
Universitat Politècnica de València

# Diseño de una heurística basada en *landmarks* para un planificador multiagente

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

*Autor:* Ana Oropesa Física

*Directora:* Eva Onaindía de la Rivaherrera

15 de septiembre de 2014

*Recuerda el pasado.*
*Vive el presente.*
*Construye el futuro.*

# Agradecimientos

Quisiera dedicar unas líneas a todas aquellas personas que han participado en mi etapa universitaria.

A mis compañeros, Alejandro y Pablo, que han estado siempre a mi lado para todas las cosas que he necesitado, a cualquier hora y desde cualquier lugar del mundo. Gracias por vuestra dedicación, paciencia y ayuda. Gracias, de corazón, por todo.

A la directora de mis dos mejores proyectos estudiantiles, Eva Onaindía. Su ayuda, consejo, comprensión y aguante han sacado lo mejor de mí. Gracias por acompañarme durante estos dos años.

A mi profesor en la vida, Alberto Conejero, por guiarme en este sendero con la mejor heurística existente.

A mis compañeros de clase y amigos, porque aunque estemos a miles de kilómetros sois la mejor compañía que alguien pudiera soñar.

A todas esas personas que han colaborado directa o indirectamente en este trabajo final de máster. Habéis hecho que el camino sea mucho menos duro.

Por último, quisiera agradecer su apoyo incondicional a mi motor económico, moral y emocional. Papá y mamá, sin vosotros no habría llegado hasta aquí.

**Resumen**

Este trabajo fin de máster presenta el diseño de una heurística basada en *landmarks* y su posterior adaptación a un planificador multiagente. Finalmente, se mostrarán resultados del rendimiento aportado por la función heurística en el planificador.

# Índice general

# Capítulo 1

# Introducción

El proyecto desarrollado en el presente trabajo se enmarca en el área de la Inteligencia Artificial (IA) y más concretamente en la planificación multiagente. En este capítulo se explica la motivación del trabajo y un resumen de la organización del documento.

## 1.1. Motivación

El objetivo de la planificación automática en IA es encontrar un conjunto de acciones o plan que, aplicado a una situación inicial de un problema, permite obtener el conjunto de objetivos definidos en dicho problema.

Un *landmark* es un hecho que tiene de ser cierto en algún punto en cualquier plan solución de un mismo problema. La información proporcionada por los *landmarks* se ha utilizado para el diseño de numerosas heurísticas de planificación en contextos monoagente, consiguiendo mejoras significativas en el rendimiento de los planificadores. Un ejemplo de una exitosa heurística es la utilizada por el planificador LAMA [RW10], del cual se hablará en la sección 2.4.2. A tenor de estos buenos resultados, el objetivo que se plantea en este trabajo es diseñar, implementar y adaptar una heurística basada en la información extraída de *landmarks* a un planificador multiagente.

La heurística diseñada se ha aplicado en el planificador multiagente FMAP, el cual ha sido desarrollado por el grupo de planificación GRPS-AI[1] del Departamento de Sistemas Informáticos y Computación[2]. FMAP es un planificador cooperativo diseñado para que varios agentes resuelvan conjuntamente

[1] `http://users.dsic.upv.es/grupos/grps/`
[2] `https://www.upv.es/entidades/DSIC/index.html`

una tarea de planificación. En FMAP cada agente contribuye a la construcción incremental del plan conjunto mediante un planificador completo de orden parcial hacia delante. La búsqueda del proceso de planificación se guía mediante una heurística basada en información sobre los cambios de las variables estado del problema de planificación. Además, la heurística utilizada por FMAP permite evaluar planes en entornos con información incompleta o privada.

FMAP está integrado en la plataforma multiagente PlanInteraction [3], resultado de un proyecto de investigación desarrollado dentro del grupo GRPS-AI[4]. La plataforma PlanInteraction proporciona un conjunto de APIs y métodos que facilitan la coordinación y gestión de información entre los agentes.

En resumen, el trabajo a realizar consiste en varias fases. En primer lugar, se implementará una versión distribuida del algoritmo de cálculo del grafo de *landmarks*[SO03]. Una vez obtenidos los grafos *landmarks* para cada uno de los agentes participantes en el problema, se implementará una heurística multiagente que combine la información proporcionada por dichos grafos con la heurística existente en FMAP. Por último, se expondrán los resultados de la experimentación con dicha heurística para determinar su rendimiento.

## 1.2. Organización del documento

En esta sección, se muestra la organización general del documento, el cual se divide en los siguientes capítulos:

- **Capítulo 1: Introducción**

  En este capítulo se presenta una introducción sobre el trabajo, su motivación y un resumen de los temas a tratar.

- **Capítulo 2: Estado del arte**

  Este capítulo presenta un breve estado del arte de la planificación automática, la definición y utilización de *landmarks* en planificación así como la heurística basada en *landmarks* del planificador LAMA.

[3] `http://servergrps.dsic.upv.es/planinteraction/`
[4] `http://users.dsic.upv.es/grupos/grps/`

- **Capítulo 3: Entorno de planificación multiagente**

  En este capítulo se presenta el funcionamiento del planificador FMAP y su integración en la plataforma PlanInteraction.

- **Capítulo 4: Extracción multiagente de *landmarks***

  En este capítulo se detalla el diseño y la implementación del cálculo distribuido de *landmarks* para cualquier problema de planificación dentro de la plataforma PlanInteraction. Además, en este capítulo se presenta también el diseño de la nueva heurística FMAP-Land. Esta heurística combinará la información proporcionada por los *landmarks* extraídos de un problema con la heurística existente de FMAP.

- **Capítulo 5: Resultados experimentales**

  En este capítulo se comentan los resultados obtenidos de la comparación entre FMAP y FMAP-Land. Los dominios utilizados serán *Rovers*, *Openstacks* y *MA-Blocksworld*, *benchmarks* en la Competición Internacional de *Planning* [5].

- **Capítulo 6: Conclusiones y trabajo futuro**

  En este capítulo se detalla un resumen del trabajo realizado, de las conclusiones extraídas del mismo y de las líneas de trabajo futuro abiertas.

[5] `http://ipc.icaps-conference.org/`

# Capítulo 2

# Estado del arte

La planificación en Inteligencia Artificial (IA) se encarga de estudiar algoritmos y técnicas para la construcción de planes. Un plan es un conjunto de acciones que, aplicadas en un estado inicial de un problema, conduce a la obtención de un conjunto de metas deseadas. En la práctica, la planificación se sustenta en el uso de funciones de utilidad, también conocidas como heurísticas, que permiten evaluar la selección de acciones o estados de acuerdo a la utilidad que ofrecen al agente de planificación [GNT04].

## 2.1. Planificación monoagente

El problema de la planificación en IA se define como sigue [Wel99]:

"Dada una descripción del estado inicial del mundo (en algún lenguaje formal), una descripción de la meta del agente (el comportamiento deseable), y una descripción de las posibles acciones (atómicas) que pueden llevarse a cabo modelada a través de funciones de transformación de estados, el objetivo es obtener un plan. Es decir, un conjunto de acciones que transformen el estado inicial en un estado en el que las metas del agente se cumplan."

La planificación clásica puede verse como un proceso de búsqueda en el que un único agente sintetiza un conjunto de acciones que le permiten conseguir sus objetivos a partir de una situación inicial. Los planificadores clásicos se enfrentan a dos importantes escollos: la definición de lenguajes robustos y significativos para modelar las acciones, y el desarrollo de técnicas eficientes para la resolución del problema.

A lo largo de los años, la investigación en planificación clásica se ha centrado en el estudio de diferentes paradigmas de resolución de problemas. Estos paradigmas pueden clasificarse de acuerdo a los siguientes conceptos [Sap05]:

- **Representación y búsqueda basada en estados:** Los planificadores operan sobre un árbol de búsqueda donde un nodo del árbol representa un estado o situación concreta, donde un nodo del árbol de búsqueda representa una situación concreta del mundo. En este caso, un plan es una secuencia de acciones representada en una rama del árbol de búsqueda.

- **Representación y búsqueda basada en planes:** Los planificadores construyen planes como secuencias parcialmente ordenadas de acciones. En este caso, un nodo del árbol de búsqueda representa un plan de orden parcial construido con las acciones existentes.

Los primeros planificadores que se construyeron utilizaban el paradigma de representación y búsqueda basada en estados, de modo que las transiciones entre estados se producen por la aplicación de acciones. Posteriormente, la necesidad de manipular planes de orden parcial durante la búsqueda condujo al desarrollo de algoritmos de búsqueda en el espacio de planes [PW92, BW94, Wel94]. El paradigma de Planificación de Orden Parcial (POP) refina los planes parciales a través de la adición de acciones, enlaces causales y restricciones de orden. También existen enfoques adicionales que aplican otro tipo de refinamientos, como el paradigma Hierarchical Task Network (HTN) [EHN94], que reemplaza acciones abstractas por fragmentos de plan de bajo nivel o acciones primitivas.

La investigación en planificación ha introducido otros enfoques que incrementan significativamente la eficiencia de los sistemas de planificación. Entre estos modelos, destacan SATPLAN [KS96], GRAPHPLAN [BF97], y la planificación heurística [BG01]. SATPLAN convierte el problema de planificación en un problema de satisfacción de restricciones; GRAPHPLAN utiliza grafos de planificación, y los planificadores heurísticos son, generalmente, planificadores basados en estados. El objetivo de la planificación heurística es buscar funciones de utilidad o heurísticas que permitan guiar la búsqueda hacia el objetivo. Este enfoque ha resultado muy exitoso, como se ha demostrado con los planificadores FF [HN01] y LAMA [RW10].

### 2.1.1. Modelización de un problema de planificación

A continuación se verá un ejemplo de modelización de un problema de planificación monoagente. En un problema monoagente existe una única entidad de planificación, la cual tiene información completa sobre el entorno en el que se desarrolla el problema, así como del estado inicial del mismo.

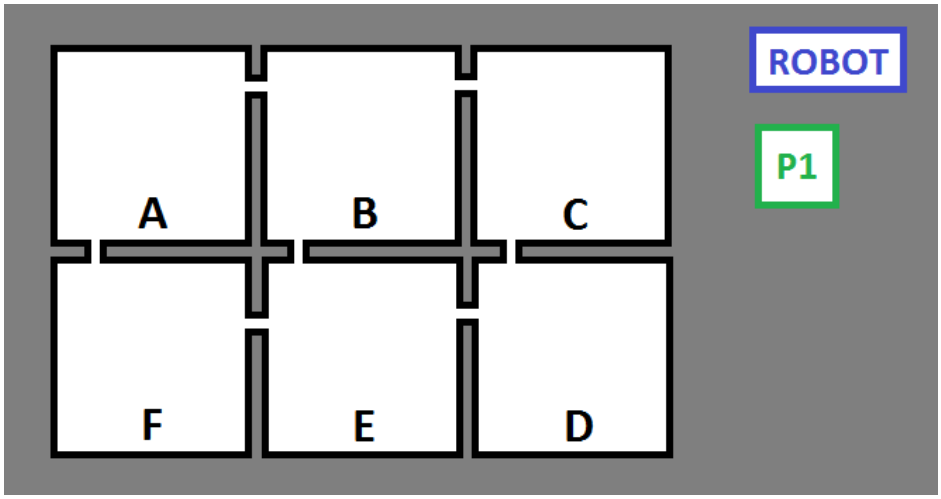


Figura 2.1: Problema1 del dominio Robot

La figura 2.1 muestra gráficamente el entorno en el que se desarrolla el *problema1*. Este problema consta de una serie de áreas delimitadas: A, B, C, D, E y F entre las cuales se puede transitar si, y sólo si, están conectadas entre ellas. Además, el problema consta de dos elementos más: un paquete y un robot. El único agente de este problema es el robot. Este agente puede realizar un conjunto de acciones tales como: cargar un paquete, descargar un paquete y moverse de un área a otra. El paquete puede estar en cualquiera de las áreas o ser trasportado por el robot.

Existen diversos lenguajes para la modelización de los problemas de planificación. El lenguaje más conocido es PDDL[GHK+98], el cual se utiliza en la mayoría de los planificadores independientes del domino. La versión más reciente de *PDDL* es *PDDL3.1*. El objetivo de esta versión es enriquecer el lenguaje con una representación de problemas al estilo de SAS+ [BN95]. Sin embargo, SAS+ permite usar variables de modo muy limitado, ya que no permite anidar, y sólo permite comparaciones y asignaciones con constantes. Como mejora, *PDDL3.1* introduce variables objeto. Las variables objeto son variables asociadas a un dominio finito de valores, una solución flexible inspi-

rada por el formalismo *Functional Strips* [Gef00]. Además, *PDDL3.1* permite especificar costes numéricos en las acciones del dominio, de modo que los costes de las acciones adquieren importancia para determinar la calidad del plan.

Una tarea de planificación se define mediante dos ficheros, un fichero para describir el dominio del problema y el otro para describir el problema concreto de planificación a resolver.

#### 2.1.1.1. Dominio de planificación

El fichero del dominio contiene la siguiente información:

- Las clases de objeto (tipos) existentes en el dominio.
- Los predicados del dominio.
- Las variables estado (funciones) del dominio.
- Las acciones del dominio.

La figura 2.2 muestra el código del dominio del problema Robot:

```
(define (domain Robot)
(:requirements :typing :equality :fluents)
(:types box place - object
     area agent - place
     robot - agent)

(:predicates
     (connected ?l1 - area ?l2 - area))

(:functions
     (pos ?x - box) - place
     (at ?x - robot)  - place)

(:action go-to
     :parameters (?r - robot ?from ?to - place)
     :precondition (and (= (at ?r) ?from)
                        (connected ?from ?to))
     :effect (and (assign (at ?r) ?to)))

(:action pick-up
     :parameters (?r - robot ?bk - box ?l - place)
     :precondition (and (= (pos ?bk) ?l) (= (at ?r) ?l))
     :effect (and (assign (pos ?bk) ?r)))
```

```
(:action drop
     :parameters (?r - robot ?bk - box ?l - place)
     :precondition (and (= (pos ?bk) ?r) (= (at ?r) ?l))
     :effect (and (assign (pos ?bk) ?l))))
```

Figura 2.2: Fichero del dominio Robot

Para entender mejor el fichero, se explicará detalladamente el significado de cada una de sus partes.

Todo objeto participante en un problema está asociado a una clase de objeto o tipo. Al definir las clases de objetos, existe la posibilidad de que una clase englobe otra como una relación de herencia. De cualquier modo, la clase raíz por defecto que engloba todas las clases de objetos o tipos definidos por el usuario se denomina `"object"`. En este problema, los objetos se asocian a las clases existentes de la siguiente forma:

- El tipo o clase `"robot"` se define como un subtipo del tipo `"agent"`.
- Dado que el tipo `"place"` engloba los tipos de `"area"` y `"agent"` todos los elementos asociados a estas clases serán también del tipo `"place"`.
- Por último, los tipos `"box"` y `"place"`, al igual que todos los objetos y clases de objetos que incluyen, son considerados del tipo global `"object"`.

En la sección `:predicates` se pueden ver los predicados del dominio. Éstos expresan una relación entre tipos de objetos. En este problema se especifica el predicado `"connected"` para indicar las áreas concretas del problema que están conectadas entre sí.

En la sección `:functions` se describen las funciones del dominio. Una variable estado modela el estado concreto de una característica de un objeto de un problema. Para representar una variable estado, PDDL hace uso de las funciones. En este caso, se definen dos variables en el dominio del problema: `(pos ?x - box) - place` y `(at ?x - robot) - place`. La variable `(pos ?x - box) - place` indica que el lugar donde se encuentra el paquete es un objeto de tipo `"place"` mientras que la variable `(at ?x - robot) - place` indica que el lugar donde se encuentra el robot también es de tipo `"place"`. Por ejemplo, si el paquete está en el área `A` y el `Robot` está en el área `B`, los valores asociados a las variables serían los siguientes:

- `(= (pos paquete) A)`
- `(= (at robot) B)`

Las acciones en PDDL se definen siempre con los siguientes elementos: parámetros, precondiciones y efectos. Para explicar la función que realiza cada elemento se tomará como ejemplo la acción `go-to` de la figura 2.2:

- **Parámetros:** Son un conjunto de variables junto con su correspondiente tipo que indican los objetos que intervienen en la acción. Nótese que la acción `go-to` puede ser instaciada para cualquier robot y dos posiciones.
- **Precondiciones:** Una precondición establece una condición que debe cumplirse en un estado para que la acción pueda ejecutarse. Si en una acción hubiese varias precondiciones deberían cumplirse todas para poder ejecutar la acción. La acción `go-to` tiene dos precondiciones: una especifica el área donde debe estar el robot, y la otra, establece que debe existir una conexión entre el área donde está actualmente el robot y el área hacia donde quiere dirigirse. En tiempo de ejecución, para comprobar el cumplimiento de las precondiciones de una acción en un estado del problema, se aplica un proceso de *"pattern-matching"*, instanciando cada una de las precondiciones de la acción con las variables o hechos correspondientes.
- **Efectos:** Como su nombre indica, son los efectos resultantes de aplicar la acción. El resultado de la aplicación de la acción `go-to` cambia la posición del robot indicada por la variable estado `(at ?r)`, la cual representa la posición del robot `?r`. De este modo, el valor de la posición del robot `(at ?r)` cambiará de su valor actual indicado por el parámetro `?from` a su nueva posición indicado por el parámetro `?to`.

Las restantes acciones que puede realizar el agente, definidas en el fichero del dominio 2.2, son las siguientes:

La acción `pickup` puede realizarse siempre que un paquete se encuentre en la misma área en la que se encuentra un robot. El resultado de esta acción cambia la posición en la que se encuentra el paquete. De este modo, el paquete deja de estar en el área donde se encontraba inicialmente y ahora es sujetado por el robot.

Por último, la acción `drop` puede realizarse cuando un robot sujeta un paquete. El resultado de esta acción hace que el robot deposite el paquete en el área donde se encuentra.

#### 2.1.1.2. Problema de planificación

Además de la definición del dominio, el agente debe conocer los elementos del problema concreto de planificación a resolver. Estos elementos son los objetos que intervienen en el problema, el estado inicial y el estado final que el agente ha de conseguir. Toda esta información se encuentra en el fichero PDDL del problema tal y como se muestra en la figura 2.3.

```
(define (problem problema1)
(:domain Robot)
(:objects
       robot - robot
       p1 - box
       A B C D E F - place)
(:init
       (connected A B) (connected B A)
       (connected B C) (connected C B)
       (connected C D) (connected D C)
       (connected D E) (connected E D)
       (connected E F) (connected F E)
       (connected F A) (connected A F)
       (= (pos p1) B)
       (= (at robot) A))

(:global-goal
       (and
           (= (pos p1) C)
           (= (at robot) D))))
```

Figura 2.3: Fichero del problema a resolver

En la sección `:objects` de la figura 2.3 se muestran los nombres que identifican los objetos del problema junto a su clase o tipo. En este problema, los objetos son:

- Áreas: `A, B, C, D, E`
- Robot: `robot`
- Paquete: `p1`

El estado inicial del problema, sección `:init` de la figura 2.3, contiene instanciaciones de predicados, denominadas proposiciones, y los valores asignados a las variables estado. A estos conjuntos <variable,valor> se les denomina *fluents*. Por ejemplo, se estudiará elemento a elemento el *fluent* `(= (pos p1) B)`. En este caso, la variable de este *fluent* es `(pos p1)`, y su valor, el área `B`. Este conjunto <`(pos p1)`,`B`> indica que el área donde se encuentra el paquete `p1` es el área `B`. El estado inicial de un problema se indica mediante *fluents*. En este caso, en el estado inicial del problema se señalan las conexiones entre las áreas y las localizaciones iniciales del paquete y el robot.

Por último, en la sección del estado final `:global-goal` de la figura 2.3 se observan las metas que han de cumplirse para finalizar el problema. Dichos estados final o metas también se representadan mediante *fluents*. En este caso, las metas del problema requieren que el paquete esté en el área `C`, representado por el *fluent* `(= (pos p1) C)`, y que el robot se encuentre en el área `D`, cuya representación viene dada por el *fluent* `(= (at robot) D))))`.

## 2.2. Formalización de elementos básicos en la planificación

A continuación, se presenta la formalización de una tarea de planificación monoagente y de variable instanciada o *fluent*.

**Definición 1.** *(**Variable instanciada o** **fluent**) Una **variable instanciada** o **fluent** del problema es una tupla de la forma $\langle v, d \rangle$, donde $v \in \mathcal{V}$, $d \in \mathcal{D}_v$. $\mathcal{V}$ es un conjunto finito de variables estado que modelan los estados del mundo. Además, una variable estado $v \in \mathcal{V}$ está asociada a un dominio finito de valores mutuamente exclusivos $\mathcal{D}_v$. Siendo $\mathcal{O}$ un conjunto finito de objetos que modelan los elementos del dominio de planificación sobre los que actúan las acciones de planificación, el valor en el dominio de una variable se corresponde con un de estos objetos, esto es, $\forall v \in \mathcal{V}$, $\mathcal{D}_v \subseteq \mathcal{O}$.*

**Definición 2.** *(**Tarea de planificación monoagente**) Una **tarea de planificación monoagente** es una tupla $\mathcal{T} = \langle \mathcal{O}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \rangle$ donde cada elemento tiene el siguiente significado:*

- *$\mathcal{O}$ es un conjunto finito de objetos. Estos objeto modelan los elementos del dominio de planificación sobre los que actúan las acciones de planificación.*

- *$\mathcal{V}$ es un conjunto finito de variables estado que modelan los estados del mundo.*
- *$\mathcal{A}$ es el conjunto de acciones deterministas del agente.*
- *$\mathcal{I}$ es el conjunto de valores iniciales asignados a las variables estado $\mathcal{V}$ y representa el estado inicial de la tarea de planificación multiagente $\mathcal{T}$.*
- *$\mathcal{G}$ es el conjunto de metas de la tarea de planificación monoagente que el agente deben satisfacer; $\mathcal{G}$ representa los valores que las variables estado deben adquirir en el estado final.*

*La información del estado del mundo que posee el agente se modela mediante un conjunto de variables instanciadas,* fluents*. Del mismo modo se modela el estado inicial $\mathcal{I}$ y las metas $\mathcal{G}$.*

## 2.3. Planificación multiagente

La planificación multiagente cooperativa extiende el problema clásico de planificación monoagente a un entorno multiagente, donde cada agente es una entidad con capacidades de planificación. Una tarea de planificación multiagente consiste en realizar conjuntamente entre diferentes entidades la construcción de un plan solución que alcance las metas del problema. De este modo, el problema de planificación multiagente cooperativo puede definirse como el problema de coordinar un conjunto de agentes, en un entorno compartido donde la información está distribuida, tal que entre todos sean capaces de construir un plan que resuelva el problema dado. En la planificación multiagente se enfatizan dos aspectos claves que no están presentes en la planificación monoagente: la coordinación y la distribución de la información entre todos los agentes.

En líneas generales, resolver una tarea de planificación multiagente requiere realizar todos o algunas de las siguientes tareas:

1. Refinamiento de las metas
2. Distribución de tareas
3. Coordinación antes de planificar
4. Planificación individual

5. Coordinación después de planificar

6. Ejecución del plan

Algunas de las tareas anteriores pueden ser eliminados o fusionados según el tipo de problema o las metas que se quieran conseguir. Por ejemplo, algunos problemas no distribuyen las metas explicítamente, por lo que la tarea 2 no se realizaría.

### 2.3.1. Modelización de un problema multiagente

En la modelización de un problema multiagente, cada agente es una entidad independiente con capacidad de visión y acción sobre el mundo con el que va a interactuar. Dichas capacidades se definen en el fichero del dominio del agente, tal y como está expresado en la sección 2.1.1.1. Además, cada agente, al igual que en la modelización monoagente, tiene conocimiento sobre el estado inicial y las metas a conseguir del problema a resolver.

La razón de realizar una modelización multiagente de un problema es, en gran parte, la distribución de información. Hay dos tipos de distribución de la información: funcional y espacial.

Si existe una distribución funcional, significa que los agentes tienen habilidades distintas y que éstos pueden realizar acciones distintas. En dicho caso, se define un fichero de dominio para cada tipo de agente, y en cada fichero, se especifica las acciones que el tipo de agente puede realizar. Por ejemplo, en un dominio donde existan varios tipos de agentes, como por ejemplo vehículos, aviones, trenes y camiones, existirá, para cada tipo de agente, un fichero del dominio en el que se especificará las acciones que puede realizar cada agente.

Si los agentes definidos en el problema de planificación están distribuidos espacial o geográficamente, dichos agentes tendrán un conocimiento diferente del mundo y del estado inicial. De este modo, algunos agentes pueden conocer ciertos elementos del problema parcialmente e incluso pueden desconocer otros totalmente.

El objetivo en un problema de planificación multiagente cooperativa es que todos los agentes cooperen entre ellos para resolver un conjunto de metas globales. Por lo tanto, las metas, apartado `:global-goal`, serán iguales para todos los agentes.

Por tanto, un problema de planificación multiagente es aquel en el que dos o más agentes parten de unos datos iniciales y cooperan entre sí para conseguir unas metas comunes. En la planificación multiagente, todos los agentes han de coordinarse entre sí y compartir la información que generan con los demás agentes del problema para resolver las metas de éste. De esta forma, puede darse el caso de que la cooperación entre agentes sea necesaria para conseguir un objetivo en particular porque ninguno de los agentes tiene todas las capacidades necesarias para resolver el objetivo por sí mismo.

Aunque en una tarea de planificación cooperativa todos los agentes son conocedores de todos los objetivos del problema, dichos agentes pueden tener un conocimiento distinto del mismo. El hecho de que los agentes puedan tener un conocimiento distinto de un mismo problema se debe a que, o bien son agentes distribuidos espacialmente con visiones diferentes del problema, o bien son agentes que tienen habilidades distintas. Por esta razón, se define un fichero de problema para cada agente en el que se plasman los hechos que conoce y de los que tiene visión dicho agente. Además, al ser un problema multiagente, es necesario diferenciar a los agentes mediante un predicado en su fichero del problema. Esta diferenciación se explicará en la sección 2.3.1.2.

En cada fichero del problema se especificará el estado inicial de cada agente en la sección `:init`, el cual dependerá de la distribución de la información del problema. Cuanto más distribuida esté la información, más diferentes serán los estados iniciales.

#### 2.3.1.1. Privacidad

Una de las principales características en la planificación multiagente es el manejo de la información privada. Al tratarse de información distribuida, puede darse el caso de que ningún agente tenga conocimiento sobre todas las variables del problema y los valores de los dominios de dichas variables. La privacidad se mide siempre entre dos agentes y el conocimiento que ambos tienen sobre un fluent o par <variable,valor>. Hay varios tipos de privacidad: nula, parcial y total. Para denotar la visión de un agente cualquiera, $i$, sobre las variables de un problema y sus correspondientes valores se usará la notación de superíndice $x^i$ para cada aspecto $x$. A continuación, se explicará un ejemplo de cada tipo de privacidad mediante dos agentes, $i$ y $j$, y un *fluent* $<v,d>$.

Los valores que un agente $i$ conoce de una variable $v$ se define como $D_v^i$. Estos valores son un subconjunto de los valores del dominio de la variable en la tarea $\mathcal{D}_v^i \subseteq \mathcal{D}_v$.

Cabe destacar también los significados de $\mathcal{V}^{ij}$, conjunto intersección de variables visibles por los agentes $i$ y $j$, y $\mathcal{D}^{ij}$, conjunto intersección de valores conocidos de la variable $v$ por los agentes $i$ y $j$.

Sea un *fluent* $< v, d >$ totalmente conocido por el agente $i$, es decir, $v \in \mathcal{V}^i$ y $d \in \mathcal{D}_v^i$.

- **Privacidad nula**: Si $v \in \mathcal{V}^{ij}$ y $d \in \mathcal{D}_v^{ij}$, el *fluent* $< v, d >$ es conocido por ambos agentes y, por lo tanto, es un *fluent* público. Al ser un *fluent* público, toda información sobre dicho *fluent* es intercambiable entre los agentes $i$ y $j$.
- **Privacidad parcial**: Si $v \in \mathcal{V}^{ij}$ y $d \notin \mathcal{D}_v^{ij}$, el *fluent* $< v, d >$ es parcialmente privado del agente $i$ con respecto al agente $j$, ya que el agente $j$ desconoce el valor $d$. En este caso, el agente $i$ sólo podría compartir con el agente $j$ el *fluent* $< v, \perp >$, donde $\perp$ es un valor indefinido.
- **Privacidad total**: Si $v \notin \mathcal{V}^{ij}$, es decir, $v \in \mathcal{V}^i$ y $v \notin \mathcal{V}^j$, el *fluent* $< v, d >$ no es conocido por el agente $j$, ya que dicho agente ni siquiera conoce la existencia de la variable $v$. En este caso, el agente $i$ no podría compartir con el agente $j$ ninguna información sobre el *fluent* $< v, d >$. De esta manera, se dice que el *fluent* $< v, d >$ es un *fluent* privado del agente i.

#### 2.3.1.2. Dominio de planificación multiagente

Como ejemplo de modelización de un problema multiagente, se analizará una extensión del problema monoagente de la sección 2.1.1. Esta vez, el problema dispone de dos agentes del mismo tipo. Ambos agentes disponen además de las mismas habilidades, por lo que sólo hay un fichero del dominio. Estos agentes, `robot1` y `robot2`, poseen las capacidades descritas en el dominio Robot. Además de los dos robots asociados a los agentes, un robot por agente, en el problema intervienen tres paquetes y seis áreas. Se remarcarán los cambios de la extensión multiagente en el código PDDL correspondiente.

La figura 2.4 muestra el código PDDL del dominio multiagente Robot:

```
(define (domain Robot)
(:requirements :typing :equality :fluents)
(:types box place - object
        area agent - place
        robot - agent)

(:predicates
       (connected ?l1 - area ?l2 - area)
       (my-agent ?r - robot))

(:functions
       (pos ?x - box) - place
       (at ?x - robot)  - place)

(:action go-to
       :parameters (?r - robot ?from ?to - place)
       :precondition (and (= (at ?r) ?from)
                          (connected ?from ?to)(my-agent ?r))
       :effect (and (assign (at ?r) ?to)))

(:action pick-up
       :parameters (?r - robot ?bk - box ?l - place)
       :precondition (and (= (pos ?bk) ?l) (= (at ?r) ?l)
                          (my-agent ?r))
       :effect (and (assign (pos ?bk) ?r)))

(:action drop
       :parameters (?r - robot ?bk - box ?l - place)
       :precondition (and (= (pos ?bk) ?r) (= (at ?r) ?l)
                          (my-agent ?r))
       :effect (and (assign (pos ?bk) ?l)))
```

Figura 2.4: Fichero del dominio multiagente Robot

Como se puede ver en la sección `:predicates` de la figura 2.4, existe un predicado más en el dominio. Este predicado, **`(my-agent ?r - robot)`**, es necesario para diferenciar qué robot está asociado a cada agente. Todos los dominios multiagente necesitan un predicado de este tipo para poder diferenciar los objetos asociados a cada agente. Además, todas las acciones utilizan dicho predicado para ligar la acción al agente y a los objetos que éste pueda poseer.

#### 2.3.1.3. Problema de planificación multiagente

Como se ha comentado anteriormente, en los problemas multiagente existe un fichero del problema por cada agente. Estos ficheros contienen el estado inicial y las metas a conseguir para cada uno de los agentes. En este caso, el agente `robot1` no conoce las áreas `E` y `F` y el agente `robot2` desconoce tanto las áreas `A` y `B` como la existencia del paquete `package2`. Se mostrarán tanto el fichero del agente `robot1` como el del agente `robot2` para ilustrar ejemplos de los tipos de privacidad.

La figura 2.5 muestra el fichero del problema del agente `robot1`.

```
(define (problem problema1)
(:domain Robot)
(:objects
     robot1 robot2 - robot
     package1 package2 package3- box
     A B C D - place)

(:shared-data
  ((pos ?x - box) - place)
  (at ?x - robot)  - place) - robot2)

(:init
     (my-agent robot1)
     (connected A B) (connected B A)
     (connected B C) (connected C B)
     (connected C D) (connected D C)
     (= (pos package1) C)
     (= (pos package2) C)
     (= (pos package3) C)
     (= (at robot1) C)
     (= (at robot2) D))

(:global-goal
     (and
          (= (pos package1) A)
          (= (pos package2) D)
          (= (pos package3) E)
          (= (at robot1) B)
          (= (at robot2) F))))
```

Figura 2.5: Fichero del problema agente `robot1`

La sección `:domain` de la figura 2.5 indica el dominio del agente `robot1`. En este caso, el agente `robot1` tendrá las capacidades descritas en el dominio

Robot de la sección 2.3.1.2.

La sección `:objects` de la figura 2.5 muestra los elementos del problema que son conocidos por el agente `robot1`. En este caso, el agente `robot1` no conoce ni tiene acceso a las áreas `E` y F, por lo que no figuran en esta sección.

En la sección `:init` de la figura 2.5 se encuentra el estado inicial del problema del agente `robot1`. En esta sección se indican las áreas que están conectadas entre sí, la posición inicial de los tres paquetes del problema, todos ubicados en el área `C`, y las posiciones iniciales del agente `robot1` y del agente `robot2`. Estas posiciones son el área `C` y el área `D` respectivamente. Respecto a las áreas `E` y F, no se define ninguna propiedad, ya que son desconocidas para el agente `robot1`.

La sección `:shared-data` de la figura 2.5 muestra la información compartida por el agente `robot1` con el agente `robot2`. En este caso, tal y como se puede ver en la figura 2.5, el agente `robot1` comparte con el agente `robot2` la posición de los paquetes del problema y su posición.

Dados los datos de las secciones anteriores, `:objects`, `:init` y `:shared-data`, a continuación se muestran algunos ejemplos de privacidad de fluents del agente `robot1` respecto al agente `robot2`.

El *fluent* `(= (pos package1) C)` es un *fluent* público ya que tanto el agente `robot1` como el agente `robot2` conocen la variable `(pos package1)` y el valor `C`. Por tanto, entre el agente `robot1`, el agente `robot2` y el *fluent* `(= (pos package1) C)` existe una relación de privacidad nula. Dada este relación de privacidad el agente `robot1` puede compartir con el agente `robot2` cualquier información sobre el *fluent*.

El *fluent* `(= (pos package1) A)` es conseguido por el agente `robot1`, razón por la cual este agente tiene conocimiento tanto de la variable como del valor del mismo. Sin embargo, el agente `robot2`, aunque conoce la variable del *fluent*, no conoce su valor, porque desconoce el área `A`. Ya que ambos agentes conocen la variable del *fluent* pero el agente `robot2` no conoce su valor, el fluent (= (pos package1) A) es parcialmente privado para el agente `robot1` hacia el agente `robot2`. En este caso, el agente `robot1` sólo podría compartir con el agente `robot2` la siguiente información sobre este *fluent*: `(= (pos package1)` $\perp$`)`, siendo $\perp$ un valor indefinido.

Finalmente, el agente `robot1` mantiene una relación de privacidad total sobre el *fluent* `(= (pos package2) B)` con el agente `robot2`. Este *fluent* es conseguido por el agente `robot1`, por lo que este agente conoce tanto la variable como el valor del *fluent*. Por el contrario, el agente `robot2` no conoce la variable `(pos package2)`, ya que desconoce que exista un paquete `package2`. De este modo, ya que el agente `robot2` desconoce la variable, el agente `robot1` no puede compartir ninguna información sobre este *fluent* con él.

Respecto a la sección `:global-goal` de la figura 2.5, se observa como, al ser metas comunes y una planificación cooperativa, pueden existir metas con variables o valores desconocidos para el agente `robot1`. Ya que el agente `robot1` no tiene visibilidad sobre alguna de las metas del problema, como por ejemplo el *fluent* `(= (pos package3) E)`, es necesario que ambos agentes, `robot1` y `robot2`, colaboren en la consecución de dichos *fluents* y poder así conseguir todas las metas del problema. La metas que el agente `robot1` puede alcanzar por sí mismo son: `(= (pos package1) A)`, `(= (pos package2) D)` y `(= (at robot1) B)`.

La figura 2.6 muestra el fichero del problema del agente `robot2`.

```
(define (problem problema1)
(:domain Robot)
(:objects
     robot1 robot2 - robot
     package1 package3- box
     C D E F - place)

(:shared-data
  ((pos ?x - box) - place)
  (at ?x - robot)  - place) - robot2)

(:init
     (my-agent robot2)
     (connected C D) (connected D C)
     (connected D E) (connected E D)
     (connected E F) (connected F E)
     (= (pos package1) C)
     (= (pos package3) C)
     (= (at robot1) C)
     (= (at robot2) D))

(:global-goal
     (and
```

```
        (= (pos package1) A)
        (= (pos package2) D)
        (= (pos package3) E)
        (= (at robot1) B)
        (= (at robot2) F))))
```

Figura 2.6: Fichero del problema del agente `robot2`

En la figura 2.6 se puede observar como el problema, sección `problem`, es el mismo que el del agente `robot1`. Los dos agentes también comparten dominio, indicado en la sección `:domain` de la figura 2.6.

En la sección `:objects` de la figura 2.6 se muestra que el agente `robot2` no tiene conocimiento de la existencia las áreas `A` y B ni del paquete `package2`, razón por la cual no figuran en esta sección.

La sección `:init` de la figura 2.6 muestra información similar a la de la figura 2.5. Sin embargo, en este caso se omite la información sobre las áreas `A` y `B` y el paquete `package2`, ya que el agente `robot2` no tiene conocimiento de estos elementos. Además, se indica mediante el predicado `(my-agent robot2)` que el robot asignado a este agente es el `robot2`.

La sección `:shared-data` de la figura 2.6 muestra la misma información que en la figura 2.5, sólo que ahora será compartible con el agente `robot1`.

Dados los datos de las secciones anteriores, `:objects`, `:init` y `:shared-data`, se darán unos ejemplos de la privacidad del agente `robot2` con respecto al agente `robot1` sobre unos ciertos *fluents* <variable,valor>.

El *fluent* `(= (pos package1) D)` es un *fluent* público, ya que tanto el agente `robot1` como el agente `robot2` conocen la variable `(pos package1)` y su valor `D`. Por tanto, entre el agente `robot2`, el agente `robot1` y el *fluent* `(= (pos package1) D)` existe una relación de privacidad nula y toda información sobre él es compartible.

Un ejemplo privacidad parcial del agente `robot2` con respecto al agente `robot1` es el *fluent* `(= (pos package3) F)`. Este *fluent* es conseguido por el agente robot2, por lo que el agente robot2 conoce tanto la variable como el valor del *fluent*. Sin embargo, el agente `robot1`, aunque conoce la variable, desconoce del valor de este *fluent*, ya que no conoce la existencia del área `F`. En este caso, el agente `robot2` sólo puede compartir con el agente `robot1` la siguiente información sobre el *fluent*: `(= (pos package3)` $\perp$`)`, siendo $\perp$ un

valor indefinido.

Por último, las metas de la sección `:global-goal` de la figura 2.6 son idénticas a las de la figura 2.5. La metas que el agente `robot2` puede alcanzar por sí mismo en este problema son: `(= (pos package3) E)` y `(= (at robot2) F)`. Para conseguir el resto de metas, el agente `robot2` tendrá que coordinarse con el agente `robot1` para alcanzar los *fluents* no alcanzables por sí mismo.

## 2.4. Heurísticas en planificación

Las heurísticas de planificación son una parte importante del proceso de resolución, ya que son las funciones que permiten orientar la búsqueda del planificador hacia los objetivos. Existen múltiples trabajos e investigaciones en planificación heurística donde se han desarrollado numerosas y diferentes heurísticas con diferentes propiedades (monotonicidad, optimalidad, etc.).

### 2.4.1. Landmarks

Este trabajo se centra en las heurísticas basadas en *landmarks*. Un *landmark* es una variable con un valor determinado, *fluent*, que ha de cumplirse en todo plan solución de un problema. Existen diferentes tipos de *landmarks*, aunque los utilizados habitualmente para el diseño de heurísticas son *landmarks* calculados a partir de estados de planificación. Por ese motivo, las heurísticas basadas en *landmarks* se han utilizado frecuentemente en planificadores basados en estados. No obstante, también pueden utilizarse en otro tipo de planificadores como, por ejemplo, *planificadores de orden parcial (POP)* donde se calculan estados únicamente para la aplicación de heurísticas.

Unos de los trabajos más relevantes sobre el cálculo y extracción de *landmarks* es la tesis doctoral [SO03] de la investigadora del DSIC Laura Sebatiá Tarín, y el trabajo *Ordered Landmarks in Planning* publicado en la revista JAIR[1] donde ella misma participó [HPS04].

Un *landmark* es un *fluent* que debe aparecer en todos los planes solución de una tarea de planificación. En los citados trabajos, se muestra un algo-

[1] http://www.aaai.org/Press/Journals/jairvol22.php

ritmo para el cálculo y la extracción de *landmarks*. Dicho algoritmo extrae todos los *landmarks* de un problema mediante una serie de validaciones, y establece un orden entre los *landmarks* extraídos. Más adelante, en la sección 4.2 del capítulo 4, se explicará con profundidad este algoritmo.

Dado que un *landmark* es un *fluent* que tiene que aparecer en cualquier plan solución de un problema de planificación, en el caso que alguno de los *landmarks* extraídos no aparezca, no se podrían alcanzar todas las metas del problema. Además, por definición, todos los *fluents* meta e iniciales son *landmarks*. A continuación se presentará un ejemplo para entender mejor el concepto de *landmark*.

El dominio de este problema *problema2* es el dominio Robot, visto anteriormente en la sección 2.1. Este problema consta de una serie de áreas delimitadas: `A, B, C, D, E` y `F`, un agente `Robot` y un paquete `p1`. En este problema se puede transitar de un área a otra, sí y solo sí, se sigue el sentido de las fechas dibujadas en la figura 2.7. El paquete `p1` puede ubicarse en alguna de las áreas nombradas anteriormente, o puede ser sujetado por el robot. El agente `Robot` será el único agente de este ejemplo, el cual puede realizar el conjunto de acciones vistas anteriormente:

- **go-to:** Desplazarse de un área a otra.
- **pickup:** Recoger un paquete de un área.
- **drop:** Depositar un paquete en un área.

A continuación se mostrarán visualmente los estados inicial y final del problema *problema2*. El estado inicial del problema *problema2* se muestra en la figura 2.7.

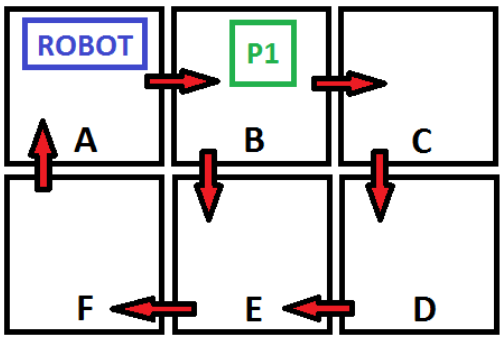


Figura 2.7: Estado inicial del *problema2* del dominio Robot

La figura 2.8 muestra el estado final de problema *problema2* con los objetivos a alcanzar.

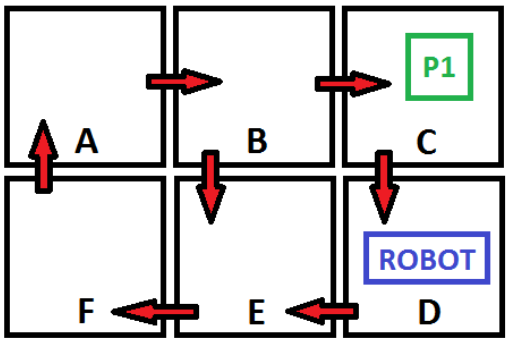


Figura 2.8: Estado final del *problema2* del dominio Robot

Un posible plan solución para este problema sería la siguiente secuencia de acciones:

- `go-to Robot A B`
- `go-to Robot B E`
- `go-to Robot E F`
- `go-to Robot F A`
- `go-to Robot A B`
- `pick-up Robot P1 B`
- `go-to Robot B C`
- `drop Robot P1 C`
- `go-to Robot C D`

Otro posible plan constaría de las siguientes acciones:

- `go-to Robot A B`
- `pick-up Robot P1 B`
- `go-to Robot B C`

- `drop Robot P1 C`
- `go-to Robot C D`

Considerando los planes anteriores, se puede observar cómo la eliminación de las acciones ***go-to Robot B E*** y ***go-to Robot E F*** no afecta a que un plan siga siendo un plan solución. El motivo de que se puedan eliminar estas acciones es porque los efectos de dichas acciones, `(= (at robot) E)` y `(= (at robot) F)`, no son *landmarks*. Conseguir o no estos efectos no influye en que el plan sea una solución para el problema. Dicho de otro modo, conseguir o no estos efectos no influye en que el plan pueda alcanzar todas las metas del problema *problema2*.

En este problema, se puede identificar el siguiente conjunto de *landmarks* en el estado inicial y las metas:

- `(= (pos p1) B)`
- `(= (at Robot) A)`
- `(= (pos p1) C)`
- `(= (at Robot) D)`

Además, se observa otro conjunto de *landmarks* que no están ni en el estado inicial ni en las metas. Sin estos *fluents* no se conseguirían las metas:

- `(= (at Robot) B)`
- `(= (at Robot) C)`
- `(= (pos p1) Robot)`

De este modo, pueden existir más *landmarks* que los iniciales y finales, siendo estos tan importantes como los anteriores. Si alguno de los *fluents* de los conjuntos de *landmarks* citados no se encuentra en un plan, ese plan nunca será un plan solución para el problema *problema2*, ya que es imposible que dicho plan contenga las metas a alcanzar.

### 2.4.2. Heurística del planificador LAMA

LAMA[RW10] es un planificador clásico de búsqueda hacia delante basado en el sistema *Fast Downward* [Hel06]. LAMA hereda de este planificador su estructura general y gran parte de su funcionalidad y, al igual que el sistema *Fast Downward*, acepta problemas de planificación en el formato PDDL [GHK+98], más concretamente en la versión PDDL2.2. Además, LAMA se amplió para el IPC 2008 [HDR08] y desde entonces puede manejar acciones con un coste asociado.

El proceso de búsqueda en LAMA está implementado por dos algoritmos. El primero, implementa una búsqueda avariciosa de "primero el mejor" que intenta encontrar una solución lo más rápido posible, y el segundo, implementa un búsqueda A* ponderada que permite ajustar la velocidad de la búsqueda con la calidad de la solución que se desea encontrar. Ambos algoritmos siguen los estándares de los algoritmos de búsqueda, usando listas de nodos abiertos y cerrados.

En el primer algoritmo, búsqueda "primero el mejor", se expande siempre un sólo estado en cada iteración. Este estado es el que tiene el menor valor heurístico $h$ de entre todos los estados de la lista de nodos abiertos. Para mejorar la relación "coste-eficiencia" del plan solución, en caso de un empate heurístico, se expandirá el estado alcanzado con menor coste en operaciones. La búsqueda A* ponderada asocia los costes con los estados y expande el estado con menor valor $f$, donde $f = w * h + g$. El valor $w$ es el peso que se le da al valor de la heurística $h$. Una restricción exige que el valor $w$ sea siempre $w \geq 1$. El valor $g$ es el mejor coste para llegar desde el estado inicial al estado considerado.

LAMA usa dos funciones heurísticas para guiar el proceso de búsqueda: la heurística de *landmarks* y una variante de la heurística FF[HN01]. Ambas heurísticas utilizan un cola propia, explotando así los puntos fuertes de cada una de ellas. Se evalúan todos los estados de la lista de nodos abiertos con las dos heurísticas, independientemente de los resultados dados por cada una. A la hora de elegir un estado para evaluarlo y expandirlo el algoritmo de búsqueda alterna entre las dos colas, basándose en prioridades númericamente asignadas a cada cola.

En LAMA los nodos son añadidos a la lista de nodos abiertos con el valor heurístico heredado de su nodo padre. Sólo cuando estos nodos hijo son seleccionados y eliminados de la lista de nodos abiertos, se les evalúa y se les

asigna su verdadero valor heurístico. Con este nuevo valor serán añadidos los nodos hijo del nodo evaluado a la lista de nodos abiertos.

La heurística de FF está basada en una relajación de la tarea de planificación, en la que se ignoran los efectos de eliminación. Esta relajación implica que las variables estado pueden tener varios valores diferentes al mismo tiempo. El valor heurístico de FF se calcula en dos fases. En una primera fase "hacia delante", se calcula una estimación del coste de conseguir cada meta de la tarea de planificación desde el estado $s$ a evaluar en una tarea de planificación relajada. En esta fase se calculan los operadores con menor coste, llamados *best support*, para producir cada hecho. En la segunda fase, se construye un plan para la tarea relajada en el que para conseguir cada hecho se utiliza el operador con menor coste de dicho hecho para conseguirlo. Este plan se logra mediante un mecanismo "hacia atrás", dicho mecanismo comienza en las metas y llega hasta el estado $s$ a evaluar. La longitud del plan relajado construido es el valor heurístico calculado por FF para cada estado a evaluar.

La heurística de *landmarks* cálcula el valor heurístico de la siguiente forma. En cada estado se calcula cuantos *landmarks* se han conseguido de los totales del problema a resolver y también los *landmarks* que, aunque han sido conseguidos alguna vez, se requiere que sean conseguidos al menos una vez más en el problema. En la mecánica del conteo de *landmarks*, a cada estado, antes de ser evaluado, se le asociada el valor de *landmarks* conseguidos por su nodo padre. Cuando se evalúa un nodo, se vuelve a calcular para dicho estado cuantos *landmarks* quedan por conseguir, así como los *landmarks* ya alcanzados que se requiere que sean conseguidos al menos una vez más. El valor heurístico asignado a cada nodo evaluado es el coste estimado de las acciones necesarias para conseguir los *landmarks* aún no alcanzados y los *landmarks* que han de ser conseguidos al menos una vez más.

El funcionamiento de la heurística de LAMA es el siguiente. Primero, evalúa los planes utilizando la heurística de FF. Cuando la aplicación de la heurística FF devuelve el mismo valor heurístico para varios planes, utiliza la heurística de *landmarks* para desempatar y elegir uno entre ellos. Con la estimación obtenida a través de los *landmarks*, LAMA selecciona los estados que han conseguido más *landmarks* que los demás, ya que dichos estados están más cercanos a ser plan solución. Por ejemplo, si hubiese que desempatar entre dos planes, *p1* y *p2*, que han alcanzado cinco y siete *landmarks* respectivamente, LAMA elegiría el plan *p2*. LAMA priorizaría el plan *p2* porque es el plan más cercano a alcanzar todos los *landmarks* de un problema, y por lo tanto, más cercano de ser un plan solución del problema a considerar.

# Capítulo 3

# Entorno de planificación multiagente

Los principales componentes que se han utilizado para el desarrollo de este trabajo son el planificador multiagente FMAP y la plataforma multiagente PlanInteraction, actualmente en desarrollo por el grupo de planificación del DSIC[1] (GRPS-AI)[2]. En concreto, este trabajo ha empleado el planificador FMAP integrado en la plataforma PlanInteraction como punto de partida para la implementación.

A continuación se presentará la formalización de varios conceptos de planificación multiagente y se explicará con más detalle el planificador FMAP, la plataforma PlanInteraction y la versión integrada de la que parte este trabajo. Para denotar las acciones, metas, etc, de un agente $i$ se usará la notación de superíndice $x^i$ para cada aspecto $x$.

## 3.1. Formalización

A continuación se define el concepto de tarea de planificación multiagente que extiende de la definición de "Tarea de planificación monoagente" vista en la sección 2.2. Además, se formaliza el concepto de "acción", "plan de orden parcial" y "plan solución".

**Definición 3.** *(**Tarea de planificación multiagente**) Una **tarea de planificación multiagente** es una tupla* $\mathcal{T} = \langle \mathcal{AG}, \mathcal{O}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$.

[1] `https://www.upv.es/entidades/DSIC/index.html`
[2] `http://users.dsic.upv.es/grupos/grps/`

- $\mathcal{AG} = \{1, \ldots, n\}$ *es un conjunto finito no vacío de agentes de planificación.*
- $\mathcal{V} = \cup_{i \in \mathcal{AG}} \mathcal{V}^i$*, donde* $\mathcal{V}^i$ *es el conjunto de variables estado conocidas por el agente* $i$*.*
- $\mathcal{O}$ *es un conjunto finito de objetos. Dichos objetos modelan los elementos del dominio de planificación sobre los que actúan las acciones de planificación.*
- $\mathcal{A} = \cup_{i \in \mathcal{AG}} \mathcal{A}^i$ *es el conjunto de acciones deterministas de los agentes, donde* $\mathcal{A}^i$ *es el conjunto de acciones realizables por el agente* $i$*.*
- $\mathcal{I} = \cup_{i \in \mathcal{AG}} \mathcal{I}^i$ *es el conjunto de fluents que representan el estado inicial de la tarea de planificación multiagente* $\mathcal{T}$*.* $\mathcal{I}^i$ *es el conjunto de fluents del estado inicial de los cuales el agente* $i$ *tiene conocimiento.*
- $\mathcal{G}$ *es el conjunto de metas comunes de la tarea de planificación multiagente que los agentes deben satisfacer;* $\mathcal{G}$ *representa los valores que las variables estado deben adquirir en el estado final.*

Cada estado del problema se define mediante un conjunto de *fluents*. Dichos conjuntos indican la situación en la que se encuentran, para ese estado, los elementos del problema. Una de las funcionalidades de FMAP es que puede trabajar con *fluents* positivos y negativos. Un *fluent* negativo toma la forma $\langle v, \neg d \rangle$, mientras que *fluent* positivo tiene la forma de $\langle v, d \rangle$. Un *fluent* positivo indica que la variable $v$ toma el valor $d$, mientras que un *fluent* negativo $\langle v, \neg d \rangle$ indica que la variable $v$ no toma el valor $d$. Por tanto, el modelo de representación de FMAP adopta la asunción de mundo abierto, considerando que la información que no está explícitamente almacenada en el modelo interno de los agentes es desconocida para ellos.

La información del estado inicial $\mathcal{I}$ se modela, por tanto, mediante un conjunto de *fluents* positivos y negativos. Esta información está distribuida entre los agentes bajo la asunción de que el conocimiento parcial de los agentes sobre $\mathcal{I}$ es consistente, es decir, no hay información contradictoria entre los mismos. Una tarea de planificación multiagente se define de modo que todos los agentes tienen una visión completa del estado inicial $\mathcal{I}$, es decir, $\forall i \in \mathcal{AG},\ \mathcal{I}^i = \mathcal{I}$.

**Definición 4.** *(**Acción**) Una **acción** $\alpha \in \mathcal{A}$ es una tupla $\langle PRE(\alpha), EFF(\alpha) \rangle$.* $PRE(\alpha) = \{p_1, \ldots, p_n\}$ *es un conjunto de condiciones de la forma* $< v, d >$ *o*

$< v, \neg d >$ *que representan las precondiciones de* $\alpha$*, mientras que* $EFF(\alpha) = \{e_1, \ldots, e_m\}$ *es un conjunto de operaciones de la forma* $(v = d)$ *o* $(v \neq d)$*,* $v \in \mathcal{V}$*,* $d \in \mathcal{D}_v$*, que representan las consecuencias de ejecutar* $\alpha$*.*

El conjunto de precondiciones de una acción $\alpha$, $PRE(\alpha)$, representa los *fluents* que deben figurar en un estado $S$ para que $\alpha$ sea aplicable en dicho estado. Una precondición positiva de la forma $\langle v, d\rangle$ indica que la variable $v$ debe tener el valor $d$ en $S$, mientras que una precondición negativa $\langle v, \neg d\rangle$ indica que la variable $v$ no debe tener el valor $d$ en el estado $S$. Nótese que la existencia de un *fluent* positivo $\langle v, d\rangle$ implica también la existencia de un *fluent* negativo $\langle v, \neg d'\rangle$ para el resto de valores en el dominio de la variable, es decir, $(\exists \langle v, d\rangle \in S) \Rightarrow (\forall d' \in \mathcal{D}_v,\, d' \neq d,\, \exists \langle v, \neg d'\rangle \in S)$.

El resultado de ejecutar $\alpha$ en un estado $S$ es un nuevo estado del mundo $S'$ que surge tras la aplicación de $EFF(\alpha)$ en $S$, es decir, $S'$ se genera actualizando las variables instanciadas en $S$ de acuerdo a los efectos de $\alpha$:

- Una operación $(v = d) \in EFF(\alpha)$ implica la adición de un *fluent* $\langle v, d\rangle$ y un conjunto de *fluents* $\langle v, \neg d'\rangle$, $\forall d' \in \mathcal{D}_v \mid d' \neq d$ al estado $S'$. Si $\langle v, d'\rangle \in S$ o bien $\langle v, \neg d\rangle \in S$, $d' \neq d$, la operación $(v = d)$ implica que los *fluents* $\langle v, \neg d\rangle$ y $\langle v, d'\rangle$ no se añadirán a $S'$.
- Una operación $(v \neq d) \in EFF(\alpha)$ implica la adición de un *fluent* $\langle v, \neg d\rangle$ al estado $S'$. Si $\langle v, d\rangle \in S$, la operación $(v \neq d)$ implica que el *fluent* $\langle v, d\rangle$ no se añadirá a $S'$. Nótese que la sola existencia de una variable instanciada $\langle v, \neg d\rangle$ en un estado $S$ indica que el valor de la variable $v$ es desconocido en $S$.

Una acción $\alpha$ es pública si dos o más agentes la comparten, esto es, $\alpha \in A^i \wedge \alpha \in A^j$, $i \neq j$. $\alpha \in \mathcal{A}^i$ es privada para un agente $i$ si y sólo si $\alpha \notin \mathcal{A}^j, \forall j \neq i$. Una acción $\alpha \in \mathcal{A}^i$ denota que el agente $i$ posee la capacidad expresada en $\alpha$. Si $\alpha$ forma parte de un plan solución, el agente $i$ es también responsable de ejecutar $\alpha$.

**Definición 5.** *(**Plan de orden parcial**) Un* ***plan de orden parcial o plan parcial*** *es una tupla* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL}\rangle$*.* $\Delta \subseteq \mathcal{A}$ *es el conjunto de acciones en* $\Pi$*.* $\mathcal{OR}$ *es un conjunto de restricciones de orden* $(\prec)$ *en* $\Delta$*.* $\mathcal{CL}$ *es un conjunto de enlaces causales sobre* $\Delta$*. Un enlace causal toma la forma* $\alpha \overset{\langle v,d\rangle}{\rightarrow} \beta$ *o bien* $\alpha \overset{\langle v,\neg d\rangle}{\rightarrow} \beta$*, donde* $\alpha \in \mathcal{A}$ *y* $\beta \in \mathcal{A}$ *son acciones en* $\Delta$*.* $\alpha \overset{\langle v,d\rangle}{\rightarrow} \beta$ *indica que hay una operación* $(v = d)$ *tal que* $v \in \mathcal{V}$*,* $d \in \mathcal{D}_v$*,* $(v = d) \in EFF(\alpha)$ *y una precondición* $\langle v, d\rangle \in PRE(\beta)$*.* $\alpha \overset{\langle v,\neg d\rangle}{\rightarrow} \beta$ *indica*

*que la precondición* $\langle v, \neg d \rangle$ *tal que* $v \in \mathcal{V}$, $d \in \mathcal{D}_v$, $\langle v, \neg d \rangle \in PRE(\beta)$ *está soportada por una operación* $(v \neq d) \in EFF(\alpha)$ *o una operación* $(v = d') \in EFF(\alpha)$, $d' \in \mathcal{D}_v$, $d' \neq d$.

Esta definición de plan parcial muestra que un plan puede verse como un grafo dirigido acíclico, donde $\Delta$ representa los nodos del grafo (acciones) y $\mathcal{OR}$ y $\mathcal{CL}$ son conjuntos de aristas dirigidas que representan las precedencias y enlaces causales entre acciones, respectivamente.

Un plan parcial *vacío* se define como $\Pi_0 = \langle \Delta_0, \mathcal{OR}_0, \mathcal{CL}_0 \rangle$, donde $\Delta_0$ contiene $\alpha_0$ y $\alpha_f$, las acciones inicial y final del plan, respectivamente. $\alpha_0$ y $\alpha_f$ son acciones ficticias que no pertenecen al conjunto de acciones de ningún agente. $\mathcal{OR}_0$ contiene la restricción de orden $\alpha_0 \prec \alpha_f$ y $\mathcal{CL}_0$ es un conjunto vacío. De este modo, un plan $\Pi$ para una tarea de planificación multiagente $\mathcal{T}$ contendrá siempre las dos acciones ficticias, de forma que $PRE(\alpha_0) = \emptyset$, $EFF(\alpha_0) = \mathcal{I}$, $PRE(\alpha_f) = \mathcal{G}$, y $EFF(\alpha_f) = \emptyset$; es decir, $\alpha_0$ representa la situación inicial de la tarea de planificación multiagente $\mathcal{T}$, y $\alpha_f$ representa las metas globales de $\mathcal{T}$.

Asumiendo que $\mathcal{G} \neq \emptyset$, un plan no vacío es incompleto si al menos una precondición de sus acciones no está soportada aún por un enlace causal.

**Definición 6.** *(**Plan solución**) Un plan* $\Pi$ *es un* ***plan solución*** *para una tarea de planificación multiagente* $\mathcal{T}$ *si soporta todas las precondiciones de la acción final* $\alpha_f$*, es decir, si resuelve todas las metas de la tarea (*$\Pi$ *es un plan completo).*

Nótese que requerimos que $\Pi$ sea un plan completo, de modo que no puede quedar ninguna precondición de las acciones de dicho plan pendiente. En consecuencia, las precondiciones de la acción ficticia final $\alpha_f$ estarán también resueltas, lo que garantiza que $\Pi$ resuelve la tarea de planificación multiagente $\mathcal{T}$.

## 3.2. FMAP

FMAP es un sistema de planificación multiagente cooperativo en el que un conjunto de agentes de planificación diseña, de forma coordinada, un curso de acción o plan para alcanzar un conjunto de objetivos o metas a partir de una situación inicial dada.

FMAP está basado en técnicas de planificación de orden parcial (POP) [PW92]. A diferencia del paradigma clásico POP, que plantea la construcción de los planes de forma regresiva a partir de las metas, FMAP realiza una búsqueda progresiva, de modo similar a los planificadores basados en estados, manteniendo al mismo tiempo la filosofía de menor compromiso que caracteriza a los planificadores de orden parcial.

El manejo de información privada en FMAP adopta la definición descrita en la sección 2.3.1.1. Además, en FMAP todo *fluent* $l$ tiene una lista de agentes productores $\mathcal{P}_l$ asociada, dicha lista indica los agentes capaces de conseguir el *fluent* $l$ tal que $\mathcal{P}_l \subseteq \mathcal{AG}$ .

### 3.2.1. Etapas principales en FMAP para resolver un problema de planificación

Antes de empezar la ejecución, FMAP crea todos los agentes que van a intervenir en la tarea de planificación a resolver. Una vez creados los agentes, cada uno sigue las tres etapas en las que se divide FMAP. En primer lugar se ejecuta la etapa *Parsing*, en segundo lugar la etapa *Grounding* y por último la etapa de *Planificación*. A continuación se explican las tres etapas principales y la relación existente entre cada una de ellas.

#### 3.2.1.1. *Parsing*

En esta fase, cada agente recibe como entrada su fichero de dominio y su fichero del problema. Todos los agentes ejecutan esta etapa sin interacción con los demás.

En esta etapa cada agente lee los ficheros y guarda la información contenida en dichos ficheros en memoria. De esta manera el agente almacena los predicados, variables $\mathcal{V}$, funciones del dominio, objetos del problema $\mathcal{O}$, estado inicial $\mathcal{I}$ y metas $\mathcal{G}$ en memoria .

#### 3.2.1.2. *Grounding*

En esta etapa cada agente recibe como entrada su dominio y su problema procesado. Para obtener todas las acciones $\mathcal{A}^i$ posibles del problema, el agente combina los objetos del problema $\mathcal{O}$ que hacen matching con los parámetros de cada operador de acción del dominio.

El agente realiza también un análisis de alcanzabilidad para verificar las acciones que son alcanzables desde su estado inicial $\mathcal{I}^i$ del problema. Es en este punto donde se construye un grafo de planificación relajado (RPG) [ZNK07] distribuido.

Para la construcción del RPG distribuido (disRPG) son necesarias las comunicaciones entre todos los agentes. En estas comunicaciones los agentes se intercambian la información compartible con el agente correspondiente de acuerdo a la privacidad existente entre los dos agentes y el *fluent* en cuestión a compartir.

La salida de este procedimiento es la tarea $T^i$ con los elementos instanciados correspondientes al problema.

#### 3.2.1.3. *Planificación*

La entrada que los agentes utilizan en esta etapa es la tarea $\mathcal{T}^i$ de la fase anterior. Los componentes de esta tarea $\mathcal{T}^i$ serán: $\mathcal{AG}$, $\mathcal{O}$,$\mathcal{V}^i$,$\mathcal{A}^i$, $\mathcal{I}^i$, $\mathcal{G}$.

Antes de empezar la planificación todos los agentes crean un "plan base" inicial. Se trata de un plan vacío que contiene una acción ficticia inicial $\alpha_i$ y una acción ficticia $\alpha_j$. La acción $\alpha_i$ tiene como efectos todos los *fluents* iniciales $\mathcal{I}$ de la tarea mientras que la acción $\alpha_j$ tiene como precondiciones todas las metas $\mathcal{G}$ de la tarea.

Partiendo del plan base inicial, FMAP realiza un proceso de búsqueda en un espacio de planes. De esta manera, los agentes participantes construyen de forma cooperativa un árbol de búsqueda donde cada nodo refina a su nodo padre. Cada nodo del árbol representa un plan de orden parcial que puede contener información proporcionada por uno o varios agentes.

El nodo raíz del árbol de búsqueda es el plan vacío. Desde el nodo raíz se comienza un proceso de búsqueda en el que en cada iteración se expande el nodo mejor evaluado, según una función de evaluación heurística. Al expandir un nodo $\Pi_p$ cada agente refina el plan de dicho nodo, añadiendo un nodo hijo $\Pi_h$ que añade una nueva acción $a$ sobre el plan del nodo padre. De esta forma, el plan del nodo hijo $\Pi_h$ será $\Pi_h = \Pi_p \cup a$. La acción $a$ puede añadirse en $\Pi_p$ si sus precondiciones $PRE(a)$ se soportan con la información contenida en $\Pi_p$, y la inserción de $a$ no genera ningún conflicto en $\Pi_h$. Cada agente añadirá tantos nodos hijos como acciones $a$ distintas satisfagan todas sus precondiciones en el plan del nodo padre $\Pi_n$.

Este proceso finaliza cuando se encuentra un plan solución que consiga todas las metas $\mathcal{G}$ del problema.

### 3.2.2. Función de evaluación heurística

FMAP aplica una estrategia de búsqueda heurística para explorar el árbol de planes multiagente. Cada plan $\Pi$ del árbol se evalúa mediante una función de utilidad $f(\Pi) = g(\Pi) + h(\Pi)$, donde $g(\Pi)$ mide el coste del plan $\Pi$ como el número de acciones de de $\Pi$, mientras que $h(\Pi)$ estima el coste de alcanzar un plan solución desde $\Pi$. FMAP realiza estas estimaciones mediante una novedosa función heurística basada en Grafos de Transición de Dominio (DTG).

Para evaluar cada plan $\Pi$, FMAP calcula el estado frontera asociado a dicho plan, $FS(\Pi)$. $FS(\Pi)$ es el estado que se alcanzaría después de ejecutar, desde el estado inical $\mathcal{I}$, todas las acciones que contiene el plan $\Pi$. Desde ese estado frontera, $FS(\Pi)$, la heurística utilizada por FMAP primero calcula el $DTG_v$ para cada variable $v$ involucrada en $\mathcal{G}$. Después, a partir de cada $DTG_v$, se obtiene el camino de coste mínimo desde $< v,d >$ en el estado frontera $FS(\Pi)$ hasta $< v,d' >$ en el estado final o meta del problema.

De esta manera, el valor heurístico, $h_{DTG}$, que devuelve la función de evaluación heurística basada de DTGs para un plan $\Pi$, será la suma de los costes mínimos calculados anteriormente de cada variable $v$ involucrada en las metas $\mathcal{G}$.

## 3.3. PlanInteraction

PlanInteraction [3] es un trabajo coordinado entre las universidades UPV [4], UC3M [5] y la UGR [6]. El interés del proyecto PlanInteraction es desarollar nuevas técnicas y tecnologías basadas en dinámicas sociales para el diseño e implementación de una plataforma de planificación multiagente compuesta de entidades de planificación autónomas.

[3] `http://servergrps.dsic.upv.es/planinteraction/`
[4] Universidad Politécnica de Valencia
[5] Universidad Carlos III de Madrid
[6] Universidad de Granada

El diseño de PlanInteraction, desde la visión más general, cuenta con un entorno en el que coexisten un conjunto de agentes con capacidades de planificación y/o ejecución, una serie de elementos de control, un simulador y una interfaz de interacción con el mundo real. Esta arquitectura permite crear múltiples configuraciones según el tipo de problema que se vaya a tratar.

PlanInteraction dispone de agentes genéricos propios (GA). Estos GA estan desarrollados sobre los agentes conversacionales (CAgents) proporcionados por la plataforma Magentix2[7]. Los CAgents implementan una serie de mecanismos de comunicación tales como protocolos, recepción y envío de mensajes y gestión de múltiples conversaciones simultáneas.

Respecto al almacenamiento interno, cada GA dispone de una memoria interna. Esta memoria almacena toda aquella información que el agente percibe del entorno, ya sea mediante sensorización o por comunicación con otros agentes. También puede almacenarse información auxiliar relativa a procesos internos del agente.

### 3.3.1. Comunicaciones

La especificación y control de las comunicaciones entre agentes viene heredado de los mecanismos proporcionados por Magentix2 [8]. El broker de comunicaciones utilizado es Apache Qpid y los mensajes intercambiados entre los agentes siguen el estándar FIPA-ACL de la tabla 3.1 [9].

| Headers | Meta-info |
|---|---|
| ConvId | Identificador de la conversación |
| Performative | Performativa del mensaje |
| Language | Lenguaje del mensaje |
| Ontology | Ontología del mensaje |
| Protocol | Protocolo del mensaje |
| Sender | Identificador del agente emisor del mensaje |
| Receivers | Lista de identificadores de los receptores |
| Content | Contenido del mensaje |

Tabla 3.1: Formato de mensaje FIPA-ACL

[7] `http://www.gti-ia.upv.es/sma/tools/magentix2/index.php`
[8] `http://www.gti-ia.upv.es/sma/tools/magentix2/`
[9] `http://www.fipa.org/repository/aclspecs.html`

El broker de comunicaciones establece una dirección única para cada agente, utilizando el identificador de éste. Por este motivo, es necesario que cada agente tenga un identificador único, para poder identificar a todos los agentes de manera inequívoca. Los mensajes recibidos por el broker son redirigidos a la cola de mensajes del receptor correspondiente, donde cada agente procesará los mensajes de su cola de forma secuencial y asíncrona.

### 3.3.2. Comportamientos

Un comportamiento es una rutina bien definida con unos objetivos determinados. Los GA disponen de una memoria que almacena todos los comportamientos que puede realizar el agente. La arquitectura PlanInteraction proporciona unas APIs de definición de comportamientos como grafos de estados. Los tipos de estados que se emplean para definir los comportamientos se muestran en la tabla 3.2:

| | |
|---|---|
| BEGIN | Estado inicial |
| FINAL | Estado final |
| ACTION | Estado de propósito general |
| WAIT | Estado de espera en la recepción de mensajes *(timeout)* |
| RECEIVE | Estado de recepción y manejo de mensajes |
| SEND | Estado de construcción y envío de mensajes |
| NAM | Estado de absorción de mensajes *(Not Accepted Message)* |

Tabla 3.2: Estados conversacionales de Magentix2

Se puede establecer un *timeout* para el estado WAIT, esto es, un tiempo de espera máximo en el que un agente se encuentra en ese estado para recibir un mensaje. Así mismo cabe destacar la particularidad de la recepción de un mensaje. Para poder recibir y tratar un mensaje, un agente ha de encontrarse necesariamente en un estado WAIT y transitar a un estado RECEIVE. Será en el estado RECEIVE donde el agente pueda tratar el mensaje recibido.

Desde cualquier estado de cualquier comportamiento se puede acceder a la memoria interna del agente, así como a las funcionalidades que proporcionan los módulos de planificación/ejecución.

La ejecución de un determinado comportamiento puede estar motivado por una decisión propia del agente o por una reacción ante una interacción

con otros agentes. De esta forma, se puede distinguir entre *comportamientos proactivos* y *comportamientos reactivos*.

Los GA disponen de una cola interna donde se añaden los comportamientos proactivos que se van a ejecutar. Los métodos de razonamiento del agente son los que determinan el orden en el que se tienen que ejecutar. Si el agente no implementa una lógica de decisión para la selección de comportamientos, la cola interna funciona según la técnica *First in, First out*. Una alternativa a este funcionamiento sería gestionar la cola interna conforme a un orden de prioridades preestablecido. Otra decisión a tener en cuenta en el diseño de un GA es la acción que se debe realizar cuando la cola interna queda vacía.

Los comportamientos reactivos se ejecutan de forma asíncrona bajo petición expresa de otros agentes. Normalmente son peticiones de información o realización de procesos a modo de servicios. La ejecución de un comportamiento reactivo puede provocar la incorporación de un comportamiento proactivo a la cola de comportamientos.

En cuanto a la ejecución ordenada de métodos, funciones o subtareas multiagente se necesita una sincronización entre los agentes involucrados para su realización. Para llevar a cabo esta sincronización, PlanInteraction utiliza unos mecanismos para garantizar que la sincronización entre agentes sea la adecuada. En estos casos, uno de los agentes asumirá el rol de coordinador, encargándose de liderar el proceso. Este rol rotará entre los agentes si es necesario para la realización de las tareas multiagente. En el siguiente capítulo 4 se verán ejemplos de procesos con un agente coordinador y agentes participantes.

## 3.4. Integración de FMAP en PlanInteraction

FMAP está integrado en la plataforma PlanInteraction. Así pues, los mensajes que utiliza el planificador serán los mensajes que proporciona dicha plataforma. En PlanInteraction, los mensajes quedan encapsulados en un método que permite indicar tanto el nombre del agente que envía el mensaje como el tipo de mensaje, nombre o nombres de los agentes a los que va dirigido el mensaje, contenido del mensaje y el nodo de tipo RECEIVE al que irá (o irán) los agentes al recibir dicho mensaje.

### 3.4.1. Modelización de las etapas principales

Cada una de las etapas de FMAP se modela como un comportamiento proactivo en PlanInteraction. De esta manera, en la cola interna de comportamientos de cada agente se añaden los comportamientos resultantes en el mismo orden en el que se ejecutan en FMAP: *Parsing*,*Grounding* y *Planificación*.

#### 3.4.1.1. *Comportamiento Parsing*

Este comportamiento modeliza la etapa de *Parsing* descrita en la sección 3.2.1.1. Como no es necesaria comunicación entre los agentes para el comportamiento de *Parsing*, cada agente la hace por separado.

Una vez acabado el comportamiento cada agente guarda en su memoria interna su dominio y su problema procesado.

#### 3.4.1.2. *Comportamiento Grounding*

El siguiente comportamiento modeliza la etapa de *Grounding* descrita en la sección 3.2.1.2. Al principio de este comportamiento, cada agente recoge de su memoria interna su dominio y su problema procesado calculado en la etapa anterior.

En la primera parte de este comportamiento de *Grounding*, donde se obtienen todas las acciones $\mathcal{A}^i$ posibles para cada agente, no es necesaria la comunicación entre los agentes. Sin embargo, en la segunda parte, en la que se construye el grafo de planificación relajado [ZNK07] distribuido (disRPG), sí es necesaria la comunicación entre todos agentes. Para dicha construcción se emplea el proceso descrito en la sección 4.3.10.3.

Cuando este comportamiento termina, cada agente $i$ almacena en su memoria interna la tarea $\mathcal{T}^i$ con los valores instanciados correspondientes a su problema.

#### 3.4.1.3. *Comportamiento Planificación*

Este comportamiento modeliza la etapa de *planificación* descrita en la sección 3.2.1.3. En este comportamiento cada agente $i$ utiliza su tarea $\mathcal{T}^i$, la

cual está almacenada en su memoria interna.

En el comportamiento que modela la etapa de *planificación* se utiliza el paso de mensajes tanto en el refinamiento de planes como en el cálculo de la heurística. En ambos casos se utilizan métodos de sincronización para llevar a cabo una ejecución multiagente ordenada tanto del refinamiento de planes como del cálculo de la heurística. Dichos métodos se explicaran en profundidad en el capítulo siguiente.

# Capítulo 4

# Extracción multiagente de *landmarks*

Como se indicó en la sección 1.1, este trabajo se centra en el diseño e implementación de una versión distribuida del algoritmo de cálculo del grafo de *landmarks* [SO03]. Una vez obtenidos los grafos de *landmarks* para cada uno de los agentes participantes en el problema, se implementará una heurística multiagente que combine la información proporcionada por dichos grafos con la heurística existente en FMAP, con el objetivo de mejorar la calidad de las estimaciones y, por tanto, el rendimiento global del planificador.

En este capítulo se explicará en profundidad el diseño de la versión multiagente del algoritmo de extracción de *landmarks* presentado en la Tesis Doctoral de Laura Sebastián Tarín [SO03], su implementación en PlanInteraction y el diseño de la heurística implementada.

En dicho trabajo [SO03], se define el mecanismo para identificar los *landmarks* de un problema de planificación y establecer órdenes necesarios entre ellos, donde un orden necesario indica que un *landmark* debe aparecer necesariamente antes que otro en cualquier plan solución.

## 4.1. Formalización de los componentes en el proceso de extracción multiagente de *landmarks*

A continuación se formalizará el concepto de *landmark*, orden necesario y grafo de *landmarks*.

**Definición 7.** *(**Landmark**) Un fluent **l** es un **landmark**, si y sólo si **l** es cierto en algún punto en todos los planes solución de la tarea de planificación a resolver.*

Existen dos tipos de *landmarks*, disyuntivos y simples.

***Landmark* simple:** Un *landmark* simple es una variable con un valor determinado, *fluent*, que ha de cumplirse en todo plan solución de un problema. Si alguno de dichos *fluents* no apareciese en un plan, éste no sería un plan solución de la tarea. Además, todos los *fluents* del estado inicial $\mathcal{I}$ y de las metas $\mathcal{G}$ también son considerados *landmarks* simples.

***Landmark* disyuntivo:** Un *landmark* disyuntivo es un conjunto de variables con un valor determinado, *fluents*, de los cuales al menos uno ha de cumplirse en todo plan solución. Los *fluents* pertenecientes a este conjunto no tienen un rol de *landmarks* por sí mismos, pero sí como conjunto. Estos conjuntos de *fluents* suelen agruparse por variables.

Dada la definición de *landmark*, los agentes productores de un *landmark* disyuntivo serán todos aquellos agentes que puedan, al menos, producir uno de los *fluents* del conjunto que caracteriza a un *landmark* disyuntivo. En el caso de los *landmarks* simples, FMAP calcula los productores de ese *landmark* de igual manera que al *fluent* que representa. Además, las acciones productoras de un *landmark* disyuntivo serán aquellas que, al menos, tengan como efecto un *fluent* de los del conjunto de dicho *landmark* disyuntivo.

**Definición 8.** *(**Orden necesario**) Sean dos fluents **l** y **l'**. Un **orden necesario** entre **l** y **l'**, denotado como $\mathbf{l} \leq_n \mathbf{l'}$, implica que siempre que se consiga **l'** en el plan, **l** deberá ser cierto en el estado inmediatamente anterior.*

**Definición 9.** *(**Grafo de landmarks**) Se define **grafo de landmarks** como **LG = (N,E)**, donde N es el conjunto de landmarks de $\mathcal{T}$, y E es el conjunto de órdenes necesarios establecidos entre pares de landmarks.*

En el algoritmo de cálculo del grafo de *landmarks*[SO03], se utiliza una estructura llamada "*estructura de landmarks por niveles*" para su ejecución. A continuación se definirá dicha estructura para una mejor comprensión del algoritmo y su ejecución.

**Definición 10.** ***(Estructura de landmarks por niveles)***

*Se define una* ***estructura de landmarks por niveles (LL)*** *como una tabla donde cada landmark es insertado según el nivel mínimo* $\boldsymbol{n}$ *en el que es encontrado por un agente en su grafo del planificación relajado [ZNK07]. El nivel mínimo* $\boldsymbol{n}$ *de un* landmark $l$ *se define de la siguiente manera:*

$$n = \min^{a}(nivel(l)),\ a \in \mathcal{AG},\ l \in landmarks(\mathcal{T})$$

## 4.2. Algoritmo de extracción de *landmarks*

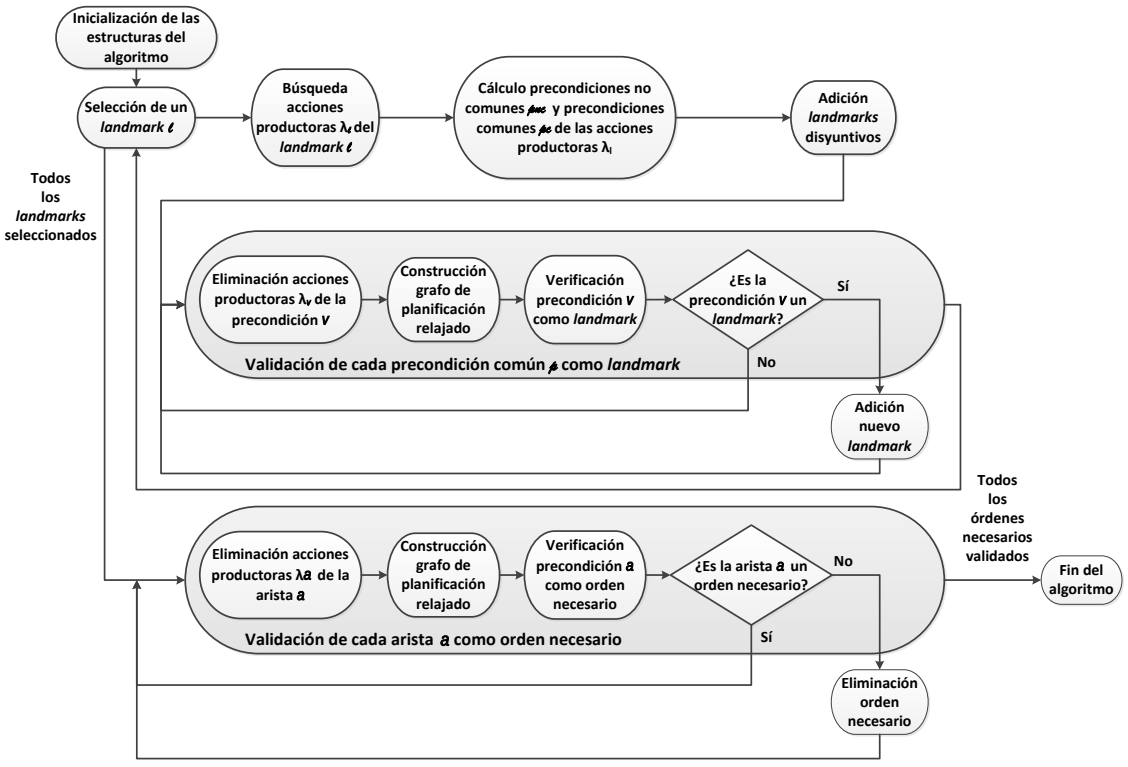


Figura 4.1: Algoritmo de extracción de *landmarks*

El algoritmo de extracción de *landmarks* original [SO03], en el que se basa este trabajo, se ilustra en la figura 4.1. A continuación, se explicarán las diferentes fases del algoritmo de extracción de *landmarks* original [SO03].

### 4.2.1. Inicio de las estructuras del algoritmo

El algoritmo calcula en primer lugar un grafo de planificación relajado [BF97] con todas las acciones $\mathcal{A}$ del problema. Ya que por definición todos los *fluents* del estado inicial y las metas son *landmarks*, se añaden directamente a la estructura de *landmarks* por niveles $LL$ y al grafo de *landmarks* $LG$. Los *landmarks* del estado inicial se almacenan en el nivel 0 de la estructura auxiliar $LL$, mientras que las metas se almacenan en la estructura $LL$ de acuerdo al nivel que ocupan en el grafo de planificación relajado.

### 4.2.2. Selección de un landmark $\ell$

En esta etapa se escoge un *landmark* $\ell$ del conjunto de *landmarks* de $LL$. Este *landmark* debe tener el mayor nivel posible y no debe haber sido escogido aún. No se escogerán los *landmarks* iniciales (nivel 0), pues de ellos no se pueden obtener más *landmarks*.

### 4.2.3. Cálculo de precondiciones no comunes $\rho_{nc}$ y comunes $\rho_c$

De entre todas las acciones del problema, $\mathcal{A}$, se selecciona el subconjunto $\lambda_l \subseteq \mathcal{A}$ de acciones que tienen $\ell$ como efecto, es decir, $\gamma \in \lambda_l \leftrightarrow \ell \in EFF(\gamma)$. Estas acciones, $\lambda_l$, serán las acciones productoras del landmark $\ell$ .

Una vez calculadas las acciones productoras $\lambda_l$ del landmark $\ell$, se calculan las precondiciones no comunes a todas las acciones de dicho conjunto. El método para calcular estas precondiciones no comunes es el siguiente.

1. Se calculan las variables comunes $\vartheta_l$ a todas las acciones del conjunto $\lambda_l$.

2. Una vez calculadas las variables comunes $\vartheta_l$, se agrupan todas las precondiciones de las acciones $\lambda_l$ según su variable $var$, siempre que $var \in \vartheta_l$.

3. Una vez extraídas todas las precondiciones no comunes $\rho_{nc}$ y agrupadas en conjuntos por su variable, se creará un *landmark* disyuntivo por cada

agrupación, ya que al menos un *fluent* del conjunto debería estar en todo plan para que éste sea un plan solución de un problema.

Después, se calculan las precondiciones comunes a todas la acciones del conjunto $\lambda_l$, lo que da lugar a un conjunto de *fluents* $\rho_c$. Las precondiciones en $\rho_c$ aparecerán en cualquier plan en que figure también $\ell$.

### 4.2.4. Adición de nuevos *landmarks* disyuntivos $\varsigma$

Esta etapa recoge los *landmarks* disyuntivos $\varsigma$ calculados la etapa anterior. Por definición, ya que los *landmarks* disyuntivos calculados no necesitan verificación, son, sin necesidad de calcular nada más, *landmarks* diyuntivos. Así pues, cada *landmark* disyuntivo $\varsigma_i$ del conjunto $\varsigma$, donde $\varsigma_i \in \varsigma$, se añade al grafo de *landmarks* $LG$. Además, se añade también un posible orden necesario $a$, tal que $\varsigma_i \leq_n \ell$, al conjunto $E$ del grafo de *landmarks* $LG$, dado que $\varsigma_i$ aparecerá antes que $\ell$ en cualquier plan solución.

### 4.2.5. Validación de precondiciones comunes $\rho_c$

En cada iteración de esta fase, se analiza un *fluent* $p \in \rho_c$ para validar si es un *landmark*. Esta validación se lleva a cabo mediante la construcción de un grafo de planificación relajado [ZNK07]. Este procedimiento consta de las siguientes etapas:

- **Eliminación de acciones productoras del *fluent* $p$**: De todas las acciones del problema, $\mathcal{A}$, se eliminan aquellas acciones que producen el *fluent* $p$ como efecto, es decir, el subconjunto $\lambda_p \subseteq \mathcal{A}$. Esto da lugar a un subconjunto de acciones $\mathcal{A}_{\lambda_p} = (\mathcal{A} \setminus \lambda_p)$ que se utilizará para construir el grafo de planificación relajado.

- **Construcción grafo de planificación relajado**: Se construye un grafo de planificación relajado [BF97] usando sólo aquellas acciones que no tienen como efecto el *fluent* $p$, $\mathcal{A}_{\lambda_p}$.

- **Verificación del *landmark***: Si todas las metas figuran en el grafo de planificación relajado, el *fluent* $p$ no será considerado *landmark*. Esto se debe a que al haberse alcanzado todas las metas sin contar con el *fluent* $p$, éste no es necesario para obtener un plan solución, por lo que no cumple la definición de *landmark*. Si, por el contrario, no se alcanza alguna de las metas en el grafo, el *fluent* $p$ será considerado un *landmark*.

### 4.2.6. Adición del nuevo *landmark* $p$

Si el *fluent* $p$ es un *landmark*, se guardará en la estructura auxiliar $LL$ de acuerdo a su nivel en el grafo relajado. Del mismo modo, $p$ se añadirá al grafo de *landmarks* $LG$, estableciéndose también un posible orden necesario $a$, tal que $p \leq_n \ell$, al conjunto $E$ del grafo de *landmarks* $LG$, dado que $p$ aparecerá antes que $\ell$ en cualquier plan solución.

### 4.2.7. Validación de órdenes necesarios $E$

En cada iteración de esta fase, se analiza una arista $a \in E$ para validar si es un orden necesario. No se calculará la validación de las aristas que contengan al menos un *landmark* disyuntivo, al carecer de valor. Esta validación se lleva a cabo mediante la construcción de un grafo de planificación relajado [ZNK07]. Este procedimiento consta de las siguientes etapas:

- **Eliminación de acciones productoras de la arista $a$, tal que $l_{prec} \leq_n l_{efec}$**: De todas las acciones del problema, $\mathcal{A}$, se eliminan aquellas acciones que tienen como precondición el *fluent* $l_{prec}$ y como efecto el *fluent* $l_{efec}$, es decir, el subconjunto $\lambda_a \subseteq \mathcal{A}$. Esto da lugar a un subconjunto de acciones $\mathcal{A}_{\lambda a} = (\mathcal{A} \setminus \lambda_a)$ que se utilizará para construir el grafo de planificación relajado.

- **Construcción grafo de planificación relajado**: Se construye un grafo de planificación relajado [BF97] usando sólo aquellas acciones que no tienen como precondición el *fluent* $l_{prec}$ y como efecto el *fluent* $l_{efec}$, $\mathcal{A}_{\lambda a}$.

- **Verificación del orden necesario $a$**: Si todas las metas figuran en el grafo de planificación relajado, la arista $a$ no será considerada un orden necesario.

  Esto se debe a que al haberse alcanzado todas las metas sin contar con las acciones productoras de la arista $a$, que indica que para conseguir el *fluent* $l_{efec}$ es necesario alcanzar antes el *fluent* $l_{prec}$, dicho enlace no es necesario para obtener un plan solución y, por lo tanto, no cumple la definición de orden necesario.

  Si, por el contrario, no se alcanza alguna de las metas en el grafo, la arista $a$ será considerada un orden necesario.

### 4.2.8. Eliminación de la arista $a$

Si la arista $a$ no es un orden necesario, dicha arista es eliminada del conjunto de órdenes $E$ calculados anteriormente de la estructura $LG$. De esta forma, al terminar el proceso de validación de todas las aristas, se habrán calculado todos los órdenes necesarios de un problema.

## 4.3. Adaptación multiagente

Esta sección describe los cambios realizados al algoritmo de extracción de *landmarks* presentado en la sección anterior (ver Figura 4.1) para llevar a cabo su adaptación a un contexto multiagente. La figura 4.2 refleja las fases del algoritmo multiagente.

Al igual que en el caso del grafo de planificación relajado, cada agente dispone de su propia versión del grafo de *landmarks*, cuyos contenidos dependen de la visión parcial del problema que mantiene el agente.

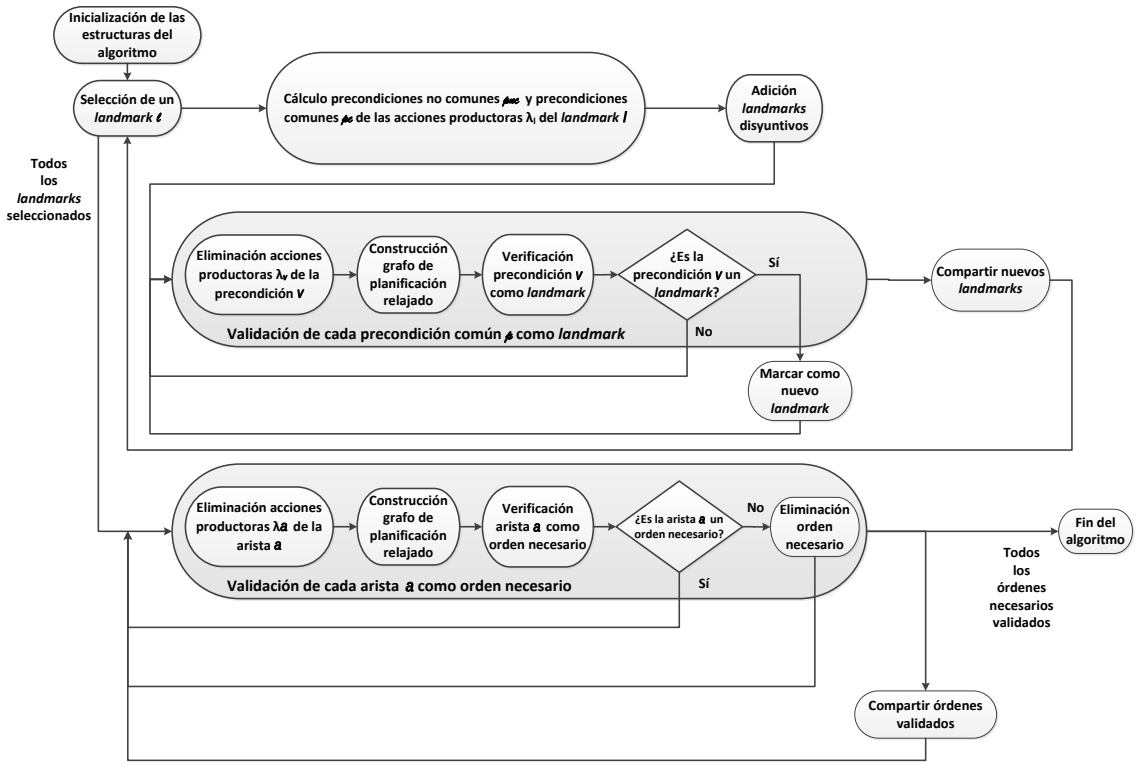


Figura 4.2: Algoritmo de extracción de *landmarks* multiagente

### 4.3.1. Comportamiento Landmarks-Mining

Para la adaptación multiagente del algoritmo de extracción de *landmarks* [SO03], se creó un comportamiento con las mismas características que los utilizados en PlanInteraction(sección 3.3.2). Este comportamiento emplea a todos los agentes implicados en un problema para extraer todos los *landmarks* que existan en dicho problema.

Este nuevo comportamiento, llamado *Landmarks-Mining*, se ejecutará en PlanInteraction después de la etapa de *Grounding*, descrita en la sección 3.2.1.2, y antes de la etapa de *Planificación*, detallada en la sección 3.2.1.3.

El comportamiento *Landmarks-Mining* interactúa con las demás etapas de PlanInteraction de la siguiente forma: el comportamiento *Landmarks-Mining* de cada agente $i$ recibe como entrada su tarea de planificación instanciada $\mathcal{T}^i$. Dicha tarea $T^i$ es calculada conjuntamente por todos los agentes en la etapa *Grounding*.

El comportamiento *Landmarks-Mining* proporciona como salida el grafo de landmarks $LG^i$ de cada agente $i$, que se utilizará en la etapa de Planificación para mejorar la precisión de la búsqueda heurística de FMAP. La sección 4.4 detalla cómo se utiliza la información extraída de los grafos de *landmarks* en la nueva heurística.

A continuación se detallarán aspectos generales de las figuras que describen los procesos del comportamiento *Landmarks-Mining*.

### 4.3.2. Aspectos generales de los diagramas

En todas las figuras de esta sección se emplea una nomenclatura para diferenciar los tipos de nodos que representan estados de PlanInteraction.

Los cuadros de las figuras de esta sección hacen referencia a un nodo o estado en PlanInteraction. Los nodos de tipo ACTION son de color rojo, los nodos de tipo SEND son de color amarillo, los de tipo RECEIVE son de color verde y el nodo WAIT es de color azul. Además, aunque sólo existe un nodo de tipo WAIT en el comportamiento *Landmarks-Mining*, este nodo se representará varias veces en una misma figura cuando resulte conveniente para facilitar su legibilidad. Asimismo, para simplificar los diagramas de flujo se representa la mayor parte de tareas multiagente como subprocesos. Estos

subprocesos se ilustran mediante recuadros redondeados de color gris.

Para lograr la ejecución ordenada de los subprocesos o fases del comportamiento los agentes asumirán el rol de coordinador o el rol participante en cada una de las fases. En todos los casos que vienen a continuación, uno de los agentes asumirá el rol de coordinador, encargándose de liderar el proceso, y el resto de agentes asumirán el rol de partipante, esperando peticiones del agente coordinador. Este rol rotará entre los agentes si es necesario para la realización de las tareas multiagente. Para hacer más comprensible el flujo de ejecución, en todos las gráficas se etiquetarán las aristas entre nodos y/o subprocesos con el rol del agente que la transitará.

A continuación, se describen en detalle las fases que componen el algoritmo multiagente de extracción de *landmarks*, representadas en la figura 4.2.

### 4.3.3. Inicio de las estructuras del algoritmo

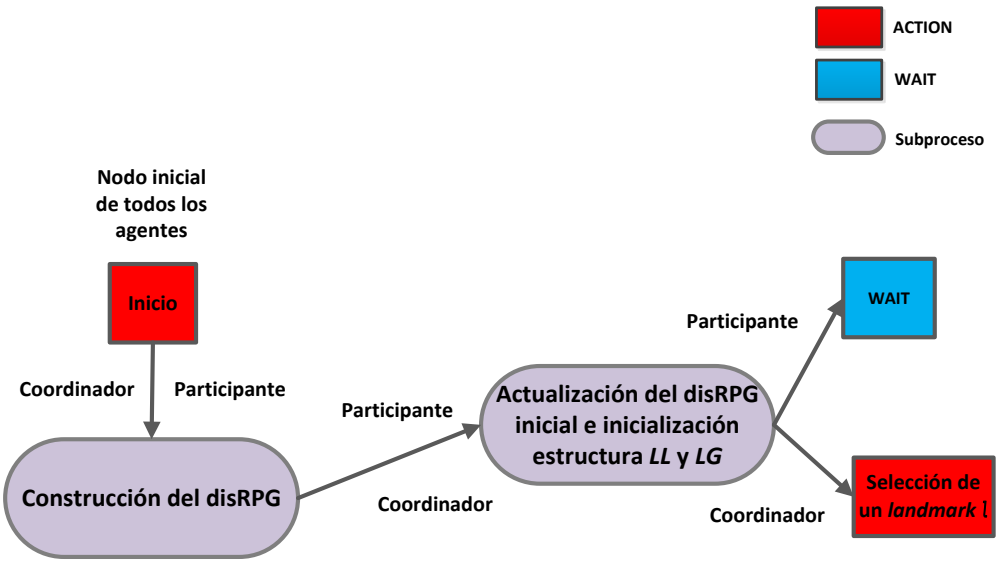


Figura 4.3: Inicio de las estructuras del algoritmo

Al igual que en la fase de "Inicio de las estructuras del algoritmo" del algoritmo monoagente, en la fase multiagente también se inicializan las estructuras $LL$ y $LG$. A continuación, se describen las etapas que constituyen este proceso, representadas en la figura 4.3. En este proceso, todos los agentes

empiezan en el nodo Inicio.

- **Inicio:** Todos los agentes comienzan ejecutando en paralelo el nodo Inicio. En él, cada agente obtiene su tarea $T^i$ con los elementos instanciados correspondientes a su problema: $\mathcal{AG}$, $\mathcal{O}$,$\mathcal{V}$,$\mathcal{A}^i$, $\mathcal{I}$, $\mathcal{G}$. El significado de todos estos elementos puede verse en la sección 3.1 del capítulo 3.

  Después de ejecutar el nodo Inicio, tanto los agentes participantes como el agente coordinador comenzarán el subproceso de construcción de un grafo de planificación relajado distribuido (disRPG).

- **Construcción del disRPG:** En este subproceso todos los agentes, tanto el agente coordinador como los agentes participantes, participan y construyen un grafo de planificación relajado distribuido (disRPG) tal y como se explica en la sección 4.3.10.3. Una vez calculados dichos grafos, tanto el agente coordinador como los agentes participantes transitarán al subproceso de "Actualización del disRPG inicial e inicialización de las estructuras $LL$ y $LG$".

- **Actualización del disRPG inicial e inicialización de las estructuras $LL$ y $LG$:** En este subproceso los agentes realizan una tarea multiagente donde sobreescriben el grafo de planificación relajado (disRPG) calculado inicialmente e inicializan sus estructuras $LL$ y $LG$. Al igual que en el procedimiento monoagente, el estado inicial y las metas se almacenan por niveles en la estructura $LL$. Los *fluents* del estado inicial se guardan en el nivel 0, mientras que las metas se almacenan según su nivel en el grafo de planificación relajado multiagente. Para finalizar, el agente coordinador se dirigirá hacía el nodo "Selección de un *landmark* $\ell$", donde comenzará la extracción de nuevos *landmarks*. Los agentes participantes volverán al nodo WAIT.

### 4.3.4. Selección de un *landmark* $\ell$

En este proceso el agente coordinador explora su estructura $LL^{coordinador}$ de forma regresiva, comenzando por el último nivel (en adelante nivel $n$). A continuación pueden darse varios casos que determinan el flujo de la ejecución:

- **Si $LL^{coordinador}(n) \neq \emptyset$:** Si aún quedan *landmarks* por analizar en el nivel $n$ de la estructura $LL^{coordinador}$, el agente coordinador selecciona

un *landmark* $l$ de entre ellos. A continuación, el agente coordinador intentará extraer nuevos *landmarks* a partir de dicho *landmark* $l$. Para llevar a cabo este proceso, que se verá en la sección 4.3.5, el agente coordinador transita al subproceso "Cálculo de precondiciones comunes $\rho_c$ y no comunes $\rho_{nc}$" junto con los agentes participantes.

- **Si $LL^{coordinador}(n) = \emptyset$ y quedan agentes por explorar el nivel $n$:** En este caso, el agente coordinador empezará un subproceso de "Rotación del agente coordinador" junto con los demás agentes participantes. De este modo, el nuevo agente coordinador explorará el nivel $n$ de su estructura $LL$ en busca de más *landmarks* por analizar.

- **Si $LL^{coordinador}(n) = \emptyset$ y todos los agentes han explorado ya el nivel $n$:** En este caso, el agente coordinador empieza el subproceso "Actualizar último nivel a observar en la estructura $LL$" junto con los demás agentes participantes. En dicha tarea multiagente todos los agentes decrementan el nivel a explorar, $n = n-1$. Es decir, los agentes pasarán a analizar el nivel anterior de $LL$.

- **Si $n = 0$:** En este caso, el proceso de exploración de la estructura $LL$ termina, y el agente coordinador inicia, junto a los demás agentes, el subproceso "Validación de órdenes necesarios $E$". Este subproceso se detallará en la sección 4.3.9.

### 4.3.5. Cálculo de precondiciones comunes $\rho_c$ y no comunes $\rho_{nc}$

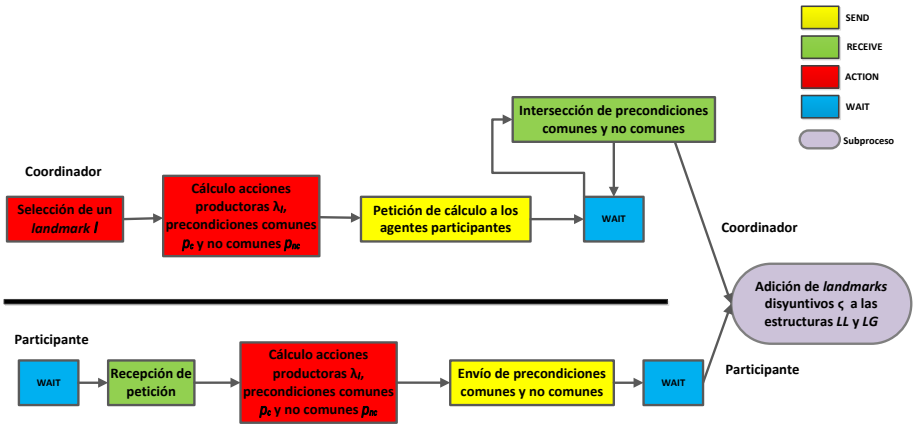


Figura 4.4: Cálculo de precondiciones no comunes $\rho_{nc}$ y comunes $\rho_c$

Los nodos correspondientes al subproceso de cálculo de precondiciones comunes y no comunes se muestran en la figura 4.4. En este subproceso, los agentes participantes parten del nodo WAIT mientras que el agente coordinador inicia el subproceso en el nodo "Seleccionar *landmark* $\ell$".

- **Cálculo acciones productoras $\lambda_l$, precondiciones comunes $\rho_c$ y no comunes $\rho_{nc}$:** Tras seleccionar un *landmark* $\ell$ en el nodo "Selección de *landmark* $\ell$" (descrito en el apartado 4.3.4), el agente coordinador transita a este nodo para calcular las acciones $\lambda_l^{coordinador}$ de su dominio que producen como efecto dicho *landmark* $\ell$. Asimismo, el coordinador calcula las precondiciones no comunes $\rho_{nc}^{coordinador}(l)$ y comunes $\rho_c^{coordinador}(l)$ a esas acciones (ver sección 4.2.3). Una vez calculados dichos elementos, el agente coordinador se dirige el nodo "Petición de cálculo a los agentes participantes".

- **Petición de cálculo a los agentes participantes:** En este nodo, el agente coordinador envía un mensaje al resto de agentes que producen el *landmark* $\ell$ solicitando que calculen las precondiciones comunes y no comunes a sus acciones productoras del *landmark* $\ell$. Una vez enviados los mensajes, el agente coordinador transita al estado WAIT.

- **Recepción de petición:** En este nodo, cada agente participante productor de $\ell$ recibe la petición enviada por el agente coordinador en el nodo anterior, y transita al nodo "Cálculo acciones productoras $\lambda_l$, precondiciones comunes $\rho_c$ y no comunes $\rho_{nc}$". Este nodo es idéntico al homónimo previamente detallado en esta sección. Una vez calculados dichos elementos, cada agente participante se dirige el nodo "Envío de precondiciones comunes y no comunes".

- **Envío de precondiciones comunes y no comunes:** En este nodo, cada agente participante envía sus preconciones comunes $\rho_{nc}^{participante}(l)$ y no comunes $\rho_c^{participante}(l)$ al agente coordinador, para que éste calcule la intersección de todas ellas. Tras enviar las precondiciones, los agentes participantes se dirigen al nodo WAIT.

- **Intersección de precondiciones comunes y no comunes:** Cada vez que el agente coordinador recibe un mensaje de un agente participante, transita al nodo "Intersección precondiciones comunes y no comunes". En dicho nodo, el agente coordinador calcula la intersección de sus conjuntos de precondiciones comunes y no comunes y los del

agente emisor del mensaje.

El agente coordinador obtiene como resultado de este subproceso el conjunto $\rho_{\cap_c}(l)$ de precondiciones comunes al *landmark* $l$, es decir, $\rho_{\cap_c}(l) = \bigcap_{\forall i \in \mathcal{AG} | i \in a_l} \rho_c^i(l)$.

En el caso de los *landmarks* correspondientes a metas del problema, el cálculo varía ligeramente, dado que las metas son visibles para todos los agentes, pero no necesariamente son alcanzables por todos ellos. Para asegurar que se calculan las precondiciones comunes correctamente, sólo se intersecan los conjuntos $\rho_c^i(l)$ no vacíos. Por tanto, dado un *landmark* $g$ tal que $g \in \mathcal{G}$, $\rho_{\cap_c} = \bigcap_{\forall i \in \mathcal{AG} | i \in a_l \wedge \rho_c^i(l) \neq \emptyset} \rho_c^i(l)$.

Los elementos del conjunto $\rho_{\cap_c}$ son los nuevos candidatos a *landmarks* simples. Antes de ser añadidos al grafo de *landmarks*, estos candidatos deberán ser verificados en una fase posterior, detallada en la sección 4.3.7.

Para calcular las precondiciones no comunes al *landmark* $\ell$, el agente coordinador realiza el siguiente proceso para cada mensaje recibido:

- Se calculan las variables comunes $\vartheta_l$ a todos los *fluents* de los conjuntos $\rho_{nc}^{coordinador}(l)$ y $\rho_{nc}^{participante}(l)$.
- Una vez calculadas las variables comunes $\vartheta_l$, se agrupan todos los *fluents* de los conjuntos $\rho_{nc}^{coordinador}(l)$ y $\rho_{nc}^{participante}(l)$ según su variable $var$, para toda $var \in \vartheta_l$.

Una vez calculadas las precondiciones comunes $\rho_{\cap_c}$ y no comunes $\rho_{\cap_{nc}}$ a todos los agentes, el agente coordinador inicia el subproceso "Adición de *landmarks* disyuntivos $\varsigma$ a las estructuras $LL$ y $LG$". En dicho subproceso, los agentes generan los nuevos *landmarks* disyuntivos (conjuntos de precondiciones no comunes $\rho_{\cap_{nc}}$ cuyas variables hayan sido comunes a todos los mensajes). Este subproceso se detallará en la sección 4.3.6.

### 4.3.6. Adición de *landmarks* disyuntivos $\varsigma$ a las estructuras $LL$ y $LG$

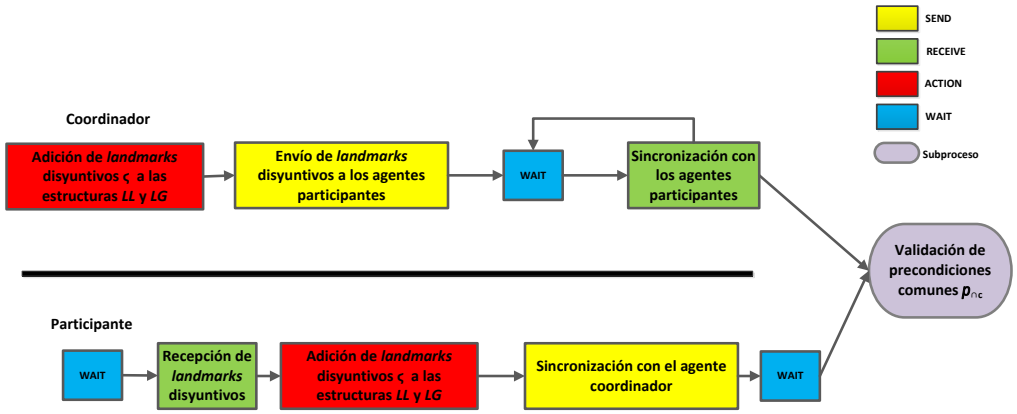


Figura 4.5: Adición nuevos *landmarks* disyuntivos $\varsigma$

La figura 4.5 representa el proceso multiagente de adición de los nuevos *landmarks* disyuntivos, $\varsigma$, cuyo cálculo se describe en la sección 4.3.5. En este subproceso, el agente coordinador parte del nodo "Adición de *landmarks* disyuntivos a las estructuras $LL$ y $LG$", mientras que los agentes participantes parten del nodo WAIT.

- **Adición de *landmarks* disyuntivos a las estructuras $LL$ y $LG$:** En este nodo, el agente coordinador añade todos los *landmarks* disyuntivos $\varsigma$ en su estructura $LL^{coordinador}$. Dichos *landmarks*, $\varsigma$, son ubicados en el nivel $n-1$ (donde $n$ es el nivel que está explorando el agente coordinador). Además, se añade también un posible orden necesario $a = \varsigma_i \leq_n \ell$, al conjunto $E$ del grafo de *landmarks* $LG^{coordinador}$. Este orden indica que $\varsigma_i$ aparecerá antes que $\ell$ en cualquier plan solución.

- **Envío de *landmarks* disyuntivos a los agentes participantes:** En este nodo el agente coordinador envía a todos los agentes participantes los *landmarks* disyuntivos de los que son productores. Después del envío de mensajes, el agente coordinador se dirige al nodo WAIT.

- **Recepción de *landmarks* disyuntivos:** En este nodo cada agente participante recibe un mensaje con los *landmarks* disyuntivos de

los cuales es productor. Una vez recibido el mensaje, el agente transita al nodo "Adición de *landmarks* disyuntivos a las estructuras $LL$ y $LG$", donde añade dichos *landmarks disyuntivos* a sus estructuras $LL^{participante}$ y $LG^{participante}$. En funcionamiento de dicho nodo es idéntico al nodo homónimo detallado anteriormente en esta sección. Una vez añadidos los *landmarks* disyuntivos, cada agente participante se dirige al nodo "Sincronización con el agente coordinador".

- **Sincronización con el agente coordinador:** En este nodo cada agente participante envía un mensaje al agente coordinador indicándole que ha terminado de añadir las *landmarks* disyuntivos. Después de enviar dicho mensaje, el agente participante transitará al nodo WAIT.
- **Sincronización con los agentes participantes:** El agente coordinador recibirá en este nodo un mensaje por cada agente participante. Una vez recibidos todos los mensajes, el agente coordinador se dirigirá al subproceso de "Validación de precondiciones comunes $\rho_{\cap_c}$" junto con los agentes participantes. El agente coordinador esperará en el nodo WAIT hasta recibir todos los mensajes de los agentes participantes.

### 4.3.7. Validación de precondiciones comunes $\rho_{\cap_c}$

En cada iteración de esta fase se analiza un *fluent* $p \in \rho_{\cap_c}$ para validar si es un *landmark*. Esta validación se lleva mediante un proceso de tres etapas: "Eliminación de acciones productoras del *fluent* $p$", "Construcción grafo de planificación relajado distribuido (disRPG)" y "Verificación del *landmark*".

#### 4.3.7.1. Eliminación de acciones productoras del *fluent* $p$

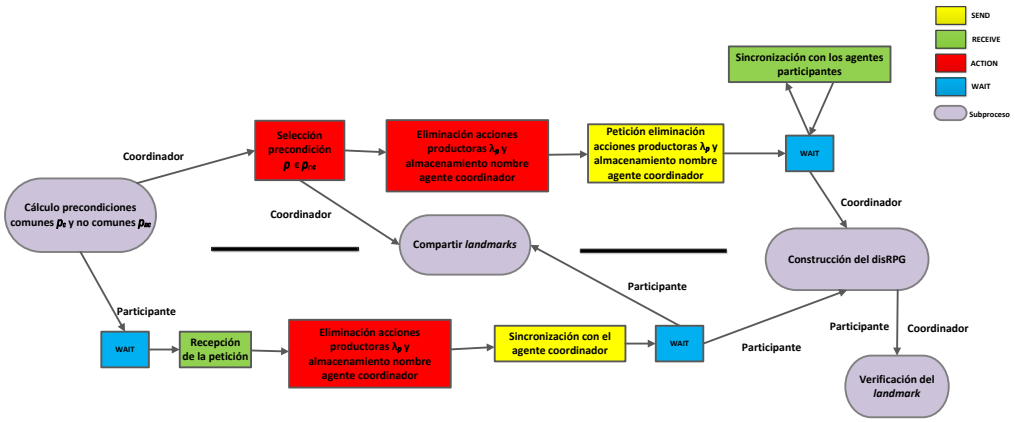


Figura 4.6: Eliminación de acciones productoras del *fluent* $p$

La figura 4.6 ilustra la etapa de eliminación de las acciones productoras de la precondición a validar y la preparación de los agentes para la construcción de un nuevo grafo de planificación relajado multiagente. A continuación, se describe el funcionamiento de los nodos de la figura 4.6. Tras ejecutar el subproceso "Cálculo de precondiciones comunes $\rho_c$ y no comunes $\rho_{nc}$", el agente coordinador se dirige al nodo "Selección precondición $p \in \rho_{\cap_c}$", mientras que los agentes participantes transitan el nodo WAIT.

- **Selección de la precondición $p \in \rho_{\cap_c}$:** En dicho nodo, el agente coordinador extrae una precondición $p$ del conjunto de precondiciones comunes $\rho_{\cap_c}$ para verificar si $p$ es un *landmark*. Tras escogerla, el agente coordinador se dirige al nodo "Eliminación de acciones productoras $\lambda_p$ y almacenamiento nombre agente coordinador".

- **Eliminación de acciones productoras $\lambda_p$ y almacenamiento nombre agente coordinador:** En este nodo, el agente coordinador calculará un conjunto de acciones en el cual se eliminan aquellas acciones que producen el *fluent* $p$ como efecto, es decir, el subconjunto $\lambda_p^{coordinador} \subseteq \mathcal{A}^{coordinador}$. Esto da lugar a un subconjunto de acciones $\mathcal{A}_{\lambda_p}^{coordinador} = (\mathcal{A}^{coordinador} \setminus \lambda_p^{coordinador})$ que se utilizará para construir el grafo de planificación relajado distribuido. A continuación, este agente guarda el nombre del agente coordinador en su almacenamiento interno. De esta forma, llegado el caso, se podría reestablecer el actual agente coordinador si fuese necesario en algún punto del algoritmo. Por último, el agente coordinador transitará al nodo "Petición eliminación acciones productoras $\lambda_p$ y almacenamiento nombre agente coordinador".

- **Petición eliminación acciones productoras $\lambda_p$ y almacenamiento nombre agente coordinador:** En este nodo, el agente coordinador manda un mensaje a todos los agentes participantes del problema de planificación pidiéndoles que creen un nuevo conjunto de acciones sin sus acciones productoras de la precondición $p$ y guarden el nombre del actual agente coordinador.

- **Recepción de la petición:** En este nodo, cada agente participante recibe la petición del agente coordinador, y transita al nodo "Eliminación acciones productoras $\lambda_p$ y almacenamiento nombre agente coordinador". En este nodo el agente participante ejecutará el mismo proceso que el descrito anteriormente en el nodo homónimo. Después de que

el agente participante haya calculado su nuevo conjunto de acciones $\mathcal{A}_{\lambda_p}^{participante} = (\mathcal{A}^{participante} \setminus \lambda_p^{participante})$ y haya guardado en su almacenado el nombre del actual agente coordinador, el agente participante transitará al nodo "Sincronización con el agente coordinador".

- **Sincronización con el agente coordinador:** En este nodo, cada agente participante envía un mensaje al agente coordinador indicándole que ha terminado de calcular el conjunto de acciones $\mathcal{A}_{\lambda_p}^{participante} = (\mathcal{A}^{participante} \setminus \lambda_p^{participante})$ y ha guardado en su memoria interna el nombre del actual agente coordinador. Después de enviar dicho mensaje, el agente participante transitará al nodo WAIT.

- **Sincronización con los agentes participantes:** El agente coordinador recibe en este nodo de recepción un mensaje por cada agente participante. Cuando haya recibido todos los mensajes, el agente coordinador se dirigirá al subproceso de "Construcción disRPG" junto con los agentes participantes. Durante la sincronización, el agente atiende de los mensajes recibidos en el nodo de recepción. Una vez atendido el mensaje, el agente coordinador espera nuevos mensajes en el nodo WAIT.

#### 4.3.7.2. Construcción de un grafo de planificación relajado distribuido (disRPG)

Para la validación de una precondición $p$ los agentes construirán el disRPG usando sólo los conjuntos de acciones calculados en la etapa anterior. Es decir, el agente coordinador utilizará las acciones $\mathcal{A}_{\lambda_p}^{coordinador}$ y los agentes participantes las acciones $\mathcal{A}_{\lambda_p}^{participante}$. El proceso multiagente de construcción de un disRPG se describe con detalle en la sección 4.3.10.3.

Una vez realizada la construcción del disRPG, el agente coordinador actual, que puede no ser el que comenzó el proceso de validación del *fluent* $p$, comienza la última etapa de la validación, el subproceso "Verificación del *landmark*".

#### 4.3.7.3. Verificación del *landmark*

Una vez realizada la construcción del disRPG, se verifica si el *fluent* $p$ es un *landmark*. Si se han alcanzado todas las metas del problema en el disRPG, no es necesario generar el *fluent* $p$ en cualquier plan solución para alcanzar los objetivos, y por tanto $p$ no es un *landmark* . Si, por el contrario, no se

alcanza alguna de las metas en el grafo, el *fluent* $p$ será considerado un *landmark*.

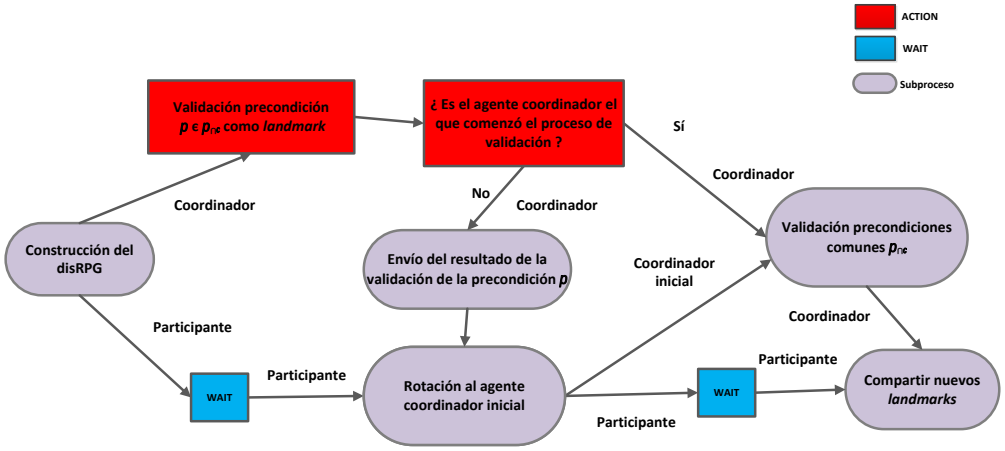


Figura 4.7: Verificación de la precondición $p$

La verificación del candidato a *landmark* $p$ se ilustra en la figura 4.7. En este subproceso el agente coordinador parte del nodo "Validación precondición $p \in \rho_{\cap_c}$ como *landmark*" y los agentes participantes del nodo WAIT. A continuación, se describen los nodos que transitan los agentes en esta etapa:

- **Validación precondición $p \in \rho_{\cap_c}$ como *landmark*:** En este nodo, el actual agente coordinador comprueba si el *fluent* $p$ es un *landmark* o no. Para ello, el agente coordinador verifica si el disRPG construido en la etapa anterior ha alcanzado todas las metas del problema.

  Una vez realizada dicha comprobación, el actual agente coordinador transita al nodo "¿ Es el agente coordinador el que comenzó el proceso de validación ?", donde comprueba si él mismo comenzó el proceso de validación o fue otro agente.

  Esta comprobación es necesaria pues el agente coordinador inicial debe determinar si $p$ es un *landmark*. El proceso de validación se ha diseñado de modo que el agente coordinador inicial es el encargado de informar al resto de agentes acerca de qué precondiciones comunes se han confirmado como *landmarks*.

Si el actual agente coordinador es el que comenzó este proceso, dicho agente transita al subproceso "Validación precondiciones comunes $\rho_{\cap_c}$", donde seleccionará otra precondición e iniciará de nuevo el proceso de validación para analizar la siguiente precondición en $\rho_{\cap_c}$. En caso contrario, el agente coordinador transita al nodo "Envío del resultado de la validación de la precondición $p$".

- **Envío del resultado de la validación de la precondición $p$:** En este nodo, el agente coordinador actual comunica al agente que inició el proceso de validación el resultado de la verificación de la precondición $p$. Tras ser comunicado el resultado de la verificación de la precondición $p$, todos los agentes se dirigen al subproceso "Rotación al agente coordinador inicial".

- **Rotación al agente coordinador inicial:** Este proceso es inmediato, dado que todos los agentes almacenaron el nombre del agente iniciador de la validación en el proceso de "Eliminación de acciones productoras del *fluent* $p$". Una vez realizada la rotación al nuevo agente coordinador, dicho agente se dirigirá al subproceso "Validación precondiciones comunes $\rho_{\cap_c}$" para comenzar otra vez el proceso de validación, descrito en la sección 4.3.7, con otra precondición.

Una vez el proceso de validación de todas las precondiciones comunes $\rho_{\cap_c}$ haya concluido, el agente coordinador procederá a compartir los nuevos *landmarks* con el resto de agentes.

### 4.3.8. Compartir nuevos *landmarks*

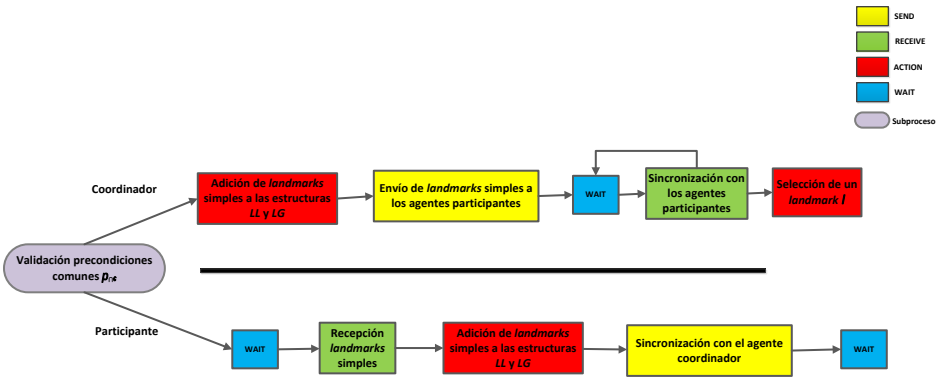


Figura 4.8: Compartir nuevos *landmarks*

La figura 4.8 muestra el funcionamiento en PlanInteraction del proceso que permite al agente coordinador compartir con el resto de agentes los nuevos *landmarks* extraídos previamente (ver sección 4.3.7). A continuación, se describen los nodos de este proceso, ilustrados en la figura 4.8. Después de ejecutar el subproceso "Validación precondiciones comunes $\rho_{\cap_c}$" el agente coordinador se dirige al nodo "Adición de *landmarks* simples a las estructuras $LL$ y $LG$" y los agentes participantes al nodo WAIT.

- **Adición de *landmarks* simples a las estructuras $LL$ y $LG$:** En este nodo, el agente coordinador añade todos los *landmarks* simples $lc^{coordinador}$ en su estructura $LL^{coordinador}$. Dichos *landmarks*, $lc^{coordinador}$, son ubicados en el nivel anterior al actual. Además, se añade también un orden necesario $a = lc^{coordinador} \leq_n \ell$, al conjunto $E$ del grafo de *landmarks* $LG^{coordinador}$, dado que $lc^{coordinador}$ aparecerá antes que $\ell$ en cualquier plan solución. Los órdenes necesarios deberán ser verificados en una fase posterior (ver sección 4.3.9). Una vez añadidos los *landmarks*, el agente coordinador transitará al nodo "Envío de *landmarks* simples a los agentes participantes".

- **Envío de *landmarks* simples a los agentes participantes:** En este nodo el agente coordinador envía a todos los agentes participantes los *landmarks* simples de los que son productores. Después del envío de mensajes, el agente coordinador se dirige al nodo WAIT.

- **Recepción *landmarks* simples:** En este nodo cada agente participante recibe un mensaje con los *landmarks* simples de los cuales es productor, $lc^{participante}$. Una vez recibido el mensaje, el agente transita al nodo "Adición de *landmarks* simples a las estructuras $LL$ y $LG$", donde introducirá los *landmarks* de $lc^{participante}$ en $LL^{participante}$ y $LG^{participante}$. El funcionamiento de dicho nodo es idéntico al nodo homónimo detallado anteriormente en esta sección. Una vez añadidos los *landmarks* simples, cada agente participante se dirigirá al nodo "Sincronización con el agente coordinador".

- **Sincronización con el agente coordinador:** En este nodo cada agente participante enviará un mensaje al agente coordinador indicándole que ha terminado de añadir los *landmarks* simples. Después de enviar dicho mensaje, el agente participante transitará al nodo WAIT.

- **Sincronización con los agentes participantes:** El agente coordinador recibirá en este nodo un mensaje por cada agente participante.

Cuando haya recibido todos los mensajes, el agente coordinador se dirigirá al nodo de "Selección de un *landmark* $\ell$" junto con los agentes participantes (ver sección 4.3.4). Si no ha recibido todos los mensajes, el agente coordinador espera el siguiente mensaje en el nodo WAIT.

### 4.3.9. Validación de órdenes necesarios $E$

El proceso de validación de órdenes necesarios $E$ es muy similar al descrito en la sección 4.3.7, en el cual se validaba si una precondición común $p \in \rho_{\cup_c}$ era o no un *landmark*. Los cambios más relevantes con respecto al proceso de validación de precondiciones comunes son los siguientes:

- **Eliminación de acciones productoras de la arista** $a = l_{prec} \leq_n l_{efec}$: En primer lugar, se eliminan las acciones productoras de la arista $a$. Por tanto, se eliminan aquellas acciones $\lambda_a$ que tienen como precondición el *fluent* $l_{prec}$ y como efecto $l_{efec}$. Es decir, todo agente $i$ del problema operará en este proceso sólo con las acciones $\mathcal{A}_{\lambda_a}^i = (\mathcal{A}^i \setminus \lambda_a^i)$.
- **Verificación de la arista** $a = l_{prec} \leq_n l_{efec}$: Una vez realizada la construcción del grafo de planificación relajado distribuido (disRPG), mecanismo descrito en la sección 4.3.10.3, se comprueba si el disRPG contiene todas las metas del problema. Si es así, la arista $a$ no se considera un orden necesario y por lo tanto se elimina del conjunto $E$ del grafo de landmarks $LG$. Si, por el contrario, no se alcanza alguna de las metas en el grafo, la arista $a$ queda validada como orden necesario, y por tanto, se mantiene en el conjunto E del grafo de landmarks $LG$. Este mecanismo sigue el mismo proceso que el ilustrado en la figura 4.7.
- **Compartición de resultados de verificación de aristas**: Los agentes establecen un proceso como el ilustrado en la figura 4.3.10.1, donde cada agente $i$ envía a los demás agentes los resultados de todas las aristas $a^i \in E^i$ que ha verificado. De esta forma, si un agente $j$ añadió la misma arista que el agente $i$, no será necesario verificarla de nuevo.

  Así pues, cada agente $j$ recibe una serie de aristas que comparará con las suyas propias $a^j \in E^j$. Si alguna arista recibida es exactamente igual a una que el agente $j$ tiene, el agente $j$ examinará el resultado de la validación de la arista recibida, y la eliminará de su conjunto $E^j$ si es necesario. Una vez terminado este proceso, cada agente $i$ guarda su grafo de *landmarks* $LG^i$ y termina el algoritmo multiagente de extracción de *landmarks*, saliendo del comportamiento *Landmarks-Mining*.

### 4.3.10. Subprocesos más importantes en *Landmarks-Mining*

En esta sección se detallarán los subprocesos más importantes del comportamiento *Landmarks-Mining*. Para cada subproceso, se describen los nodos de PlanInteraction que lo conforman.

#### 4.3.10.1. Sincronización de los agentes

En muchas ocasiones los agentes necesitan sincronizarse para ejecutar un proceso o tarea multiagente. Para ello, uno de los agentes asume el rol de coordinador, liderando de este modo el proceso de sincronización.

La figura 4.9 ilustra el mecanismo de sincronización utilizado habitualmente en PlanInteraction. Cada recuadro de la figura representa un nodo en PlanInteraction. Como se describe en la sección 3.3.2, PlanInteraction incluye, entre otros, 4 tipos de nodos: envío de un mensaje (SEND), recepción de un mensaje (RECEIVE), desarrollo de una acción concreta (ACTION) y espera (WAIT). El comportamiento desarrollado incluye un único nodo de tipo WAIT; no obstante, este nodo se replica en varias ocasiones en la figura 4.9 para facilitar su legibilidad.

En este proceso, el agente coordinador partirá del nodo "Realización de la tarea", mientras que el resto de agentes participantes se encuentran en el nodo WAIT esperando.

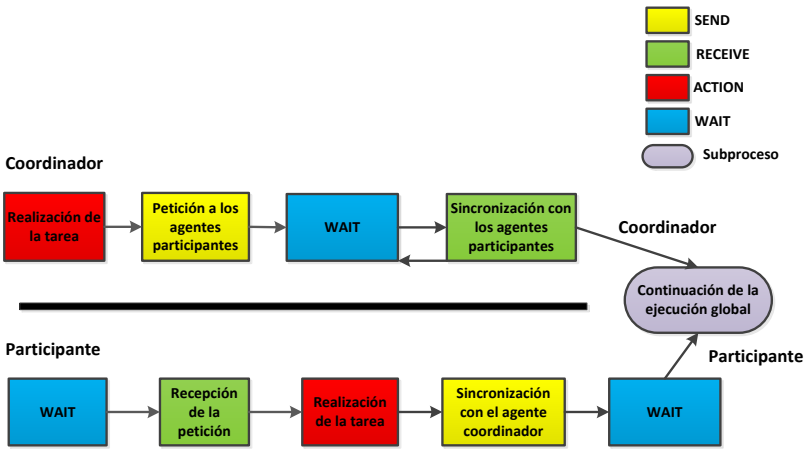


Figura 4.9: Mecanismo de sincronización de agentes en PlanInteraction

- **Realización de la tarea:** En este nodo, el agente coordinador ejecuta

la subtarea a realizar y después transita al nodo "Petición a los agentes participantes".

- **Petición a los agentes participantes:** En este nodo, el agente coordinador manda un mensaje a los demás agentes del problema para que ellos también la realicen la subtarea. Más tarde, el agente coordinador transita al nodo de tipo WAIT y entra en estado de espera.

- **Recepción de la petición:** En este nodo, cada agente participante involucrado en la subtarea transita del nodo WAIT al nodo "Recepción de la petición" para procesar el mensaje enviado por el agente coordinador. Después de procesar el mensaje, cada agente participante transita al nodo "Realización de la tarea".

- **Realización de la tarea:** En este nodo, cada agente participante ejecuta la subtarea a realizar y, tras completarla, transita al nodo "Sincronización con el agente coordinador".

- **Sincronización con el agente coordinador:** En ese nodo, cada agente participante envía un mensaje al agente coordinador con el fin de avisarle de que ha completado la subtarea que lo tenía ocupado. Después de enviar dicho mensaje, el agente participante transita al nodo de tipo WAIT.

- **Sincronización con los agentes participantes:** El agente coordinador espera en el nodo WAIT a recibir mensajes de los participantes. Una vez recibido un mensaje, el agente coordinador lo procesa transitando al nodo "Sincronización con los agentes participantes".

  Tras procesar el mensaje, el agente coordinador efectua un conteo de los mensajes recibidos en este nodo. Si no ha recibido un mensaje de cada uno de los agentes involucrados en la subtarea, el coordinador espera nuevos mensajes en el estado WAIT. En caso contrario, el agente coordinador abandona el proceso de sincronización, pasando a ejecutar la siguiente subtarea junto con todos los agentes participantes (ver figura 4.9).

#### 4.3.10.2. Sincronización de los agentes en la rotación del agente coordinador

Al igual que en el anterior subproceso de sincronización de la sección anterior 4.3.10.1, uno de los agentes asumirá el rol de coordinador, liderando

así el procedimiento.

.

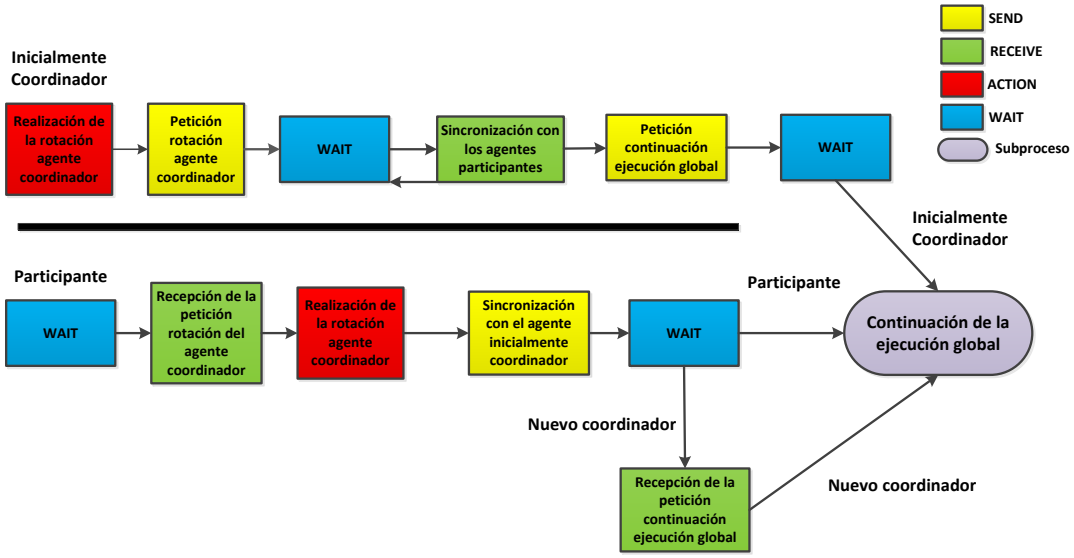


Figura 4.10: Mecanismo de sincronización de los agentes en la rotación del agente coordinador

La figura 4.10 ilustra el mecanismo de sincronización para la rotación del agente coordinador, en el que participan todos los agentes del problema. El coordinador inicia el proceso en el nodo "Realización de la rotación del agente coordinador", mientras que los agentes participantes esperan en el nodo WAIT.

- **Realización de la rotación agente coordinador:** En este nodo el agente coordinador realiza la rotación del agente coordinador. A continuación, transita al nodo "Petición rotación agente coordinador".

- **Petición rotación agente coordinador:** En este nodo el agente inicialmente coordinador envía un mensaje al resto de agentes involucrados en el problema de *planning*, identificándose. Tras enviar el mensaje, transita al nodo WAIT.

- **Recepción de la petición rotación del agente coordinador:** Para procesar el mensaje recibido, cada participante transita del nodo

WAIT al nodo "Recepción petición y realización de la tarea". En este nodo, cada agente recibe el mensaje enviado por el coordinador, lo analiza y almacena el nombre del agente coordinador. Después, transita al nodo "Realización de la rotación agente coordinador", donde el rol de coordinador es asumido por uno de los participantes. Una vez realizada la rotación cada agente se dirige "Sincronización con el agente inicialmente coordinador".

- **Sincronización con el agente inicialmente coordinador:** En este nodo cada agente envía un mensaje, gracias al nombre anteriormente almacenado, al agente inicialmente coordinador, con el fin de indicar que ya ha realizado la rotación del agente coordinador. Después, los agentes se dirigen al estado WAIT.

- **Sincronización con los agentes participantes:** En este nodo, el agente inicialmente coordinador recibe un mensaje de un agente participante. Para recibir dicho mensaje, el agente inicialmente coordinador transita del nodo WAIT al nodo "Sincronización con los agentes participantes". El mensaje recibido indica que un agente ya ha actualizado la rotación del agente coordinador. Cuando el agente inicialmente coordinador haya recibido todos los mensajes, transita al nodo "Petición continuación ejecución global".

- **Petición nuevo agente coordinador:** En este nodo, el agente inicialmente coordinador envía un mensaje al actual agente coordinador (que todos los agentes conocen pues todos han ejecutado la rotación) con el fin de indicarle que han completado la rotación y se puede proseguir con la ejecución global del comportamiento. Después de enviar el mensaje, el agente inicialmente coordinador se dirige al nodo WAIT.

- **Recepción de la petición continuación ejecución global:** En este nodo, el nuevo agente coordinador recibe un mensaje del agente inicialmente coordinador. Para poder procesar este mensaje, el nuevo agente coordinador transitará del nodo WAIT al nodo "Recepción de la petición continuación ejecución global". Este mensaje indica que el agente coordinador ya puede continuar con la ejecución global del comportamiento junto con los demás agentes.

#### 4.3.10.3. Construcción de un grafo de planificación relajado distribuido

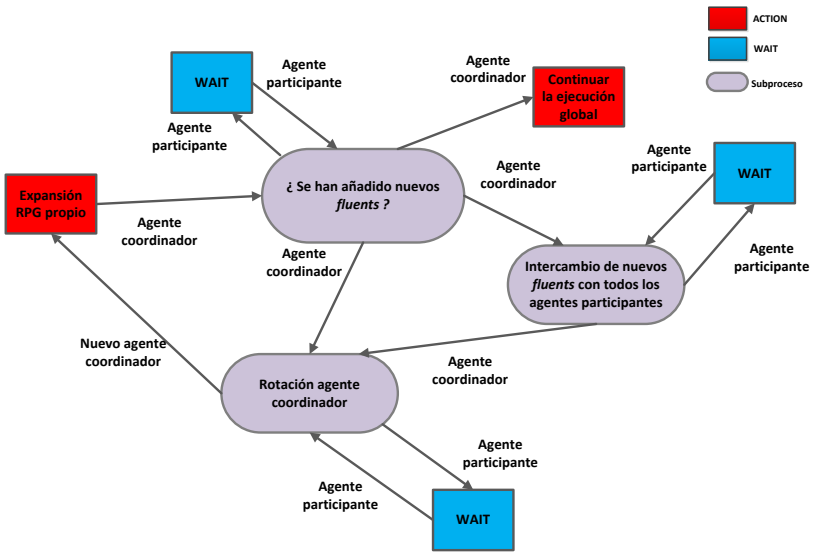


Figura 4.11: Mecanismo de construcción de un grafo de planificación relajado distribuido

La figura 4.11 refleja el mecanismo de construcción de un grafo de planificación relajado distribuido (disRPG). En este procedimiento interactúan todos los agentes involucrados en un problema de planificación.

El agente coordinador empieza la subtarea en el nodo"Expansión propio RPG". Los participantes se encuentran en el nodo WAIT.

- **Expansión RPG propio:** En este nodo, el agente coordinador calcula su RPG individualmente. A continuación, transita al subproceso "¿Se han añadido nuevos *fluents*?".

- **¿Se han añadido nuevos *fluents*?:** En este subproceso intervienen todos los agentes. Los agentes verifican si el agente coordinador ha generado *fluents* nuevos en su RPG, y almacenan este hecho en memoria.

  Si el agente coordinador tiene noción de que ningún agente ha añadido nuevos *fluents* al expandir su RPG, el subproceso habrá terminado, y el agente coordinador se dirigirá al nodo "Continuar la ejecución global"

para seguir con la ejecución global del comportamiento.

Sin embargo, si el agente coordinador ha creado nuevos *fluents*, se iniciará el subproceso de "Intercambio de nuevos *fluents* con todos los agentes participantes".

Por último, si no se da ninguno de los dos casos anteriores, se iniciará el subproceso de "Rotación agente coordinador".

- **Intercambio de nuevos *fluents* con todos los agentes participantes:** En este subproceso intervienen tanto todos los agentes participantes como el agente coordinador. Si el agente coordinador ha creado nuevos *fluents*, compartirá con los demás agentes los nuevos *fluents* según la privacidad existente con cada uno. Una vez compartidos los *fluents*, cada agente ubicará sus *fluents* alcanzados en el menor nivel posible de su RPG. Es decir, si un agente $i$ consigue un literal $l$ en el nivel 1 de su grafo, y un agente $j$ lo consigue en el nivel 2, ambos guardarán el *fluent* en el nivel 1.

  Después de este subproceso, el agente coordinador transitará al subproceso "Rotación agente coordinador" mientras que los agentes participantes se quedaran en el nodo WAIT.

- **Rotación agente coordinador:** En este subproceso se realiza la rotación del agente coordinador. Después de que todos los agentes realicen la rotación, el nuevo agente coordinador se dirigirá al nodo "Expansión RPG propio" y los demás agentes participantes se quedarán esperando en el nodo WAIT.

## 4.4. Aplicación del grafo de *landmarks* a la heurística FMAP

El grafo de *landmarks* obtenido tiene como objetivo mejorar la calidad de las estimaciones de FMAP. En el presente trabajo, se ha desarrollado una nueva estrategia de búsqueda heurística que combina la heurística existente en FMAP, $h_{DTG}$, con un nuevo valor heurístico. El nuevo valor heurístico, llamado $h_{land}$, devolverá, para cada plan $\Pi$, el número de *landmarks* simples por alcanzar en dicho plan.

Se considerará que un *landmark* simple ha sido alcanzado cuando, además de ser obtenido, se hayan alcanzado también todos sus predecesores en el grafo de *landmarks*.

Al evaluar un plan Π, se calculan tanto el valor heurístico $h_{land}$ como el valor heurístico $h_{DTG}$ del mismo. La estrategia diseñada ordena los nodos hoja del árbol de búsqueda de FMAP de acuerdo al valor de $h_{land}$. En caso de empate, se prioriza el plan con un menor valor de $f = g + h_{DTG}$. Esta nueva estrategia de búsqueda, por tanto, establece una *priorización* de aquellos planes que han alcanzado un mayor número de *landmarks*.

La versión original de FMAP desarrolla una búsqueda de tipo A, ordenando los nodos de acuerdo a una función $f = g + h_{DTG}$. Sin embargo, la estrategia de priorización de *landmarks* conlleva un comportamiento mucho más voraz por parte de FMAP, ya que el planificador se centra en expandir aquellos nodos hoja del árbol de búsqueda que minimizan el número de *landmarks* por alcanzar. Las consecuencias de este cambio en la estrategia de búsqueda se detallan en la siguiente sección, que compara los resultados obtenidos con ambas versiones de FMAP.

# Capítulo 5

# Resultados experimentales

En este capítulo se comparan los resultados obtenidos con la nueva estrategia heurística implementada, FMAP-LAND, y la versión original de FMAP, que utiliza una heurística basada en DTGs. Para ello, se han utilizado 3 dominios empleados en la Competición Internacional de Planning [1]: *Rovers*, *MA-blocksworld* y *Openstacks* [2].

Estos dominios ofrecen una amplia variedad de problemas multiagente: en algunos casos, los agentes tienen diferentes capacidades, de forma que cada meta sólo puede ser satisfecha por uno de ellos (dominio *Rovers*); en otros problemas, las metas pueden ser alcanzadas por todos los agentes, que tienen las mismas capacidades (dominio *MA-blocksworld*); por último, algunos de los problemas requieren la colaboración de los agentes para alcanzar las metas (dominio *Openstacks*). Los dominios utilizados tienen las siguientes características:

- **Rovers:** Este dominio cuenta con un agente por cada rover. Los rovers recogen muestras de suelo y roca y no interactúan entre sí en la mayoría de los casos. La interacción entre los rovers se limita a que un agente no podrá recoger una muestra de suelo o de la roca si antes un agente diferente ha recogido dicha muestra. El número de agentes varía de 1 a 8 rovers por tarea. Toda información referente a la posición o estado del rover es privada para el agente, sólo la información relacionada con las muestras recogidas es de carácter público.

[1] http://ipc.icaps-conference.org/

[2] Todos los experimentos se han ejecutado en una única máquina con procesador Intel Core i7 de 4 núcleos a 3,40 GHz y 8 GB de memoria RAM (2 GB de RAM asignados a la máquina virtual de Java).

- **MA-Blocksworld:** La versión multiagente de este dominio presenta un conjunto de agentes robot (cuatro agentes por tarea), teniendo cada uno un brazo para organizar bloques. A diferencia del dominio original, la versión de multiagente del dominio Blocksworld permite el manejo de más de un bloque a la vez. Toda la información de las tareas se considera pública.
- **Openstacks:** Este dominio incluye dos agentes especializados en todas las tareas; *manager*, encargado de manejar los pedidos, y *manufacturer*, que controla las diferentes pilas y fabrica los productos. Ambos agentes necesitan la colaboración del otro para llevar a cabo sus actividades, lo que resulta en tareas fuertemente acopladas con objetivos inherentemente cooperativos. Ya que ambos agentes que necesitan compartir información con el otro para alcanzar las metas comunes, la mayor parte de la información relativa a las diferentes pedidos y productos es pública.

Para todos los problemas de cada dominio se han tomado las siguientes medidas:

- ***Makespan*:** Duración de un plan.
- **Iteraciones:** Número de iteraciones requeridas en la etapa de planificación para encontrar la solución del problema.
- **Acciones:** Número total de acciones del plan solución encontrado.
- **Tiempo de ejecución:** Tiempo total de la ejecución de cada problema.
- **Problemas resueltos en el dominio:** Número de problemas resueltos en el dominio.
- ***Landmarks* simples extraídos:** Número de *landmarks* simples extraídos del problema, sin contar las metas.
- ***Landmarks* meta:** Número de metas del problema. Por definición, toda meta es considerado un *landmark* simple.

Para mejorar la legibilidad de los datos, para cada dominio se plasman las medidas de *landmarks* simples extraídos y metas, *makespan*, iteraciones

y acciones en una tabla y la medida del tiempo de ejecución en una gráfica. Si un problema no ha podido ser resuelto se marcarán sus medidas con el símbolo "-" y no se ilustrará en las gráficas de tiempo. Todo problema ha de ser resuelto en menos de 30 minutos. Si un problema sobrepasa el límite de 30 minutos sin hallar una solución se indica como "no resuelto". Todos estos resultados se muestran en las tablas 5.4, 5.2 y 5.3 y en la gráfica 5.1. Además, en la tabla 5.1 se muestran lo valores medios de las medidas mostradas en tablas de las pruebas.

| Dominio | Problemas Resueltos | | Iteraciones | | Acciones | | Makespan | |
|---|---|---|---|---|---|---|---|---|
| | FMAP | FMAP-Land | FMAP | FMAP-Land | FMAP | FMAP-Land | FMAP | FMAP-Land |
| Rovers | 20/20 | 17/20 | 331.59 | 136.76 | 35.94 | 36.06 | 15.47 | 15.65 |
| Openstacks | 19/20 | 20/20 | 320.00 | 65.50 | 54.92 | 55.29 | 43.13 | 44.83 |
| MA-Blocksworld | 23/34 | 28/34 | 2012.32 | 1265.68 | 19.82 | 30.09 | 15.18 | 23.23 |

Tabla 5.1: Resultados medios de los dominios *Rovers*, *Openstacks* y *MA-Blocksworld*

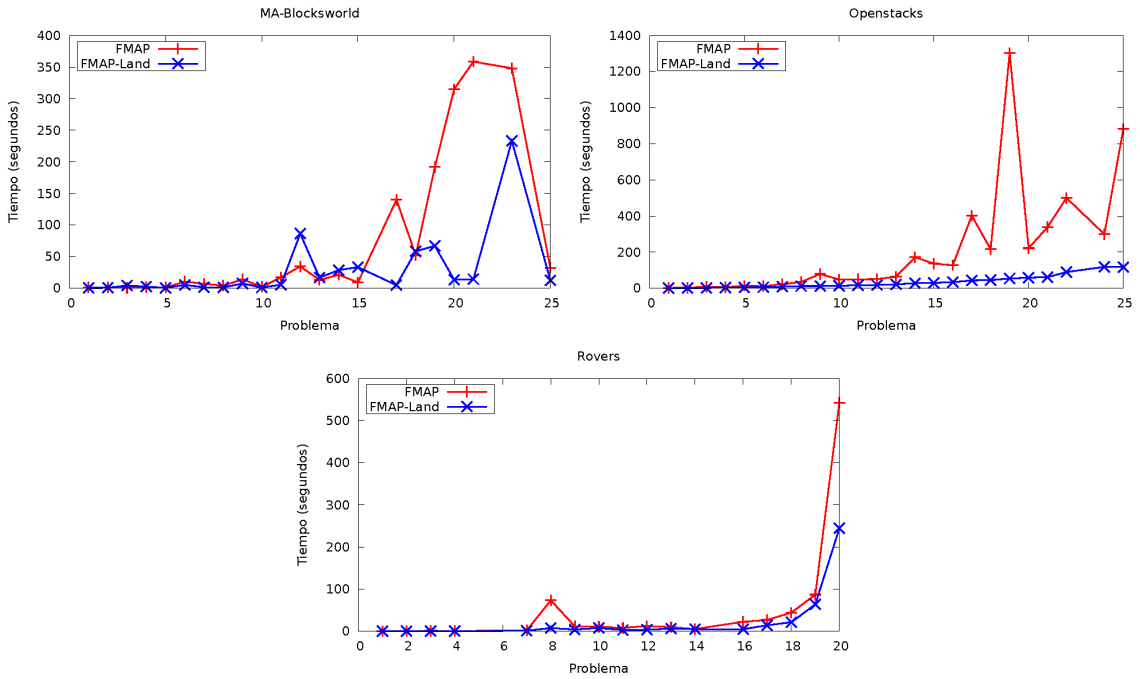


Figura 5.1: Tiempo de ejecución consumido en los dominios *MA-blocksworld*, *Openstacks* y *Rovers*

| Tarea | Ag. | Landmarks | | Makespan | | Iteraciones | | Acciones | |
|---|---|---|---|---|---|---|---|---|---|
| | | Simples | Metas | FMAP | FMAP-Land | FMAP | FMAP-Land | FMAP | FMAP-Land |
| 1 | 2 | 11 | 5 | 14 | 14 | 21 | 20 | 17 | 18 |
| 2 | 2 | 13 | 6 | 16 | 16 | 28 | 21 | 21 | 20 |
| 3 | 2 | 15 | 7 | 19 | 19 | 46 | 24 | 26 | 23 |
| 4 | 2 | 17 | 8 | 22 | 25 | 38 | 37 | 27 | 27 |
| 5 | 2 | 19 | 9 | 24 | 25 | 62 | 33 | 31 | 31 |
| 6 | 2 | 21 | 10 | 27 | 28 | 57 | 39 | 32 | 33 |
| 7 | 2 | 23 | 11 | 29 | 32 | 81 | 46 | 40 | 38 |
| 8 | 2 | 25 | 12 | 33 | 34 | 104 | 54 | 42 | 42 |
| 9 | 2 | 27 | 13 | 32 | 35 | 275 | 50 | 42 | 43 |
| 10 | 2 | 29 | 14 | 37 | 37 | 178 | 51 | 45 | 47 |
| 11 | 2 | 31 | 15 | 38 | 41 | 115 | 64 | 49 | 49 |
| 12 | 2 | 33 | 16 | 42 | 45 | 162 | 59 | 52 | 52 |
| 13 | 2 | 35 | 17 | 45 | 46 | 152 | 61 | 56 | 55 |
| 14 | 2 | 37 | 18 | 46 | 45 | 310 | 63 | 58 | 60 |
| 15 | 2 | 39 | 19 | 47 | 51 | 216 | 68 | 65 | 65 |
| 16 | 2 | 41 | 20 | 52 | 53 | 239 | 74 | 67 | 66 |
| 17 | 2 | 43 | 21 | 53 | 57 | 648 | 82 | 70 | 73 |
| 18 | 2 | 45 | 22 | 58 | 60 | 428 | 84 | 72 | 72 |
| 19 | 2 | 47 | 23 | 58 | 61 | 1844 | 90 | 76 | 75 |
| 20 | 2 | 49 | 24 | 61 | 63 | 326 | 88 | 79 | 80 |
| 21 | 2 | 51 | 25 | 66 | 65 | 464 | 92 | 79 | 82 |
| 22 | 2 | 53 | 26 | 69 | 71 | 649 | 110 | 84 | 88 |
| 23 | 2 | 55 | 27 | - | 71 | - | 108 | - | 88 |
| 24 | 2 | 57 | 28 | 73 | 77 | 321 | 160 | 92 | 90 |
| 25 | 2 | 59 | 29 | 74 | 76 | 916 | 102 | 96 | 98 |
| 26 | 2 | 61 | 30 | 75 | 82 | 1513 | 114 | 97 | 99 |
| 27 | 2 | 63 | 31 | 79 | 82 | 4416 | 126 | 101 | 104 |
| 28 | 2 | 65 | 32 | 84 | 82 | 4610 | 133 | 104 | 107 |
| 29 | 2 | 67 | 33 | 87 | 89 | 1868 | 162 | 108 | 109 |
| 30 | 2 | 69 | 34 | 89 | 94 | 4940 | 144 | 109 | 111 |
| **Media** | | | | **43.13** | **44.83** | **320.00** | **65.50** | **54.92** | **55.29** |

Tabla 5.2: Resultados de los problemas del dominio *Openstacks*

| Tarea | Ag. | Landmarks | | Makespan | | Iteraciones | | Acciones | |
|---|---|---|---|---|---|---|---|---|---|
| | | Simples | Metas | FMAP | FMAP-Land | FMAP | FMAP-Land | FMAP | FMAP-Land |
| 4-0 | 4 | 0 | 3 | 5 | 8 | 26 | 12 | 6 | 8 |
| 4-1 | 4 | 3 | 3 | 6 | 9 | 28 | 21 | 8 | 12 |
| 4-2 | 4 | 1 | 3 | 4 | 13 | 11 | 191 | 6 | 16 |
| 5-0 | 4 | 3 | 4 | 9 | 15 | 36 | 109 | 10 | 20 |
| 5-1 | 4 | 2 | 4 | 8 | 8 | 28 | 9 | 8 | 8 |
| 5-2 | 4 | 4 | 4 | 12 | 21 | 293 | 247 | 14 | 30 |
| 6-0 | 4 | 4 | 5 | 10 | 17 | 174 | 48 | 12 | 20 |
| 6-1 | 4 | 1 | 5 | 12 | 12 | 97 | 29 | 14 | 14 |
| 6-2 | 4 | 5 | 5 | 13 | 24 | 387 | 359 | 20 | 28 |
| 7-0 | 4 | 6 | 6 | 14 | 16 | 46 | 31 | 18 | 20 |
| 7-1 | 4 | 5 | 6 | 16 | 17 | 471 | 235 | 20 | 22 |
| 7-2 | 4 | 5 | 6 | 17 | 27 | 1007 | 4850 | 24 | 44 |
| 8-0 | 4 | 4 | 7 | 17 | 22 | 346 | 832 | 18 | 30 |
| 8-1 | 4 | 4 | 7 | 11 | 33 | 590 | 1429 | 18 | 40 |
| 8-2 | 4 | 3 | 7 | 13 | 24 | 226 | 1805 | 20 | 32 |
| 9-0 | 4 | 7 | 8 | - | 43 | - | 8451 | - | 54 |
| 9-1 | 4 | 8 | 8 | 23 | 24 | 4079 | 125 | 26 | 30 |
| 9-2 | 4 | 7 | 8 | 21 | 44 | 1486 | 2663 | 30 | 60 |
| 10-1 | 4 | 8 | 9 | 21 | 33 | 5138 | 3364 | 32 | 46 |
| 10-2 | 4 | 8 | 9 | 26 | 27 | 8970 | 369 | 30 | 34 |
| 11-0 | 4 | 8 | 10 | 20 | 36 | 10135 | 519 | 32 | 44 |
| 11-1 | 4 | 7 | 10 | - | 38 | - | 2926 | - | 46 |
| 11-2 | 4 | 9 | 10 | 28 | 48 | 9924 | 10287 | 36 | 58 |
| 12-0 | 4 | 9 | 11 | - | 49 | - | 12708 | - | 72 |
| 12-1 | 4 | 10 | 11 | 28 | 33 | 773 | 311 | 34 | 46 |
| 13-0 | 4 | 10 | 12 | - | 53 | - | 4672 | - | 72 |
| 13-1 | 4 | 11 | 12 | - | 51 | - | 23194 | - | 74 |
| 14-0 | 4 | 11 | 13 | 29 | - | 13021 | - | 38 | - |
| 14-1 | 4 | 9 | 13 | - | 54 | - | 16419 | - | 70 |
| 15-0 | 4 | 9 | 14 | - | - | - | - | - | - |
| 15-1 | 4 | 10 | 14 | - | - | - | - | - | - |
| 16-1 | 4 | 13 | 15 | - | - | - | - | - | - |
| 16-2 | 4 | 13 | 15 | - | - | - | - | - | - |
| 17-1 | 4 | 14 | 16 | - | - | - | - | - | - |
| **Media** | | | | **15.18** | **23.23** | **2012.32** | **1265.68** | **19.82** | **30.09** |

Tabla 5.3: Resultados de los problemas del dominio *MA-Blocksworld*

| Tarea | Ag. | Landmarks | | Makespan | | Iteraciones | | Acciones | |
|---|---|---|---|---|---|---|---|---|---|
| | | Simples | Metas | FMAP | FMAP-Land | FMAP | FMAP-Land | FMAP | FMAP-Land |
| 1 | 1 | 16 | 3 | 8 | 8 | 11 | 11 | 10 | 10 |
| 2 | 1 | 13 | 3 | 4 | 4 | 9 | 9 | 8 | 8 |
| 3 | 2 | 5 | 3 | 9 | 9 | 22 | 14 | 14 | 13 |
| 4 | 2 | 4 | 3 | 4 | 5 | 9 | 9 | 8 | 8 |
| 5 | 2 | 7 | 7 | 8 | - | 31 | - | 22 | - |
| 6 | 2 | 15 | 10 | 14 | - | 75 | - | 38 | - |
| 7 | 3 | 0 | 6 | 7 | 7 | 25 | 25 | 19 | 19 |
| 8 | 4 | 0 | 8 | 8 | 9 | 1352 | 186 | 28 | 27 |
| 9 | 4 | 7 | 8 | 18 | 15 | 153 | 53 | 34 | 32 |
| 10 | 4 | 0 | 11 | 20 | 18 | 82 | 71 | 38 | 38 |
| 11 | 4 | 8 | 9 | 22 | 16 | 129 | 41 | 35 | 32 |
| 12 | 4 | 0 | 6 | 15 | 9 | 226 | 92 | 24 | 25 |
| 13 | 4 | 0 | 12 | 25 | 29 | 162 | 120 | 47 | 46 |
| 14 | 4 | 6 | 8 | 19 | 11 | 72 | 79 | 31 | 30 |
| 15 | 4 | 6 | 10 | 23 | - | 197 | - | 44 | - |
| 16 | 4 | 0 | 11 | 17 | 26 | 408 | 85 | 46 | 45 |
| 17 | 6 | 8 | 13 | 18 | 15 | 161 | 86 | 53 | 58 |
| 18 | 6 | 0 | 11 | 19 | 16 | 431 | 252 | 50 | 52 |
| 19 | 6 | 0 | 17 | 26 | 26 | 464 | 414 | 71 | 73 |
| 20 | 8 | 0 | 20 | 24 | 43 | 1921 | 778 | 95 | 97 |
| **Media** | | | | **15.47** | **15.65** | **331.59** | **136.76** | **35.94** | **36.06** |

Tabla 5.4: Resultados de los problemas del dominio *Rovers*

En general, dados los resultados de los tres dominios, cuantos más *landmarks* simples se extraigan en un problema mejor guiará la heurística FMAP-Land la búsqueda del plan solución. Se observa también que la calidad de la información que ofrecen los grafos de *landmarks* extraídos aumenta o disminuye según el tipo de dominio y que existen casos especiales en los que la heurística FMAP ha tomado un comportamiento diferente al previsto.

## 5.1. Análisis de la información aportada por los grafos de *landmarks*

En esta sección se muestra como varía la calidad de la información aportada por los grafos de *landmarks* según el dominio, y los factores que la condicionan.

En dominios donde los agentes tienen que interactuar entre ellos para alcanzar las metas del problema, el grafo de *landmarks* proporciona al planificador un esqueleto del plan solución a construir. Un claro ejemplo de este hecho es el problema número 1 del dominio *Openstacks*. La figura 5.2 muestra el grafo de *landmarks* del problema, que es idéntico para los dos agentes participantes en la tarea, el *Manager* y el *Manufacturer*. Los *landmarks* simples no numerados se consiguen en el estado inicial.

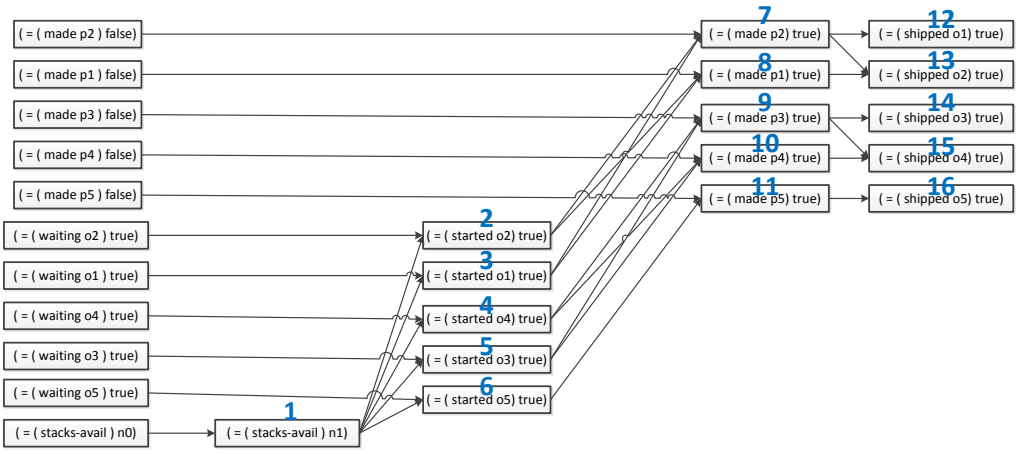


Figura 5.2: Grafo de landmarks de los agentes Manager y Manufacturer del problema 1 del dominio *Openstacks*

La siguiente figura 5.3 muestra el plan solución obtenido por FMAP-Land para el problema 1 del dominio *Openstacks*.

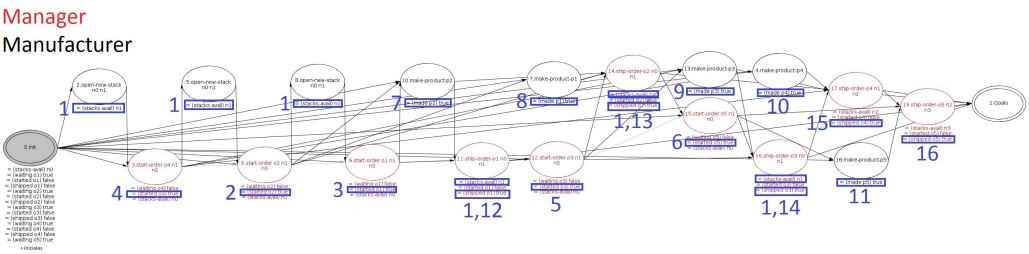


Figura 5.3: Plan solución del problema 1 del dominio *Openstacks*

Como se puede observar en la figura 5.3, el plan solución obtenido por FMAP-Land alcanza los *landmarks* simples en el mismo orden que aparecen en el grafo de *landmarks*. Por ejemplo, para poder alcanzar el *landmark* número 7, el plan alcanza antes los *landmarks* número 2 y 3, tal como indica el grafo de *landmarks*.

En este tipo de dominios, los agentes extraen un número elevado de *landmarks* simples, ya que cada agente tiene capacidades únicas y se requiere cooperación para alcanzar las metas. Los *landmarks* simples y los ordenes necesarios indican al *planner* de forma prácticamente unívoca la siguiente acción a aplicar, que será aquella que alcance el siguiente *landmark* en el grafo del agente.

En el ejemplo anterior, la primera acción que debería añadir el plan vacío, considerando que los *landmarks* no numerados se consiguen en el estado inicial, es la que produzca el *landmark* número 1. Después de esta acción, se deberá añadir al plan una acción que produzca alguno de los *landmarks* sucesores del *landmark* número 1, esto es, los *landmarks* 2,3,4,5 ó 6. Gracias a esta información, FMAP-Land alcanza los *landmarks* del problema ejemplo mostrado a razón de un *landmark* por acción añadida.

En dominios en los que varios agentes tienen las mismas capacidades y pueden conseguir cualquier meta, como el dominio *MA-Blocksworld*, la información aportada por los grafos de *landmarks* no es tan potente como en el caso anterior. En este tipo de dominios, los *landmarks* simples extraídos están mucho más inconexos, creando una serie de "metas intermedias" que han de ser alcanzadas para estar más cerca de poder alcanzar las metas del problema. Estas "metas intermedias" no proporcionan un esqueleto del plan solución como en el caso anterior, pero permiten guíar la planificación de forma efectiva, como mostrarán los resultados del dominio *MA-Blocksworld*. A continuación, en la figura 5.4, se verá un ejemplo basado en el problema 4-1 del dominio *MA-Blocksworld*. La figura 5.4 ilustra una fragmento del grafo de *landmarks* obtenido por el agente `r0` del problema 4-1 del dominio *MA-Blocksworld*.

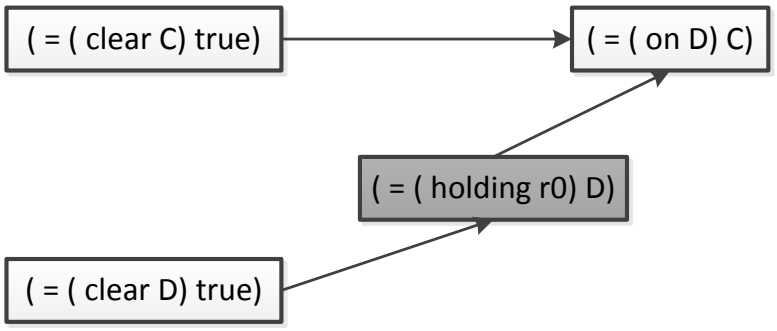


Figura 5.4: Fragmento del grafo de *landmarks* del agente `r0` del problema 4-1 del dominio *MA-Blocksworld*

En esta figura se observa que la meta `(= (on D) C)`, que indica que el bloque D ha de estar encima del bloque C, requiere de dos hechos para poder ser alcanzada: que el bloque C no tenga ningún bloque encima y que un robot tenga cogido el bloque D. Es decir, para que el *landmark* meta, `(= (on D) C)`, pueda ser alcanzado, se necesita que el *landmark* simple `( = (clear C) true)` y el *landmark* disyuntivo `( = (holding r0) D)` (este

*landmark* puede ser obtenido por cualquiera de los robots `(= (holding r1) D)`, `(= (holding r2) D)`, etc) sean alcanzados antes. Además, se observa como, para que algún robot coja el bloque D, dicho bloque no ha de tener ningún bloque encima. Es decir, para poder alcanzar algún *fluent* de este *landmark* disyuntivo es necesario antes alcanzar el *landmark* `( = (clear D) true)`. Dado que FMAP-Land, como el resto de planificadores basados en *landmarks*, sólo considera los *landmarks* simples, no se tiene en cuenta el hecho de que algún agente sostenga el bloque D. Así pues, en este caso, para poder alcanzar la meta `(= (on D) C)` se priorizará antes la consecución de la "meta intermedia" `( = (clear D) true)`.

Por último, algunos dominios multiagente, como el dominio *Rovers*, tienen características mixtas, de forma que en algunos problemas los agentes pueden o no tener capacidades diferenciadas. Por tanto, en función de las características del problema, cada meta podrá ser conseguida por un único agente, un subconjunto de estos, o todos los agentes que intervienen en la tarea. De esta forma, en el dominio *Rovers*, sólo se extraen *landmarks* simples cuando hay un único rover en el problema capacitado para resolver una meta concreta. Por ejemplo, esto ocurre cuando se dispone de un único rover capacitado para analizar una muestra de tierra. De esta forma, FMAP-Land guiará la planificación de forma más efectiva cuando el grafo de *landmarks* disponga de un número significativo de *landmarks* simples (a priori tan bien como en el dominio *OpenStacks*). Sin embargo, el rendimiento de FMAP-Land puede verse comprometido en aquellos problemas en los que no se consigan *landmarks* simples al margen de las metas.

En este dominio, dado que se tiene que cumplir una condición especial para que existan *landmarks* simples, en muchos de los problemas no se obtienen *landmarks* de este tipo.

## 5.2. Análisis de los resultados

En esta sección se detalla la comparativa entre la heurística FMAP-Land y FMAP, el análisis de los resultados para cada uno de los dominios y los casos especiales observados en los resultados.

Como se ha expuesto en la sección 4.4, la versión original de FMAP realiza una búsqueda de tipo A, seleccionando en cada iteración el plan que minimiza la función $f = g + h_{DTG}$, donde $g$ es el número de acciones del plan, y $h_{DTG}$ es el resultado de la aplicación de la heurística basada en diagramas

de transición de dominio. Por su parte, FMAP-Land prioriza los planes que minimizan $h_{land}$ (aquellos con un mayor número de *landmarks* alcanzados), y aplica $f$ para desempatar planes con el mismo valor de $h_{land}$. Por este motivo, FMAP-Land muestra un comportamiento más voraz que FMAP, lo que condiciona en parte los resultados obtenidos. Los resultados de los tres dominios analizados muestran que, en general, FMAP-Land mejora notablemente su rendimiento en los problemas con un mayor número de *landmarks* simples extraídos.

En aquellos casos en los que no se extrae ningún *landmark* simple, FMAP-Land ofrece un rendimiento similar a FMAP. En estos casos, FMAP-Land se comporta de forma equivalente a FMAP hasta generar un plan que alcance alguna de las metas. A partir de este punto, FMAP-Land da prioridad a los planes que alcanzan un mayor número de metas del problema, esto es, los planes que minimizan $h_{land}$.

Respecto a los dominios analizados, FMAP-Land muestra un mejor comportamiento en dominios donde es necesaria la interacción entre los agentes para alcanzar las metas. Concretamente, FMAP-Land muestra un rendimiento muy superior a FMAP en el dominio *Openstacks*, debido a la gran cantidad y calidad de *landmarks* obtenidos en cada problema. Como se ha descrito en la sección anterior, los *landmarks* en este dominio guían la búsqueda de forma muy precisa, indicando de forma prácticamente unívoca la siguiente acción a añadir en cada punto del proceso de planificación.

Como se aprecia en la tabla 5.2, FMAP-Land resuelve los problemas del dominio *Openstacks* en alrededor de 65 iteraciones de media, por 320 iteraciones de media en el caso de FMAP. Esto se traduce en tiempos de ejecución mucho más bajos (ver figura 5.1), y una cobertura superior (20 problemas resueltos por 19 en el caso de FMAP).

Respecto al dominio *MA-Blocksworld*, la sección 5.1 muestra que los *landmarks* simples obtenidos no aportan una información tan valiosa para guiar la búsqueda como en el caso de *Openstacks*. En este caso, la mayor parte de *landmarks* simples extraídos en los problemas son del tipo `(=(clear B1)true)`, lo que indica que un bloque B1 no tiene ningún bloque encima. Este hecho puede conseguirse con muchas acciones diferentes, como son: apilar el bloque B1 en otro bloque B2, apilar el bloque B1 en la mesa y desapilar un bloque B2 que está justo encima de B1. Por tanto, a diferencia del dominio *Openstacks*, los *landmarks* en el dominio *MA-Blocksworld* no determinan las acciones que debe añadir el planificador, dado que hay diversas acciones

que generan estos *landmarks* como efectos. En conclusión, en este dominio los *landmarks* no resultan tan útiles para guiar la búsqueda como en el caso del dominio *Openstacks*.

Pese a esta limitación, la tabla 5.3 muestra que el comportamiento de FMAP-Land es superior a FMAP, obteniendo un número medio de iteraciones claramente inferior (1265 por 2012), y tiempos de ejecución mucho menores en los problemas de mayor complejidad. La cobertura es claramente favorable a FMAP-Land, que resuelve 28 problemas, por 23 en el caso de FMAP.

Finalmente, los *landmarks* obtenidos en el dominio *Rovers* demuestran ser un recurso útil para guiar la búsqueda. Sin embargo, como se explicó en la sección anterior, las características de los problemas de este dominio condicionan el número de *landmarks* obtenido. Por ello, en algunos problemas, el algoritmo de extracción de *landmarks* no encuentra ningún *landmark* simple, al margen de las metas.

La tabla 5.4 muestra que FMAP-land requiere un número medio de iteraciones muy inferior a FMAP (136 por 331), lo que redunda en tiempos de ejecución inferiores (ver figura 5.1). Sin embargo, en este caso FMAP resuelve un mayor número de problemas que FMAP-Land (20 por 17).

Respecto a la calidad de las soluciones, únicamente en el dominio *MA-Blocksworld* se observan diferencias entre FMAP y FMAP-Land. En este caso, la heurística FMAP-Land extrae planes de peor calidad. Este hecho es debido a la débil guía que ofrecen los *landmarks* extraídos en ese dominio y a la estrategia voraz de la heurística de FMAP-Land. Al aplicar una estrategia de tipo A, FMAP expande planes de distintas zonas del árbol de búsqueda, mientras que FMAP-Land se limita a expandir los nodos que minimizan $h_{land}$. Ello redunda también en soluciones de mejor calidad para FMAP en *MA-Blocksworld*.

Finalmente, conviene reseñar algunos casos particulares en los que FMAP-Land no encuentra solución, a pesar de extraerse un número significativo de *landmarks* simples. Concretamente, se aprecia este comportamiento en los dominios *Rovers* y *MA-Blocksworld* (ver tablas 5.4 y 5.3). Este hecho es debido la estrategia voraz de FMAP-Land, que prioriza los nodos con menor número de *landmarks* por alcanzar. Por esta razón, puede que la búsqueda se fije en una rama del árbol donde pueden desarrollarse múltiples planes sucesores (introduciendo acciones reversibles) que no llegan a alcanzar una

solución. De esta forma, en estos casos se explora el subárbol conformado por los planes que minimizan $h_{land}$ sin llegar a alcanzar planes solución.

A pesar de ese inconveniente, y a tenor de los resultados de los tres dominios en cuanto a iteraciones y tiempos de ejecución, se observa que FMAP-Land actúa con más rapidez que FMAP, ofreciendo soluciones de calidad similar en cuanto a acciones y makespan.

Por último, existen casos en el dominio *MA-Blocksworld* en los que ninguna de las dos aproximaciones encuentran un plan solución. En estos casos, la estrategia voraz que sigue FMAP-Land no ha sido suficiente para lograr una solución o se ha dado el caso de que la búsqueda se ha fijado en una rama del árbol donde se han desarrollado múltiples planes sucesores que no llegan a progresar hacia una solución.

# Capítulo 6

# Conclusiones y trabajo futuro

En este capítulo se detalla un resumen del trabajo realizado, de las conclusiones extraídas del mismo y de las líneas de trabajo futuro abiertas.

## 6.1. Resumen del trabajo realizado

En este trabajo se ha diseñado e implementado una heurística basada en *landmarks* para el planificador FMAP integrado en PlanInteraction. Para dotar de información sobre los *landmarks* de un problema a la nueva heurística, se diseñó un algoritmo multiagente de extracción de *landmarks*, posteriormente implementado en un comportamiento de PlanInteraction. Este comportamiento, llamado *Landmarks-Mining*, preserva la privacidad de los agentes durante el proceso de extracción multiagente de *landmarks* de un problema. Cada grafo de *landmarks* es almacenado internamente por cada agente, y, durante la planificación, se utilizan estos grafos para calcular un nuevo valor heurístico $h_{land}$ (número de *landmarks* simples por alcanzar en un plan).

El valor, $h_{land}$, junto al ya implementado en FMAP, $h_{DTG}$, son los elementos básicos de la nueva heurística implementada, FMAP-Land. En dicha heurística, al evaluar un plan $\Pi$, se calculan ambos valores heurísticos, y la estrategia diseñada ordena los nodos hoja de acuerdo al valor de $h_{land}$. En caso de empate, se priorizan los planes que minimicen $g + h_{DTG}$, donde $g$ es el número de acciones del plan.

Para validar el trabajo desarrollado, se realizó una amplia batería de pruebas en tres dominios adaptados de la competición Internacional de Planificación [1], comparando el rendimiento de la versión original de FMAP y la nueva estrategia heurística implementada en FMAP-Land.

## 6.2. Conclusiones

Las conclusiones extraídas del presente trabajo son las siguientes: el uso de la plataforma PlanInteraction ha facilitado la implementación del trabajo, dada su capacidad de modelización del flujo de ejecución para varios agentes.

En cuanto a los resultados de la nueva heurística FMAP-Land, se observa como, aunque la calidad de las soluciones es similar a la calidad de las extraídas por FMAP, el rendimiento de FMAP-Land en cuanto a tiempo de ejecución y número de iteraciones ha mejorado significativamente (en función de la cantidad de *landmarks* simples encontrados en cada problema y de la calidad de información que dan éstos en cada tipo de dominio).

A pesar de las evidentes mejoras observadas en los tres dominios de planificación analizados, se observaron algunos defectos de la heurística FMAP-Land debido a su estrategia voraz. Esta voracidad causa, en algunos casos, que el planificador se centre en desarrollar una zona concreta del árbol de búsqueda donde los planes sucesores obtenidos no llegan a progresar hacia una solución.

## 6.3. Trabajo Futuro

De entre las futuras líneas de trabajo a seguir, se destacan las siguientes:

- Cambiar la estrategia de búsqueda, priorizando esta vez los nodos hijo con menor valor $g + h_{DTG}$, y en caso de empate, priorizar los planes con menor número de *landmarks* simples por alcanzar ($h_{land}$).

- Mantener dos ordenaciones de planes independientes, de modo que el planificador mantendría en todo momento dos colas de planes, ordenadas según $g + h_{DTG}$ y $h_{land}$. El planificador utilizaría en primera instancia la cola ordenada por el valor $g + h_{DTG}$ y, cuando el proceso de búsqueda se encuentre en un *plateau* (esto es, cuando el proceso de

[1] http://ipc.icaps-conference.org/

búsqueda no consiga reducir el valor de $g + h_{DTG}$ durante un número significativo de iteraciones), utilizaría la cola ordenada por el valor $h_{land}$ hasta salir del *plateau*. Esta estrategia ha sido implementada en planificadores secuenciales como LAMA [RW10].

# Bibliografía


[BF97] A. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[BG01] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

[BN95] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[BW94] A. Barrett and D. S. Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[EHN94] K. Erol, J. Hendler, and D. Nau. UCMP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 249–254, 1994.

[FN71] R. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971.

[Gef00] H. Geffner. Functional strips: a more flexible language for planning and problem solving. pages 187–209, 2000.

[GHK+98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *AIPS-98 Planning Committee*, 1998.

[GNT04] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning. Theory and Practice*. Morgan Kaufmann, 2004.

[HDR08] M. Helmert, M. Do, and I Refanidis. Ipc 2008, deterministic part. 2008.

[Hel06] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26(1):191–246, 2006.

[HN01] J. Hoffmann and B. Nebel. The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[HPS04] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)*, 22:215–278, 2004.

[KS96] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 1194–1201, 1996.

[PW92] J.S. Penberthy and D.S. Weld. UCPOP: A sound, complete, partial order planner for ADL. *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114, 1992.

[RW10] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

[Sap05] O. Sapena. *Planificación Independiente del Dominio en Entornos Dinámicos de Tiempo Restringido*. PhD thesis, Universidad Politécnica de Valencia, 2005.

[SO03] Laura Sebastià and Eva Onaindía. Descomposición y resolución concurrente de problemas de planificación independiente del dominio. 2003.

[Wel94] D.S. Weld. An introduction to least commitment planning. *AI magazine*, 15(4):27, 1994.

[Wel99] D.S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[ZNK07] J.F. Zhang, X.T. Nguyen, and R. Kowalczyk. Graph-based multi-agent replanning algorithm. In *Proceedings of the 6th Conference on Autonomous Agents and Multiagent Systems*, 2007.