y. Chen. New features in SGPlan for handling preferences and constraints in PDDL3.0. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pages 39–41, 2006.

[IPC, 2006] IPC. IPC: International planning competition, 2006. http://ipc.icaps-conference.org.

[Jónsson *et al.*, 2000] A. Jónsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in interplanetary space: Theory and practice. In *Proc. 5th Int. Conference on AI Planning Systems (AIPS-2000)*. AAAI Press, 2000.

[Kambhampati and Sanchez Nigenda, 2000] S. Kambhampati and R. Sanchez Nigenda. Distance based goal ordering heuristics for Graphplan. In *Proc. Int. Conference on Artificial Intelligence Planning and Scheduling*, pages 315–332, Menlo Park, CA, 2000. AAAI Press.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. 10th European Conference on AI*, pages 359–363, Vienna, Austria, 1992. Wiley.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proc. 13th Nat. Conference on AI*, pages 1194–1201, 1996.

[Kautz and Selman, 1998] H. Kautz and B. Selman. BLACKBOX: a new approach to the application of theorem proving to problem solving. In *Proc. Workshop on Planning as Combinatorial Search*, 1998.

[Kautz and Selman, 2006] H. Kautz and B. Selman. SATPLAN04: Planning as satisfiability. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pages 45–46, 2006.

[Knoblock, 1994] C.A Knoblock. Automatically generating abstraction for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[Knoblock, 2003] C.A. Knoblock. Deploying information agents on the web. In *Proc. Int. Joint Conference on AI (IJCAI-2003)*, pages 1580–1586, Acapulco, Mexico, 2003. Morgan Kaufmann.

[Köehler *et al.*, 1997] J. Köehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In Springer-Verlag, editor, *Lecture Notes in Artificial Intelligence (1348)*, pages 273–285. 1997.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. 9th Nat. Conference on AI*, pages 634–639, Anaheim, CA, 1991.

[McCarthy and P.J., 1969] J. McCarthy and Hayes P.J. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[McDermott, 1998] D. McDermott. *PDDL - The Planning Domain Definition Language*. AIPS-98 Planning Competition Committee, 1998.

[Muscettola, 1994] N. Muscettola. HSTS: Integrating planning and scheduling. In M. Zweben and M.S. Fox, editors, *Intelligent Scheduling*, pages 169–212. Morgan Kaufmann, San Mateo, CA, 1994.

[Nareyek, 2004] A. Nareyek. Artificial intelligence in computer games - state of the art and future directions. *ACM Queue*, 1(10:58–64, 2004.

[Nebel *et al.*, 1997] B. Nebel, Y. Dimopoulos, and J. Köehler. Ignoring irrelevant facts and operators in plan generation. In *Proc. European Conference on Planning (ECP-97)*, pages 338–350, Toulousse, France, 1997.

[Newell and Simon, 1963] A. Newell and H. Simon. *GPS, a Program that Simulates Human Thought*. McGraw Hill, NY, 1963.

[Nguyen *et al.*, 2002] X. Nguyen, S. Kambhampati, and R.S. Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135:73–123, 2002.

[Nilsson, 2000] N. Nilsson. *Inteligencia Artificial. Una Nueva Síntesis*. Mac Graw-Hill, 2000.

[Pednault, 1989] E. Pednault. ADL: Exploring the middle ground between strips and the situation calculus. In *Proc. Int. Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 324–332, San Francisco, CA, 1989. Morgan Kaufmann.

[Penberthy and Weld, 1992] J. Penberthy and D.S. Weld. UCPOP: a sound, complete, partial-order planner for ADL. In *Proc. Int. Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114, Los Altos, CA, 1992. Kaufmann.

[Penberthy and Weld, 1994] J. Penberthy and D. Weld. Temporal planning with continuous change. *Proc. 12th Nat. Conference on AI*, 1994.

[Penberthy, 1993] J. Penberthy. Planning with continuos change. Technical Report Ph.D. dissertation 93-12-01, Dept. of Computer Science and Engineering, U. Washington, 1993. Ph.D. dissertation.

[Refanidis and Vlahavas, 1999] I. Refanidis and I. Vlahavas. GRT: a domain independent heuristic for strips worlds based on greedy regression tables. In S. Biundo and M. Fox, editors, *Proc. European Conference on Planning (ECP-99)*, pages 346–358. Springer, 1999.

[RoboCup Rescue Technical Committee, 2000] RoboCup Rescue Technical Committee. Robocup-rescue simulator manual, 2000. Available at http://robomec.cs.kobe-u.ac.jp/robocup-rescue.

[Russell and Norvig, 2004] S. Russell and P. Norvig. *Inteligencia Artificial. Un Enfoque Moderno*. Pearson Educación, 2004.

[Sacerdoti, 1974] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[Sacerdoti, 1977] E.D. Sacerdoti. *A Structure for Plans and Behavior*. American Elsevier, New York, 1977.

[Smith and Weld, 1999] D.E Smith and D.S. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. 16th Int. Joint Conference on AI (IJCAI-99)*, pages 326–337, Stockholm, Sweden, 1999.

[Tate, 1977] A. Tate. Generating project networks. In *Proc. Int. Joint Conference on AI (IJCAI-77)*, pages 888–893, Cambridge, MA, 1977.

[Vere, 1983] S. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:246–267, 1983.

[Vidal and Ghallab, 1996] T. Vidal and M. Ghallab. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proc. 12th European Conference on AI (ECAI-96)*, pages 48–52, 1996.

[Vidal, 2004] V. Vidal. A lookahead strategy for heuristic search planning. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2004)*, pages 150–159, 2004.

[Weld and Smith, 1998] A. Weld and D.E. Smith. Extending Graphplan to handle uncertainty and sensing actions. In *Proc. AAAI-98*, pages 897–904, Madison, WI, 1998.

[Weld, 1994] D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):93–123, 1994.

[Wilkins, 1988] D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, CA, 1988.

[Xing *et al.*, 2006] Z. Xing, Y. Chen, and W. Zhang. MaxPlan: Optimal planning by decomposed satisfiability and backward reduction. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2006) – International Planning Competition*, pages 53–55, 2006.

[Yang and Tenenberg, 1990] Q. Yang and J.D. Tenenberg. ABTWEAK: Abstracting a nonlinear, least commitment planner. In *Proc. American Conference on Artificial Intelligence (AAAI-90)*, pages 204–209, Boston, MA, 1990. AAAI Press.

[Yang, 1997] Q. Yang. *Intelligent Planning. A Decomposition and Abstraction Based Approach.* Springer-Verlag, Berlin, Heidelberg, 1997.