

Maestría en Inteligencia Artificial

Razonamiento y Planificación Automática

Índice

Esquema. Tema 1	3
Ideas clave. Tema 1	4
1.1. ¿Cómo estudiar este tema?	4
1.2. Problemas de toma de decisiones	4
1.3. Arquitectura de un agente inteligente	15
1.4. Tipos de agentes inteligentes	30
1.5. Referencias bibliográficas	32
Esquema. Tema 2	36
Ideas clave. Tema 2	37
2.1. ¿Cómo estudiar este tema?	37
2.2. Técnicas de representación simbólica	38
2.3. Clases de conocimiento	41
2.4. Tipos de razonamiento	43
2.5. Razonamiento lógico deductivo	48
2.6. Razonamiento lógico inductivo	50
2.7. Razonamiento lógico abductivo	52
2.8. Referencias bibliográficas	55
Esquema. Tema 3	58
Ideas clave. Tema 3	59
3.1. ¿Cómo estudiar este tema?	59
3.2. Tipos de lógica	59
3.3. Lógica matemática	65
3.4. Lógica de descripción ALC	70
3.5. Lógica de orden superior	76
3.6. Lógica multivaluada y lógica difusa	78
3.7. Referencias bibliográficas	80
Esquema. Tema 4	82
Ideas clave. Tema 4	83
4.1. ¿Cómo estudiar este tema?	83

4.2. Descripción general de un problema de búsqueda	84
4.3. Búsqueda en amplitud	96
4.4. Búsqueda en profundidad	99
4.5. Búsqueda de coste uniforme	101
4.6. Referencias bibliográficas	104
 Esquema. Tema 5	 106
 Ideas clave. Tema 5	 107
5.1. ¿Cómo estudiar este tema?	107
5.2. Tipos de heurísticas	108
5.3. Búsqueda A*	114
5.4. Búsqueda por subobjetivos	119
5.5. Búsqueda <i>online</i>	123
5.6. Referencias bibliográficas	128
 Esquema. Tema 6	 131
 Ideas clave. Tema 6	 132
6.1. ¿Cómo estudiar este tema?	132
6.2. Introducción	132
6.3. Búsqueda minimax	137
6.4. La poda alfa-beta	142
6.5. Búsqueda expectiminimax	146
6.6. Referencias bibliográficas	151
 Esquema. Tema 7	 153
 Ideas clave. Tema 7	 154
7.1. ¿Cómo estudiar este tema?	154
7.2. ¿Qué es un problema de planificación?	154
7.3. Tipos de problemas de planificación	156
7.4. Planificadores de orden total y de orden parcial	160
7.5. Referencias bibliográficas	167
 Esquema. Tema 8	 169
 Ideas clave. Tema 8	 170

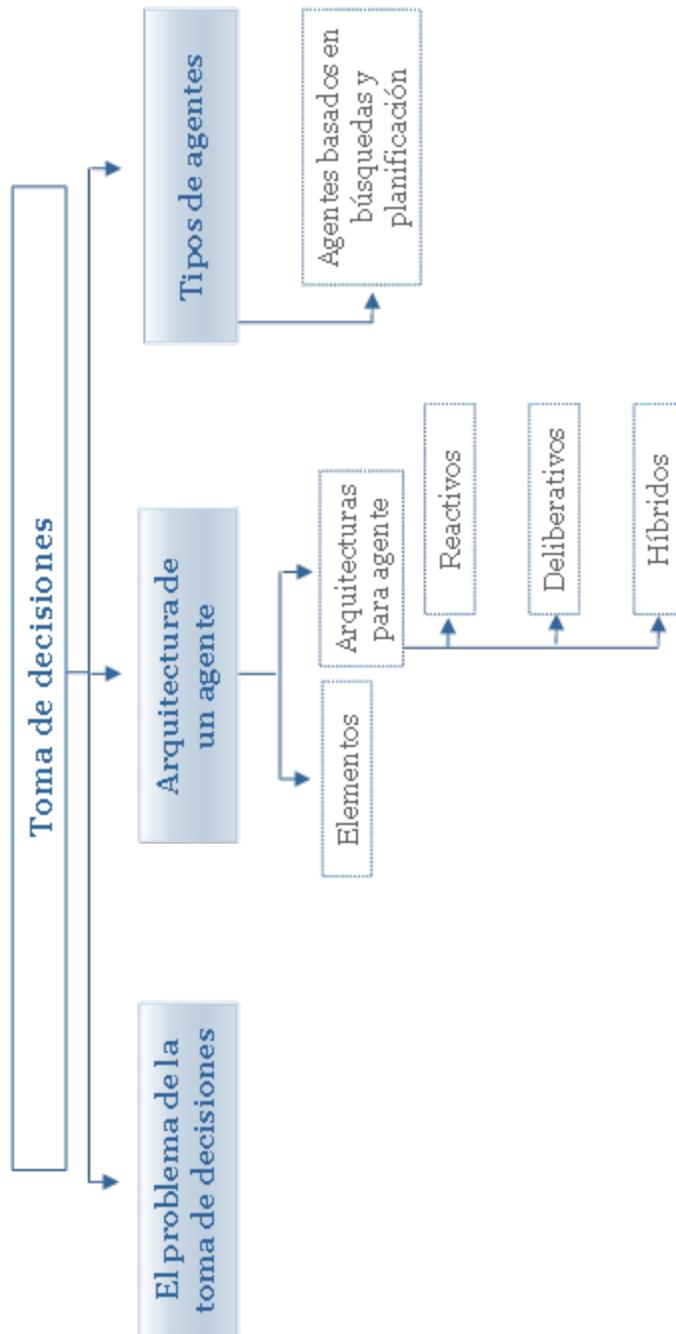
8.1. ¿Cómo estudiar este tema?	170
8.2. STRIPS	170
8.3. PDDL	180
8.4. GOAP	183
8.5. Referencias bibliográficas	189
Esquema. Tema 9	192
Ideas clave. Tema 9	193
9.1. ¿Cómo estudiar este tema?	193
9.2. Definición	193
9.3. Planificación por medio de red de tareas	195
9.4. Descomposición jerárquica	197
9.5. Referencias bibliográficas	200
Esquema. Tema 10	203
Ideas clave. Tema 10	204
10.1. ¿Cómo estudiar este tema?	204
10.2. ¿Qué es la planificación multi agente?	204
10.3. Planificación multi agente antes de la ejecución	206
10.4. Planificación multi agente durante ejecución	213
10.5. Referencias bibliográficas	214
Esquema. Tema 11	218
Ideas clave. Tema 11	219
11.1. ¿Cómo estudiar este tema?	219
11.2. Planificadores distribuidos	219
11.3. Formalización	221
11.4. Protocolos	225
11.5. FMAP	226
11.6. Referencias bibliográficas	229

Esquema. Tema 12	232
Ideas clave. Tema 12	233
12.1. ¿Cómo estudiar este tema?	233
12.2. Estructuras de reparación reactiva	233
12.3. Dificultades en la reparación multi agente	235
12.4. Búsqueda multi agente durante la ejecución	236
12.5. Referencias bibliográficas	240

Razonamiento y Planificación Automática

Introducción a la toma de decisiones

Esquema. Tema 1



Ideas clave. Tema 1

1.1. ¿Cómo estudiar este tema?

En este tema estudiaremos los principios necesarios para entender el problema de la toma de decisiones para los sistemas inteligentes. Describiremos la problemática humana de la toma de decisiones y su influencia al construir agentes inteligentes.

Partiremos de las estrategias mentales que pueden ayudar a estructurar la toma de decisiones y plantearemos los mecanismos tradicionales para estructurar los problemas, dividiendo en etapas el proceso de definición del problema.

Completaremos los conceptos iniciales con la definición de **agente inteligente** y de los tipos de agentes inteligentes que nos podemos encontrar según su funcionalidad o servicio que ofrece. Presentaremos los distintos elementos que componen estas arquitecturas y su utilidad.



Accede a los ejercicios de autoevaluación a través del aula virtual

1.2. Problemas de toma de decisiones



Accede al vídeo «Problemas de toma de decisiones» a través del aula virtual

La complejidad en el proceso de toma de decisiones es evidente, ya que es necesario evaluar y conocer las condiciones del entorno. Como guías podemos emplear la intuición y las reglas, pero en ocasiones es necesario realizar un análisis exhaustivo de la totalidad de los posibles escenarios. Es en este contexto cuando las matemáticas aplicadas a la toma de decisiones pueden servir de ayuda.

Todos tenemos dificultades para tomar decisiones, especialmente cuando la decisión debe tomarse frente a una situación de riesgo. En este tipo de casos, la intuición no suele ser una herramienta y las matemáticas son un buen recurso para ayudar a tomar decisiones.

En la actualidad, existen varios desafíos que generan situaciones de riesgo (terrorismo, crisis económico-financieras, ciberseguridad, cambio climático...) y la toma de decisiones en estos casos es muy importante.

Cuando una situación necesita ser analizada y una decisión tomada, es necesario asumir un riesgo; por tanto, es necesario evaluar e identificar el riesgo involucrado y decidir cuáles son las medidas a adoptar para que este sea mínimo. La capacidad que tenemos para tomar decisiones está relacionada con la asunción de riesgo, la creatividad y la búsqueda de alternativas a los retos que aún no existen.

Los elementos para tener en cuenta en la toma de decisiones son:

- ▶ Efecto futuro: cómo afectará la decisión en el futuro. Cualquier decisión influye: algunas lo hacen a corto plazo (decisiones de bajo nivel), pero otras influyen a largo plazo (decisiones de alto nivel).
- ▶ Reversibilidad: es el esfuerzo necesario para revertir una decisión y la velocidad con la que una decisión puede revertirse. Cuando es difícil revertir, es mejor realizar decisiones a alto nivel. Si es simple, será necesario tomar decisiones a bajo nivel.
- ▶ Impacto: está relacionado con otras actividades y áreas que se ven afectadas por la decisión tomada. El impacto puede ser: extenso (abarca varias áreas o actividades y es recomendable aplicar decisiones a alto nivel) o localizado, único (solo afecta a una o pocas áreas o actividades y se asocia con una decisión de bajo nivel).
- ▶ Calidad: la ética, la legalidad, los principios que gobiernan el comportamiento, la imagen y las relaciones laborales, entre otras, entran en juego en este punto. Cuando muchos de estos factores están involucrados, es necesario tomar

decisiones a alto nivel. En cambio, cuando hay pocos implicados, es recomendable tomar la decisión a bajo nivel.

- ▶ Periodicidad: este elemento se refiere a la frecuencia o excepcionalidad de una decisión. Las decisiones excepcionales son de alto nivel, mientras que las decisiones que deben ser tomadas con mayor frecuencia son decisiones de bajo nivel.

La toma de decisiones a **alto nivel** requiere seguir un proceso de **análisis serio, buscar alternativas, planificar, ejecutar y evaluar**. La toma de decisiones a **bajo nivel** implica **poco esfuerzo** y es posible tomar este tipo de decisiones en **poco tiempo**. En el mundo de planning, las decisiones de alto nivel se toman con planificadores deliberativos, es decir planificadores que pueden emplear todo el tiempo necesario para encontrar una solución. Por el contrario, en las decisiones a bajo nivel se emplean planificadores reactivos. Los cuales son planificadores que deben encontrar una solución en un tiempo muy corto.

Resumen de elementos de toma de decisiones de alto nivel y decisiones de bajo nivel:

Alto Nivel	Bajo Nivel
Afectan al futuro.	No afectan al futuro.
Reversibilidad difícil.	Reversibles.
Impacto amplio.	Poco impacto.
Afectan a muchos factores importantes de calidad.	Afectan a pocos factores importantes de calidad.
Excepcionales.	Frecuentes.

Tabla 1. Comparativa entre las decisiones de alto y bajo nivel.

Por otra parte, es posible clasificar las decisiones en dos tipos: **decisiones programadas y decisiones no programadas**.

Las decisiones **programadas** son de rutina y se repiten periódicamente. Tienen que ver con problemas bien definidos y no requieren un proceso de decisión complejo.

Normalmente se administran con una secuencia de pasos, hay un método que es capaz de manejar este tipo de decisiones y existen reglas que permiten dirigir el pensamiento en una dirección específica definida. Por ejemplo: en una emergencia en el hogar, llame al número de emergencia.

Las decisiones no **programadas** están relacionadas con condiciones o entornos desconocidos, situaciones nuevas y no existen reglas o métodos establecidos que puedan servir como guía. Este tipo de decisiones son exclusivas de la situación, también única.

Las decisiones no programadas, en general, necesitan más tiempo para tomarse, ya que hay múltiples variables que deben valorarse y ponderarse, y también la información disponible es incompleta, ya que no es posible anticipar cuál será el impacto en el resultado de la decisión tomada. Por ejemplo: frente al menú de un restaurante que es la primera vez que visitamos.

En cuanto al problema, podemos diferenciar el **problema estructurado** del **no estructurado**.

- ▶ Problema estructurado: el enunciado contiene toda la información necesaria para poder resolverlo.
- ▶ Problema no estructurado: el enunciado no contiene toda la información que sería necesaria para resolverlo. Es necesario que la persona que se enfrenta a él busque información adicional para añadirla.

Etapas necesarias para la resolución de problemas

A continuación, vamos a hablar sobre las diferentes etapas necesarias para la resolución de problemas.

Primera etapa

Comprender la complejidad del problema es el objetivo de esta etapa. Analizaremos el problema, sintetizándolo por medio del pensamiento hipotético-deductivo, obteniendo una visión general del mismo.

- ▶ Identificar el problema.

Formularemos e intentaremos verificar la veracidad de una hipótesis que ilustre el problema. La mayoría de las veces, en nuestro día a día, los problemas se presentan de un modo claro y estructurado (en general, se encuentran a nuestro alrededor, esperando ser descubiertos). Es importante tener una actitud activa y tener la intención de hacer x o querer hacer y.

La visión correcta del problema es observar las tendencias, analizar la evolución de los hechos y usar la creatividad para anticipar e imaginar qué es posible que suceda. Es necesario tomar distancia al analizar los hechos, realizar un análisis con frialdad y confiar en las experiencias pasadas para predecir lo que sucederá en el futuro, pero sin que estas o los datos entren en conflicto con decisiones futuras. La materia prima para tomar decisiones es la información y, cuanto mejor sea, mejor será la calidad de la decisión.

La **información** es primordial en la toma de decisiones: a mayor calidad de esta, mejor es la calidad de la toma de decisiones. La **creatividad**, la **experiencia** y la **intuición**, junto con el conocimiento de las **experiencias pasadas**, ayudan a la hora de elaborar las predicciones de lo que puede suceder en el futuro.

Todas estas claves deberán considerarse necesarias a la hora de crear mecanismos automáticos para la toma de decisiones y resolución inteligente de problemas, ya sea de forma natural o artificial.

¿Cómo identificar el problema?

En la siguiente tabla se muestran preguntas que sirven para ayudar a delimitar un problema.

	¿Dónde se encuentra el problema?	¿Dónde no está el problema?
Origen	Determinar los aspectos donde es visible la situación que está provocando el problema.	En qué aspectos no se manifiesta el problema.
Magnitud	Número de personas afectadas y sus características (edad, género, contexto socioeconómico...).	En qué áreas no aparece el problema.
Lugar	Determinar el área geográfica (oficina, provincia, escuela...).	A qué personas no afecta.
Foco	Determinar, del mismo modo, grupos de afectados y el número de integrantes de los grupos.	Dónde no aparece el problema.
Historia	Determinar si es un hecho puntual y reciente, si perdura y se alarga en el tiempo...	En qué momento o momentos no se produce la situación problemática.

Tabla 2. Preguntas para delimitar un problema.

El **análisis DAFO** es otra técnica empleada para diseñar una estrategia para determinar problemas. Se trata de realizar una valoración en grupo sobre el presente y el futuro, valorando lo positivo y lo negativo.

Debilidades.	Amenazas.
Fortalezas.	Oportunidades.

Tabla 3. Análisis DAFO.

Por último, dentro de las técnicas para determinar problemas, vamos a describir la **reunión de discusión**. Consiste en compartir ideas cuando se va a comenzar un proyecto. Requiere una preparación previa concretando los temas que preocupan al equipo, disponer a los asistentes en círculo para favorecer la comunicación y que todos se vean las caras, no superar las dos horas de reunión y no complicarse demasiado realizando los preparativos. Este tipo de técnica se emplea con mucha frecuencia en la metodología Scrum.

Puedes ver más sobre Scrum en el siguiente enlace:

<https://proyectosagiles.org/que-es-scrum/>

Como vemos, existen varias técnicas para delimitar problemas: preguntarse por el origen, lugar, magnitud, foco o historia; ponerse en el lugar del cliente; análisis DAFO y reunión de discusión.

- ▶ Explicar el problema.

Es necesario explicar el problema para definir dónde se origina, dónde ocurre, cómo se produce y a quién afecta. Es hora de realizar tareas para profundizar en el problema, comprenderlo y desarrollar una explicación satisfactoria basada en los elementos que lo causan (las causas). De esta manera, será posible comenzar a crear una estrategia para resolverlo.

Para **explicar el problema**, hay que ir más allá de los síntomas y poner el foco en los aspectos que lo provocan: **indagar las causas**. Para lograr explicarlo se pueden seguir los siguientes pasos:

- Valorar la importancia del problema: es clave determinar la importancia de un problema con los criterios objetivos, haciendo un juicio de la importancia del problema en nuestro objetivo.
- Definir límite: encontrar el límite concreto del problema. Los problemas se originan de una manera difusa, mal definida. Antes de encontrar la solución, debemos conocer bien el problema y definir cuál es y cuál no, y buscar el objeto relacionado con el problema.
- Detectar las causas y las consecuencias: al reducir el problema (definir el límite), también lo describimos. Pero es necesario detectar y analizar las causas, ir al origen del problema, determinar y ayudar a predecir las consecuencias.

Existen varias **técnicas para dar explicación a los problemas**: clasificación ERIM de problemas, método de los seis interrogantes, las veinte causas, diagrama de espina de pez y mapas mentales.

- ERIM: se basa en clasificar los problemas según dos criterios (urgencia e importancia), elaborando una matriz del tipo:

		Urgente	
		+	-
Importante	+		
	-		

Tabla 4. Clasificación ERIM.

- Seis interrogantes sobre el problema: ¿cuál es?, ¿cómo sucede?, ¿por qué ocurre?, ¿a quién afecta?, ¿dónde sucede? y ¿cuándo pasa?
- Las veinte causas: hay que crear una lista de los posibles elementos que causan el problema, posteriormente se van eliminando causas para quedarse con las veinte más importantes.
- Diagrama de espina de pez: también conocido como diagrama Ishikawa, permite mostrar las causas de un problema, organizarlas y asignarles unos pesos o valores. Consiste en una técnica llevada a cabo en varias sesiones divididas en fases.

Primero, se debe dibujar un diagrama en blanco y luego escribir brevemente el problema a analizar. Se identifican las categorías del problema que se consideran apropiadas. Se determina una causa principal y se pregunta sobre el origen de esa causa, esto ayudará a establecer causas secundarias cuya consecuencia es la causa principal (análisis de dispersión). Se analizan en detalle todas las causas, tanto principales como secundarias.

Ejemplo de un diagrama de pez (causa-efecto) o diagrama Ishikawa:

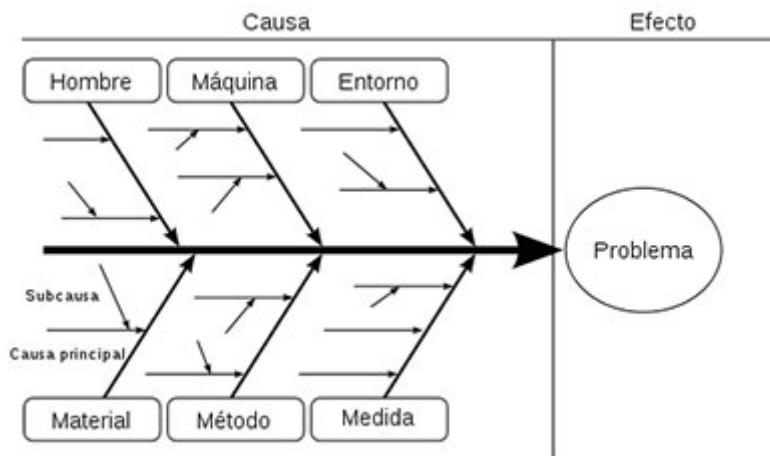


Figura 1. Diagrama espina de pez.

Fuente: (Ishikawa, 1976)

- Mapas mentales: sirven fundamentalmente para crear ideas a partir del análisis sistemático y detallado del problema. Los mapas mentales son una forma de expresar un razonamiento radiante (desde el interior hasta el final). La siguiente imagen muestra el mapa mental para hacer un DAFO.



Figura 2. Ejemplo de mapa mental para especificar un análisis DAFO.

Fuente: (Buzan, 1996)

Un problema bien definido: la capacidad de explicar el problema nos llevará a construir definiciones consistentes y bien definidas, de tal manera que los agentes inteligentes puedan diseñarse con un objetivo claro, entendiendo correctamente cuál es el objetivo final de su desempeño y su entorno real de desarrollo.

Deberemos, por tanto, crear definiciones de problemas que presenten características que nos permitan trabajar con ellos de modo eficiente. Algunas características o asunciones que se pueden tener en entornos simples (problemas bien definidos) son:

- Discreto:
 - Se puede concebir el mundo en estados.
 - En cada estado hay un conjunto finito de percepciones y acciones.
- Accesible: el agente puede acceder a las características relevantes del entorno.
 - Puede determinar el estado actual del mundo.
 - Puede determinar el estado del mundo que le gustaría alcanzar.
- Estático y determinista: no hay presión temporal ni incertidumbre.
 - El mundo cambia solo cuando el agente actúa.
 - El resultado de cada acción está totalmente definido y es previsible.

Segunda etapa

Crear una estrategia para resolver el problema que reduzca al mínimo los efectos negativos y haga que sea posible alcanzar los logros.

Con el problema bien definido y delimitado, ingresamos en la fase que permite encontrar las soluciones.

- ▶ Idear estrategias.

Existen muchas técnicas para encontrar estrategias y soluciones. Algunas de las empleadas en la toma de decisiones estratégica pueden ser: *brainstorming*, 4x4x4, la pecera y seis sombreros para pensar. Son interesantes para reflexionar acerca de la complejidad de la toma de decisiones.

- ▶ Elegir estrategia y tomar una decisión.

Para tomar una decisión, es necesario elegir de entre una serie de posibilidades aquella que sea la más adecuada para resolver el problema. En este punto, es necesario valorar las estrategias que han surgido, usando aquella metodología elegida en la fase de idear estrategias y tomar la decisión. Para elegir hay que valorar:

- Beneficios.
- Probabilidad de éxito.
- Dependencias internas y externas (¿depende de nosotros llevarlo a cabo o depende de más personas?).
- Medios, materiales y personas que son necesarios para llevar a cabo la estrategia.
- Estimación de tiempo necesario para poner en marcha la estrategia.
- Costes.

Desde el punto de vista de los agentes inteligentes, veremos que la estrategia a utilizar dependerá mucho del problema al que nos enfrentemos, de ahí la importancia de definir correctamente este.

Muchas técnicas son generalmente aplicables a muchos problemas, pero existen técnicas que mejoran las posibilidades de éxito frente a ciertos problemas. Por ejemplo, si tenemos que resolver un problema en el que el orden en el que se toman las decisiones no es importante, pero sí el resultado final de ciertas variables, son más eficientes las técnicas de CSP (Problema de Satisfacción de Restricciones) (Manya, 2003) o los algoritmos genéticos. Si encontramos un conocimiento *a priori* alto sobre el problema, pero una definición vaga de los predicados y términos utilizados, deberíamos buscar modelos de inferencia difusos, etc.

Por tanto, es cometido del especialista el conocer bien el problema antes de plantear estrategias. El especialista debe saber también el mayor número de técnicas posibles para emplear la más conveniente a cada problema.

- ▶ Diseñar intervención.

A continuación, deberemos planificar las acciones que se van a realizar, determinando una hoja de ruta. Deberemos emplear el mejor diseño arquitectónico posible para poder garantizar la mejor toma de decisiones, de modo eficiente y organizado.

Tercera etapa

Consiste fundamentalmente en conseguir resolver el problema. Es la etapa en la cual se implementa la estrategia óptima ejecutando la intervención. Por último, el proceso de toma de decisiones se cerrará con una evaluación de logros que servirá para mejorar y como aprendizaje.



Accede a los ejercicios de autoevaluación a través del aula virtual

1.3. Arquitectura de un agente inteligente



Accede al vídeo «Arquitectura de un agente inteligente» a través del aula virtual

Definición de agente inteligente

De las múltiples interpretaciones y técnicas empleadas para construir sistemas que resuelvan problemas, la **inteligencia artificial** es la disciplina derivada de los conocimientos científicos e ingenieriles que estructura las metodologías, técnicas y algoritmos que apoyan la resolución de estos problemas, intentando emplear mecanismos sofisticados de razonamiento y usando de modo eficiente recursos y conocimientos.

Dentro de la inteligencia artificial, la **teoría de agentes inteligentes** diseña los sistemas de toma de decisiones por medio de la filosofía de inmersión de un elemento (agente) en un entorno, con el cual debe interactuar por medio de sus sistemas de percepción (que le proporcionan información de estado de dicho entorno) y sistemas actuadores (que le permiten modificar el estado para conseguir un objetivo).

Es importante resaltar la **definición de agente inteligente** que emplearemos a lo largo de este curso. Para nosotros, un sistema se considerará inteligente siempre que intente obtener una meta por medio de un comportamiento racional. Evitaremos las definiciones de inteligencia artificial «fuerte» que consideran factores más allá de la racionalidad para clasificar un agente como inteligente, dado que necesitaríamos de definiciones reflejo del comportamiento humano para considerar estos agentes «fuertes».

Por nuestra parte, un agente será más inteligente cuanto más autónomo sea, es decir, que necesite menos conocimiento *a priori* para encontrar una solución racional a su problema. El concepto de conocimiento *a priori* surge de cualquier información previa sobre el comportamiento del entorno que se le tenga que proporcionar al agente para tomar decisiones. Todo conocimiento de este tipo que le ofreczcamos a un agente le resta autonomía y, por tanto, capacidad de tomar decisiones.

La **racionalidad del agente** se medirá a través de una medida de desempeño (normalmente nos referiremos a ella como métrica o utilidad).



Figura 3. Representación gráfica de agente inteligente.

Elementos de un agente inteligente

En términos generales, un agente inteligente debe organizar su proceso de toma de decisiones en tres fases principales:



Figura 4. Fases en las que un agente inteligente debe organizar su proceso de toma de decisiones.

- ▶ **Sentir:** le proporciona al agente la percepción del entorno. Esta información tiene que ser procesada por el sistema de percepción que debe:
 - Extraer en estructuras de datos la información percibida. Las estructuras de datos deben ser «amigables» para el razonamiento.
 - Procesar dichas estructuras extraídas para generar una representación reducida de elementos relevantes para el razonamiento, conceptos de alto nivel relacionados con el proceso de razonamiento del agente.

- ▶ Pensar: es el proceso de toma de decisiones racionales del agente. Empleando la información disponible del sistema de percepción y la **memoria** interna del agente, inicia un proceso deliberativo o reactivo en el cual debe:
 - Razonar, extrayendo de los hechos relevantes observados y recordados conclusiones nuevas, por ejemplo, de la posición de un nuevo agente respecto a la antigua: su trayectoria.
 - Decidir: empleando mecanismos para la toma de decisión de acciones (los algoritmos tradicionales de búsqueda, planificación...).
- ▶ Actuar: por medio de sus sistemas actuadores, el agente interactuará con el entorno para producir aquellos cambios que le permitan obtener un estado que mejore su valoración del desempeño para alcanzar sus metas. Para ello deberá:
 - Generar sus acciones de alto nivel en estructuras de datos que puedan ser manejadas en el futuro, empleando árboles, grafos, etc.
 - Convertir las acciones empleadas por el agente como resultado del razonamiento en elementos entendibles por el sistema de actuadores (curvas de animación, trazas de información del sistema, variable de estado...).

Arquitectura de un agente Inteligente

Una arquitectura de agentes es una metodología empleada para construir agentes. Las arquitecturas de agente tienen un importante papel en el razonamiento práctico. En ellas se especifica cómo el agente puede ser descompuesto en un conjunto de módulos, y cómo estos módulos pueden interactuar. El conjunto total de módulos y sus interacciones determinan las acciones y estados internos futuros. (Wooldridge, 1994) proponen, una clasificación de arquitectura de agentes en tres categorías principales:

- ▶ Arquitecturas deliberativas.
- ▶ Arquitecturas reactivas.

- Arquitecturas híbridas.

Arquitecturas de agentes deliberativos

Contienen una representación explícita y lógica del mundo, con lo cual las decisiones (por ejemplo, acerca de las acciones a realizar) son hechas por medio de un razonamiento lógico. Se basan en un razonamiento práctico, en el que se decide el conjunto de acciones a realizar para conseguir los objetivos. No existe límite de tiempo para calcular el conjunto de acciones.

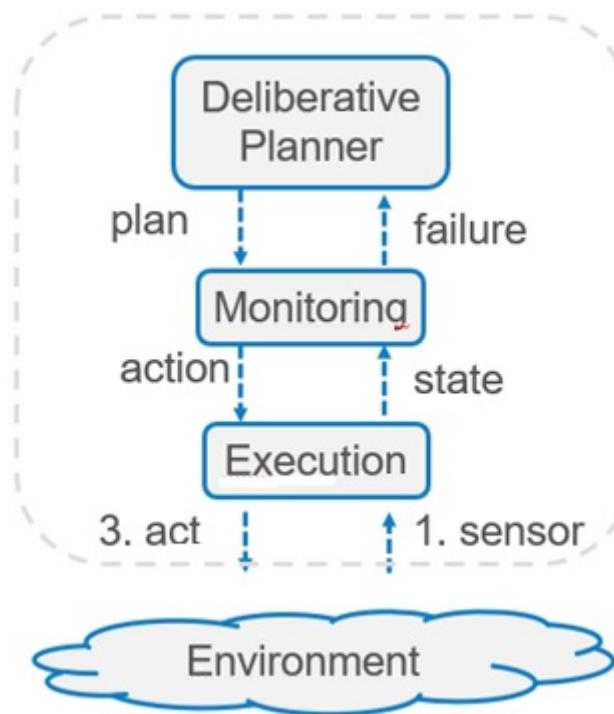


Figura 5. Arquitectura de un Agente deliberativo.

Fuente: (Guzman, 2020).

- Ejemplos de arquitecturas deliberativas:

- Agentes basados en planificación (*planning agents*): la comunidad de inteligencia artificial dedicada a la planificación, desde principios de los años setenta, ha estado estrechamente relacionada con el diseño de agentes. Parece razonable, entonces, que muchas de las innovaciones en el diseño de agentes

provengan de esa comunidad. El objetivo es describir a los agentes que pueden encontrar un curso de acciones que, cuando se ejecutan, logran algún objetivo deseado.

- Belief, Desire & Intention (BDI) (Georgeff, Pell, Pollack, Tambe y Wooldridge, 1999): la descripción del estado de un agente de estas características se realiza por medio de:
 - Creencias: que representan el conocimiento que el agente tiene de sí mismo y de su entorno. Estas creencias pueden ser incompletas y/o incorrectas.
 - Deseos: objetivos que el agente desea cumplir a largo plazo.
 - Intenciones: subconjunto de las metas que el agente intenta conseguir.

Además de estos elementos básicos, los agentes BDI generan planes por medio de explorar metas intermedias.

- Planes: combinación de intenciones. Las intenciones crean subplanes del plan global del agente, de manera que el conjunto de todos los planes refleja las intenciones del agente.
- Metas: subconjunto de los deseos que el agente puede conseguir. Han de ser realistas y no tener conflictos entre ellas.

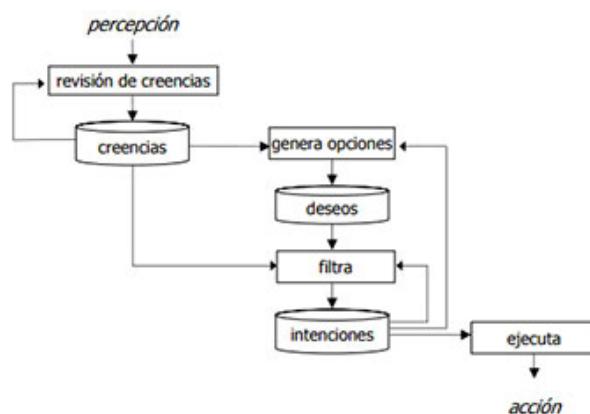


Figura 6. Modelo BDI de agentes.

Fuente: Pavón (2006).

Arquitecturas de agentes reactivos

Una arquitectura para agente reactivo es aquella que no utiliza razonamiento simbólico complejo. Pretenden resolver el problema puntual que se encuentran en cada instante por medio de una toma de decisiones rápida.

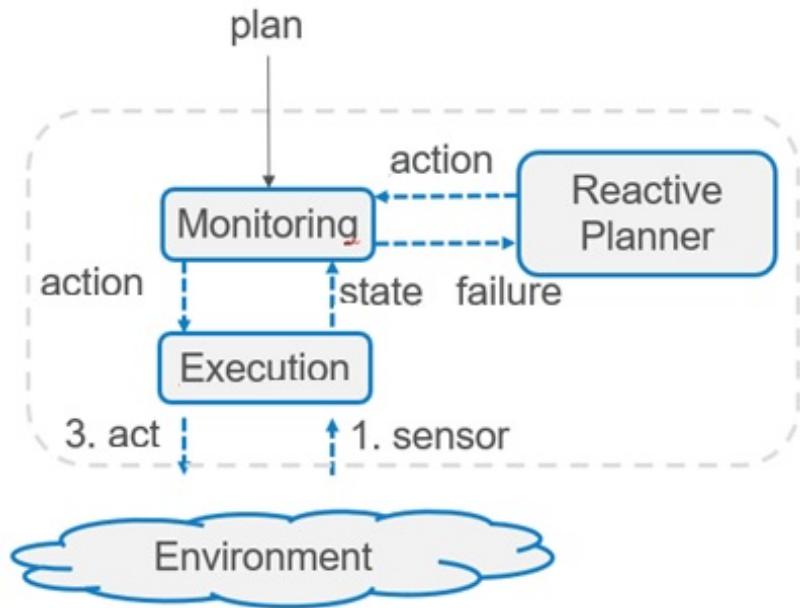


Figura 7. Arquitectura de un Agente reactivo.

Fuente: (Guzman, 2020).

► Ejemplos:

- *Sumbsumption architecture:* consiste en una jerarquía de comportamientos para la consecución de tareas. Cada comportamiento intenta tomar el control del agente en cada instante. Capas menores representan comportamientos de tipo más primitivo (tal como evitar obstáculos, por ejemplo) y tienen precedencia sobre las capas superiores de la jerarquía.
- Arquitectura de red de agentes: Pattie Maes (Maes, 1990) ha desarrollado una arquitectura de agentes en la cual un agente se define como un conjunto de módulos de competencia. Estos módulos asemejan ligeramente el comportamiento de la arquitectura subsunción. Cada módulo es especificado

por el diseñador en términos de pre- y poscondiciones, y un nivel de activación, que da una indicación real de la relevancia del módulo en una situación particular.

- Reactive execution model (Guzman, 2015): consiste en una arquitectura de ejecución reactiva para el control de la ejecución de planes. Es independiente del dominio y opera con unas estructuras precalculadas en tiempo de ejecución.

Arquitecturas para agentes híbridos

Un enfoque totalmente deliberativo o reactivo no es adecuado para diseñar agentes en entornos complejos. En este tipo de entornos, se necesitan sistemas híbridos que intentan unir estos enfoques. Un enfoque obvio es construir un agente compuesto por dos subsistemas o planificadores:

- ▶ Deliberativo, que contiene un módulo simbólico del mundo, que desarrolla planes y efectúa decisiones de la manera propuesta por la inteligencia artificial simbólica.
- ▶ Reactivo, que es capaz de reaccionar a eventos que ocurren en el ambiente sin necesitar un razonamiento complejo. A menudo, al componente reactivo se le da cierto grado de precedencia sobre el deliberativo, de tal manera que pueda proveer una pronta respuesta a eventos ambientales importantes.

Este tipo de arquitecturas presenta controladores reactivos en los niveles de toma de decisiones locales o inmediatas, delegando conductas planificadas a las capas superiores con controladores de deliberación. Existen mecanismos para enviar información entre capas, en muchos casos como entradas a capas inferiores que se combinan con las percepciones recibidas en los sensores.

Veamos algunos ejemplos:

- Procedural Reasoning System (PRS): PRS (Georgeff, 1987) es una arquitectura BDI, es decir, basada en creencias, deseos e intenciones, que incluye una librería de planes, así como una explícita representación simbólica de las creencias, deseos e intenciones.

Las creencias son hechos tanto derivados de la percepción del entorno como del estado interno del agente. Estos hechos son expresados mediante lógica de primer orden. Los deseos son representados como comportamientos del sistema. La librería de planes del PRS contiene un conjunto de planes parcialmente elaborados, llamados áreas de conocimiento (*Kas, por sus siglas en inglés*), cada uno de los cuales se asocia con una condición para ser invocado.

Estas condiciones determinan cuando el *KA* debe ser activado. Los *KAs* pueden ser activados por objetivos o por datos. Pueden ser también reactivos, permitiendo que el PRS responda rápidamente a cambios en su ambiente. Las intenciones son el conjunto actual de *KAs*.

- COSY (Burmeister, 1992): la arquitectura COSY es una arquitectura BDI. Aunque está relacionada con agentes interactivos, también se considera una arquitectura híbrida que incluye elementos tanto de PRS (Georgeff, 1987) como de IRMA (Bratman, 1987). Tiene cinco componentes principales:

- Sensores: reciben entradas perceptibles no comunicativas.
- Actuadores: permiten al agente realizar acciones no comunicativas.
- Comunicaciones: permiten al agente enviar mensajes.
- Cognición: es responsable de mediar entre las intenciones del agente y sus conocimientos acerca del mundo, y de elegir una acción apropiada para realizar.
- Intención: contiene objetivos a largo plazo, actitudes, responsabilidades y los elementos de control que toman parte en el razonamiento y la toma de decisiones del componente cognición.

Otros tipos de Arquitecturas

Arquitectura de tres capas (*three-layer architecture*)

Es posible agrupar los datos por niveles. Los robots móviles se basan en este tipo de arquitectura.

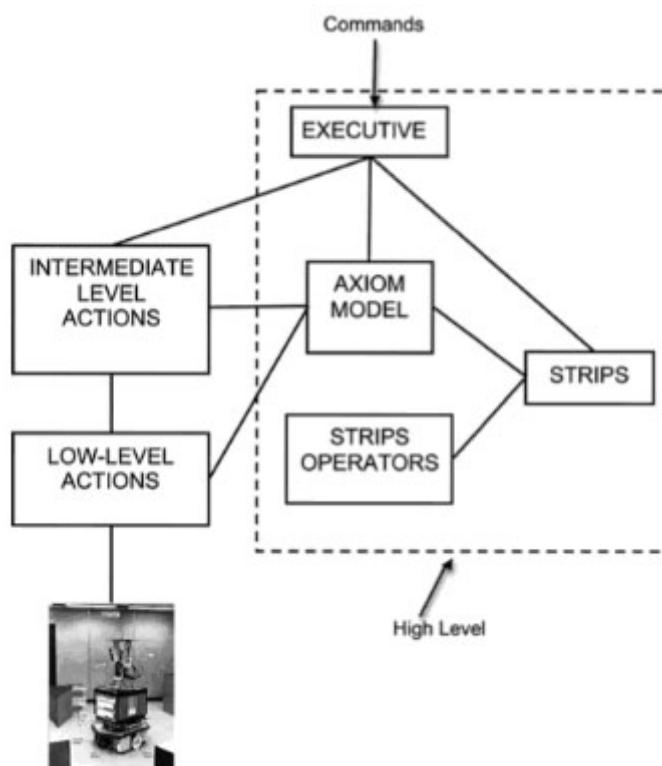


Figura 8. Modelos y arquitecturas de agentes.

Fuente: Pavón (2006).

Niveles:

- ▶ De control reactivos.
- ▶ Gobernar secuencias.
- ▶ Consumidores de tiempo.

Arquitecturas multicapa (*multilayer architectures*) (Albus y Barbera, 2005).

Arquitectura de modelo de referencia, múltiples capas de sistemas de control en tiempo real. Generación de comportamiento, modelado del mundo, procesamiento sensorial, juicio de valor.

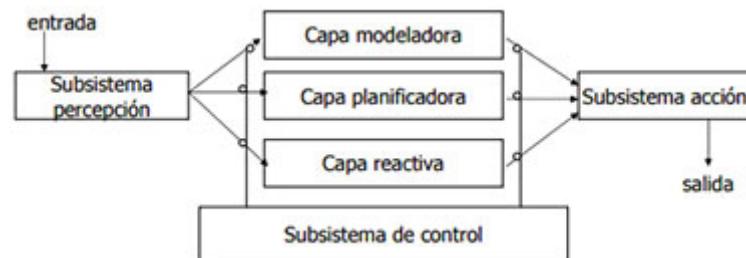


Figura 9. Modelos y arquitecturas de agentes.

Fuente: Pavón (2006).

Arquitectura triple torre:

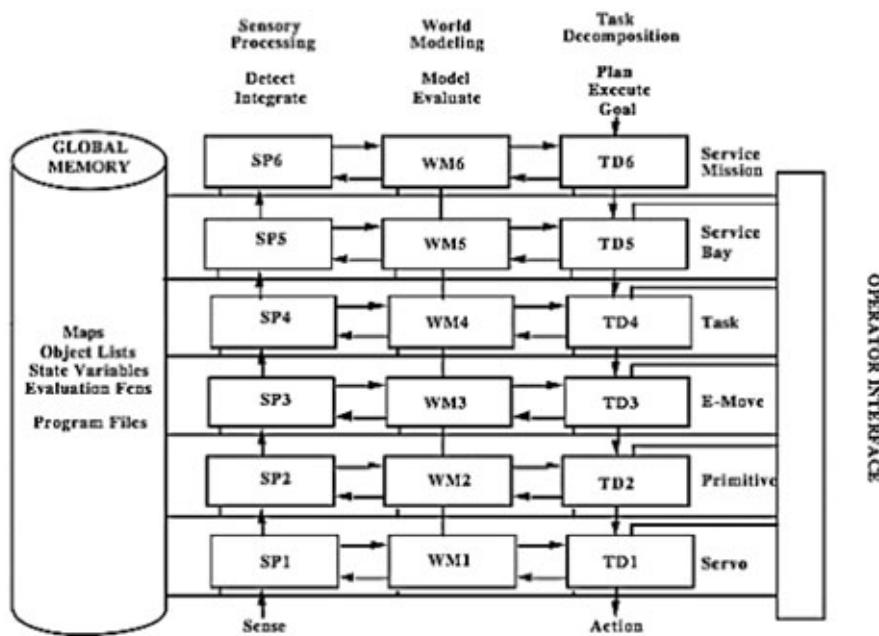


Figura 10. Arquitectura triple torre.

Fuente: Pavón (2006).

Arquitecturas cognitivas

Su objetivo es definir las operaciones cognitivas y perceptivas básicas que posee la mente humana. En teoría, cada tarea que los humanos pueden realizar debe consistir

en una serie de estas operaciones discretas. Los modelos formalizados pueden usarse para refinar aún más una teoría integral de la cognición y, más inmediatamente, como un modelo comercialmente utilizable.

El Institute of Creative Technologies (s.f.) define la arquitectura cognitiva como:

«hipótesis sobre las estructuras fijas que proporcionan una mente, ya sea en sistemas naturales o artificiales, y cómo funcionan juntas —junto con el conocimiento y las habilidades incorporadas dentro de la arquitectura— para producir un comportamiento inteligente en una diversidad de entornos complejos».

Ejemplos:

- ▶ ACT-R: es una arquitectura cognitiva desarrollada principalmente por John Robert Anderson en la Universidad Carnegie Mellon.

La suposición más importante de ACT-R es que el conocimiento humano se puede dividir en dos tipos de representaciones irreductibles: **declarativo** y de **procedimiento**.

Dentro del código ACT-R, el conocimiento declarativo se representa en forma de fragmentos, es decir, representaciones de vectores de propiedades individuales, cada una de ellas accesible desde una ranura etiquetada.

Los fragmentos se mantienen y se hacen accesibles a través de búferes, que son el *front-end* de lo que son módulos, es decir, estructuras cerebrales especializadas y en gran medida independientes.

Hay dos tipos de módulos:

- Módulo perceptivo-motor: encargado de la interacción con el entorno. Gestiona el flujo de percepción y acción necesario para conectar al agente con el mundo.
- Módulo de memoria: la memoria está dividida en dos tipos. Por un lado, la **memoria a largo plazo** o memoria productiva, que contiene las reglas de producción y, por otro, la **memoria a corto plazo** (memoria de trabajo o memoria declarativa), que contiene los hechos actuales del mundo.

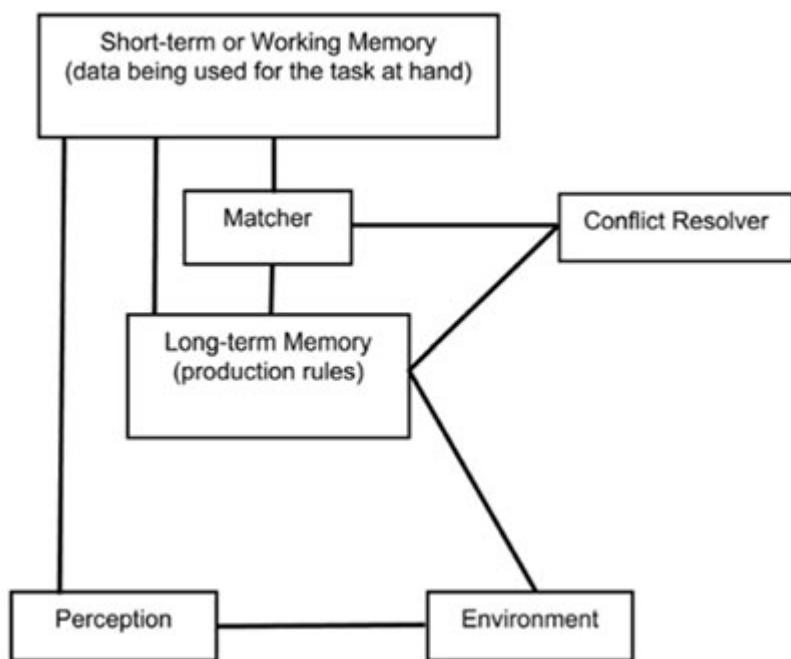


Figura 11. Esquema simplificado de arquitectura cognitiva.

Fuente: Institute of Creative Technologies (s.f.)

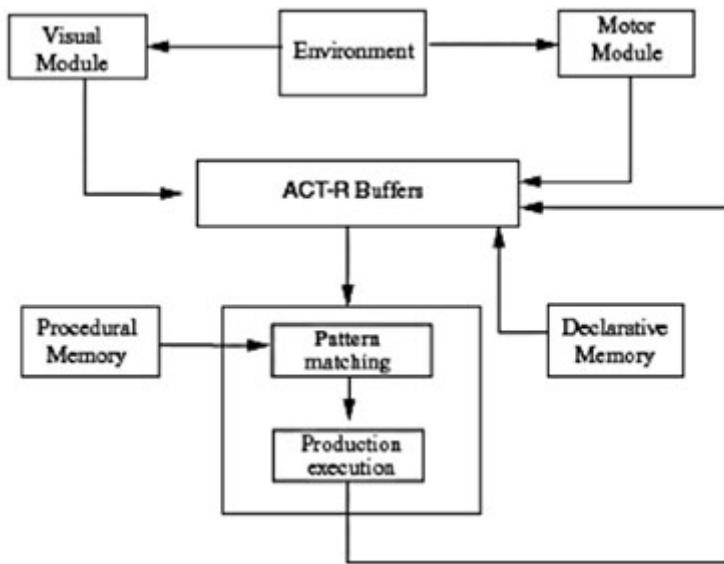


Figura 12. Esquema de un sistema ACT-R.

Fuente: <http://act-r.psy.cmu.edu/about/>

- ▶ SOAR (Laird, Newell y Rosenbloom, 1987).: también creada en la Universidad de Carnegie Mellon por Laird y Newell. El objetivo final de una arquitectura cognitiva es proporcionar una base para un sistema capaz de un comportamiento inteligente general. Es decir, el objetivo es proporcionar la estructura subyacente que permitiría a un sistema realizar toda la gama de tareas cognitivas, emplear toda la gama de métodos de resolución de problemas y representaciones apropiadas para las tareas, y conocer todos los aspectos de la tarea y su rendimiento en ellos.

Para alcanzar este objetivo se desarrollan los bloques de construcción computacionales fijos necesarios para los agentes inteligentes generales: agentes que pueden realizar una amplia gama de tareas y codificar, utilizar y aprender todo tipo de conocimientos para comprender la gama completa de capacidades cognitivas que se encuentran en humanos, como la toma de decisiones, la resolución de problemas, la planificación y la comprensión del lenguaje natural. Es, a la vez, una teoría de lo que es la cognición y una implementación computacional de esa teoría.

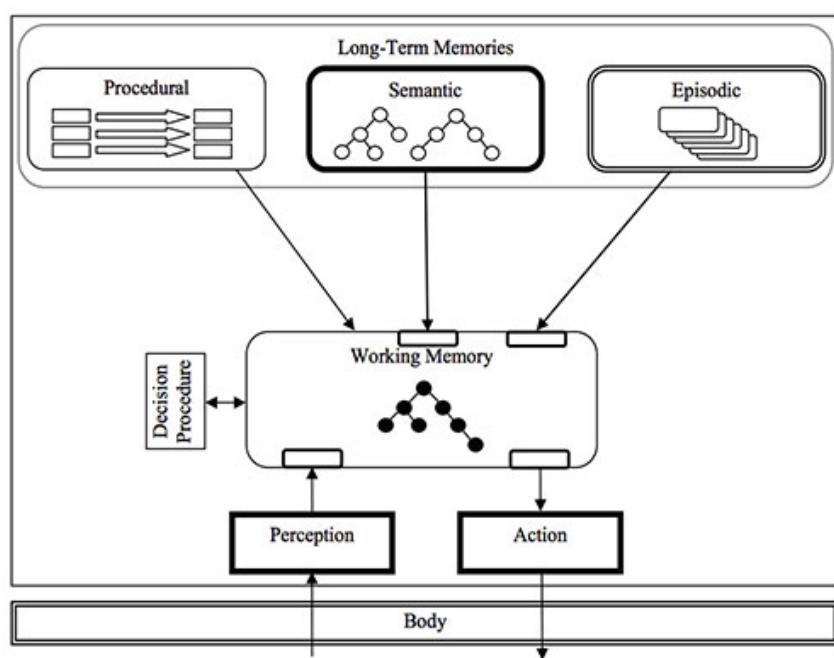


Figura 13. Arquitectura SOAR.

Fuente: Laird, Newell y Rosenbloom (1987)

El diseño de SOAR se basa en la hipótesis de que todo comportamiento deliberado orientado a **objetivos** puede ser entendido como la selección y aplicación de **operadores** a un **estado**. Un estado es una representación de la situación actual del problema a resolver, un operador transforma un estado (realiza cambios en la representación) y un objetivo es un resultado deseado para el problema.

SOAR se ejecuta continuamente intentando aplicar el operador actual y seleccionar el siguiente operador (un estado puede tener solo un operador a la vez), hasta que se logre el objetivo.

SOAR tiene memorias separadas (con diferentes modos de representación) para la descripción de su estado actual y de su conocimiento *procedural* a largo plazo. La **memoria a corto plazo** almacena los datos producidos por los sensores, los resultados de inferencias intermedias sobre los datos más actuales, las metas actualmente activas para el agente y los operadores (acciones y planes) que están también activos.

Mientras que la **memoria a largo plazo o memoria de producción** mantiene el conocimiento de responder a situaciones por medio de procedimientos que almacenan el conocimiento específico para la resolución del problema. Esto lo lleva a cabo por medio de reglas de inferencia y conocimiento para la selección y aplicación de operadores frente a estados específicos.



Accede a los ejercicios de autoevaluación a través del aula virtual

1.4. Tipos de agentes inteligentes



Accede al vídeo «Tipos de agentes inteligentes» a través del aula virtual

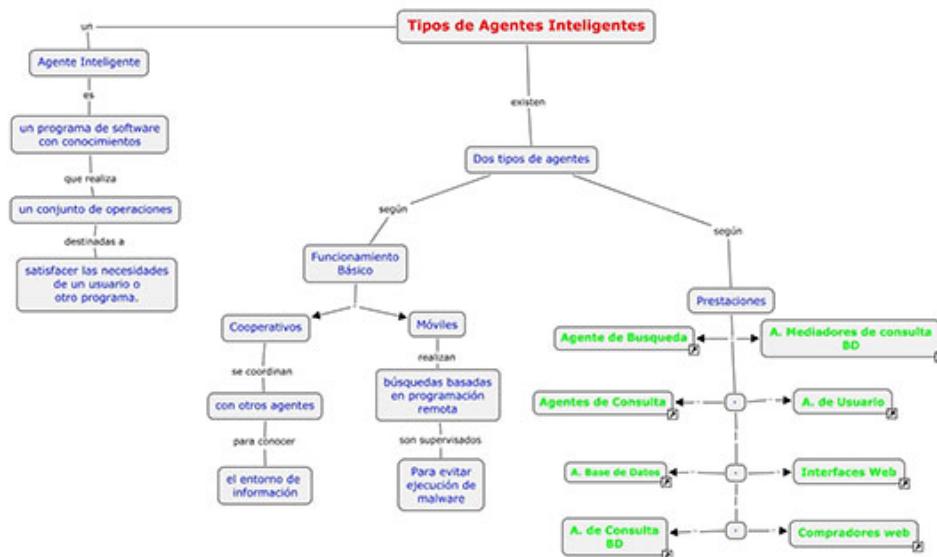


Figura 14. Tipos de agentes inteligentes.

Fuente: (Heredia, s.f.).

Puedes consultar el mapa conceptual completo y navegar a través de él para ver en detalle los tipos de agentes inteligentes en:

<http://cmapspublic.ihmc.us/rid=1LN08Q062-151SR17-2805/tipos%20de%20agentes%20inteligentes.cmap>

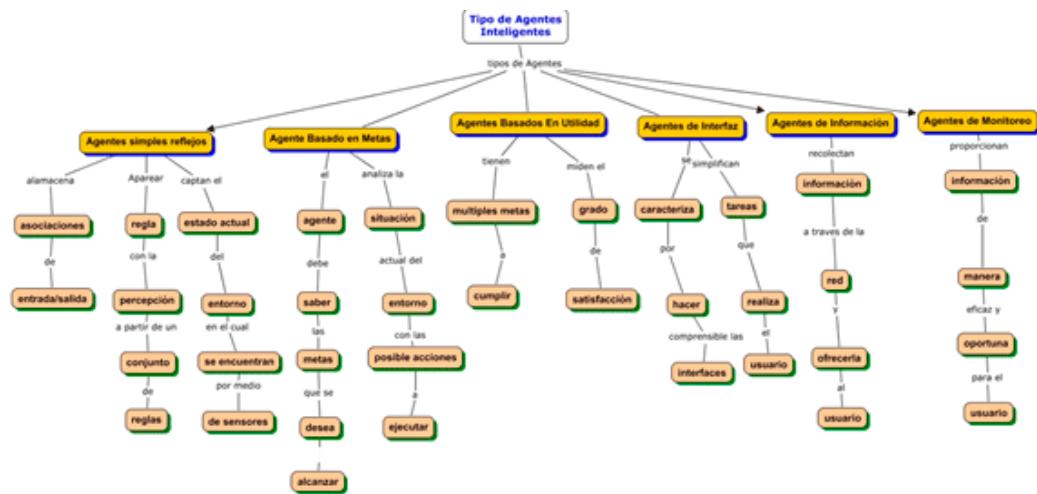


Figura 15. Tipos de agentes inteligentes.

Fuente: (Heredia, s.f.).

Puedes consultar el mapa conceptual completo y navegar a través de él para ver en detalle los tipos de agentes inteligentes en:

<https://cmapspublic.ihmc.us/rid=1LNTSCHKK-LLS690-2JGW/Tipos%20de%20Agentes%20Inteligentes.cmap>

Agentes basados en búsquedas y planificación

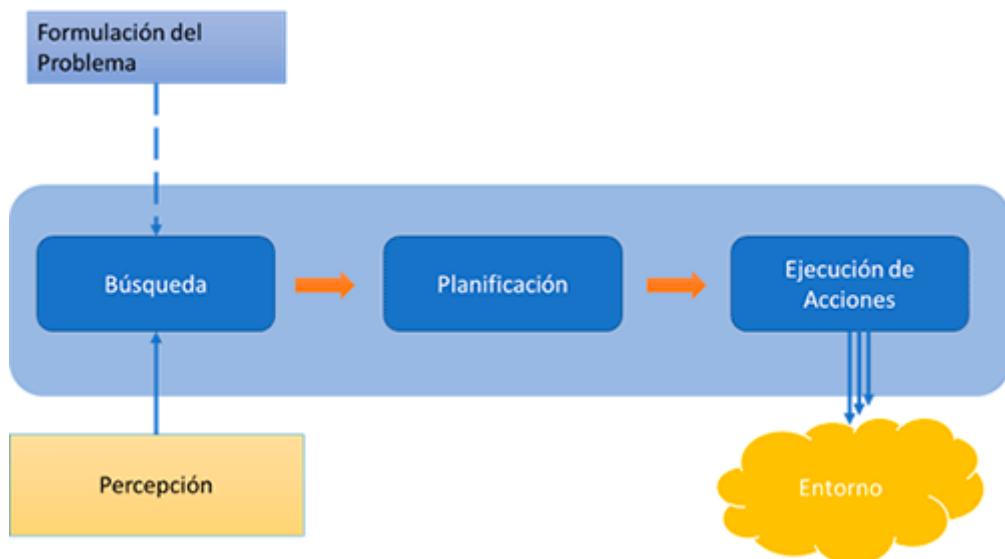


Figura 16. Agentes basados en búsquedas y planificación.

Estos agentes resuelven el problema de alcanzar una meta u objetivo deseado en el entorno en el que se encuentran por medio de la exploración del espacio de estados.

Para ello realizan una búsqueda de las posibles alternativas (acciones disponibles) dado el estado en el que están. Con esta decisión simulan o estiman el estado resultante del mundo al operar de esa manera e intentan encontrar qué secuencia de acciones es la que les permite alcanzar la meta que desean.

Los algoritmos de búsqueda emplean la descripción del estado como punto de partida para toda exploración, pero pueden tomar o no en consideración el hecho de que los estados del mundo presentan estructura.

Empezaremos no teniendo en cuenta dicha estructura y presentaremos los algoritmos generales de búsqueda en espacios de estados (amplitud, profundidad, A*, multiagentes...), y luego introduciremos aquellos modelos de razonamiento en los cuales la estructura del estado nos permita alcanzar unos mejores modelos de exploración del espacio de estados, acelerando el proceso de toma de decisiones.



Accede a los ejercicios de autoevaluación a través del aula virtual

1.5. Referencias bibliográficas

Albus, J. y Barbera, A. (2005). RCS: A cognitive architecture for intelligent multi agent systems. *Annual Reviews in Control*, 29(1), 87-99.

Bratman, M. E. (1987). Toward an architecture for resource-bounded agents. *SRI and Stanford Univ., Stanford, CA, Centre for the Study of Language Information*, (pp. 87-104).

Burmeister, B. &. (1992). Cooperative problem-solving guided by intentions and perception. *ACM SIGART Bulletin*, (pp. 13(3), 10).

Buzan, T. &. (1996). *El libro de los mapas mentales*. Barcelona: Ediciones Urano.

Georgeff, M. P. (1987). Reactive reasoning and planning. *AAAI*, (pp. Vol. 87, pp. 677-682).

Georgeff, M., Pell, B., Pollack, M., Tambe, M. y Wooldridge, M. (1999). *The Belief-Desire-Intention Model of Agency*. Conferencia presentada en Intelligent Agents V, Agent Theories, Architectures, and Languages, París.

Guzman, C. C. (2015). Reactive execution for solving plan failures in planning control applications. *Integrated Computer-Aided Engineering*, 22(4), 343-360.

Heredia, C. (s.f.) *Tipos de agentes inteligentes*. Recuperado de <http://cmapspublic.ihmc.us/rid=1LN08Q062-151SR17-2805/tipos%20de%20agentes%20inteligentes.cmap>

Ishikawa, K. (1976). *Guide to Quality Control*. Asian Productivity Organization.

Laird, J., Newell, A. y Rosenbloom, P. (1987). *SOAR: An Architecture for General Intelligence*. Michigan: Carnegie-Mellon University. Recuperado de <https://pdfs.semanticscholar.org/c86e/4e9f058f06e2e452ff34c08b2f29b957acb9.pdf>

Maes, P. (1990). *Designing Autonomous Agents*. Cambridge, MA: The MIT Press.

Manya, F. &. (2003). Técnicas de resolución de problemas de satisfacción de restricciones. *Revista Iberoamericana de Inteligencia Artificial*, 7(19), 0.

Müller, J. (1996). *The design of intelligent agents: a layered approach*. Lecture Notes in Computer Science, Vol. 1177. Berlín: Springer-Verlag.

Pavón, J. (2006). *Agentes inteligentes. Modelos y arquitecturas de agentes*. Recuperado de

<http://www.fdi.ucm.es/profesor/jpavon/doctorado/arquitecturas.pdf>

USC Institute for Creative Technologies. (s. f.) Cognitive Architecture. Recuperado de
<http://cogarch.ict.usc.edu/>

Wooldridge, M. y. (1994). Agent Theories, Architectures and Languages: A Survey.
Intelligent Agents ECAI. Ámsterdam: Recuperado de
<https://www.csee.umbc.edu/~finin/papers/atal.pdf>.

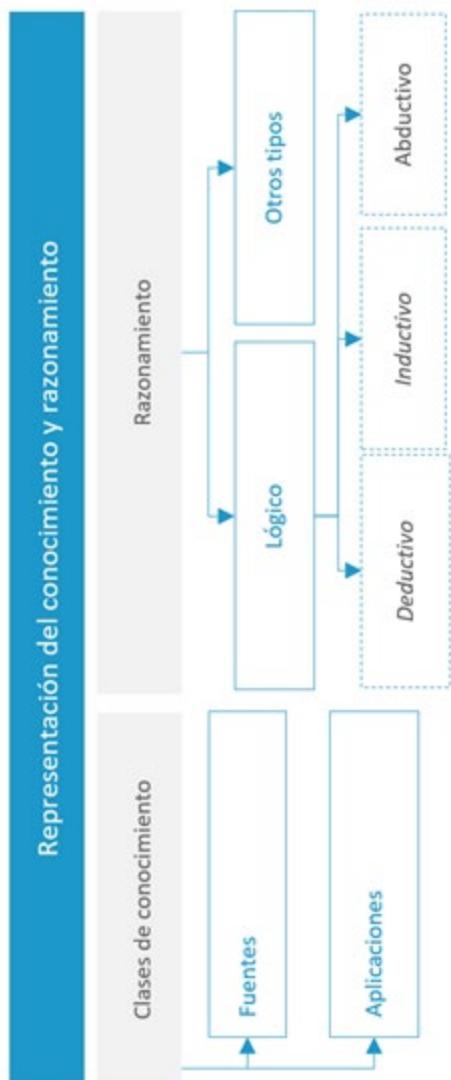


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Representación del conocimiento y razonamiento

Esquema. Tema 2



Ideas clave. Tema 2

2.1. ¿Cómo estudiar este tema?

En este tema analizaremos cuatro puntos fundamentales para poder entender los mecanismos de representación de la información para agentes inteligentes:

- ▶ La representación de la información: ¿por qué es necesaria?, ¿cómo puede representar lo que configura el entorno?
- ▶ Del conocimiento que tenemos *a priori* y que podemos proporcionar al agente: ¿cuál es útil?, ¿cómo lo expreso?
- ▶ De la información que recibimos en cada instante y que procesamos: ¿cómo podemos almacenarla?, ¿en qué influye lo que ya ha experimentado un agente?
- ▶ Con toda la información que tenemos extraída de la percepción, ¿cómo podemos crear más información?

Veremos a lo largo del tema que los distintos enfoques de agente que queramos emplear para resolver un problema influyen notablemente en los esquemas de diseño de la información y en cómo gestionamos su almacenamiento e inferimos nuevos datos desde ello.

Además, trataremos brevemente los distintos **tipos de razonamiento** que emplea el ser humano para resolver un problema. Describiremos en detalle los razonamientos lógicos (inductivo, deductivo y abductivo), básicos para entender procesos necesarios a la hora de implementar sistemas inteligentes.



Accede a los ejercicios de autoevaluación a través del aula virtual

2.2. Técnicas de representación simbólica



Accede al vídeo «Técnicas de representación simbólica» a través del aula virtual

Si tomamos en consideración el esquema de agente que debe realizar una toma de decisiones de acuerdo a las percepciones recibidas de los cambios del entorno en el que se encuentra, nos encontramos con un primer problema: ¿cómo representamos la información que caracteriza el entorno y el conocimiento que tenemos del problema?

Ya hemos visto la importancia de definir bien un problema. Ahora detallaremos los modelos estándar de representación de dicha información y especificación del problema.

¿Qué es la representación de la información?

Cualquier entidad inteligente que desee razonar sobre su mundo encuentra un hecho importante e ineludible: el razonamiento es un proceso que ocurre internamente, mientras que la mayoría de las cosas sobre las que desea razonar existen solo externamente. Por ejemplo, Un agente (o persona) dedicado a planificar el montaje de una bicicleta, puede tener que razonar sobre entidades como ruedas, cadenas, piñones, manubrios, etc., pero tales cosas existen solo en el mundo externo.

Esta inevitable dicotomía es una razón fundamental y un rol para una representación: funciona como un sustituto dentro del razonador, un sustituto de las cosas que existen en el mundo. Las operaciones en y con representaciones sustituyen las operaciones en lo real, es decir, sustituyen la interacción directa con el mundo. En este punto de vista, el razonamiento en sí mismo es, en parte, un sustituto de la acción en el mundo, cuando no podemos o no (todavía) queremos tomar esa acción (Brachman, 1985).

Las representaciones que tengamos de la información funcionan como sustitutos de nociones abstractas como acciones, procesos, creencias, causalidad, categorías, entre otros. Lo que les permite ser descritas dentro de una entidad para que pueda razonar sobre ellas.

Técnicas de representación

La representación explícita y formal del conocimiento sobre un problema requiere el uso de técnicas particulares. En el campo de la representación simbólica del conocimiento dentro de la inteligencia artificial, se han propuesto diversas formas de representación (Molina, 2006). Como muestra de estas técnicas tenemos, por ejemplo:

Marcos: que son representaciones estereotipadas de situaciones, conceptos, ideas u objetos. Los procesos de inferencia se realizan por medio de la jerarquía u ordenación de los mismos.

Lógica: representación e inferencia basada en modelos lógicos formales.

Reglas: por medio del encadenamiento hacia adelante se produce la inferencia de sentencias condicionales basadas en estructuras de la forma SI-ENTONCES.

Restricciones: interrelaciones entre variables por medio de un dominio de valores posibles. La inferencia se realiza a través de técnicas de propagación de restricciones o consistencia de arco.

Red bayesiana: red causal basada en valoraciones cuantitativas probabilistas de la influencia o relaciones entre antecedentes y consecuentes.

Lógica difusa: valoraciones semánticas experienciales que nos identifican valores de pertenencia a conjuntos de características borrosas. El proceso de inferencia se realiza por medio de modelos matemáticos de lógica especializada para estos entornos.

Figura 1. Técnicas de representación.

En general, una forma de representación del conocimiento debe satisfacer los siguientes requisitos (Molina, 2006):

- ▶ Formal: la representación no debe presentar ambigüedades. Por ejemplo, el lenguaje natural no se considera representación del conocimiento debido a las ambigüedades que presenta.
- ▶ Expresiva: la representación debe ser suficientemente rica como para capturar los diferentes aspectos que sea necesario distinguir. Por ejemplo, las fórmulas lógicas de cálculo de predicados constituyen una representación más expresiva que la que se maneja en cálculo proposicional.
- ▶ Natural: la representación debe ser suficientemente análoga a formas naturales de expresar conocimiento. En este sentido, las representaciones matemáticas tradicionales y cuantitativas (por ejemplo, las matrices) pueden resultar muy artificiales para emular procesos de razonamiento.
- ▶ Tratable: la representación se debe poder tratar computacionalmente, es decir, deben existir procedimientos suficientemente eficientes para generar respuestas a través de la manipulación de los elementos de las bases de conocimiento.

En general, es conveniente crear representaciones de información basadas en una única representación. Esto mejora el mantenimiento de la base de conocimiento. Pero, en algunos casos, intentar representar todo el conocimiento en una única representación nos puede limitar la aplicación de técnicas y algoritmos.

Para sistemas complejos que requieren tomar decisiones a distintos niveles, es común emplear la idea de generar **agentes multicapa** (ver Figura 2) que descomponen el problema en niveles y, para cada uno de ellos, establece un tratamiento orientado a un agente especializado que necesita de una representación concreta de parte de la información del entorno (Molina, 2006).

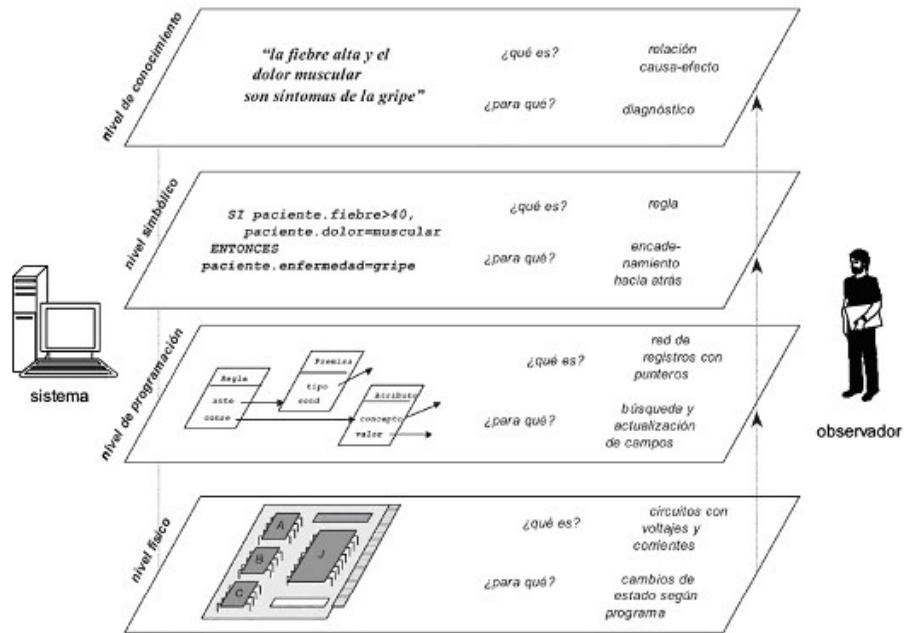


Figura 2. Sistema modular multicapa. Fuente: (Molina, 2006).

Debido a esto, la propuesta actual es diseñar sistemas inteligentes basados en arquitecturas que distribuyan y traten el problema de modo más organizado y emplear técnicas especializadas para cada subtarea o subproblema.



Accede a los ejercicios de autoevaluación a través del aula virtual

2.3. Clases de conocimiento



Accede al vídeo «Clases de conocimiento» a través del aula virtual

Para resolver problemas de modo natural es necesario llevar a cabo un **análisis preciso del conocimiento**. Para ello, debemos tener en cuenta las distintas clasificaciones de conocimiento que se emplean en la inteligencia artificial (Mira, 1995).

En el trabajo de (Molina, 2006) hay un detalle más extenso de los tipos de conocimiento. Lo pueden ver en la Figura 3.

Conocimiento de dominio

- Son aquellos criterios específicos de un problema o entorno concreto.
- Su representación es declarativa.
- No tienen que presentar orden o relación entre ellos y pueden ser no completos.

Conocimiento explícito

- Extraído del análisis introspectivo de los propios razonamientos.
- Conociendo los procesos realizados para resolver un problema, se extraen las relaciones por medio de un análisis detenido.
- Se suele expresar por medio de marcos o reglas.

Conocimiento implícito

- Aquel que refiere a capacidades o habilidades innatas que, por su naturaleza o complejidad, no son expresadas de modo sencillo verbalmente.
- Se emplean redes bayesianas o redes neuronales para su representación o modelización.

Conocimiento superficial

- Obtenido por medio de la experiencia en la resolución de otros problemas similares.
- Por medio de reglas se modeliza el problema sin hacer explícitos los detalles teóricos que los sustentan.

Conocimiento profundo

- Responde a un marco teórico bien estructurado.
- No está presente en muchos problemas porque no es sencillo tener un análisis teórico del funcionamiento del entorno en todos los problemas.
- Contraposición entre problemas de mecánica y/o problemas de medicina o economía.

Conocimiento de control

- Representa el modelo del problema por medio del orden de ejecución de un conjunto de pasos para resolver el problema.
- En muchos casos, puede derivar en la codificación de un programa en la secuencia de expansión de los algoritmos de búsqueda.

Metaconocimiento

- Conocimiento que permite generar nuevos modelos a partir de modelos de problemas anteriores.
- Emplea mecanismos de inferencia general sobre los problemas y modelos lógicos.
- Establece relaciones entre niveles de bases de conocimiento, es decir, se pueden expresar nuevos elementos en una base de conocimiento a partir de otros conocimientos de otra base.

Figura 3. Tipos de conocimiento.

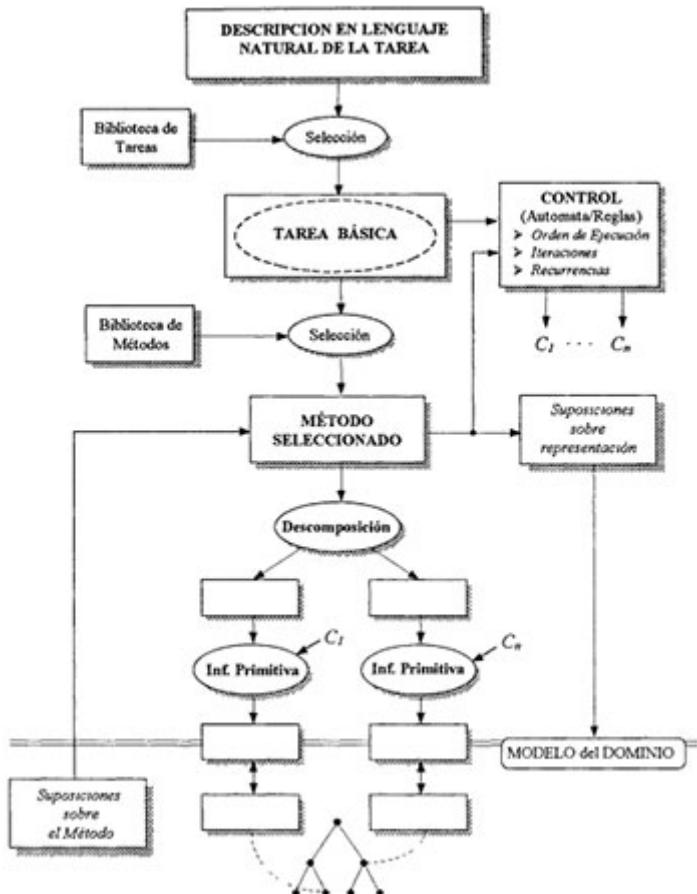


Figura 4. Esquema conceptual de las distintas fases del desarrollo de un sistema basado en el conocimiento.

Fuente: (Galán, 2008)

La Figura 4 muestra un esquema conceptual de las distintas fases del desarrollo de un sistema basado en el conocimiento.



Accede a los ejercicios de autoevaluación a través del aula virtual

2.4. Tipos de razonamiento



Accede al vídeo «Tipos de razonamiento» a través del aula virtual

Cuando hablamos de razonamiento, nos referimos a un conjunto de actividades mentales que conectan las ideas basándose en reglas que justifican una idea, permitiendo resolver problemas a través de conclusiones.

Veamos algunas interpretaciones de la definición de razonamiento a través de varios autores:

«El razonamiento es un conjunto de proposiciones relacionadas de tal manera que la proposición final (conclusión) se obtiene de las proposiciones iniciales (premisas), dando lugar a un conocimiento nuevo que supera el expresado en las premisas» (Contreras, 1992).

«El razonamiento es el acto mediante el cual progresamos en el conocimiento con ayuda de lo ya conocido. Las proposiciones de lo conocido se llaman premisas y el conocimiento que se obtiene de las premisas es la conclusión» (Napolitano, 1989).

«El razonamiento es un proceso a través del cual, dadas unas premisas verdaderas, o supuestamente verdaderas, se pasa a afirmar una nueva proposición (conclusión) que se fundamenta en las premisas» (Muñoz, 1992).

Como se puede observar en estas definiciones, los autores se refieren a los mismos conceptos implicados: premisas + conclusión.



Figura 5. Razonamiento.

En todo razonamiento existen dos elementos: **contenido y forma**.

- ▶ Contenido: es lo que hace que una proposición sea verdadera o falsa. Es la referencia a los objetos y las propiedades.
- ▶ Forma: es el nexo lógico entre los antecedentes (lo ya conocido) y los consecuentes (la conclusión inferida a partir de los antecedentes). Este nexo que implica inferencia se expresa mediante conjunciones. La forma es la que hace que la proposición sea válida, y consiste en el uso de símbolos para expresar la validez de las proposiciones.

El razonamiento puede clasificarse en: **razonamiento deductivo** y **razonamiento no deductivo**.

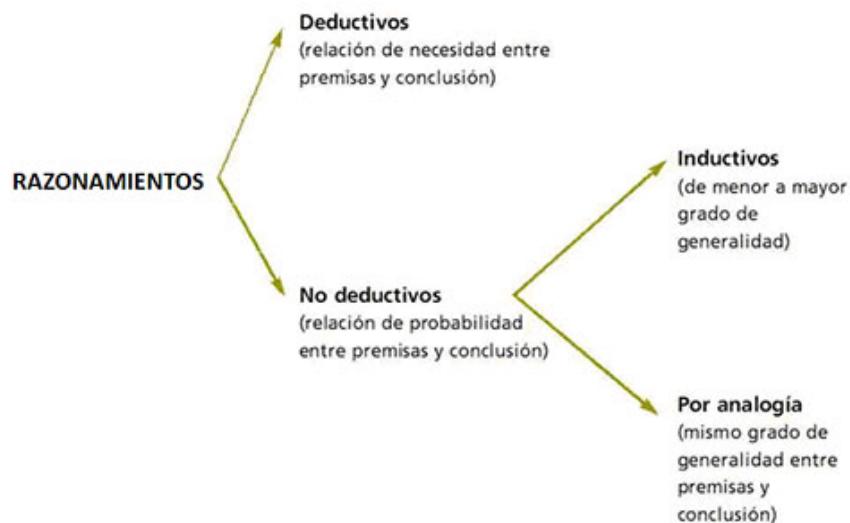


Figura 6. Clasificación del razonamiento.

Los tipos de razonamiento son:

- ▶ Analógico: método de procesamiento de la información donde se utilizan similitudes para lograr una conclusión. Se parte del conocimiento de una característica común entre dos objetos y se llega a la conclusión de que otros objetos similares tendrán las mismas características. Es **no deductivo**. Emplea comparaciones. Establece el tipo de relaciones: A es a B como C es a D.
- ▶ Válido e inválido: hablamos de **razonamiento válido** cuando la conclusión parte de las premisas. Se considera que un razonamiento es válido, partiendo de la

forma lógica, aunque la conclusión o las premisas sean falsas; mientras que un **razonamiento inválido** se produce cuando, a partir de premisas verdaderas, se obtiene una conclusión falsa.

<i>Si las premisas son...</i>	<i>Y la conclusión es...</i>	<i>El razonamiento es...</i>	<i>p</i>	<i>q</i>	<i>p → q</i>
Verdaderas	Verdadera	Válido	1	1	1
Verdaderas	Falsa	Inválido	1	0	0
Falsas	Verdadera	Válido	0	1	1
Falsas	Falsas	Válido	0	0	1

Validez o no Validez de un Razonamiento

Tabla de verdad de una proposición condicional

Tabla 1. Validez o no validez de un razonamiento. Fuente: (Sejas, 2012)

Todo argumento puede representarse mediante una proposición condicional con las premisas como antecedente y con la conclusión como consecuente.

- ▶ Lógico: también llamado razonamiento formal, es el proceso mental que implica la aplicación de la lógica. Se puede partir de una o varias premisas para obtener una conclusión verdadera, falsa o posible. También se denomina razonamiento causal cuando, a partir de uno o varios juicios, se deriva la validez, la posibilidad o la falsedad de un juicio diferente. Puede iniciarse a partir de una observación o hipótesis. Puede transformarse en un razonamiento inductivo o en un razonamiento deductivo. Son tipos de razonamiento lógico: **razonamiento inductivo, razonamiento deductivo y razonamiento deductivo**.

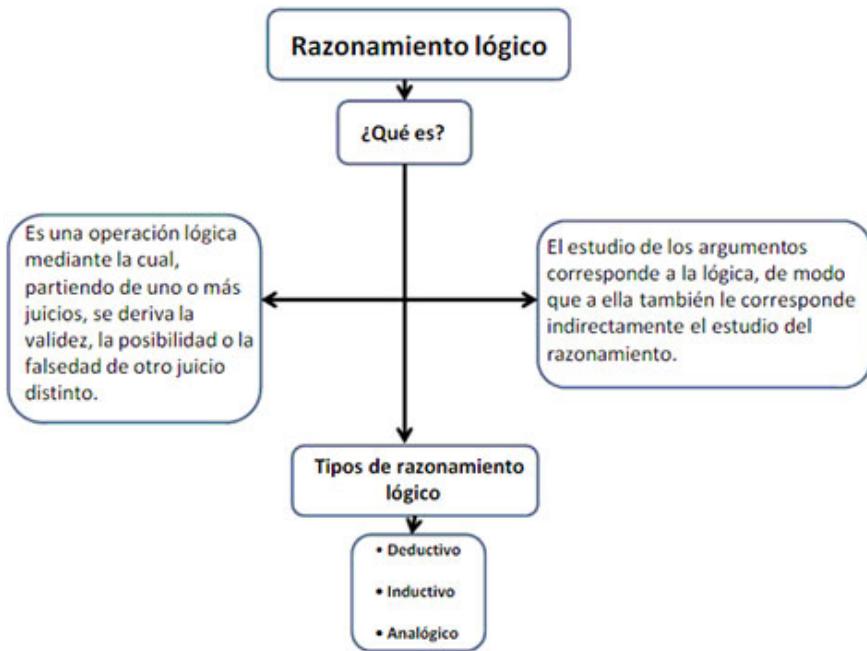


Figura 7. Esquema de razonamiento lógico.

- ▶ No-lógico: también conocido como razonamiento informal. No solo se parte de premisas (como ocurre en el razonamiento lógico), también se tiene en cuenta el contexto y la experiencia.
- ▶ Argumentativo: un argumento es toda expresión lingüística de un determinado razonamiento; por tanto, el razonamiento argumentativo refiere a la actividad lingüística donde se puede argumentar. Establece una secuencia de ideas en un argumento que, por medio de distintos mecanismos de razonamiento, intentamos proceder desde una tesis hasta una conclusión. Podemos distinguir **cinco modos de razonamiento argumentativo**: por analogía, por generalización, por signos, por causa y por autoridad.
- ▶ Por analogía: consiste en emplear el recurso de la analogía para explicar nuevos fenómenos, comparando punto por punto un fenómeno desconocido con otro conocido para llegar a comprender el primero. La analogía permite garantizar que, a partir de una tesis aceptada, la misma tesis es válida para otro fenómeno u objeto. Asumimos que, si dos cosas son muy parecidas entre sí, entonces la tesis válida para una de ellas es también válida para la otra.
- ▶ Hipotético: es un análisis de consecuencias en un escenario imaginario o ficticio.

A continuación, vamos a centrarnos en los razonamientos más importantes para la inteligencia artificial.



Accede a los ejercicios de autoevaluación a través del aula virtual

2.5. Razonamiento lógico deductivo



Accede al vídeo «Razonamiento lógico deductivo» a través del aula virtual

Deducir es elaborar inferencias. Las inferencias deductivas son aquellas que pueden ser válidas o inválidas. Es un razonamiento cuya conclusión es de consecuencia necesaria; es decir, dadas unas determinadas premisas, se dice necesariamente una conclusión (Napolitano, 1989).

Un razonamiento es deductivo cuando en él se exige que la conclusión se derive necesariamente, forzosamente de las premisas. Por ello, se le considera rigurosamente (Contreras, 1992).

Ejemplo de razonamiento deductivo:

- ▶ Si nieva, entonces hace frío.
- ▶ Nieva.
- ▶ Luego tengo frío.

Se entiende que existe validez cuando, a partir de las premisas verdaderas, no se consigue una conclusión falsa. De premisas falsas pueden derivarse conclusiones verdaderas y, sin embargo, el argumento ser válido.

La verdad se da cuando lo que se describe en las premisas se corresponde con la realidad. Este tipo de razonamiento va de lo general a lo particular.

Dentro del razonamiento deductivo se distinguen varios tipos:

- ▶ Razonamiento deductivo categórico: parte de dos premisas verdaderas que darán lugar a una conclusión verdadera.
- ▶ Razonamiento deductivo proposicional: relaciona dos premisas donde una es condición de la otra, antecedente y consecuente.
- ▶ Disyunción o dilema: la relación entre las premisas es de contrarios, por lo tanto, la conclusión descarta una de ellas.

Existen dos formas de razonamiento deductivo: **inmediato** (la única operación lógica es el cambio de juicio) y **mediato** (se establece una relación de mediación entre juicios para llegar a la conclusión).

Deducción (Peirce, 1867):

Premisa mayor	MP	Los seres humanos son mortales.
Premisa menor	SM	Los griegos son seres humanos.
Conclusión	SP	Los griegos son mortales.

Tabla 2. Deducción.

Los objetivos del razonamiento deductivo se alcanzan a través del **método deductivo**, una serie de pasos prefijados que permiten alcanzar con éxito el objetivo. El método deductivo, utilizado por Euclides (325-265 a. C.) en su geometría, y basado en la lógica de Aristóteles, deduce teoremas a partir de principios universales. Es un método que no proporciona información nueva.

Observa y analiza el siguiente esquema de funcionamiento del método deductivo:



Figura 8. El método deductivo. Fuente: (E-ducativa, 2020)

El método deductivo parte de lo general para llegar a lo particular.



Figura 9. Pasos que sigue el método deductivo.



Accede a los ejercicios de autoevaluación a través del aula virtual

2.6. Razonamiento lógico inductivo



Accede al vídeo «Razonamiento lógico inductivo» a través del aula virtual

El razonamiento inductivo crea conclusiones probables de acuerdo a las premisas dadas. Se basa en que, si diversos acontecimientos presentan la misma situación que la de sus premisas, existe la probabilidad de que el resultado sea idéntico. Inducir significa precisamente extraer conclusiones generales de experiencias particulares.

La diferencia con el razonamiento deductivo es que **la conclusión no se obtiene obligatoriamente de las premisas**. La conclusión del razonamiento inductivo se obtiene con la observación directa de casos particulares.

Observa las diferencias en el siguiente esquema:



Figura 10. Diferencias entre razonamiento inductivo y deductivo. Fuente: (Respuestas.tip, 2020)

Veamos el ejemplo de los griegos desde la perspectiva de la inducción (Peirce, 1867):

Premisa menor (original)	SM	Los griegos son seres humanos.
Conclusión (original)	SP	Los griegos son mortales.
Premisa mayor (original)	MP	Los seres humanos son mortales.

Tabla 3. Inducción.

Dentro del razonamiento inductivo hay distintos tipos:

- ▶ Razonamiento inductivo completo (también llamado razonamiento inductivo perfecto): ocurre cuando todos los casos particulares están incluidos en las premisas.
- ▶ Razonamiento inductivo incompleto o razonamiento inductivo imperfecto: en las premisas únicamente se incluyen determinados casos particulares.

Los objetivos del razonamiento inductivo se alcanzan a través del **método inductivo**, una serie de pasos prefijados que permiten alcanzar con éxito el objetivo.

El método inductivo fue propuesto por Francis Bacon para tratar de generalizar conclusiones de carácter universal a partir de la observación de casos particulares. El método inductivo tiene riqueza de información.

A continuación, se muestra un esquema de funcionamiento del método inductivo:



Figura 11. Funcionamiento del método inductivo. Fuente: (E-ducativa, 2020)

El método inductivo parte de lo particular para llegar a lo general.

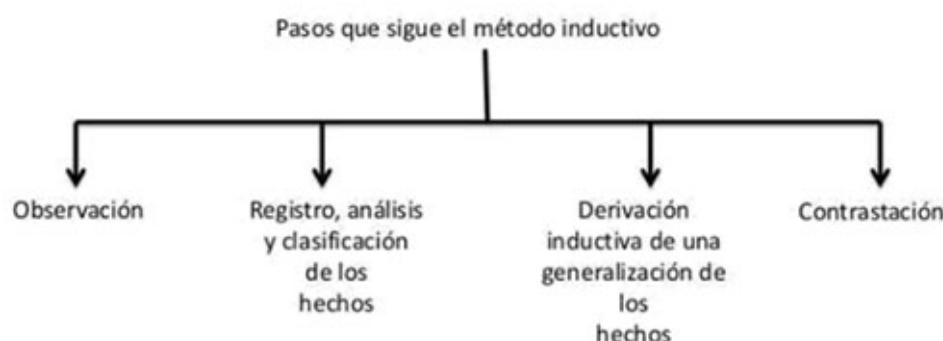


Figura 12. Pasos que sigue el método inductivo.



Accede a los ejercicios de autoevaluación a través del aula virtual

2.7. Razonamiento lógico abductivo



Accede al vídeo «Razonamiento lógico abductivo» a través del aula virtual

El concepto clave en este tipo de razonamiento es el silogismo. A través del silogismo, se considera la premisa mayor como cierta y la premisa menor como probable, lo que conduce a obtener una conclusión que posee el mismo nivel de probabilidad que la premisa menor. Es decir, el silogismo está constituido por al menos dos premisas —particulares (se refieren solo a un caso o a un único individuo) o universales (se aplican a todos los casos)— y una conclusión (resultado de comparar las premisas). El primero en dar a conocer este tipo de razonamiento fue Aristóteles.

A partir de un hecho se llega a las acciones que lo causaron. Es un método empleado para encontrar explicaciones de hechos observados.

Por tanto, la abducción consiste en elaborar una hipótesis explicativa con base al siguiente esquema: veo A con la característica Z. Como todos los A que veo son Z, entonces cualquier elemento A tiene la característica Z.

La abducción parte de hechos y busca una teoría (del efecto a la causa). Es un concepto que no puede ser directamente observado, relaciona lo observable con algo diferente que posiblemente tampoco será observado.

El razonamiento abductivo implica un proceso que abarca tres pasos:

- ▶ El objeto o hecho.
- ▶ Hipótesis de por qué ocurre el objeto o hecho.
- ▶ Afirmar que la causa fue la responsable del objeto o hecho.

Para Charles S. Peirce (Peirce, 1867), la abducción o *retroducción* es un proceso inferencial que está relacionado con la generación de hipótesis, ya sea en el razonamiento científico o en el razonamiento ordinario.

El ejemplo de los griegos desde la abducción (Peirce, 1867):

Premisa mayor (original)	SM	Los seres humanos son mortales.
Conclusión (original)	SP	Los griegos son mortales.
Premisa menor (original)	MP	Los griegos son seres humanos.

Tabla 4. Abducción.

La lógica y el razonamiento abductivo son fundamentales, ya que permiten enriquecer cualquier proceso en la fase de prueba, aportando una perspectiva de cambio.

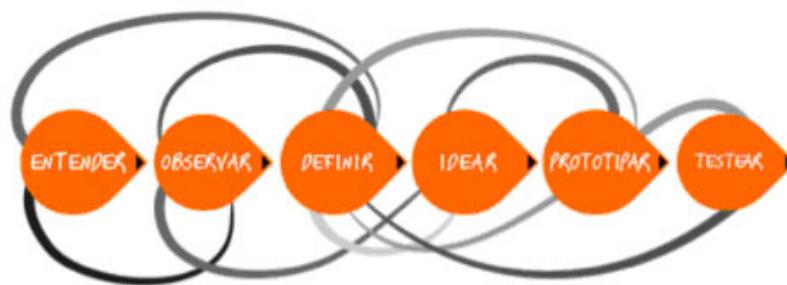


Figura 13. La lógica y el razonamiento abductivo en la innovación.

Fuente: (Sobejano, 2013)

El razonamiento abductivo permite pensar de manera alternativa, sin seguir los caminos habituales de razonamiento, lo que conduce a soluciones disruptivas y novedosas. Es contrario al razonamiento deductivo, que nos lleva a quedarnos en la zona de confort. La innovación está fuertemente ligada al razonamiento abductivo (Sobejano, 2013)

El **método abductivo** nos permite enfrentar hechos que parecen inexplicables. Es un proceso de creación basado en la observación de un fenómeno, que puede clasificarse como sorprendente, y en la investigación del hecho. Los raptos son conjetas espontáneas de la razón. Para que surjan, la imaginación y el instinto son necesarios, para ir más allá de lo que se conoce. El *musement sense*, del que habla Peirce, es un momento más intuitivo que racional en el que hay un flujo de ideas hasta que de repente la idea aparece como una iluminación, según (Peirce, 1867) «el

secuestro es el primer paso del razonamiento científico» y eso dará comienzo a una restricción de hipótesis aplicable a un fenómeno.

En la siguiente figura se resumen las diferencias entre los tres tipos de razonamiento analizados en detalle en este tema.



Figura 14. Diferencias entre tipos de razonamiento.



Accede a los ejercicios de autoevaluación a través del aula virtual

2.8. Referencias bibliográficas

Brachman, R. y. (1985). *Readings in Knowledge Representation*. Los Altos: Morgan Kaufman.

Contreras, B. (1992). *Lógica simbólica*. San Cristóbal: Universidad Católica del Táchira.
E-ducativa. (2020). <http://e-educativa.catedu.es/>. Obtenido de 4.1. El método inductivo y el método deductivo: <http://e->

ducativa.catedu.es/44700165/aula/archivos/repositorio/1000/1248/html/41_el_mtodo_inductivo_y_el_mtodo_deductivo.html

Galán, S. F. (2008). *Problemas resueltos de inteligencia artificial aplicada: búsqueda y representación*. Madrid: Pearson Educación.

Mira, J. D. (1995). *Aspectos básicos de la inteligencia artificial*. Madrid: Sanz y Torres.

Molina, M. (2006). *Métodos de resolución de problemas: aplicación al diseño de sistemas inteligentes*. Madrid: Fundación General de la UPM. . Obtenido de Recuperado de <http://oa.upm.es/14207/>

Muñoz, A. (1992). *Lógica simbólica elemental*. Venezuela: Miró.

Napolitano, A. (1989). *Lógica Matemática*. Caracas: Biosfera.

Peirce, C. (1867). On the Natural Classification of Arguments. *Proceedings of the American Academy of Arts and Sciences*, (págs. 7, 261-287).

Respuestas.tip. (2020). *¿Como diferenciar el razonamiento inductivo del deductivo?* Obtenido de Respuestas.tip: <https://respuestas.tips/como-diferenciar-el-razonamiento-inductivo-del-deductivo/>

Seijas, L. (27 de 05 de 2012). *Monografías*. Obtenido de Elementos y tipos de razonamiento: <http://www.monografias.com/trabajos72/elementos-tipos-razonamiento/elementos-tipos-razonamiento2.shtml>

Sobejano, J. (17 de 07 de 2013). *La lógica de la innovación*. Obtenido de Sintetia: <https://www.sintetia.com/la-logica-de-la-innovacion/>



Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Lógica y pensamiento humano

Esquema. Tema 3

Tipos de lógica				
Matemática	Descriptiva ALC	Orden Superior	Multivaluada y difusa	Otros

Ideas clave. Tema 3

3.1. ¿Cómo estudiar este tema?

La lógica es clave para tratar de representar el pensamiento humano. A través de la lógica podemos implementar sistemas que emulen el pensamiento. Es, por tanto, una herramienta para explicar todo el conocimiento basado en los **elementos de razonamiento**: categorías, definiciones, juicios y proposiciones.

A lo largo de la historia, desde Aristóteles hasta nuestros días, se han ido produciendo avances que han evolucionado la lógica y han permitido que se convierta en una herramienta potente a la hora de modelizar comportamiento dentro del campo de la inteligencia artificial.

En el siguiente tema veremos algunos **tipos de lógica** y detallaremos los que tienen más relevancia para inteligencia artificial.



Accede a los ejercicios de autoevaluación a través del aula virtual

3.2. Tipos de lógica



Accede al vídeo «Tipos de lógica» a través del aula virtual

La definición que podemos encontrar en el *Diccionario de la lengua española* para la palabra lógica es la siguiente: «Ciencia que expone las leyes, modos y formas de las proposiciones en relación con su verdad o falsedad».

La **lógica** (su etimología está en la palabra *lógica* del latín, que a su vez proviene del griego *logike*: «argumentativo, intelectual», y esta, a su vez, deriva de *logos*: «pensamiento, razón, palabra») es una ciencia formal basada en las leyes del conocimiento científico cuyo objetivo es estudiar los métodos y principios para identificar el razonamiento correcto.

Básicamente, la lógica permite realizar una representación formal de las relaciones existentes entre los objetos (y entre los objetos y sus propiedades).

El filósofo griego Aristóteles es considerado el padre de la lógica, ya que fue el primero en mostrar interés por el razonamiento lógico y emplear sistemas de validación de argumentos como indicadores de verdad, utilizando el silogismo (razonamiento que está formado por dos premisas y una conclusión que es el resultado lógico que se deduce de estas) como argumento válido y desarrollando un sistema lógico que ha llegado a nuestros días.

Se distinguen **varios tipos de lógica**, todas centradas en entender los razonamientos y diferenciar si son correctos o incorrectos. Para ello se basan en el estudio de los enunciados, abarcando más allá del lenguaje natural (el discurso verbal) y llegando a áreas muy variadas como las matemáticas y las ciencias de la computación, con estructuras muy diferentes.

Generalmente, dentro del término lógica se engloban varias formas:

- ▶ Lógica proposicional: se utilizan proposiciones que representan afirmaciones que pueden ser verdaderas o falsas. Las proposiciones se unen con (\wedge [y], \vee [o], \neg [no]) y se construyen reglas con el operador de implicación lógica (\rightarrow). Los mecanismos de inferencia permiten obtener nuevos datos a partir de los datos ya conocidos (por ejemplo, *modus ponens* y *modus tollens*...).
- ▶ Lógica de predicados: añade la posibilidad de utilizar cuantificadores:
 \forall (para todo).
 \exists (existe).

Emplean como mecanismos de inferencia los tradicionales: *modus ponens*, o *modus tollens*. La Figura 1 muestra un ejemplo de lenguaje de programación que emplea la modelización de lógica de predicados es PROLOG.

Para mostrar un ejemplo que contiene elementos de un lenguaje de primer orden vamos a trabajar sobre la siguiente base de conocimientos:

Hechos:

1. Atlanta se encuentra en Georgia.
2. Houston y Austin se encuentran en Texas.
3. Toronto se encuentra en Ontario.

Que, usando el predicado `located_in`, podemos representar con las siguientes clausulas:

```
located_in(atlanta,georgia). % Clause 1
located_in(houston,texas). % Clause 2
located_in(austin,texas). % Clause 3
located_in(toronto,ontario). % Clause 4
```

Reglas:

1. Lo que está en Georgia o Texas, también está en USA.
2. Lo que está en Ontario, también está en Canadá.
3. Lo que está en USA o Canadá, también está en Norte América.

Que podemos representar con las siguientes clausulas (`geo.pl`):

```
located_in(X,usa) :- located_in(X,georgia).      % Clause 5
located_in(X,usa) :- located_in(X,texas).          % Clause 6
located_in(X,canada) :- located_in(X,ontario).     % Clause 7
located_in(X,north_america) :- located_in(X,usa).   % Clause 8
located_in(X,north_america) :- located_in(X,canada). % Clause 9
```

Observa que al estar trabajando con predicados que sí reciben argumentos ya no es necesario usar la directiva vista en el ejemplo anterior.

Vamos a ver cuál es el árbol de deducción que sigue Prolog para resolver la siguiente clausula:

```
located_in(toronto,north_america).
```

Figura 1. Ejemplo de modelización en PROLOG.

Fuente: (Caparrini, 2014)

- ▶ Lógica natural: directamente relacionada con el empirismo, aprender a base del acierto-error. La lógica natural es aquella que previene al ser humano de cometer el mismo error varias veces, es razón innata.
- ▶ Lógica científica: surge como la lógica natural de la experiencia, pero además se incluye en ella la razón, creando planteamientos de todo lo que existe. Se fundamenta en encontrar los motivos o justificaciones por los cuales sucede un hecho.
- ▶ Lógica material: se estudia desde la epistemología (una de las ramas modernas de la filosofía). Incluye la incertidumbre, ya que las conclusiones implican cierto grado de duda. El objetivo es probar la validez de un racionamiento basándose en la realidad. Un ejemplo: si está nublado, es posible que llueva, como es posible que

no llueva, por lo que el razonamiento «es posible que caigan unas gotas» es correcto, pero no es válido porque no existe total seguridad de que esto vaya a suceder.

- ▶ Lógica formal: se llama a la lógica clásica (aristotélica) que estudia las proposiciones, los argumentos, desde el enfoque estructural. La motivación es determinar si un enfoque es correcto o incorrecto a través de un método para estructurar el pensamiento. El objeto de estudio no es empírico, sino que se centra en la estructura del argumento y no en la falsedad o veracidad del contenido de un argumento particular. En el ejemplo que usábamos para la lógica de materiales, bajo el mismo escenario (cielo nublado, puede llover o no), el pensamiento «es posible que caigan unas pocas gotas» sería válido. Incluidos en la lógica formal podemos encontrar dos tipos muy importantes: **Lógica deductiva y lógica inductiva**.
 - Lógica deductiva: realizar inferencias a partir de teorías que ya existen. Por ejemplo: si los humanos tienen orejas y Manuel es un humano, entonces Manuel tiene orejas.
 - Lógica inductiva: consiste en crear conceptos generales a partir de argumentos específicos. Por ejemplo: si un humano tiene orejas, existe otro humano con orejas y otro más que también tiene orejas, entonces todos los humanos tienen orejas.
- ▶ Lógica de primer orden: es un sistema formal diseñado para estudiar la inferencia en los lenguajes de primer orden; son lenguajes cuantificadores que alcanzan variables de individuos con predicados y funciones constantes o variables. En primer lugar, es importante entender el concepto **«enunciado declarativo»**, expresiones que son verdaderas o falsas enunciadas con el lenguaje natural (idioma en el que se habla), y la diferencia que existe con los **enunciados interrogativos e imperativos**. En la lógica de primer orden, se establecen una serie de objetos y relaciones entre los mismos. Son enunciados declarativos los siguientes ejemplos:

1. Laura es madre y María es hija de Laura.
2. Toda madre quiere a sus hijas.

1 se refiere a los objetos sobre los que versa el discurso, describiendo las propiedades de estos (ser madre, ser hija), mientras que 2 define las relaciones entre los objetos. Esto sería el comienzo de la lógica computacional, en la que es necesario definir las situaciones en objetos y las relaciones que se establecen entre ellos.

Cuando sobre los enunciados (predicados) se aplican reglas de razonamiento, se obtienen conclusiones. De este modo, se resuelven problemas. Sobre los enunciados que exponemos más arriba (1 y 2) es posible inferir un tercer enunciado:

3. Laura quiere a María.

También es posible utilizar funciones cuando queremos resolver un problema. Las funciones son relaciones en las que solo existe una correspondencia dado un valor. Por ejemplo, es posible definir la función madre de.

Otros conceptos para tener en cuenta, como en todo sistema formal, son la sintaxis (expresiones de lógica de primer orden) y la semántica (significado que hace que la expresión sea verdadera o falsa).

- ▶ Lógica simbólica o matemática: utiliza símbolos para construir un nuevo lenguaje en el cual se expresan los argumentos. La intención es traducir el pensamiento humano a lenguaje matemático, es decir, conseguir convertir el pensamiento abstracto en estructuras formales y emplear las leyes del cálculo (base de la exactitud), permitiendo que los argumentos sean más exactos. Este tipo de lógica, en matemáticas, se emplea para demostrar teoremas.
- ▶ Lógica de clases: la base de los principios de la lógica de clases se fundamenta en la teoría de conjuntos. Se analiza una proposición lógica sobre la pertenencia, o no

pertenencia, de un elemento o individuo a una determinada clase (conjunto de elementos o individuos que tiene en común alguna característica o propiedad particular). Es la característica (propiedad) la que define la clase. Por ejemplo, no es lo mismo decir «Einstein era un hombre» que afirmar «Einstein pertenecía a la clase de los hombres».

También se suele hablar de otros tipos y subtipos de lógica:

- ▶ Lógica informal: se centra en el lenguaje y en el significado de las construcciones semánticas y de los argumentos. Se diferencia de la lógica formal en que la lógica informal se centra en el contenido de las oraciones y no en la estructura. El objetivo es encontrar la manera de argumentar para conseguir el resultado que se desea obtener.
- ▶ Lógica moderna: nacida en el siglo XIX, se diferencia de la lógica clásica porque incluye elementos matemáticos y simbólicos, teoremas que reemplazan las carencias de la lógica formal. Se diferencian varios tipos de lógica moderna: lógica modal, lógica matemática, lógica trivalente...
- ▶ Lógica modal: este tipo de lógica emplea los argumentos y añade elementos (operadores modales) que hacen posible determinar si un enunciado es verdadero y falso. La intención es estar en consonancia con el pensamiento humano; por tanto, tiene en consideración expresiones como «siempre», «es muy probable», «a veces», «tal vez».
- ▶ Lógica computacional: deriva de la lógica simbólica (matemática o de primer orden) y se aplica al área de las ciencias de la computación. A través de la lógica, es posible trabajar con lenguajes de programación con el fin de ejecutar tareas específicas de verificación.

En este tema profundizaremos en las lógicas que están más relacionadas con la inteligencia artificial: **lógica matemática, lógica descriptiva o de descriptores ALC, lógica de orden superior, lógica multivaluada y lógica borrosa o difusa**.



Accede a los ejercicios de autoevaluación a través del aula virtual

3.3. Lógica matemática



Accede al vídeo «Lógica matemática» a través del aula virtual

La **lógica matemática** consiste en el estudio matemático de la lógica y en la aplicación de este a otras áreas de las matemáticas. Está en estrecha relación con las ciencias de la computación y la lógica filosófica.

Estudia los sistemas formales en relación con el modo en el que codifican nociones intuitivas de objetos matemáticos como conjuntos, números, demostraciones y computación.

Se le aplica a las definiciones concretas y razonamientos rigurosos propios de la matemática. Cada expresión del lenguaje tiene un significado exacto y un simbolismo apropiado, sin ambigüedades.

El siguiente esquema muestra un resumen de la lógica matemática (<https://cmapspublic.ihmc.us/rid=1LNTSCHKK-LLS690-2JGW/Tipos%20de%20Agentes%20Inteligentes.cmap>):

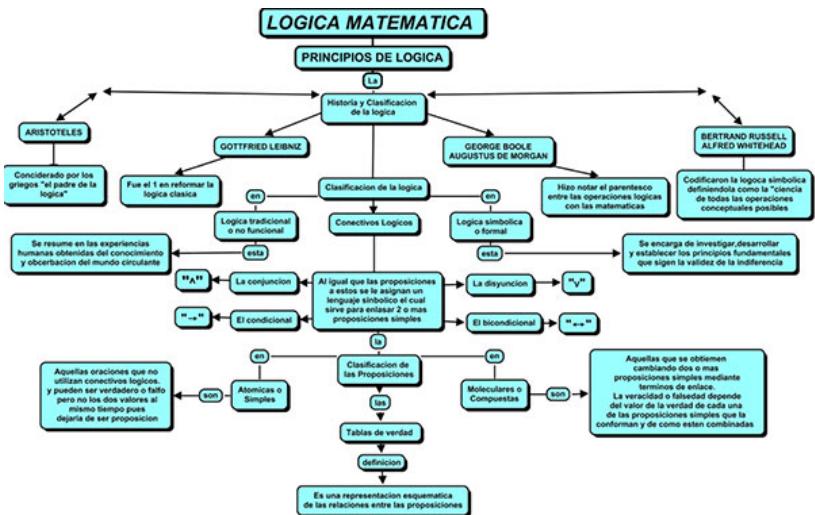


Figura 2. Resumen de la lógica matemática.

Fuente: (Reina, 2016)

Una **proposición** es toda afirmación o expresión que tiene significado y de la que podemos decir si es «falsa» (F/0) o «verdadera» (V/1). Las proposiciones pueden enlazarse entre sí mediante conectivos lógicos para formar estructuras con un significado preciso.

No son proposiciones: las frases para dar órdenes (lee esto), las frases exclamativas e interrogativas (¿cómo te llamas?), e instrucciones (vuelve hacia atrás).

Las proposiciones **se suelen representar con una letra minúscula**. Por ejemplo, p, q, r.

Relacionando proposiciones es posible obtener otras proposiciones. Todo razonamiento lógico debe partir necesariamente de una adecuada vinculación de algunas proposiciones elementales.

La lógica matemática parte de proposiciones elementales como axiomas, postulados o hipótesis y emplea razonamientos lógicos para determinar si las conclusiones obtenidas son verdaderas o falsas.

Las proposiciones se pueden clasificar en:

- ▶ Tautologías: una proposición compuesta es una tautología si es verdadera para todas las asignaciones de valores de verdad, para sus proposiciones componentes. Es decir, su valor verdadero no depende de los valores de verdad de las proposiciones que la forman, sino de la manera en que se establecen relaciones sintácticas de unas proposiciones con las otras.
- ▶ Contradicciones: aquellas proposiciones que, en todos los casos posibles de su tabla de verdad, su valor es falso siempre. Su valor falso no depende de los valores de verdad de las proposiciones que la forman, sino de la manera en que se establecen relaciones sintácticas de unas proposiciones con las otras.
- ▶ Contingentes, falacias o inconsistencias: también denominadas verdad de hecho, es aquella proposición que puede ser verdadera o falsa, combinando tautología y contradicción, según los valores de las proposiciones que la integran.

Los elementos que relacionan proposiciones son los **conectivos lógicos** u **operaciones lógicas**.

Conejtos lógicos	Símbolo
y	\wedge
o	\vee
sí...entonces	\rightarrow
sí y solo si	\leftrightarrow
no	\neg

Tabla 1. Conejtos lógicos.

Los conceptos para tener en cuenta en las relaciones entre proposiciones son:

- ▶ Implicación lógica: cualquier condicional que sea tautología.

$$\begin{aligned} A \Rightarrow B \\ A \Rightarrow B \text{ es tautología} \end{aligned}$$

- ▶ Equivalencia lógica: es toda bicondicional que sea tautología.

$$\begin{array}{c} A \leftrightarrow B \\ A \leftrightarrow B \text{ es tautología} \end{array}$$

Las relaciones entre proposiciones se pueden reflejar en tablas de verdad, que son tablas que muestran el valor de verdad de una proposición compuesta. Las tablas de verdad fueron desarrolladas por C. S. Peirce hacia 1880, pero más tarde Ludwig Wittgenstein dio el formato que es más popular, en su *Tractatus logico-philosophicus* de 1921.

Las tablas de verdad son métodos sencillos con gran potencial, ya que sirven para demostrar propiedades lógicas y semánticas de proposiciones del lenguaje humano o de cálculo proposicional:

- ▶ Ver si son tautologías, contradictorias o contingentes.
- ▶ Ver las condiciones de verdad.
- ▶ Ver el rol inferencial, las conclusiones lógicas y qué proposiciones se consiguen a partir de otras, lógicamente hablando.

A continuación, veamos las tablas de los conectivos lógicos:

- ▶ Negación (también se representa con \sim):

p	$\neg p$
0	1
1	0

Figura 3. Negación.

- ▶ Conjunción:

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Figura 4. Conjunción.

- Disyunción no exclusiva:

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

Figura 5. Disyunción no exclusiva.

- Disyunción exclusiva:

p	q	$p \Delta q$
0	0	0
0	1	1
1	0	1
1	1	0

Figura 6. Disyunción exclusiva.

- Condicional:

p	q	$p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Figura 7. Condicional.

- Bicondicional:

p	q	$p \leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

Figura 8. Bicondicional.

A continuación, se muestra un ejemplo de tabla de verdad de una relación entre proposiciones:

$$(p \rightarrow q) \leftrightarrow (\neg p \wedge q)$$

P	Q	$\neg p$	$p \rightarrow q$	$\neg p \wedge q$	$(p \rightarrow q) \leftrightarrow (\neg p \wedge q)$
V	V	F	V	F	F
V	F	F	F	F	V
F	V	V	V	V	V
F	F	V	V	F	F

Tabla 2. Tabla de verdad.

La aplicación de la lógica proposicional más común es la que se realiza en juegos, de azar o estratégicos. También se ha aplicado en inteligencia artificial. La inteligencia artificial trata de explicar cómo funciona la mente humana, ya que utiliza algoritmos que controlan diferentes funciones. Tanto la robótica como los sistemas de agentes inteligentes son sistemas creados con la idea de tomar decisiones por sí mismos; por lo tanto, la conexión entre lógica matemática y lógica computacional que se utiliza en varios niveles es clara: circuitos informáticos, programación lógica, optimización.



Accede a los ejercicios de autoevaluación a través del aula virtual

3.4. Lógica de descripción ALC



Accede al vídeo «Lógica de descripción ALC» a través del aula virtual

Las lógicas descriptivas son más que los lenguajes para formalizar conceptos, son un conjunto de lenguajes de representación del conocimiento que se utilizan para representar el conocimiento terminológico de un dominio determinado de una manera estructurada y formal, de modo que esté bien entendido.

Deben usarse para representar la ontología (formalización de un dominio) y permitir el razonamiento al respecto. Aparecen nuevos elementos de lenguaje y semántica, necesarios para formalizar las propiedades de objetos o individuos que pertenecen

al dominio y las relaciones entre conceptos y roles al establecer las bases del conocimiento.

Usa lenguajes formales para definir vocabulario de dominio, compartir significado y deducir nuevo conocimiento.

Las lógicas descriptivas **son apropiadas para la web semántica** porque son útiles para agregar razonamiento a la red de redes. Tienen una sintaxis formal que permite describir los conceptos de nociones importantes de un universo o dominio, las relaciones que surgen o existen entre ellos y los constructores de nuevos conceptos. Como toda lógica formal, hace posible razonar sobre la base de conocimiento que se haya definido como tal. Son variantes de la lógica de primer orden (Huertas, 2006).

Características formales de la lógica descriptiva o lógica de descriptores:

- ▶ Como comentamos al principio, modelan ontologías, proporcionan descripciones a los dominios y formalizan los elementos de una terminología o descripciones de una ontología. La sintaxis y la semántica no tienen ambigüedad, ya que son formales.
- ▶ Formalismo descriptivo: roles, constructores y conceptos. Ejemplo de formalización del concepto: «animal cuyos padres son humanos».

Animal \sqcap $\forall tiene.Hijo.Humano.$

Donde los diferentes elementos que aparecen son:

- Un concepto primitivo: *Animal*
- Un rol o relación: $\forall tiene.Hijo.Humano$ («todo hijo es humano»)
- Un constructor de nuevos conceptos: \sqcap (conjunción de conceptos).

- ▶ Formalismo terminológico: emplea axiomas que introducen propiedades de la terminología descriptiva y descripciones complejas. Por ejemplo:

Mujer ⊑ *Persona* («una mujer es una persona»)

Hombre ≡ *Persona* □ ¬ *Mujer* («un hombre es una persona no mujer»)

- ▶ Formalismo asertivo que incluye propiedades de individuos. Por ejemplo:

maría: Mujer («el individuo María es una mujer»)

(jesús, maría): tieneHijo («el individuo María tiene el hijo individuo Jesús»)

- ▶ Es posible inferir un nuevo conocimiento a partir del dado. Se emplean cálculos o algoritmos de conocimiento que deciden. Permiten implementar procesos automatizados.

En resumen, la lógica descriptiva se compone de: conceptos (padre, madre, humano); relaciones entre los conceptos, denominadas propiedades o roles (*tieneHijo*, *esHijoDe*), y elementos del dominio, denominados individuos (María, Jesús).

La base de conocimiento tiene dos niveles:

- ▶ TBox (términos): la descripción de los conceptos. Conjunto de axiomas terminológicos.
- ▶ ABox (aserciones): la descripción de los individuos. Conjunto de axiomas asertivos.

	Sintaxis	Semántica (I es una interpretación de los símbolos de la sintaxis)
TBox	Nombres de individuos o, p, \dots	Objetos $I(o), I(p), \dots$ son elementos de Δ (dominio de interpretación)
	$C \sqsubseteq D$	Inclusiones de conceptos o roles $I(C) \subseteq I(D)$
	$R \sqsubseteq S$	$I(R) \subseteq I(S)$
	$C \equiv D$ $R \equiv S$	Igualdades de conceptos o roles $I(C) = I(D)$ $I(S) = I(S)$
ABox	$o : C$	Instanciación de concepto: a es del concepto C $I(o) \in I(C)$
	$(o, p) : R$	Instanciación de rol: b está relacionado con a por R $(I(o), I(p)) \in I(R)$

Figura 9. TBox y ABox. Fuente: (Huertas, 2006)

Las proposiciones en lógica descriptiva pueden representarse en lógica de primer orden:

Padre ≡ Persona $\cap \exists$ tieneHijo Persona

$\forall x(\text{Padre}(x) \leftrightarrow (\text{Persona}(x) \wedge \exists y(\text{tieneHijo}(x,y) \wedge \text{Persona}(y))))$

Orgulloso ≡ Persona $\cap \exists$ tieneHijo ReciénNacido

$\forall x(\text{Orgulloso}(x) \leftrightarrow (\text{Persona}(x) \wedge \exists y(\text{tieneHijo}(x,y) \wedge \text{RecienNacido}(y))))$

ReciénNacido \subseteq Persona

$\forall x(\text{RecienNacido}(x) \rightarrow \text{Persona}(x))$

Figura 10: Lógica de primer orden.

Definición de conceptos

Subclase: $C \subseteq D$ (C está incluido en D ó D subsume a C)

Intersección: $C \cap D$

Unión: $C \cup D$

Complemento: $\neg C$

Concepto vacío: \perp

Clases Disjuntas: $C \cap D \equiv \perp$

Equivalencia: $C \equiv D$

Tabla 3. Definición de conceptos.

Existen varias propiedades:

Existencial ($\exists R C$)

x pertenece a $\exists R C$ si existe algún valor $y \in C$ tal que $R(x, y)$

Universal ($\forall R C$)

x pertenece a $\forall R C$ si para todo y , si $R(x, y)$, entonces $y \in C$

Cardinalidad ($P = n$)

x pertenece a ($P = n$) si existen n $y \in C$ tales que $R(x, y)$

Cardinalidad máxima ($P \geq n$)

x pertenece a ($P \leq n$) si existen n o menos $y \in C$ tales que $R(x, y)$

Cardinalidad mínima ($P \geq n$)

x pertenece a ($P \geq n$) si existen n o más $y \in C$ tales que $R(x, y)$

Las propiedades pueden tener los siguientes atributos:

Reflexiva	P es reflexiva $\Rightarrow \forall x P(x, x)$
Irreflexiva	P es irreflexiva $\Rightarrow \forall x \neg P(x, x)$
Simetría	Si $P(x, y)$ entonces $P(y, x)$
Asimetría	Si $P(x, y)$ entonces $\neg P(y, x)$
Transitividad	Si $P(x, y)$ y $P(y, z)$ entonces $P(x, z)$

Tabla 4. Atributos de las propiedades.

Las relaciones que se establecen entre propiedades:

Inversa	P es inversa de $Q \Rightarrow P(x, y) \Leftrightarrow Q(y, x)$
Subpropiedad	P subpropiedad de Q si $P(x, y) \Rightarrow Q(x, y)$

Tabla 5. Relaciones entre propiedades.

Y tienen una serie de propiedades de funcionalidad:

Propiedad funcional	$P(x, y) \text{ y } P(x, z) \text{ entonces } y = z$
Propiedad funcional inversa	$P(x, y) \text{ y } P(z, y) \text{ entonces } x = z$
Claves	$P(x, y) \text{ y } P(z, y) \text{ entonces } x = z$

Tabla 6. Propiedades de funcionalidad.

A partir de una base de conocimiento se puede realizar un razonamiento (inferencia):

Satisfacción de conceptos	De \sum no se deduce que $C \equiv \perp$
Subsunción	$\sum \Rightarrow C \subseteq D$
Instanciación	$\sum \Rightarrow a \in C$
Recuperación de información	Dado un concepto C , obtener a tales que $a \in C$
Comprensión	Dado un elemento a , obtener concepto más específico C tal que $a \in C$

Tabla 7. Razonamiento.

ALC es la lógica descriptiva básica. En la siguiente figura hay una presentación formal de este sistema.

<i>ALC</i>	<i>Sintaxis</i>	<i>Semántica</i> (I es una interpretación de los símbolos de la sintaxis)
Nombres de conceptos atómicos	A, B, \dots	<i>Predicados unitarios</i> $I(A), I(B), \dots$ son subconjuntos de Δ (dominio de interpretación)
Nombres de roles atómicos	R, S, \dots	<i>Predicados binarios</i> $I(R), I(S), \dots$ son relaciones binarias sobre Δ
Conceptos Universal y Vacío	\top \perp	$I(\top) = \Delta$ Universal: describe el universo del dominio $I(\perp) = \emptyset$ Vacío: describe lo contradictorio.
Conceptos complejos (se obtienen a partir de los conceptos y roles atómicos usando constructores)	(C) $\neg C$	<i>Complementario: concepto con objetos que no son de C</i> $I(\neg C) = \Delta - I(C)$
	$C \sqcap D$	<i>Intersección de conceptos: concepto con objetos de C y D</i> $I(C \sqcap D) = I(C) \cap I(D)$
	(U) $C \sqcup D$	<i>Unión de conceptos: concepto con objetos de C o D</i> $I(C \sqcup D) = I(C) \cup I(D)$
	(E) $\exists R.C$	<i>Restricción existencial: el concepto cuyos objetos son los que están relacionados por R con los de C</i> $I(\exists R.C) = \{b \in \Delta / \text{existe } c \in \Delta ((b,c) \in I(R) \text{ y } c \in I(C))\}$
	$\forall R.C$	<i>Restricción universal: el concepto cuyos objetos se relacionan por R sólo con objetos de C</i> $I(\forall R.C) = \{b \in \Delta / \text{para todo } c ((b,c) \in I(R) \text{ implica } c \in I(C))\}$

Figura 11. ALC. Fuente: (Huertas, 2006)



Accede a los ejercicios de autoevaluación a través del aula virtual

3.5. Lógica de orden superior



Accede al vídeo «Lógica de orden superior» a través del aula virtual

Una lógica de orden superior o de segundo orden es una extensión de una lógica de primer orden en la que se añaden variables para propiedades, funciones y relaciones, y cuantificadores que operan sobre esas variables. Así, se expande el poder expresivo del lenguaje sin tener que agregar nuevos símbolos lógicos (Enderton, 2009).

La necesidad de la lógica de segundo orden se refleja en el axioma de inducción de la aritmética de Giuseppe Peano.

Se requiere un lenguaje en el que los cuantificadores puedan abarcar no solo las variables que representan los elementos de hormigón, sino también las relaciones o funciones.

En la lógica de primer orden, los cuantificadores solo se pueden aplicar a los objetos (elementos de primer orden), mientras que los de mayor orden son extensiones de la lógica de primer orden que permiten que los cuantificadores se apliquen a los predicados definidos en los objetos de segundo orden.

La lógica de segundo orden tiene un **poder expresivo mayor** que la de primer orden, lo que permite crear axiomas matemáticos de sistemas complejos que no son formalizables por medio de la lógica de primer orden.

Existen **varios tipos de lógica de segundo orden** según el tipo de variables adicionales introducidas con respecto a las de la lógica de primer orden:

- ▶ La lógica de segundo orden monádica (LSOM): añadimos variables para cierto dominio dentro de sus subconjuntos.

- ▶ La lógica de segundo orden completa (LSOC): se añaden tanto variables como cuantificadores que pueden referirse a cualquiera de estas variables.

Sintaxis de la lógica de segundo orden (LSO):

Dado: vocabulario X

La lógica de segundo orden (LSO) sobre \mathcal{L} es definida como la extensión de LPO que incluye las siguientes reglas:

- ▶ Si t_1, \dots, t_k son \mathcal{L} -términos y X es una variable de segundo orden de aridad k (vale decir, una relación con $k \geq 1$ argumentos), entonces $X(t_1, \dots, t_k)$ es una fórmula en LSO
- ▶ Si φ es una fórmula en LSO y X es una variable de segundo orden de aridad k , entonces $\exists X\varphi$ y $\forall X\varphi$ son fórmulas en LSO

Figura 12. Sintaxis de la lógica de segundo orden. Fuente: (Arenas, 2010)

Semántica de la lógica de segundo orden:

Dada una estructura \mathfrak{A} con dominio A, una asignación σ es una función que asigna:

- ▶ un valor en A a cada variable x de primer orden: $\sigma(x) \in A$
- ▶ un subconjunto de A^k a cada variable X de segundo orden con k argumentos: $\sigma(X) \subseteq A^k$

Figura 13. Semántica de la lógica de segundo orden. Fuente: (Arenas, 2010)

La definición de LSO incluye tres casos extra:

Para una variable de segundo orden X con aridad k:

- ▶ $(\mathfrak{A}, \sigma) \models X(t_1, \dots, t_k)$ si y sólo si $(\sigma(t_1), \dots, \sigma(t_k)) \in \sigma(X)$
- ▶ $(\mathfrak{A}, \sigma) \models \exists X\varphi$ si y sólo si existe $S \subseteq A^k$ tal que $(\mathfrak{A}, \sigma[X/S]) \models \varphi$
- ▶ $(\mathfrak{A}, \sigma) \models \forall X\varphi$ si y sólo si para todo $S \subseteq A^k$, se tiene que $(\mathfrak{A}, \sigma[X/S]) \models \varphi$

Figura 14. Casos extra de la lógica de segundo orden. Fuente: (Arenas, 2010)



Accede a los ejercicios de autoevaluación a través del aula virtual

3.6. Lógica multivaluada y lógica difusa



Accede al vídeo «Lógica multivaluada y lógica difusa» a través del aula virtual

La **lógica multivaluada** es una lógica que permite valores intermedios (grande, tibio, lejos, pocos, muchos, etc.) y en la que se emplean más de dos valores de verdad para describir conceptos que van más allá de lo verdadero y lo falso. Las lógicas multivaluadas ofrecen herramientas conceptuales que **hacen posible describir formalmente la información difusa, vaga o incierta**.

La **lógica difusa** (también llamada lógica borrosa) es una lógica multivaluada que permite **representar matemáticamente la incertidumbre y la vaguedad**, proporcionando herramientas formales para su tratamiento. Lofti A. Zadeh es considerado el padre de la lógica difusa. Su carrera se centró en trabajos sobre conjuntos difusos y la aplicación de la lógica difusa en el razonamiento aproximado. El término «lógica difusa» aparece por primera vez en 1974.

«Cuando aumenta la complejidad, los enunciados precisos pierden su significado y los enunciados útiles pierden precisión.» Esto puede resumirse en «los árboles no te dejan ver el bosque» (Zadeh, 1973).

El modelo de caracterizar un problema por medio de lógica difusa se basa en la prerrogativa de que el mapeo entre conceptos se realiza por medio de la semántica, no de la precisión numérica. Es muy adecuado para modelizar problemas a partir del conocimiento de los expertos que, normalmente, detallan su base de conocimiento en forma de expresiones poco precisas.

La aplicación de esta lógica en la Inteligencia artificial está orientada a manejar el razonamiento bajo incertidumbre y con nociones imprecisas. También puede emplearse para la gestión de bases de datos y sistemas basados en el conocimiento cuando se sepa que la información es imprecisa.

En procesos de automatización de las técnicas de prospección de datos, que suelen estar ligadas a conjuntos difusos o multivaluados, también interesa disponer de métodos de razonamiento automático para estas lógicas.

Un esquema de funcionamiento típico para un sistema difuso podría ser:

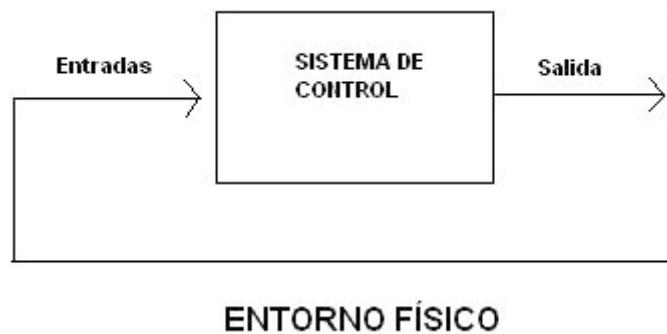


Figura 15. Esquema de funcionamiento para un sistema difuso.

Observando el esquema: el sistema de control hace los cálculos con base en sus reglas heurísticas, de la forma SI (antecedente) ENTONCES (consecuente), donde el antecedente y el consecuente son también conjuntos difusos.

Por ejemplo: SI hace calor, ENTONCES bajar temperatura. La salida final actuaría sobre el entorno físico y los valores sobre el entorno físico de las nuevas entradas (modificados por la salida del sistema de control) serían tomados por sensores del sistema.

Imaginemos que nuestro sistema difuso era el acondicionador de aire que regula la temperatura de acuerdo con las necesidades. Los chips difusos del acondicionador de aire recogen los datos de entrada, que en este caso podrían ser simplemente la temperatura y la humedad. Estos datos están sujetos a las reglas del motor de

inferencia (como se discutió anteriormente, en la forma SI... ENTONCES...), lo que deriva en un área de resultados. Desde esa área, se elegirá el centro de gravedad, proporcionándolo como una salida. Según el resultado, el acondicionador de aire podría aumentar la temperatura o disminuirla en función del grado de salida.



Accede a los ejercicios de autoevaluación a través del aula virtual

3.7. Referencias bibliográficas

Arenas, M. (12 de 12 de 2010). *Marenas.sitios.ing.uc.cl*. Obtenido de La lógica de segundo orden: Sintaxis: <http://marenas.sitios.ing.uc.cl/iic3260-10/clases/comp-lpo-ext-II.pdf>

Caparrini, F. S. (07 de 01 de 2014). *Fernando Sancho Caparrini*. Obtenido de Una introducción a Prolog: <http://www.cs.us.es/~fsancho/?e=73>

Enderton, H. (2009). Second-order and Higher-order Logic. En Zalta, E. (Ed.), *The Stanford Encyclopedia of Philosophy*. Stanford: Metaphysics Research Lab, Stanford University.

Huertas, A. (2006). Lógicas descriptivas: lógicas para la red. (págs. 8(85-102)). Azafea.
Reina, A. (22 de 02 de 2016). *DGETI - CBTIs 137: Lógica*. Obtenido de UNIDAD 4
EVALUAR ARGUMENTACIONES (EPR):
http://cbtis137logica.blogspot.com/2016_02_01_archive.html?view=classic

Zadeh, J. (1973). Outline of a new approach to the analysis of complex system. *IEEE Transaction on System Man and Cybernetics*, (págs. 1, 28-44).

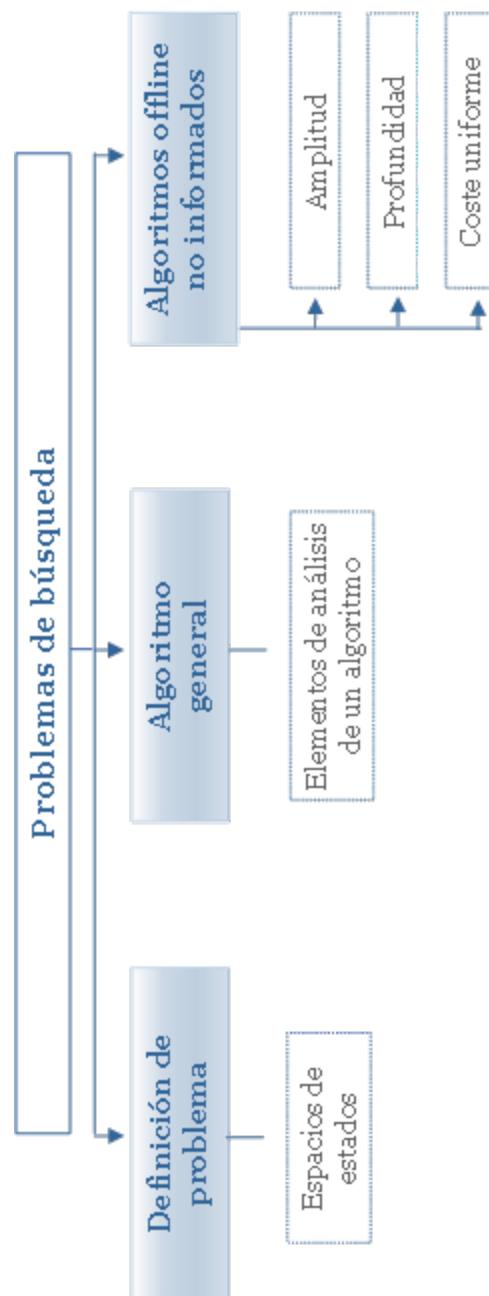


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Búsqueda no informada

Esquema. Tema 4



Ideas clave. Tema 4

4.1. ¿Cómo estudiar este tema?

En este tema presentaremos la descripción general de los problemas de búsqueda en un espacio de estados no estructurado. Este tipo de problemas representa gran parte de los mecanismos de búsqueda autónoma de los agentes inteligentes.

Los planificadores que veremos en los temas finales se apoyan en los conceptos de búsqueda que aparecen en los siguientes temas. Los agentes inteligentes deben enfrentarse muchas veces a problemas que les obligan explorar el entorno al que pueden acceder e intentar encontrar cuáles son las acciones que les permiten alcanzar sus objetivos y metas.

Principalmente este tema, así como alguno de los siguientes, se apoyan en el texto Inteligencia artificial: un enfoque moderno, de (Russell, 2004). Por lo tanto, se hace un requisito necesario durante el desarrollo de la asignatura leer este texto.

Puedes encontrar mucha información (en inglés) en la web de los autores:

<http://aima.cs.berkeley.edu/>



Accede a los ejercicios de autoevaluación a través del aula virtual

4.2. Descripción general de un problema de búsqueda



Accede al vídeo «Descripción general de un problema de búsqueda» a través del aula virtual

Los problemas de búsqueda se pueden catalogar de distintas maneras. En este tema nosotros vamos a explicar dos categorías o divisiones. La primera división toma en cuenta una particularidad del algoritmo de búsqueda, si utiliza una función heurística o no.

Función Heurística

En Inteligencia Artificial, una función heurística se define como una estimación de lo que falta para conseguir el objetivo.

En caso de no utilizar heurística, lo llamaremos una búsqueda no informada. Por el contrario, si el algoritmo utiliza una heurística, se considera una búsqueda informada.

Vamos a explicar un poco en más detalle estos dos términos:

- ▶ Búsqueda no informada: Decimos que una búsqueda es no informada cuando no emplea ningún tipo de heurística (el término heurística lo explicaremos en más adelante con más detalle). Es decir, no tienen ningún modo de poder guiar la búsqueda, siempre se evalúa el siguiente estado sin conocer a priori si este es mejor o peor que el anterior.
- ▶ Búsqueda informada: representa aquellos algoritmos que emplean una función heurística para guiar la búsqueda y de esta manera llegar a soluciones óptimas del problema.

La segunda división está más relacionada con el tipo de agente. Si es reactivo o deliberativo.

Si es deliberativo el agente realiza una búsqueda más offline, en la que puede emplear todo el tiempo necesario para encontrar un plan solución. Por el contrario, si es reactivo el agente realiza una búsqueda más online, en la que cuenta con un tiempo limitado para encontrar una solución parcial.

- ▶ Búsqueda *offline*: representa a aquellos agentes que realizan un proceso de búsqueda de la secuencia de acciones que deben desarrollar desde el principio (el estado actual en el que se encuentran) hasta el estado final (objetivo o meta que desean alcanzar) y, una vez realizado el proceso de búsqueda, empiezan a ejecutar el plan para conseguir el estado final del problema.
- ▶ Búsqueda *online*: engloba a aquellos agentes que realizan una búsqueda a «corto» plazo, que no llega necesariamente a encontrar la meta, pero nos pone en el camino para alcanzarla. Al contrario que la búsqueda offline, empieza a ejecutar acciones durante el proceso de búsqueda. Normalmente, se emplean algoritmos que realizan una búsqueda local.

Antes de empezar a discutir estos problemas y sus agentes asociados, debemos definir el problema en general al que se enfrentan los agentes basados en búsquedas.

¿Qué son los Agentes basados en búsquedas?

Son aquellos que:

- ▶ Mantienen un **modelo simbólico** del entorno. Modelo que representa solo aquella parte de la información del entorno que resulta relevante para el problema en cuestión, definiendo aquellos parámetros que permiten diferenciar un estado de otro del entorno.

- Desean **modificar el estado del entorno** de acuerdo con sus objetivos. Es decir, aplicar acciones que permitan modificar el entorno, de tal modo que se acabe alcanzando un estado meta de los que satisfacen los objetivos del agente.

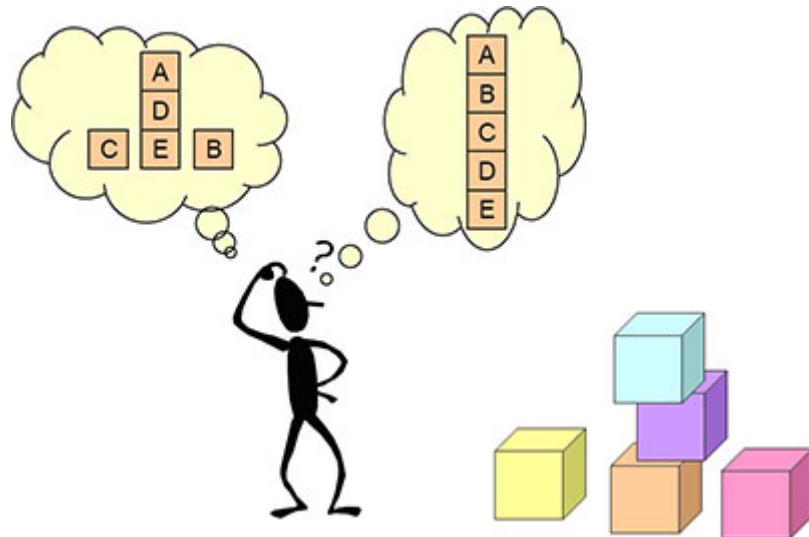


Figura 1. Dominio del mundo de bloques. El agente desea modificar el entorno para alcanzar un estado determinado (meta).

Por ejemplo, la figura 1, representa un agente que desea modificar un entorno del mundo de bloques. El entorno tiene 5 bloques, diferenciados por colores. El agente relaciona cada color con una letra, C=Amarillo, E=Naranja, B=rosa, D=Violeta, y A=Azul marino. Así, entonces en el entorno los bloques C, E y B se encuentran en la mesa. El bloque D se encuentra sobre el bloque E. Y el bloque A se encuentra sobre el bloque D. El objetivo del agente es colocar todos los bloques uno encima de otro. Más exactamente, el agente desea colocar el bloque A sobre el B, el B sobre el C, el C sobre el D, y el D sobre el E.

Para modificar el entorno de acuerdo a sus objetivos, los agentes de búsqueda **anticipan** los efectos que tendrían sus acciones sobre el mundo (por medio de su modelo del entorno), generando **planes de actuación** a través de una secuencia de acciones que los llevan desde su estado actual hasta el objetivo buscado, tal como se muestra en la figura 2. Esto lo realizan por medio de un proceso de búsqueda.

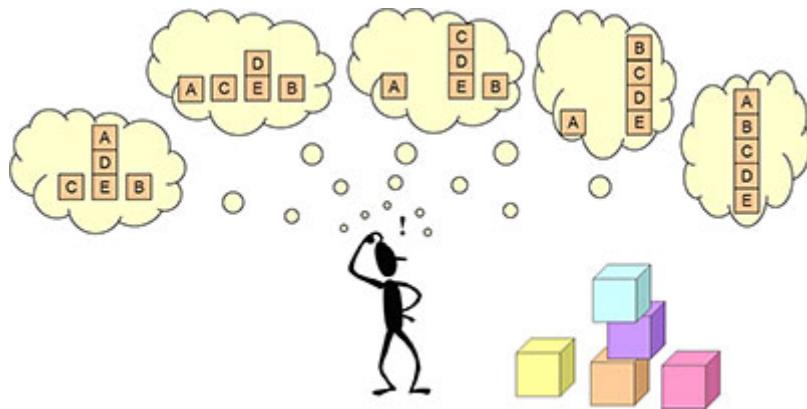


Figura 2. Dominio del mundo de bloques. Una búsqueda es la obtención de un plan de acciones para alcanzar la meta.

Si son agentes deliberativos, esta tarea la realizan antes de ejecutar las acciones del plan en el entorno real. Es decir, una vez encuentran el plan solución comienza la ejecución de este.

Si son agentes reactivos, esta tarea la van realizando a la vez que van ejecutando las acciones, corriendo el riesgo de equivocarse y tener que deshacer acciones que en muchos casos pueden resultar costosas.

Cualquiera de los dos tipos de agentes puede usar búsquedas no informadas e informadas.

Mecanismo para resolver estos problemas

Para resolver este tipo de problemas tenemos varios mecanismos que van desde aquellos que no permiten al agente ser autónomo de ninguna manera a aquellos que le permiten realizar procesos racionales de toma de decisiones.

Vamos a explicar algunos de estos mecanismos teniendo en cuenta el dominio de las torres de Hanoi mostrado en la figura 3.

Ejemplo: Torres de Hanoi

Objetivo:

- Trasladar los discos de la aguja A a B en el mismo orden

Restricción:

- un disco mayor nunca debe reposar sobre uno de menor tamaño

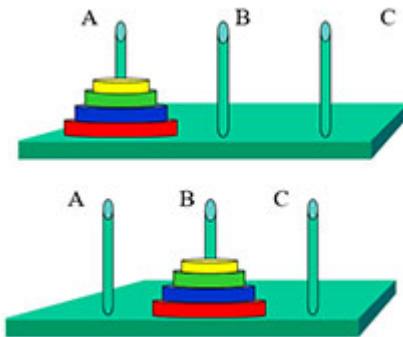


Figura 3. Domino de las Torres de Hanoi.

En este problema, se encuentran tres agujas A, B, y C. En la aguja A se encuentran 4 discos (1, 2, 3 y 4) de diferentes tamaños cada uno. Ubicados de mayor a menor, siendo 1 el de menor tamaño (el disco de color amarillo en la Figura 4). El objetivo es trasladar todos los discos a la aguja B. Los discos deben quedar en la aguja B en el mismo orden que estaban en la aguja A. Como restricción adicional se tiene que un disco mayor nunca puede estar sobre un disco de menor tamaño.

Tablas de actuación

En estos mecanismos, para cada situación hay una entrada en una tabla de actuación específica para el problema; dicha tabla presenta la secuencia de acciones completa para llegar desde el estado inicial a uno de los estados finales.

cuatro discos en A ⇒
1: (disco 1) A → C /
2: (disco 2) A → B /
3: (disco 1) C → B /
4: (disco 3) A → C /
n-1: ... /
n: (disco 1) C → B

Figura 4. Ejemplo de fragmento de tabla para el dominio de las torres de Hanoi con cuatro discos.

Figura 4, muestra un ejemplo de tablas de actuación para el dominio de las torres de Hanoi de la figura 3. Por ejemplo, la línea 1 indica que lo primero que debemos hacer es mover el disco 1 desde la aguja A a la aguja C.

Se puede mejorar la flexibilidad del agente por medio de técnicas que le permitan **aprender** nuevas entradas, pero tiene un grave problema de escalabilidad dado que rápidamente tiene problemas de memoria.

Algoritmos específicos del dominio

En esta técnica de resolución, el diseñador del agente conoce un método para resolver problemas de un dominio concreto y codifica este método en un algoritmo particular para el dominio. Generamos, en resumen, un código específico que resuelve cualquier problema concreto que le planteamos de un dominio específico.

Se puede intentar mejorar su flexibilidad por medio de crear un código que admita parámetros que configuren el problema y otro código que los resuelva de modo general para cualquier valor admitido como parámetro.

El principal problema es que el diseñador de la solución debe prever todos los escenarios posibles. En entornos reales, suele ser demasiado complejo anticipar todas las posibilidades. Además, de que solo funciona para un dominio en particular.

En la figura 5, se puede ver un algoritmo específico para solucionar cualquier problema de las torres de Hanoi.

```

PROCEDURE MoverDiscos(n:integer;
                      origen,destino,auxiliar:char);
{ Pre: n > 0
  Post: output = [movimientos para pasar n
                  discos de la aguja origen
                  a la aguja destino] }

BEGIN
  IF n = 0 THEN {Caso base}
    writeln
  ELSE BEGIN {Caso recurrente}
    MoverDiscos(n-1,origen,auxiliar,destino);
    write('Pasar disco',n,',de',origen,',a',destino);
    MoverDiscos(n-1,auxiliar,destino,origen)
  END; {fin ELSE}
END; {fin MoverDiscos}

```

Figura 5. Dominio de las Torres de Hanoi. Ejemplo de código que resuelve problemas de modo recursivo.

Como podemos ver, este algoritmo permite resolver problemas de las torres de Hanoi con N discos. Es específico del problema porque solo puede ser usado para este dominio. El mismo algoritmo no es válido para el dominio del mundo de bloques, por ejemplo.

Métodos independientes del dominio

Son aquellos métodos que emplean un modelo simbólico del dominio y problema. Por ejemplo, para el caso de las torres de Hanoi, definiríamos de modo general el problema tal como se muestra en la siguiente figura 6.

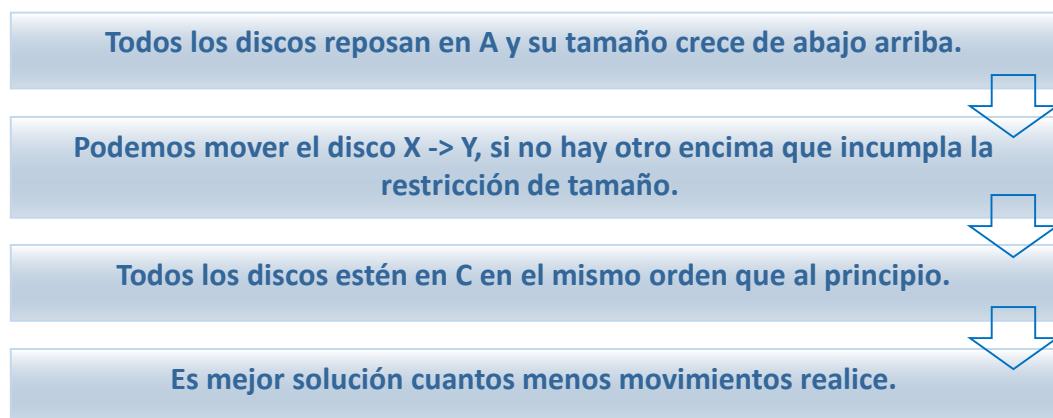


Figura 6. Dominio de las Torres de Hanoi. Definición general para un problema.

Para resolver los problemas, este método emplea un **algoritmo de búsqueda genérico**, representando el dominio y el problema mediante el modelo simbólico. Permite una mayor flexibilidad, ya que no necesitamos conocer la solución previamente y es fácil añadir nuevas características al problema.

Problemas de búsqueda en el espacio de estados

En estos problemas, nos encontramos con que el entorno se representa por medio de estados, que se deben diferenciar entre sí de modo unívoco, pero que no presentan características accesibles que los permitan diferenciar; es decir, que son distintos, pero para el agente solo son «etiquetas distintas» que representan estados distintos.

Para este tipo de problemas tenemos tres elementos que los definen:

- ▶ Espacio de estados: modelo del mundo representado por un grafo, en el cual tenemos un conjunto de elementos que representan componentes del mundo, que se traducen en elementos del modelo simbólico que simbolizaremos de una manera determinada en el grafo.



Figura 7. Espacio de estados.

- ▶ Problema de búsqueda: por medio de un mecanismo independiente del problema, exploramos el **espacio de estados** aplicando la **actitud del agente**, que representa el componente de racionalidad en el proceso de exploración.

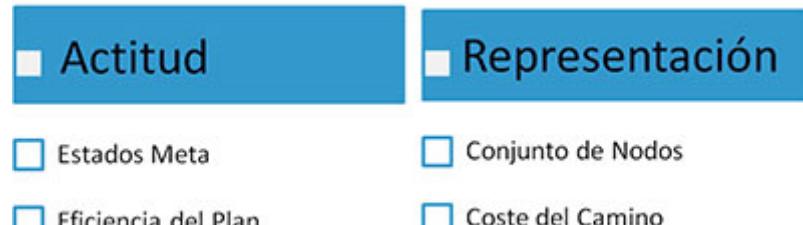


Figura 8. Problema de búsqueda.

- ▶ Objetivo: queriendo encontrar el plan más eficiente que lleve del estado inicial a un estado meta.

Por tanto, un problema de este tipo trata de encontrar el mejor camino dentro de un grafo dirigido como el presentado en la figura 9. Pero, por desgracia, **no conocemos el grafo**. Si lo conociésemos, realizaríamos una búsqueda de camino mínimo en grafos como, por ejemplo, Dijkstra (Rosettacode, 2020).

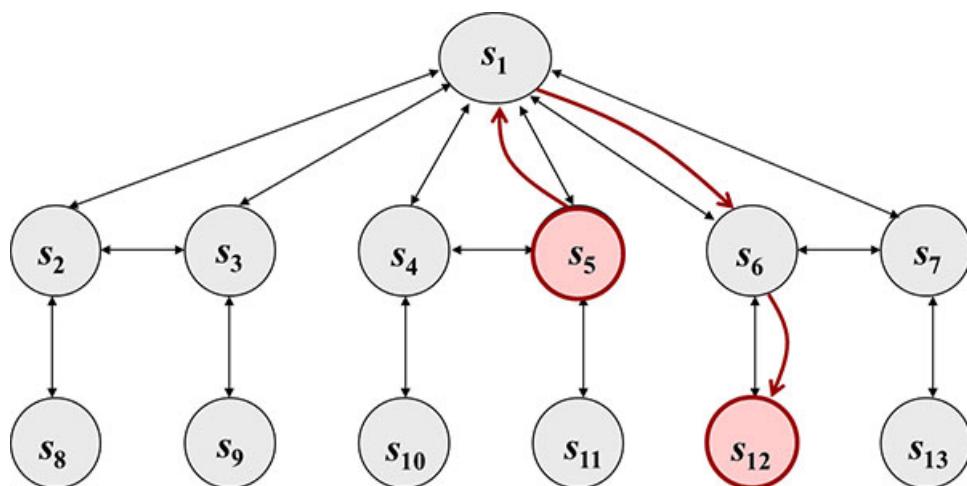


Figura 9. Grafo que representaría un espacio de estados de un problema.

Para este tipo de problemas disponemos de una serie de conocimientos *a priori* en el agente que le van a permitir realizar una estrategia de búsqueda sobre el problema,

aun desconociendo el modelo completo del mismo. Así, tendremos una representación del conocimiento del problema de **búsqueda implícita**.

Conocimiento a priori del agente	
s_0	Estado <i>inicial</i>
$\text{expandir}(s): \{s_1, \dots, s_n\}$	Conjunto de estados sucesores del estado s
$\text{meta}(s): \text{verdad} \mid \text{falso}$	Función de evaluación de s como estado meta
$c(s_i, s_j): v, v \in \mathbb{R}$	Coste de aplicar un operador que lleva de s_i a s_j
$c(s_1 s_2 \dots s_n)$ $= \sum_{k=1}^{n-1} c(s_k, s_{k+1})$	Coste del plan completo

Tabla 1. Conocimiento *a priori* del agente.

Con esta información *a priori*, emplearemos una **estrategia** basada en el **método de búsqueda**, ver figura 10, que irá explorando el espacio de estados, **expandiendo** a cada paso un estado y creando de modo progresivo un **árbol de búsqueda** tal como el mostrado en la figura 11.

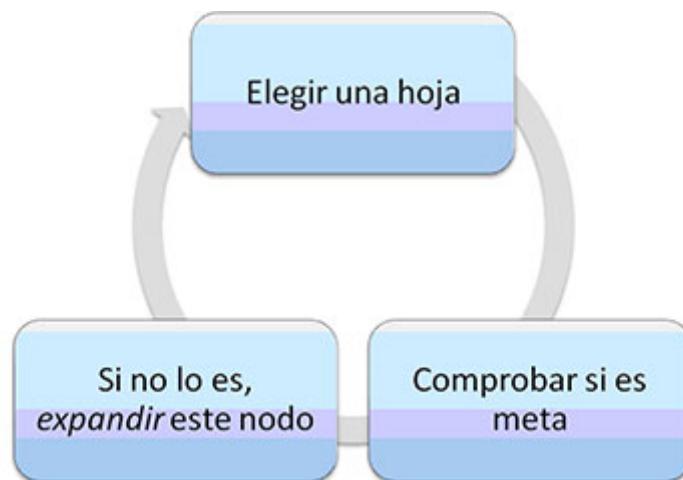


Figura 10. Método de búsqueda a partir de la elección de una hoja que representa un estado.

Árbol de búsqueda:

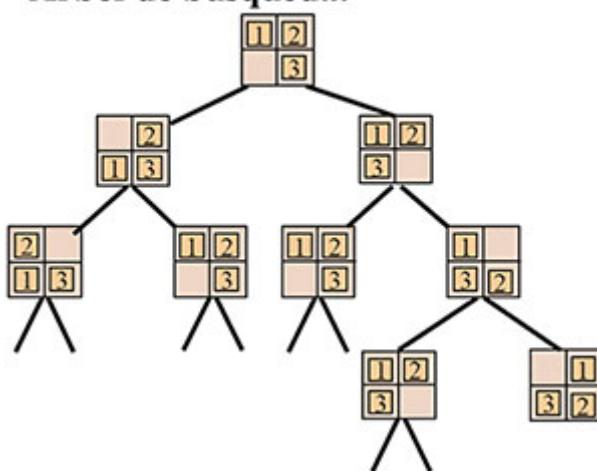


Figura 11. Árbol de búsqueda.

Algoritmo general de búsqueda

Podemos definir un algoritmo general de búsqueda tal como está en la figura 12:

Input: Estado inicial S_0 , Estado final G

```
1: colaAbierta  $\leftarrow \{S_0\}$ 
2: mientras colaAbierta  $\neq \emptyset$ 
3:   nodo  $\leftarrow$  extraer primero de colaAbierta
4:   si meta(nodo) entonces
5:     retornar camino a nodo
6:   fin si
7:   sucesores  $\leftarrow$  expandir(nodo)
8:   para cada sucesor  $\in$  sucesores hacer
9:     sucesor.padre  $\leftarrow$  nodo
10:    colaAbierta  $\leftarrow$  colaAbierta  $\cup$  sucesor
11:   fin para
12: fin mientras
13: retorna plan vacío o problema sin solución
```

Figura 12. Algoritmo general de búsqueda.

Donde el árbol se representa con base a un registro **del tipo nodo**, en su representación más simple de un nodo enlazado a su antecesor (padre) por medio de una referencia (línea 9). En este nodo almacenaremos el estado que se alcanza en este instante de la exploración.

Existe una lista de nodos «abierta» (línea 1) con las hojas actuales del árbol, es decir, aquellos estados y caminos que hemos expandido para explorar.

Si la lista está vacía, nos encontraremos con un problema sin solución. Si no está vacía extraeremos el primer elemento de la lista de hojas abiertas.

Si el nodo es meta, entonces retornamos el camino haciendo un *backtracking* sobre los nodos padres del nodo meta encontrado (línea 4).

Por último, añadiremos los estados nuevos obtenidos de la expansión en la lista de abierto (línea 10).

En las siguientes secciones mostraremos el funcionamiento de los distintos algoritmos de búsqueda no informados, que son empleados como base en muchos problemas de agentes inteligentes.

En todos los mecanismos de búsqueda tenemos presente un posible problema que puede aparecer, que son los **estados repetidos**. Para resolver este problema, que puede causar en algunos casos errores graves como entrar en bucles infinitos, tenemos distintas estrategias:

- ▶ Ignorarlo: por extraño que parezca, algunos algoritmos no tienen problemas con esta solución debido a su propio orden de exploración.
- ▶ Evitar ciclos simples: evitando añadir el padre de un nodo al conjunto de sucesores.
- ▶ Evitar ciclos generales: de tal modo que ningún antecesor de un nodo se añada al conjunto de sucesores.

- Evitar todos los estados repetidos: no permitiendo añadir ningún nodo existente en el árbol al conjunto de sucesores.

Estas estrategias deben tener en cuenta el coste que conlleva tanto explorar de más como buscar elementos repetidos para explorar menos.

En los siguientes temas y en las siguientes secciones presentaremos distintos algoritmos. Para todos ellos emplearemos un mecanismo de clasificación basado en los siguientes conceptos y características:

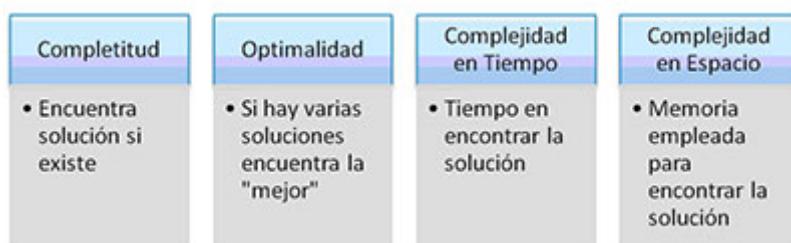


Figura 13. Características de un algoritmo.



Accede a los ejercicios de autoevaluación a través del aula virtual

4.3. Búsqueda en amplitud



Accede al vídeo «Búsqueda en amplitud» a través del aula virtual

La búsqueda en amplitud (BFS, por sus siglas en inglés) es una estrategia que genera el árbol de búsqueda por niveles de profundidad, expandiendo todos los nodos de nivel i antes de expandir los nodos de nivel $i+1$.

Considera, en primer lugar, todos aquellos estados que se encuentran en caminos de longitud 1 (es decir, aquellos caminos que solo requieren una acción), luego los de

longitud 2 (caminos que solo requieren dos acciones), etc. De este modo, se encuentra aquel estado meta que esté a menor profundidad.

La figura 14 muestra un ejemplo de cómo se va generando el árbol con este algoritmo.

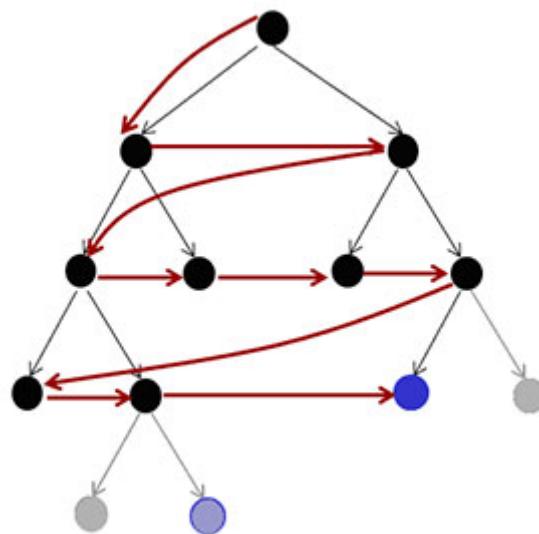


Figura 14. Esquema de búsqueda en amplitud.

El algoritmo desarrollaría un mecanismo de búsqueda que, por ejemplo, en el problema del puzzle-8, derivaría en el árbol de búsqueda mostrado en la figura 15.

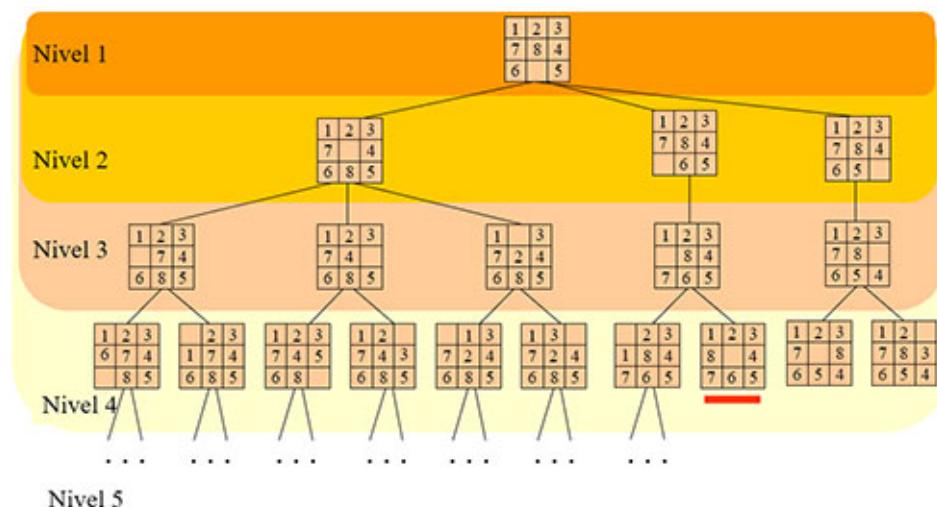


Figura 15. Árbol de búsqueda en amplitud para el puzzle-8.

Puzzle-8: es un popular juego que consiste en una matriz de 3x3, en la que se encuentran unas fichas enumeradas del 1 al 8. Adicionalmente, existe una pieza vacía o en blanco. El juego consiste en dado una configuración inicial llevar las piezas a una configuración final. En cada turno las únicas piezas que se pueden mover son las piezas adyacentes a la pieza vacía.

El algoritmo de búsqueda en amplitud lo podemos extraer del algoritmo de búsqueda general anteriormente expuesto en la figura 15. Matizando las siguientes cuestiones:

- ▶ Para añadir nuevos sucesores, lo haremos al final de la lista abierta.
- ▶ Por su parte, la lista abierta funciona como cola (insertando al final y recuperando al inicio), lo que conlleva que siempre se expandan primero aquellos nodos más antiguos (es decir, los menos profundos).
- ▶ Adicionalmente, controlaremos los nodos que se han visitado previamente.

Este algoritmo es, por tanto, **completo y óptimo**, pero presenta una complejidad en tiempo y espacio muy deficiente, ya que depende de modo proporcional al nivel de profundidad d de la solución y, por tanto, al número de nodos expandidos o factor de ramificación b .

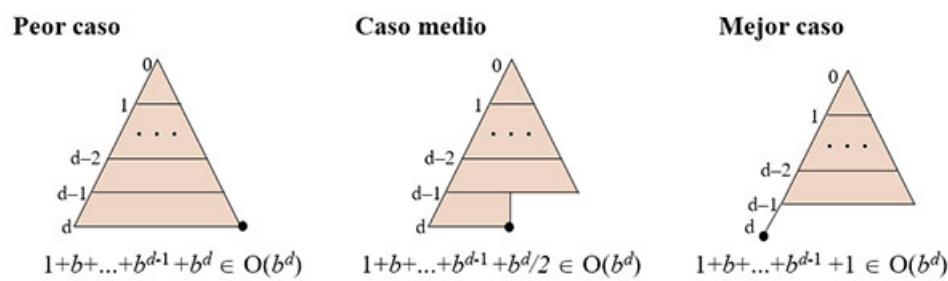


Figura 16. Complejidad espacial y temporal de un grafo sin eliminación de nodos duplicados. El factor de ramificación está en b y la solución está en una profundidad d .

En la figura 16 se puede ver un análisis más exhaustivo de la complejidad aproximada de este algoritmo para un problema general.

Ejemplo: recursos requeridos por la búsqueda en amplitud en el *peor caso*

- factor de ramificación efectivo: 10
- tiempo: 1.000.000 nodos/segundo
- memoria: 1.000 bytes/nodo

d	nodos	tiempo	memoria
2	110	0,11 ms	107 KB
4	11.110	11 ms	10,6 MB
6	10^6	1,1 s	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 horas	10 TB
12	10^{12}	13 días	1.000 TB
14	10^{14}	3,5 años	99 PB
16	10^{16}	350 años	10.000 PB

Figura 17. Análisis de la complejidad de un algoritmo de búsqueda en amplitud.

Fuente: (Russell, 2004).



Accede a los ejercicios de autoevaluación a través del aula virtual

4.4. Búsqueda en profundidad



Accede al vídeo «Búsqueda en profundidad» a través del aula virtual

La búsqueda en profundidad (DFS, por sus siglas en inglés) es otra estrategia de búsqueda no informada (sin información adicional). En ella, al contrario de la búsqueda en amplitud, se intenta desarrollar un camino de longitud indeterminada, en el cual intentamos alcanzar metas profundas (aquellas que tienen un camino largo para alcanzarlas) desarrollando las menores ramificaciones posibles.

En general, es un algoritmo que funcionará bien mezclado con otras informaciones adicionales, pero que puede resolver problemas de la misma manera que un algoritmo en amplitud.

La figura 18 muestra un ejemplo de cómo se va generando el árbol con este algoritmo.

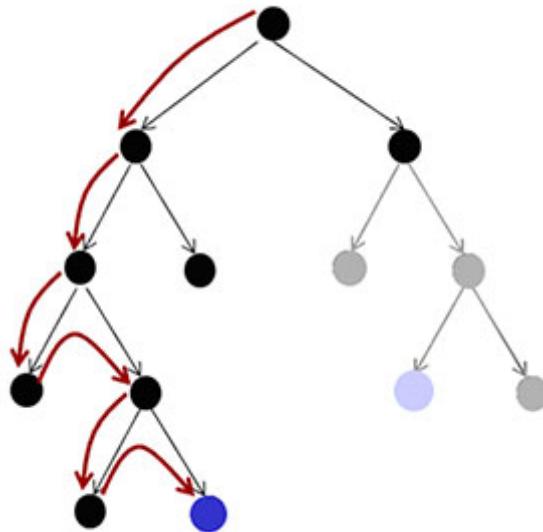


Figura 18. Esquema de búsqueda en profundidad.

En esta estrategia, se expande el árbol de «izquierda a derecha», por lo tanto, aquellos nodos más profundos se expanden primero. Si se llega a un nodo sin sucesores, se retrocede y se expande el siguiente nodo más profundo.

Como resultado, el método va explorando un «camino actual» y no siempre se encuentra el nodo de profundidad mínima.

En este caso, el algoritmo general se ve adaptado teniendo en cuenta las siguientes consideraciones:

- ▶ Los nuevos sucesores se añaden al inicio de la lista abierta,
- ▶ La lista abierta funcionará como una pila (insertando al principio y extrayendo también del principio) y siempre extraeremos el nodo más profundo. Al guardar todos los sucesores de un nodo expandido en abierta, se permite la «vuelta atrás» o *backtracking*.

- Solo procesaremos un nodo de la pila si este no ha sido visitado aún.

El análisis de este algoritmo nos muestra que es **completo** (si y solo si se garantiza la eliminación de los estados repetidos dentro de una misma rama), pero **no es óptimo** (para operadores de coste uno), dado que no garantiza que siempre se encuentre aquella solución que está a la menor profundidad.



Accede a los ejercicios de autoevaluación a través del aula virtual

4.5. Búsqueda de coste uniforme



Accede al vídeo «Búsqueda de coste uniforme» a través del aula virtual

En los casos anteriores de las búsquedas en amplitud y profundidad se empleaba una asunción de que el coste de aplicación de cualquier acción (por tanto, el coste de elegir una rama) era siempre igual a 1, por lo que el coste total del camino era el número de niveles en el que se encontraba un determinado nodo. Pero ¿qué sucede cuando el coste de transitar de un nodo a otro no es igual para todas las acciones dentro del entorno?

Por ejemplo, en el caso de la figura 19, que representa el problema de encontrar la ruta más corta en un grafo donde cada nodo es una ciudad y las aristas entre las ciudades contienen un número que representa el coste del operador, distancia por carretera de una ciudad a otra.

Estado: estancia en una ciudad
Coste de un operador: distancia por carretera a la ciudad vecina

Operadores: ir a una ciudad vecina
Coste de un plan: suma de distancias entre las ciudades visitadas

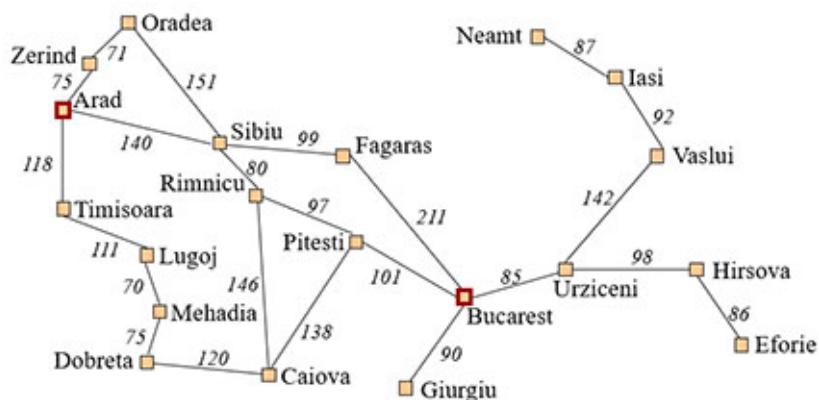
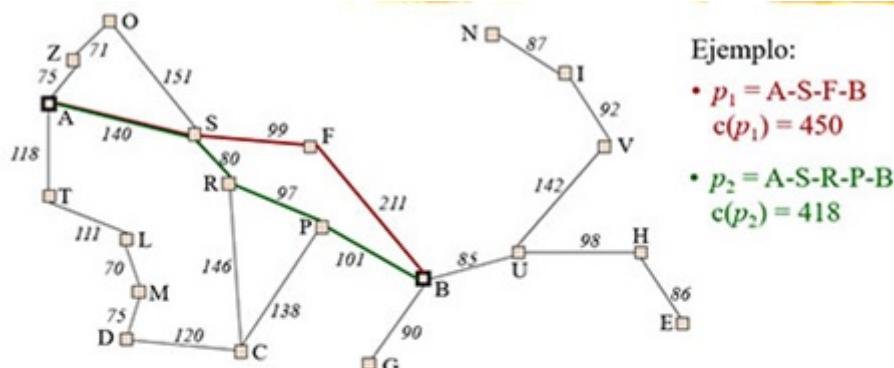


Figura 19. Grafo esquemático de carreteras.

En este caso, como se puede observar en la figura 20, la exploración de caminos empleando el algoritmo de búsqueda en amplitud que garantizaba la solución óptima falla. El algoritmo en amplitud ofrece la solución de color rojo, A - S - F - B. La solución óptima en este caso, es la que se encuentra en color verde, A - S - R - P - B.



Problema:

- La búsqueda en amplitud encuentra el nodo meta de menor profundidad; éste puede no ser el nodo meta de coste mínimo
- $\text{prof.}(B_{p1}) = 3 < 4 = \text{prof.}(B_{p2})$ / $c(p1) = 450 > 418 = c(p2)$

Figura 20. La búsqueda en amplitud, que asume coste 1, falla para encontrar el camino óptimo.

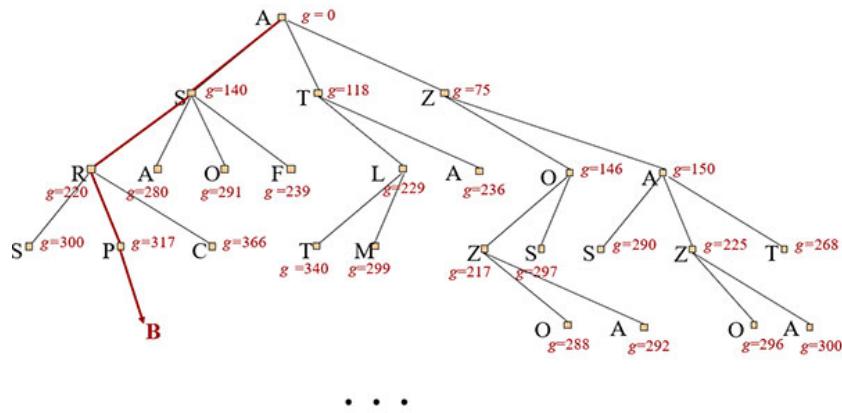
Para resolver este tipo de escenario donde el coste no es igual para todas las acciones (pero sí positivo en todos los casos), tenemos el algoritmo de búsqueda de coste uniforme (UCS por sus siglas en inglés).

Empleando el mismo algoritmo general de búsqueda, aplicamos la idea de dirigir la búsqueda por el coste de los operadores. Supondremos que existe una función de utilidad $f(n) = g(n)$ que permite calcular el coste real para llegar del nodo inicial al nodo n . En cada iteración del algoritmo expandiremos primero el nodo de menor coste f . Por simplicidad, en el resto de la explicación de este algoritmo, nos referiremos a la función de utilidad indistintamente como f o g .

Las modificaciones que se realizan del algoritmo general de búsqueda son:

1. Almacenaremos cada nodo por prioridad con base a su valor de g , por lo tanto, la inserción de nuevos nodos en la lista abierta se ordenará de modo ascendente según su valor g .
2. Lo anterior hace que la lista abierta este definida como una cola de prioridad ordenada por el valor de g .
3. Solo agregamos un nodo a la lista abierta si este no se encuentra en la misma.
4. En caso de encontrarse, reemplazamos el nodo si y solo si el coste f es menor.

La figura 21 muestra el resultado de aplicar el algoritmo UCS al problema planteado en la figura 19. En cada nodo se observa el valor real de g .





Accede a los ejercicios de autoevaluación a través del aula virtual

4.6. Referencias bibliográficas

Rosettacode. (14 de 03 de 2020). Obtenido de Dijkstra's algorithm:
http://rosettacode.org/wiki/Dijkstra%27s_algorithm

Russell, S. y. (2004). *Inteligencia Artificial: Un Enfoque Moderno*. Madrid: Pearson Educación.

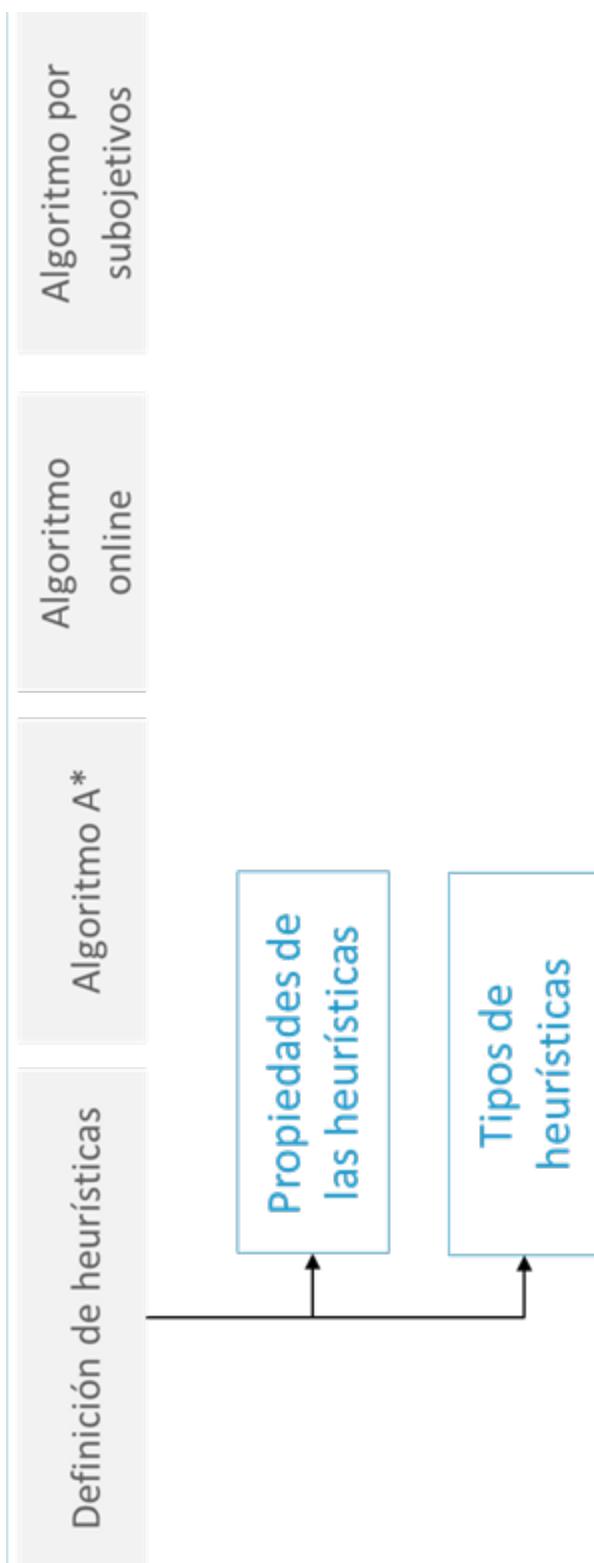


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Búsqueda informada

Esquema. Tema 5



Ideas clave. Tema 5

5.1. ¿Cómo estudiar este tema?

En este tema veremos cómo mejorar las búsquedas en entornos más complejos, donde la expansión de nodos, debido a la complejidad del entorno, puede suponer problemas para realizar la búsqueda.

Exploraremos inicialmente el concepto de **heurística**, como una formalización de la experiencia que se puede proporcionar a un agente que debe resolver un problema y que le permite guiarse a través del espacio de estados.

Con los conceptos de heurística definiremos tres nuevos modelos de búsqueda en espacios no estructurados, que permitirán acelerar el proceso de exploración incluso en entornos reactivos, donde la restricción temporal puede suponer un problema muy restrictivo.

En este tema, nos volveremos a apoyar en el libro de Russell y Norvig: *Inteligencia artificial: un enfoque moderno* (Russell, 2004).

Así, veremos los algoritmos de:

- ▶ Búsqueda A*.
- ▶ Búsqueda con horizonte limitado.
- ▶ Búsqueda *online*.
- ▶ Búsqueda por subobjetivos.



Accede a los ejercicios de autoevaluación a través del aula virtual

5.2. Tipos de heurísticas



Accede al vídeo «Heurísticas: admisibilidad y monotonía» a través del aula virtual

El término «heurística» (del griego: *heuriskein*) significa «encontrar», «descubrir». La heurística surge resolviendo problemas y observando cómo se han resuelto otros.

Podemos definir la heurística como la manera de alcanzar la solución de problemas a través de la evaluación de los progresos alcanzados durante la búsqueda del resultado definitivo.

La heurística es una capacidad típica del ser humano, que le lleva a innovar para alcanzar unos objetivos. En inteligencia artificial, muchos de los algoritmos que se emplean son heurísticos o utilizan reglas heurísticas.

El método heurístico, ya que, como comentamos, se basa en resultados anteriores para lograr nuevos objetivos, puede retornar soluciones verdaderas o falsas, aunque esté correctamente aplicado. En inteligencia artificial, y en las ciencias de la computación en general, el método heurístico se usa en determinadas circunstancias, únicamente cuando no hay una solución óptima con una serie de restricciones de partida.

Los programas heurísticos actúan encontrando algoritmos que proporcionen tiempo de ejecución y soluciones adecuadas. Por ejemplo, en juegos que intentan predecir lo que va a hacer el usuario. Para ello, se basan en la experiencia y en lo que ha hecho el usuario en otras ocasiones.

Un ejemplo de algoritmo heurístico empleado en inteligencia artificial es el encargado de determinar si un email debe ser catalogado como *spam* o como no *spam*. Y es que las reglas, si se emplean de manera independiente, pueden provocar

errores de clasificación, pero al ser utilizadas en conjunto, cuando se utilizan muchas reglas heurísticas a la vez, la solución mejora, siendo más robusta y aceptable.

En definitiva, la **heurística en inteligencia artificial representa el conocimiento extraído de la experiencia dentro del dominio del problema**.

Diferenciamos dos tipos de heurística:

- ▶ «Fuerte»: pensada para facilitar la resolución del problema, pero no nos garantiza la resolución de este, ni su completitud ni su optimalidad. Es como una «regla de tres» para solucionar un problema.
- ▶ «Débil»: buscamos aplicar un método riguroso junto a la información heurística para guiar el proceso de búsqueda. Queremos mejorar el **rendimiento medio** de un método de resolución de problemas, pero no garantiza una mejora en el peor caso.

El concepto de información heurística que aplicaremos a un método de exploración se basa en las **funciones heurísticas**.

La función heurística es dependiente del estado y se usa para evaluar cómo de prometedor es un nodo.

Estas funciones se basan en «el mejor primero», eligiendo el nodo más prometedor para expandir. Una definición simplista es interpretar una heurística como la «distancia» que separa a un nodo de la meta de modo aproximado.

$h(n): n \rightarrow G$	Coste <i>real</i> desde el nodo n hasta la meta más cercana.
$h(n)^*: n \rightarrow G$	Función heurística que estima el valor de <i>ir desde el nodo n hasta la meta</i> .

Tabla 1. Funciones de coste real y de coste estimado (heurística)

Por ejemplo, los humanos saben que para ir de Madrid a Barcelona es mejor pasar por Guadalajara que por Toledo; esto es así porque si trazamos una línea recta en el mapa (la distancia euclídea) como estimación de la distancia a Barcelona, veremos que la distancia estimada es mucho menor para Guadalajara que para Toledo.

Función heurística admisible u optimista

Definiremos una función heurística h^* como admisible si $h^*(n) \leq h(n)$ para todo n . Por ejemplo, en el dominio del mundo de bloques, una heurística admisible es considerar el número de bloques **descolocados**. O, en el caso de enfrentarnos a un problema de encontrar rutas en una red de carreteras, por ejemplo, estimaríamos la distancia en **línea recta** hasta un nodo meta. Un ejemplo se puede ver en la Figura 1, donde se muestra un grafo que representa una red de carreteras. Las aristas tienen asociado el coste real de ir desde un nodo a otro, por ejemplo, Ir de A a S tiene un costo de 140. La tabla de la derecha presenta el costo estimado para ir desde cualquier nodo al nodo B, que es el nodo objetivo. De este modo, si nos encontramos en la ciudad O y queremos ir a la ciudad B, nuestra heurística estimaría un costo de 380.

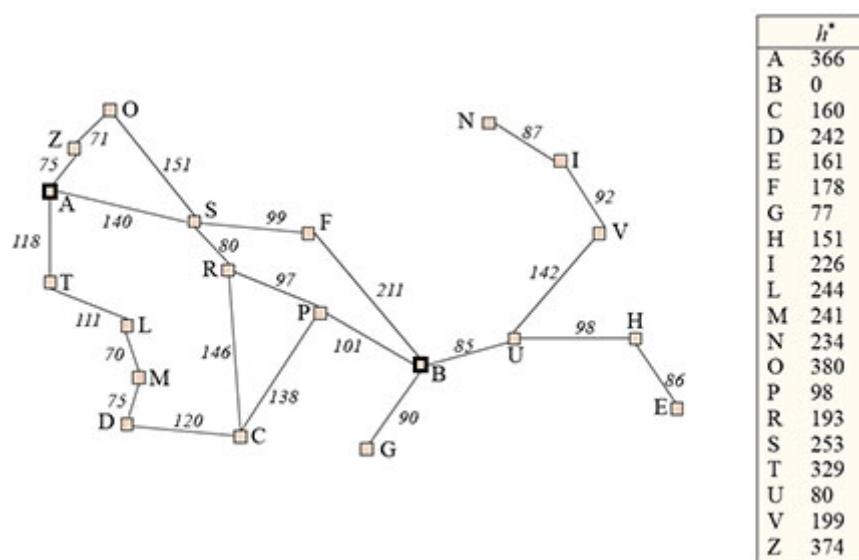


Figura 1. Ejemplo esquemático de un grafo que representaría una red de carreteras y su coste estimado para ir hasta el nodo B.

Función heurística consistente

Una heurística consistente es aquella que cumple la propiedad de que el valor de la función de evaluación $f^*(s) = g(s) + h^*(s)$ de un hijo nunca es menor que el del padre.

Por ejemplo, si el nodo s' es un sucesor de s y $f^*(s') < f^*(s)$, entonces la heurística no será consistente.

Esto implica que con una heurística consistente $f^*(s)$ es monotónicamente creciente.

Por ejemplo, supongamos una solución de coste 10 y un estado inicial a partir del cual una heurística consistente genera un valor de 6. Sus nodos sucesores podrían tener la siguiente secuencia de valores en la función de evaluación: 6, 6, 6, 7, 7, 8, 8, 8, 9, 10, 10; es decir, el valor de la función de evaluación del hijo es, al menos, tan grande como la del padre.

Dicho de otra forma, el valor heurístico del padre es menor o igual que el valor heurístico del hijo más la distancia del padre al hijo, es decir, $h^*(s) \leq c(s, s') + h^*(s')$.

Una heurística consistente siempre es admisible, pero no necesariamente ocurre lo contrario.

Cuestiones

- ▶ ¿Cómo se pueden encontrar funciones heurísticas **admisibles y/o consistentes**?
- ▶ ¿Cómo se puede distinguir entre «buenas» y «malas» funciones heurísticas?

Diseño de funciones heurísticas

Para el diseño de estas funciones, nos plantearemos el concepto de *planning de problemas relajados*. Es decir, aquellos en los que dado un problema real donde

tenemos un estado inicial, estado meta, y operadores o acciones, cada una con sus precondiciones (elementos que se deben cumplir en un estado o nodo n , para que la acción se pueda ejecutar sobre dicho nodo n generando un nodo n') y efectos (conjunto de elementos que se deben modificar en el nodo n como resultado de la ejecución del operador o acción)., relajamos los efectos de cada operador quitando cualquier efecto que elimine un elemento del nodo n donde se ejecuta la acción. . En ellos, por tanto, todas las soluciones al problema original también lo son al problema relajado y posiblemente a algunos más.

Partiremos de la idea de usar el coste exacto, $h(n)$, de llegar desde el estado n a un nodo meta para generar en el problema relajado una función heurística $h^*(n)$ que nos permita resolver el problema original.

El problema lo consideramos relajado porque utilizamos una estimación de $h(n)$.

Por construcción, una función heurística h^* implementada de esta manera se considera admisible.

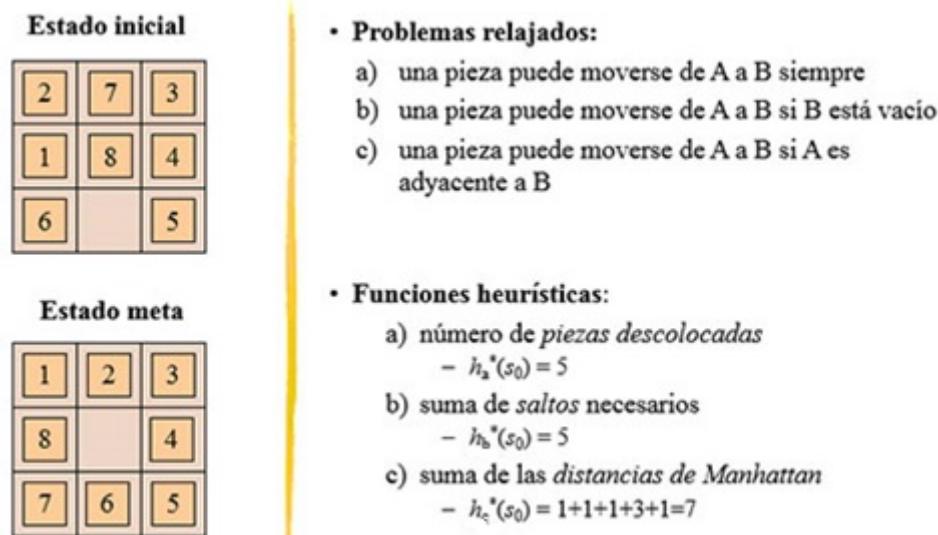


Figura 2. Ejemplo de problemas relajados para el puzzle-8.

En la figura 2, podemos ver un ejemplo de problemas relajados para el dominio del puzzle-8.

Puzzle-8: es un popular juego que consiste en una matriz de 3x3, en la que se encuentran unas fichas enumeradas del 1 al 8. Adicionalmente, existe una pieza vacía o en blanco. El juego consiste en dado una configuración inicial (Estado inicial) llevar las piezas a una configuración final (Estado meta). En cada turno las únicas piezas que se pueden mover son las piezas adyacentes a la pieza vacía.

En el ejemplo, de la figura 2, se presenta el problema relajado de tres maneras. Por ejemplo, podemos relajar el problema diciendo que a) una pieza puede moverse de A a B directamente. En este supuesto, es como si estuviéramos desarrollando una función heurística basada en a) el número de piezas descolocadas. Otra posible relajación del problema es c) que una pieza se puede mover de A a B si A es adyacente a B. En este caso, es como si desarrolláramos una función heurística basada en c) la distancia en Manhattan, donde realizamos la suma de las distancias de las fichas mal colocadas. La distancia se mide como el número de movimientos necesarios para situar la ficha en la posición final deseada.

Funciones heurísticas y su calidad

Si tenemos dos funciones heurísticas admisibles, $h1^*$ y $h2^*$, diremos que $h1^*$ es más informada que $h2^*$ si para todo nodo n se cumple que $h1^*(n) \geq h2^*(n)$.

De este modo, si $h1^*$ es más informada que $h2^*$, entonces el algoritmo que emplee $h2^*$ expande al menos tantos nodos como aquel que emplee $h1^*$.

Por tanto, será preferible elegir aquellas funciones que presenten valores de h^* grandes, siempre que se mantengan admisibles. Si hay varias funciones heurísticas admisibles, elegiríamos aquellas en cada caso que tengan mayor valor en cada momento.

$$h^*(n) = \max(h_1^*(n), h_2^*(n), \dots, h_m^*(n))$$



Accede a los ejercicios de autoevaluación a través del aula virtual

5.3. Búsqueda A*



Accede al vídeo «De búsqueda no informada a A*» a través del aula virtual

La idea es orientar la búsqueda de una solución en la que las acciones no tienen el mismo coste y desconocemos, *a priori*, el coste de llegar hasta un estado meta, pero podemos estimar, por medio de una función heurística, el coste restante que nos queda desde el nodo actual para alcanzar el nodo meta. Por tanto, nuestro objetivo es minimizar el coste estimado de un camino en el árbol de búsqueda, combinando el coste de llegar al nodo n (conocido exactamente por $g(n)$) y el coste aproximado de llegar de este nodo n hasta la meta por medio de la función heurística $h^*(n)$.

Para nuestro caso, siempre se debe establecer una función heurística $h^*(n)$ que subestime el coste real $h(n)$ que nos resta para llegar a la meta desde un nodo cualquiera n .

Por lo tanto, contamos con una función de evaluación real que se define como:

$$f(n) = g(n) + h(n):$$

coste mínimo que pasa por n , es decir, el **coste real** del plan.

Para ello, emplearemos la función de evaluación $f^*(n)$, que estimará una aproximación de $f(n)$:

$$f^*(n) = g(n) + h^*(n)$$

siendo:

- ▶ $h^*(n)$: coste estimado del nodo actual al nodo meta ($h(n)=0$ si n es un nodo meta).
- ▶ $g(n)$: coste real de llegar a n desde el estado inicial.

Por lo tanto, la función de evaluación $f^*(n)$ es el coste real de llegar al nodo n más el coste estimado de llegar al nodo meta.

Durante el proceso de búsqueda, emplearemos la estrategia de elegir, entre las hojas del árbol de búsqueda, aquella con un valor f^* mínimo.

El **algoritmo A***, se puede explicar brevemente, basándonos en la búsqueda general. Donde deberemos añadir el valor g y f^* como atributos a cada nodo expandido y ordenar la lista abierta con base a los valores crecientes de f^* . Por tanto, añadiremos los nuevos nodos en la lista abierta según sus valores crecientes de f^* .

```
// inicialmente todos los valores de g y f en los nodos son infinitos
 Grafo, Estado inicial S0, Estado final G

1: Definir colaAbierta as colaPrioridad
2:
3: S0.g ← 0
4: S0.f ← h(Grafo, S0, G)
5: colaAbierta ← {S0}
6:
7: mientras colaAbierta ≠ ∅
8:   nodo ← extrae el de menor f de colaAbierta
9:   si G ⊑ nodo
10:    retornar camino a nodo
11:   fin si
12:   sucesores ← expandir(nodo)
13:   para cada sucesor ∈ sucesores hacer
14:     tentative.e ← nodo.e + c(nodo, sucesor)
15:     si tentative.e < sucesor.e
16:       sucesor.padre ← nodo
17:       sucesor.g ← tentative.e
18:       sucesor.f ← sucesor.g + h(Grafo, sucesor, G)
19:       si sucesor no en colaAbierta
20:         colaAbierta ← colaAbierta U sucesor
21:
22: retorna plan vacío o problema sin solución
```

Figura 3. Algoritmo de búsqueda A*.

La figura 3 muestra el algoritmo de búsqueda A*. Hay que tener en cuenta que esta implementación del algoritmo no lleva un control de los nodos visitados (lista de nodos cerrados). Aspecto que es bastante importante en una implementación óptima

del algoritmo A*. El estudiante debe estar en la capacidad de buscar y agregar este añadido al algoritmo A* ofrecido en la figura 3.

La lista de nodos cerrados se utiliza para controlar los elementos que ya se han visitado. Cuando se evalúa un nodo sucesor (línea 13), siempre se verifica que en el nodo no exista en la lista de nodos cerrados con un menor valor de f^* , en cuyo caso no lo agregamos en la lista de nodos abiertos (colaAbierta). En caso contrario lo agregaríamos.

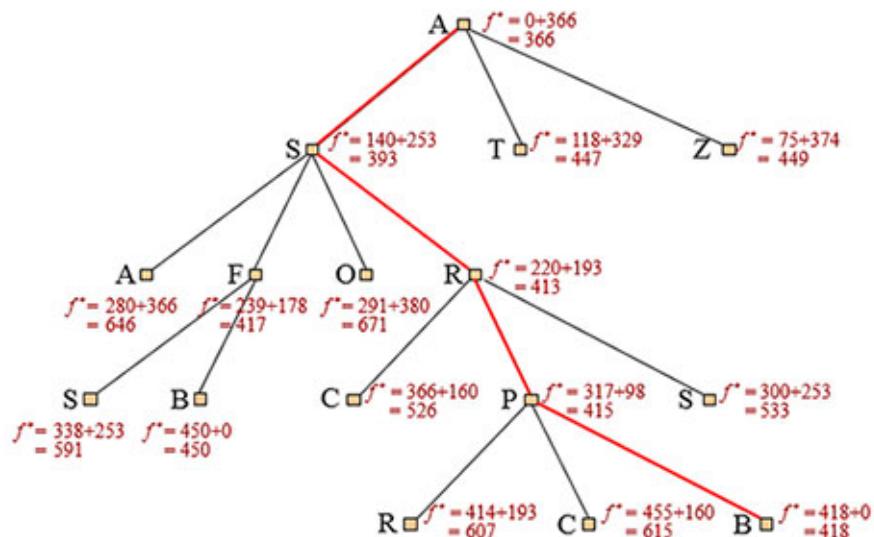


Figura 4. Ejemplo de búsqueda A*.

La **estrategia A*** continúa mientras se produce un crecimiento en el valor de f^* a lo largo de los caminos del árbol de búsqueda, tal como se muestra en la Figura 4.

Por tanto, si nos alejamos de la meta, g y h , que representan el coste real de ir avanzando hacia la meta y el coste estimado de alcanzar la meta desde la posición actual, crecen, por lo que $f^* = g + h^*$, crece en gran medida. En cambio, si nos acercamos a la meta, el valor de f^* crece poco porque g crece mientras que h^* disminuye.

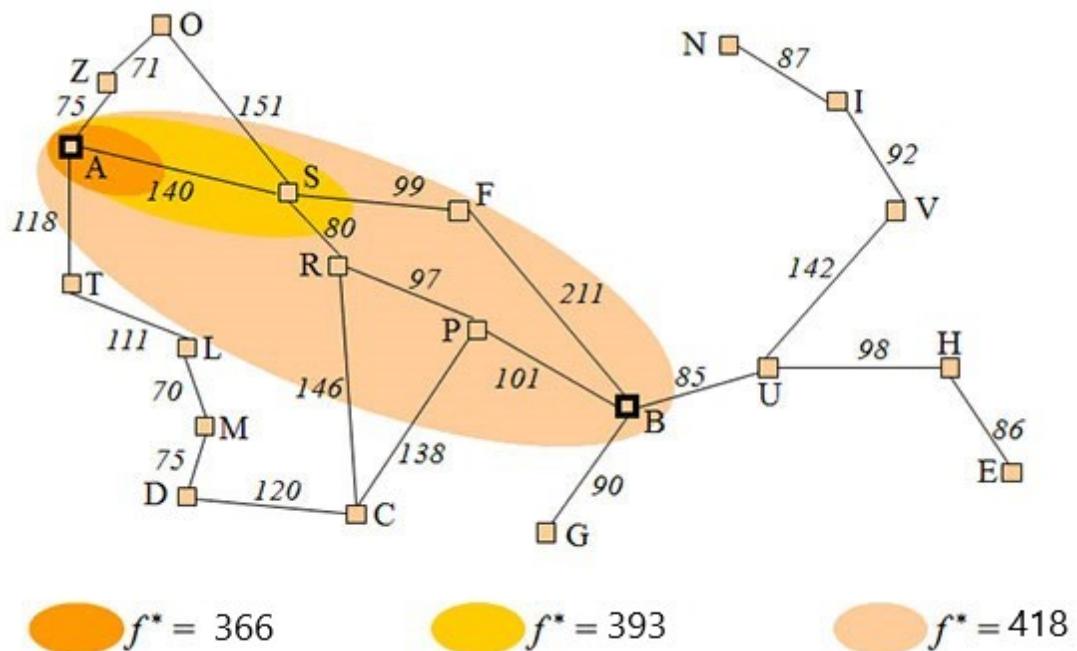


Figura 5. Ejemplo de evolución de los valores de f^* a lo largo de una búsqueda.

Si lo comparamos con la búsqueda de coste uniforme, los valores de g crecen a lo largo de todos los caminos del árbol de búsqueda porque en cada paso se suma el coste de un operador (que es un número natural positivo).

En el ejemplo anterior, figuras 4 y 5, los valores de f^* también crecen a lo largo de todos los caminos del árbol de búsqueda. Pero esto no siempre ha de ser así, tal como se muestra en la siguiente figura 6, donde se ve la diferencia entre el comportamiento esperado del aumento del coste real respecto al coste estimado.

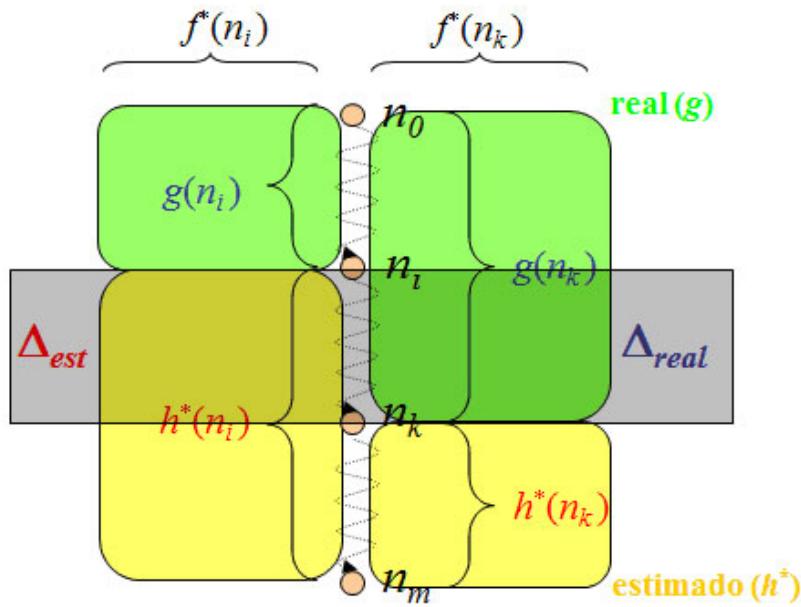


Figura 6. Comportamiento esperado del aumento del coste real respecto al coste estimado.

En un camino del árbol de búsqueda, si un nodo n_k es sucesor (no necesariamente directo) de n_i , entonces el valor de $f^*(n_i)$ contiene una **estimación** del coste de llegar de n_i a n_k (D_{est}) y $f^*(n_k)$ el coste real (D_{real}). Es decir, se puede obtener el valor de $f^*(n_k)$ a partir de $f^*(n_i)$, sumando D_{real} y restando D_{est} . Es posible que $f^*(n_k) < f^*(n_i)$, si $D_{est} > D_{real}$.

La función heurística en la que nos apoyemos debe ser admisible.

Análisis del Algoritmo A*

El algoritmo A* es:

- ▶ Óptimo: si h^* es admisible.
- ▶ Y completo, en caso de existir una solución, siempre dará con ella.

Presenta una complejidad dependiente del número de nodos expandidos, que depende, a su vez, de la precisión de h^* , de tal modo que si tuviésemos que la función heurística fuese de perfecta información ($h^*(n) = h(n)$ para todos los nodos n), la

complejidad sería lineal (sin contar la complejidad de computar h^*). Mientras que, si carecemos de información, ($h^*(n) = 0$ para todos los nodos n), la búsqueda A* degenera en la búsqueda de coste uniforme.

Aun así, suele haber una mejora notable en comparación con métodos no informados.

Por ejemplo, en la figura 7, se muestra una comparativa del número de nodos expandidos en el problema del 8-puzzle de un método no informado con el algoritmo A* usando diferentes heurísticas., Suponemos que la profundidad de la solución se encontrase tras d niveles.

d	B. no informada (prof. iterativa)	$A^*(h_a)$	$A^*(h_c)$
2	10	6	6
4	112	13	12
6	680	20	18
8	6.384	39	25
10	47.127	93	39
12	3.644.035	227	73
14	—	539	113
16	—	1.301	211
18	—	3.056	363
20	—	7.276	676
22	—	18.094	1.219
24	—	39.135	1.641

Figura 7. Media estimada para cien ejecuciones. Fuente: basado en (Russell, 2004)



Accede a los ejercicios de autoevaluación a través del aula virtual

5.4. Búsqueda por subobjetivos



Accede al vídeo «Otra aproximación a A*» a través del aula virtual

El uso de heurísticas puede reducir drásticamente la complejidad de los algoritmos de búsqueda. Por ello es una estrategia que debe considerarse en muchos problemas, ya sea con una aplicación débil o con una aplicación más fuerte.

El Algoritmo A* emplea una aplicación «débil» de información heurística. Se usan heurísticas para guiar la búsqueda, no para reducir el espacio de posibles soluciones. Se mantienen propiedades de *completitud* y *optimalidad*.

En muchos casos, se requiere/prefiere una aplicación «fuerte» de heurísticas que reduzca el espacio de búsqueda de forma efectiva (en anchura o profundidad), aunque se corra el riesgo de no explorar posibles soluciones, dado que estas son poco probables. Así, es posible que no se encuentre la mejor solución (no ser óptimos). Pero el objetivo es encontrar soluciones «buenas» (no necesariamente óptimas) y mejorar el rendimiento de forma substancial, aunque la *optimalidad* y *completitud* no se garanticen.

La **búsqueda por subobjetivos** tiene como finalidad principal **reducir la complejidad de un problema de búsqueda**. La idea se fundamenta en disminuir la profundidad de las búsquedas, subdividiendo el problema en varios subproblemas más pequeños mediante una heurística fuerte.

Intenta determinar una secuencia de estados intermedios (i_1, i_2, \dots, i_n) que con mayor probabilidad van a encontrarse en el camino óptimo, ver figura 8. Con estos estados intermedios, realizará búsquedas con un **método base** (amplitud, o profundidad, por ejemplo) desde el estado inicial s_0 a i_1 , de i_1 a i_2 , ..., y de i_n al estado meta s_m . Por ejemplo, en la Figura 9, vemos el árbol generado para solucionar el subproblema de s_0 a i_1 , con la búsqueda en amplitud.

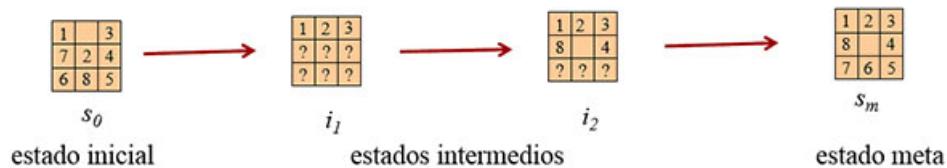


Figura 8. Subobjetivos para el problema de Puzzle-8.

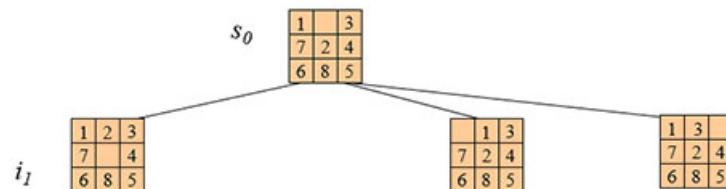


Figura 9. Método base: búsqueda en amplitud camino de s_0 a i_1 .

Otro ejemplo se puede observar en la figura 10, donde se muestra el árbol que soluciona el subproblema de encontrar un camino para ir del sub estado i_1 a i_2 .

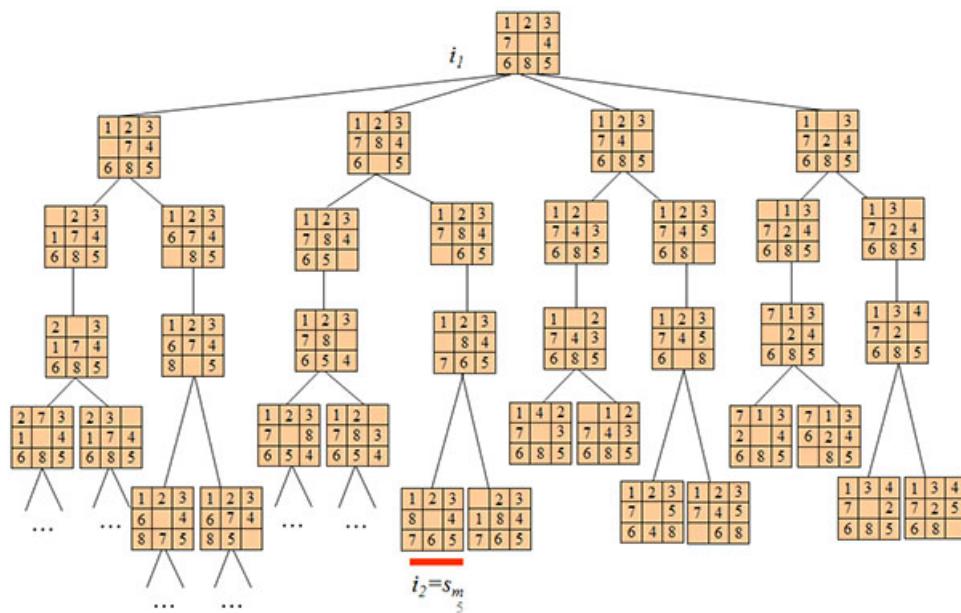


Figura 10. Camino de i_1 a i_2 .

Complejidad de búsqueda por subobjetivos

En esta técnica nos encontramos con que la complejidad en tiempo y espacio depende de condiciones diferentes:

- Complejidad en tiempo: nodos expandidos.
- Complejidad en espacio: nodos abiertos.

En general, si la elección de subobjetivos es buena, se obtiene una mejora en la complejidad, siempre que los caminos entre los pares de subobjetivos estén «más cortos» que la solución global.

Por ejemplo, en la figura 11, se muestra la complejidad algorítmica para un problema cuyo factor de ramificación fuese $b=3$, la profundidad de la solución fuese $d=20$ y creásemos $x=4$ subobjetivos de profundidad $d'=5$ (el camino encontrado tiene una longitud de 25), y además empleáramos como técnica básica un algoritmo de búsqueda en amplitud.

	General		Ejemplo	
	Bús. prof.	Bús. subob.	Bús. prof.	Bús. subob.
mejor caso: - tiempo:	$\frac{b^d - 1}{b - 1}$	$(x + 1) \frac{b^{d'} - 1}{b - 1}$	1.743.392.200	605
- espacio:	$\frac{b^{d+1} - 1}{b - 1}$	$(x + 1) \frac{b^{d'+1} - 1}{b - 1}$	5.230.176.601	1820
peor caso: - tiempo:	$\frac{b^{d+1} - 1}{b - 1} - 1$	$(x + 1) \frac{b^{d'+1} - 1}{b - 1} - 1$	5.230.176.600	1815
- espacio:	$\frac{b^{d+2} - 1}{b - 1} - b$	$(x + 1) \frac{b^{d'+2} - 1}{b - 1} - b$	15.690.529.801	5450

Figura 11. Estimaciones de complejidad algorítmica.

Análisis de búsqueda por subobjetivos:

Permite reducir de forma substancial la complejidad, especialmente en problemas de planes muy largos. La búsqueda por subobjetivos no es necesariamente completa ni óptima.

Por tanto, es completo si el método base es completo y los subobjetivos se encuentran en el camino que nos permite llegar desde el estado inicial al final. Asimismo, será óptimo si el método base es óptimo y los subobjetivos se encuentran

ordenados de tal modo que su camino sea el mínimo posible dentro del camino solución.

En el caso de utilizar búsquedas heurísticas como método base (por ejemplo, A*), se puede alcanzar una reducción aún mayor de la complejidad, pero puede llegar a ser necesario especificar distintas funciones heurísticas para cada subobjetivo.



Accede a los ejercicios de autoevaluación a través del aula virtual

5.5. Búsqueda *online*



Accede al vídeo «Otras búsquedas informadas» a través del aula virtual

En algunos entornos no es posible/útil encontrar un plan completo. Por ejemplo, en entornos reactivos, o en entornos no deterministas o dinámicos, donde no se pueden determinar los resultados de las acciones, o el entorno y/o el estado meta puede cambiar. Por ejemplo: seguir un objeto en movimiento (escalera de Hogwarts).

Tampoco es posible encontrar un plan completo en entornos (parcialmente) inaccesibles, donde se desconoce parte del entorno (estados existentes, resultados de las acciones). Por ejemplo: el dominio de un laberinto donde se conoce parcialmente el laberinto y el mismo se va descubriendo a medida que el agente camina por el laberinto.

Surgen de la necesidad que tienen los agentes reactivos de encontrar una solución en un tiempo muy corto. Esta falta de tiempo para encontrar la solución hace que en muchos casos baste con encontrar una buena acción, no un plan de acción completo bueno.

La idea básica es combinar un paso de búsqueda con un paso de actuación. En la tabla 2, podemos ver una comparación entre los dos tipos de búsqueda, online y offline.

Búsqueda online	Búsqueda offline
Repetir: 1. Percibir entorno. 2. Elegir la «mejor» acción. 3. Ejecutar acción. Fin repetir.	1. Percibir entorno. 2. Buscar plan. 3. Ejecutar plan.

Tabla 2. Comparativa entre la búsqueda online y offline.

A continuación, vamos a ver, los siguientes tipos de búsqueda online:

- ▶ Búsqueda por ascenso de colinas.
- ▶ Búsqueda por horizonte.
- ▶ Optimización mediante búsqueda online.

Búsqueda por ascenso de colinas (*hill climbing*)

Tiene como idea inicial disminuir la profundidad del árbol de búsqueda e intentar llevar a cabo la acción que parece más prometedora en cada momento, repitiendo el ciclo percepción/acción de forma continua, tal como se muestra en la figura 12.

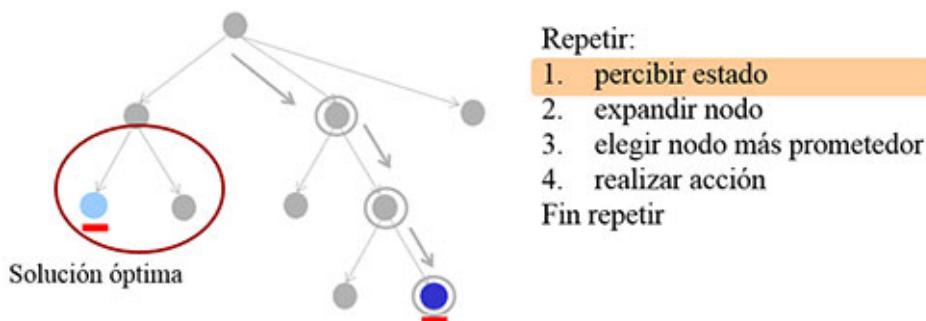


Figura 12. Comportamiento de la búsqueda por ascenso de colinas.

El algoritmo de búsqueda por ascenso de colinas (Skiena, 2010), ver figura 13, emplea una función heurística para calcular la distancia estimada al nodo meta más cercano desde el nodo n , $h^*(n)$; por otro lado, debe retornar el estado actual del problema por medio de un mecanismo de percepción ($percibir(entorno)$). En un estado determinado por el nodo n , debe evaluar cuál de las posibles acciones es la mejor por medio de la estimación heurística de las opciones presentes ($evaluar(n,h^*)$). Una vez decidida cuál es la mejor acción, la ejecutaremos dentro del entorno.

```

{búsqueda ascenso de colinas}
Repetir
    nodo  $\leftarrow$  percibir(entorno)
    Si meta?(nodo) entonces
        devolver(positivo)
    Si vacía?(sucesores) entonces
        devolver(negativo)
    mejor  $\leftarrow$  arg minnesucesores[evaluar(n,h*)]
    a  $\leftarrow$  acción(n.mejor)
    ejecutar(a, entorno)
Fin {repetir}

```

Figura 13. Algoritmo general de la búsqueda por ascenso de colinas.

Análisis de búsqueda por ascenso de colinas

En general, **no se puede asegurar optimidad ni completitud**. Pero suele producir resultados mejores, sobre todo en los casos en los que no existan bucles y la función heurística sea buena. Si h^* es completa ($h^* = h$), entonces es **óptimo y completo**. Hay que vigilar la eliminación de ciclos y eliminar los estados repetidos.

Este algoritmo es utilizado para resolver problemas donde se debe encontrar un óptimo local (una solución que no puede ser mejorada considerando una configuración de la vecindad) pero no garantiza encontrar la mejor solución posible (el óptimo o máximo global) de todas las posibles soluciones (el espacio de búsqueda). Aun así, esto se puede remediar utilizando reinicios (búsqueda local

repetida), o esquemas más complejos basados en iteraciones (Lourenço, 2019), como búsqueda local iterada, en memoria, como optimización de búsqueda reactiva (Battiti, Brunato, & Mascia, *Reactive Search and Intelligent Optimization*, 2008) y/o búsqueda tabú (Battiti & Tecchiolli, *The reactive tabu search*, 1994).

Búsqueda con horizonte

Hacemos una expansión de los nodos hasta la profundidad k (horizonte) y realizamos aquella acción que es más prometedora de todas las acciones de ese nivel, repitiendo de forma continua hasta que encontramos una solución dentro del horizonte de búsqueda.

El algoritmo general arranca en un estado s , desde el cual aplicamos una búsqueda no informada de profundidad k , que puede ser suspendida en los nodos meta. De las hojas (H) adquiridas en el nivel k , obtenemos la mejor empleando una función heurística h^* , de tal modo que optaremos por aquella acción que nos lleve a la hoja de menor coste estimado. Repetiremos este proceso hasta alcanzar la meta. A continuación, en las figuras 14, 15 y 16, se muestra un ejemplo de tres pasos.

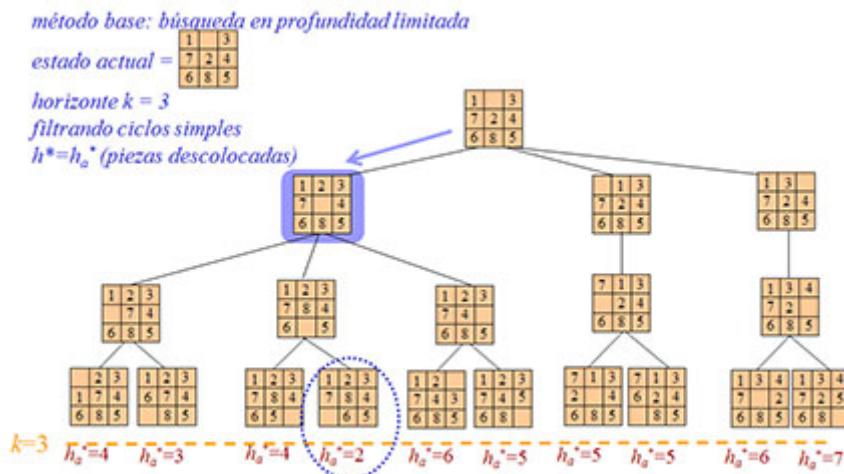


Figura 14. Ejemplo de búsqueda en profundidad limitada (paso 1).

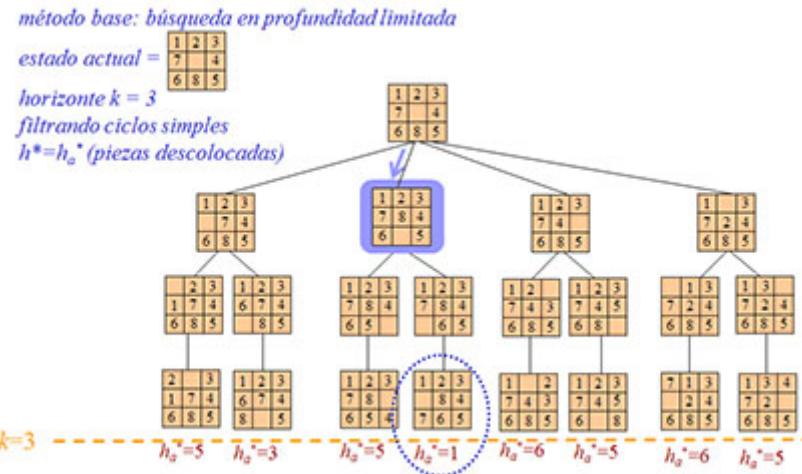


Figura 15. Ejemplo de búsqueda en profundidad limitada (paso 2).

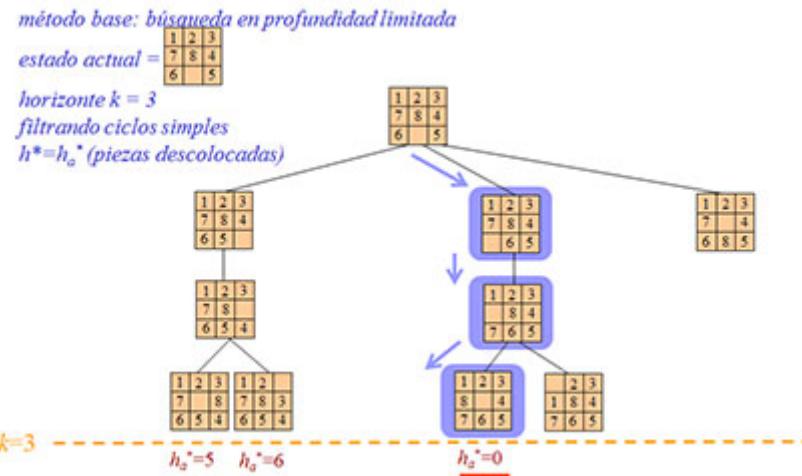


Figura 16. Ejemplo de búsqueda en profundidad limitada (paso 3, final).

Análisis de búsqueda con horizonte

En general, no se puede asegurar *optimalidad* ni *completitud*, dado que esta está ligada directamente con la función heurística. En el caso de tener una **h^* completa ($h^* = h$)**, entonces es óptimo y completo.

La búsqueda por ascenso de colinas es un subtipo de la búsqueda con horizonte (horizonte = 1). El aumento del horizonte de exploración permite evitar bucles de longitud menor que k .

Optimización mediante búsqueda *online*

Un subtipo de problemas a los que se puede aplicar con éxito las búsquedas online (horizonte/ascenso de colinas) es el de los problemas de optimización, es decir, aquellos en los que tenemos una función objetivo que deseamos alcanzar. En este tipo de problemas no necesitamos obtener el valor máximo/mínimo de la función que deseamos alcanzar, sino que nos basta con el valor que más se aproxime a dicho óptimo.

El camino para llegar al estado buscado puede ser irrelevante (coste 0). Algunos ejemplos: juegos lógicos y de configuración, n-Reinas.

En los problemas de optimización con funciones heurísticas estimas el coste del camino que parte del estado actual y va hasta el nodo meta más próximo, midiendo la calidad de un nodo con respecto a una función de evaluación objetivo.



Accede a los ejercicios de autoevaluación a través del aula virtual

5.6. Referencias bibliográficas

Battiti, R., & Tecchiolli, G. (1994). The reactive tabu search. *ORSA Journal on Computing*, 6 (2): 126-140.

Battiti, R., Brunato, M., & Mascia, F. (2008). *Reactive Search and Intelligent Optimization*. Springer Verlag ISBN 978-0-387-09623-0 archivado desde el original el 16 de marzo de 2012.

Lourenço, H. R. (2019). *Iterated local search: Framework and applications*. Springer, Cham.

Russell, S. y. (2004). *Inteligencia Artificial: Un Enfoque Moderno*. Madrid: Pearson Educación.

Skiena, S. S. (2010). *The Algorithm Design Manual, Softcover reprint of hardcover 2nd*. Springer.

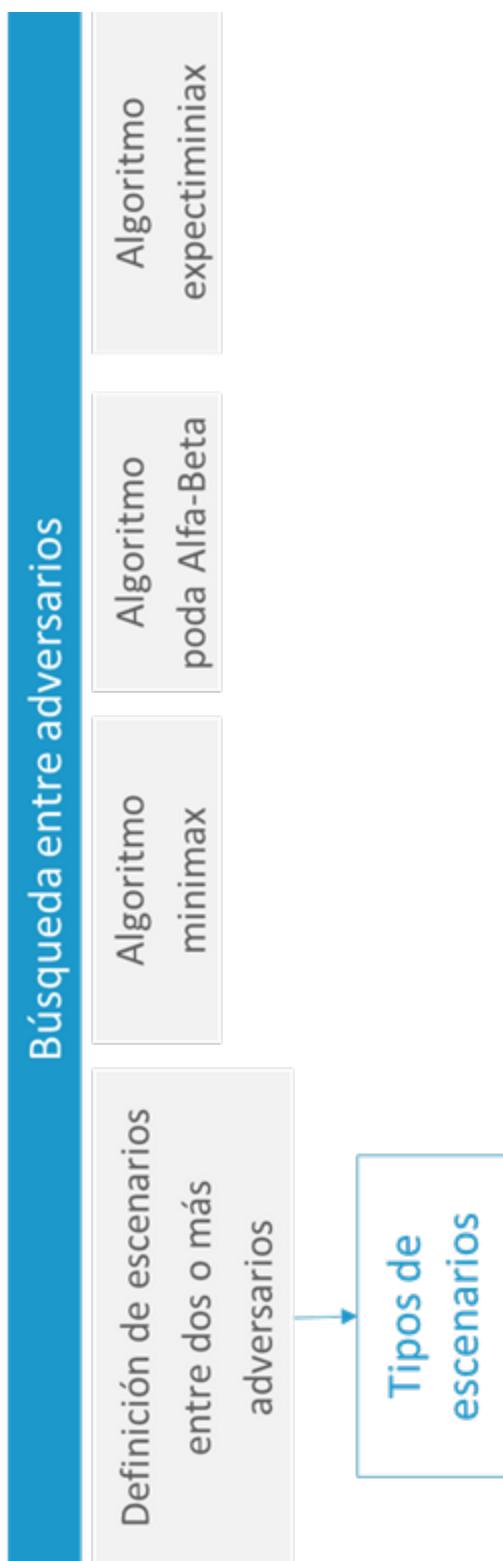


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Búsqueda entre adversarios

Esquema. Tema 6



Ideas clave. Tema 6

6.1. ¿Cómo estudiar este tema?

En este tema trataremos aquellos problemas en los que nos encontramos compitiendo con más de un agente sobre el mismo entorno. Son problemas propios de la Teoría de Juegos. En ellos deberemos tener en cuenta el objetivo de cada uno de los agentes y sus interrelaciones.

En esta clase de problemas, es obvio que los agentes no siempre tienen los mismos objetivos y, como buenos entes racionales, desean alcanzarlos del modo más eficiente posible. En la mayoría de los casos, estos problemas se definen para entornos en los que hay dos agentes enfrentados con metas contrapuestas, pero las técnicas que se emplean pueden ser las mismas en otros muchos escenarios que no son necesariamente bipersonales y/o competitivos.

Hablaremos principalmente del algoritmo minimax presentado por von Neumann, en 1928, en su demostración teórica sobre juegos bipersonales de «suma cero». (Hazewinkel, 2013).



Accede a los ejercicios de autoevaluación a través del aula virtual

6.2. Introducción



Accede al vídeo «Introducción» a través del aula virtual

Los problemas entre adversarios son aquellos en los que más de un agente especializado actúa de modo concurrente en un mismo entorno. Los podemos

considerar como problemas que suceden en entornos con múltiples agentes, donde cada agente es independiente e intenta conseguir sus propios objetivos. Cualquier acción ejecutada por un agente influye en el rendimiento de los demás agentes del entorno, que no pueden controlar las acciones que el resto va a realizar, aunque sí pueden intentar predecir el comportamiento de los otros agentes.

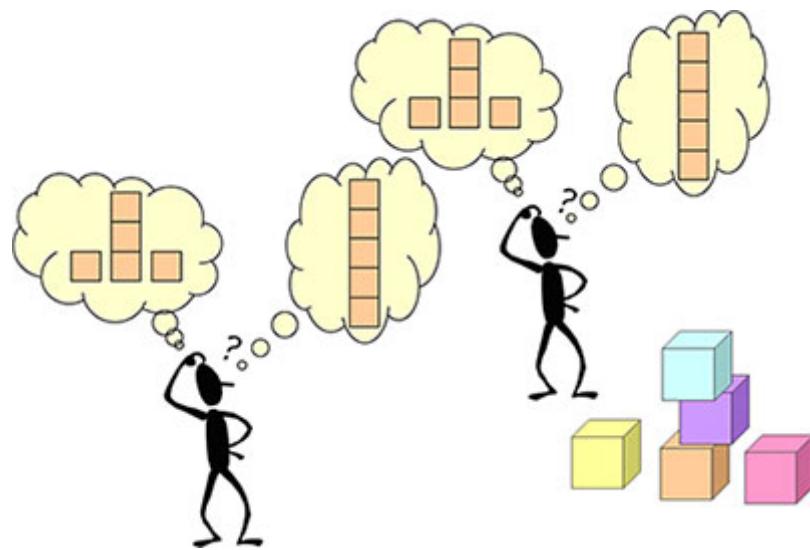


Figura 1. Varios agentes que tienen que interactuar sobre un mismo entorno pueden o no compartir objetivos.

En escenarios multi agentes, nos encontramos con distintos tipos de problemas, atendiendo al comportamiento de los agentes entre sí:

- ▶ Escenarios cooperativos: en los que los agentes tienen metas compartidas y, por tanto, las acciones de unos deberían facilitar los objetivos de todos los agentes.
- ▶ Escenarios parcialmente cooperativos: aquellos escenarios en los que algunas metas son compartidas por varios agentes, pero otras pueden ser opuestas.
- ▶ Escenarios antagónicos o competitivos: donde las metas de todos los agentes son opuestas. El ejemplo «clásico» de escenarios antagónicos es el juego de la suma nula, donde un (grupo de) jugador(es) solo puede ganar algo a costa de otro (grupo de) jugador(es).

Tal como lo mencionamos anteriormente, los algoritmos que explicaremos en este tema estarán relacionados con los escenarios antagónicos o de suma nula.

Se pueden distinguir diferentes tipos de juegos de suma nula en función de:

- ▶ Número de jugadores: juegos bipersonales (damas) frente a juegos de varios jugadores (Monopoly o Catán).
- ▶ Elementos de azar: juegos con elementos de azar (parchís) frente a juegos sin elementos de azar (ajedrez).
- ▶ Información: juegos con información perfecta (damas) frente a juegos con información incompleta (*black jack*).

Ejemplo del juego de tres en raya

Emplearemos este juego durante el tema para exemplificar el funcionamiento de los distintos algoritmos.

En el juego de las tres en raya participan dos jugadores (que denominaremos **min.** y **máx.**). Min. y máx. colocan fichas en las casillas de un tablero 3x3. Por convenio, máx. usa las fichas X y min. usa las fichas O.

En cada casilla solo puede haber una ficha. El tablero comienza vacío y máx. empieza. Posteriormente, y de modo alternado, min pondrá otra pieza. Así, hasta que se alcance un resultado (vence máx., vence min o empatan). Al ser un juego de «suma cero», estos resultados siempre implican que o bien se empata, o bien todo lo que gana un jugador es lo que pierde el otro.

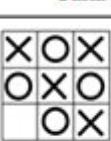
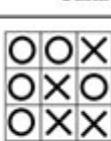
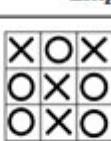
Gana Max	Gana Min	Empate
 gana max	 gana min	 empate

Figura 2. Posibilidades en el juego de las tres en raya. Fuente: (Billhardt, 2015)

Modelos de juegos bipersonales

Nos vamos a centrar en los juegos bipersonales porque permiten ver claramente cómo funcionan los distintos algoritmos que conforman el tema.

Los agentes máx. y min. tienen un conocimiento mínimo *a priori* que se puede ver como:

- s_0	<i>posición inicial</i> (estado inicial)
- <i>expandir</i> : $s \mapsto \{s_{i_1}, \dots, s_{i_n}\}$	cjto. finito de <i>posiciones sucesores</i>
- <i>terminal?</i> : $s \mapsto \text{true} \mid \text{false}$	<i>prueba terminal</i>
- $U: s \mapsto k, k \in \mathbb{N}$	función parcial de <i>utilidad</i> del juego

Figura 3. Conocimiento a priori de un agente.

La función *expandir* crea los estados derivados de emplear cualquiera de las acciones permitidas en el estado s .

En este tipo de problemas, evaluaremos el desempeño de los agentes con base a una función de utilidad, que solo se define, inicialmente, en los estados meta. Por ejemplo, en los problemas de suma nula, máx. gana si y solo si min. pierde. Por tanto, la función de utilidad de un estado s , $U(s)$, es:

gana máx: $U(s) = +\infty$	empate: $U(s) = 0$	gana min: $U(s) = -\infty$
----------------------------	--------------------	----------------------------

Tabla 1. Función de utilidad básica.

Al igual que en los algoritmos de búsqueda que vimos anteriormente, los algoritmos de resolución de escenarios bipersonales o entre adversarios emplean la construcción de árboles de búsqueda para gestionar el proceso de concesión de la meta u objetivo del agente.

En el caso de escenarios de juego como los descritos en el presente tema, el árbol que generamos se llama **árbol de juego G**. Y formalmente está formado por la tupla $G = \langle N, E, L \rangle$, donde:

- N , es un conjunto de nodos,

- ▶ $E \subseteq NxN$, es una función sobre N que asigna a cada nodo N sus hijos
- ▶ $L = \{\text{máx., min.}\}$, es una función sobre N que etiqueta a cada nodo N a que jugador le toca el turno

En este árbol, G empieza siendo vacío y etiquetaremos la raíz como máx., indicando que el turno inicial es siempre de este. Todos los sucesores se irán entregando de modo ordenado.

Cada nivel del árbol representa media jugada (un *ply*). Por otro lado, todas las hojas de un árbol, si se expande completamente, representan nodos terminales del juego.

Ejemplo de árbol de juego para tres en raya:

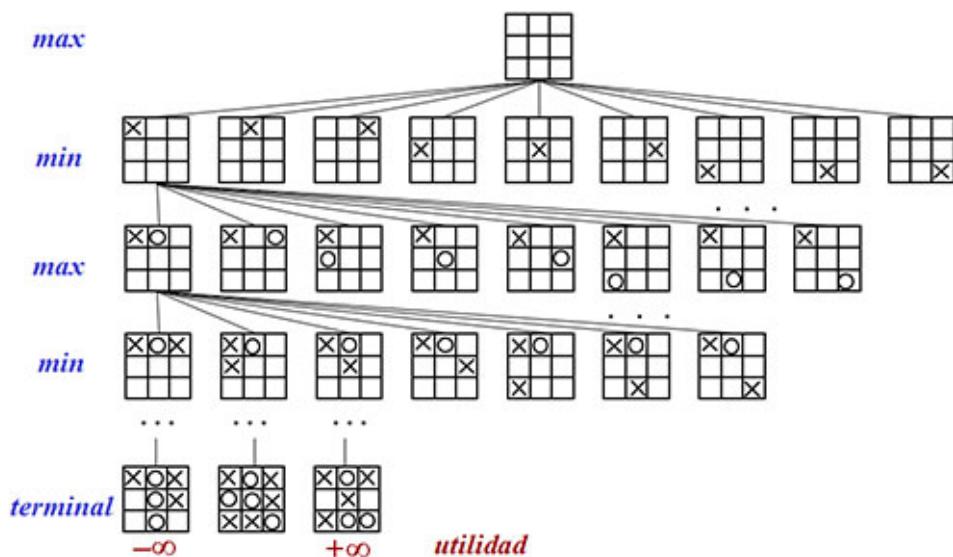


Figura 4. Árbol de juego para tres en raya.

La mejor jugada del agente máx

La estrategia se centra en un solo agente, máx. Y define las jugadas de máx en todos los estados alcanzables del juego. Es decir, da una respuesta a todas las jugadas posibles de máx y del contrincante min. Las jugadas de min serán un subárbol del árbol de juego.

Si aplicásemos únicamente algoritmos de búsqueda estándar, nos encontraríamos con que min nunca querrá realizar acciones que se encuentren en el camino principal de máx.

Estrategia óptima o racional

En los escenarios competitivos con agentes racionales siempre se asume que min realizará la mejor acción para sí mismo de aquellas que el agente máx le haya dejado en su momento.

La estrategia óptima para máx es la **estrategia minimax: maximizar la utilidad mínima en cada jugada**.



Accede a los ejercicios de autoevaluación a través del aula virtual

6.3. Búsqueda minimax



Accede al vídeo «Búsqueda minimax» a través del aula virtual

Minimax es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversario y con información perfecta. Es un cálculo que se emplea de forma recursiva. Su funcionamiento se resume en elegir el mejor movimiento para uno mismo partiendo de la suposición de que el contrincante escogerá el peor para el adversario. La figura 5, describe a modo resumen las características principales del algoritmo Mini-máx.



Figura 5. Método minimax.

En último lugar llegará el valor de utilidad desde un nodo terminal al nodo raíz. Y tendremos que la acción que lleva al camino del nodo meta con mayor valor es la jugada óptima. En la figura 6, se puede ver un ejemplo de selección de la mejor jugada de máx. Como función de utilidad tenemos valores de 0, - infinito e infinito.

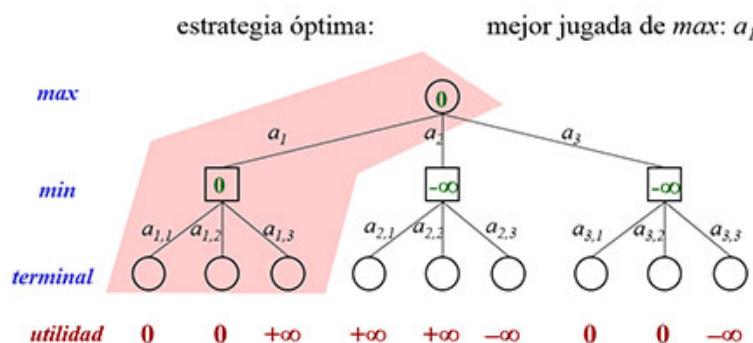


Figura 6. Estrategia óptima para una búsqueda minimax.

El algoritmo explorará los nodos del árbol y les asigna un valor numérico mediante una función de utilidad, empezando por nodos terminales y subiendo a la raíz. La función de utilidad definirá lo buena que es la posición de un jugador (min o máx) cuando la alcanza.

En la figura 7, se puede ver una versión muy preliminar del algoritmo Mini-Max.

Algoritmo:	<ul style="list-style-type: none"> • α: máximo de la utilidad de los sucesores de un nodo <i>max</i> • β: mínimo de la utilidad de los sucesores de un nodo <i>min</i>
{MaxValor en el Minimax básico}	{MinValor en el Minimax básico}
Función MaxValor(estado)	Función MinValor(estado)
Si terminal?(estado) entonces devolver (<i>U</i> (estado)) sucesores \leftarrow expandir(<i>max</i> , estado) $\alpha \leftarrow -\infty$ Para cada s \in sucesores hacer $\alpha \leftarrow \max(\alpha, \text{MinValor}(s))$ devolver (α) Fin {MaxValor}	Si terminal?(estado) entonces devolver (<i>U</i> (estado)) sucesores \leftarrow expandir (<i>min</i> , estado) $\beta \leftarrow +\infty$ Para cada s \in sucesores hacer $\beta \leftarrow \min(\beta, \text{MaxValor}(s))$ devolver (β) Fin {MinValor}

Figura 7. Algoritmo minimax (versión preliminar).

El principal problema de esta técnica es que, incluso en juegos muy simples, resulta imposible crear un árbol completo de juego. Esto es debido al crecimiento exponencial del árbol de juego.

Minimax con suspensión

Cuando el problema manifiesta unos problemas graves de expansión de los nodos, la solución se basa en heurísticas.

La idea es reemplazar la prueba terminal por una prueba de suspensión, es decir, en vez de tener que desarrollar todo el árbol de juego, realizamos una expansión hasta el nodo de profundidad *d*. Para ello deberemos emplear una **función de evaluación** *e*, que estima la utilidad esperada del juego correspondiente a una posición *s* determinada (*e* debe coincidir con la función de utilidad *u* en los nodos terminales).

Lo más habitual es emplear una función de tipo lineal ponderada:

$$e(s)w_1f_1(s) + w_2f_2(s) + \dots + w_nf_n(s)$$

Por ejemplo, para el ajedrez nos podríamos encontrar con una función de la forma:

$e(s)$ = "suma de los valores materiales en s "

Mientras que, en el juego de tres en raya, la función de evaluación puede ser de la forma:

$e(s) = \text{"número de líneas abiertas para líneas máx en } s\text{"} - \text{"número de líneas abiertas para líneas min en } s\text{"}$

Al emplear funciones heurísticas fuertes, dado que deseamos limitar la complejidad, no podremos garantizar que la estrategia minimax sea óptima. Pero claro, esto dependerá de la calidad de las heurísticas.

En el siguiente ejemplo se muestra minimax con suspensión:

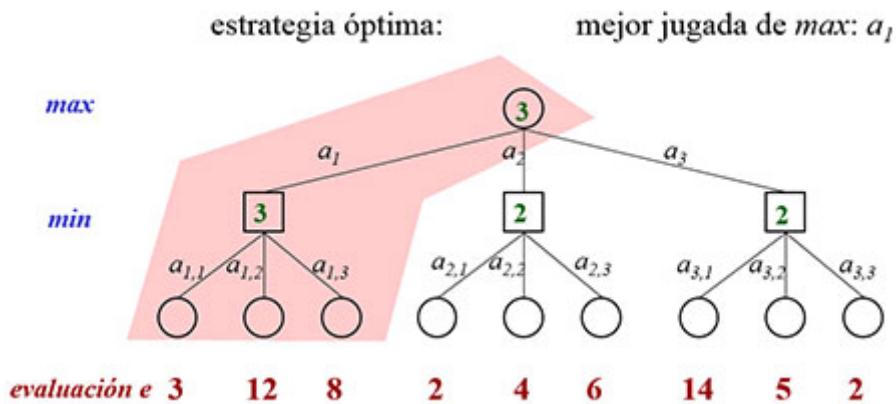


Figura 8. Minimax con suspensión.

El algoritmo lo podemos resumir de la siguiente manera:

Algoritmo:

- funciones mutuamente recursivas
- *estado* es el estado actual

- α : máximo de la evaluación de los sucesores de un nodo *max*
- β : mínimo de la evaluación de los sucesores de un nodo *min*

{MaxValor: Minimax con suspensión}

Función MaxValor(estado)

```

    Si suspensión?(estado) entonces
        devolver(e(estado))
    sucesores ← expandir(max, estado)
    α ← -∞
    Para cada s ∈ sucesores hacer
        α ← max(α, MinValor(s))
    devolver(α)
    Fin {MaxValor}
  
```

{MinValor: Minimax con suspensión}

Función MinValor(estado)

```

    Si suspensión?(estado) entonces
        devolver(e(estado))
    sucesores ← expandir(min, estado)
    β ← +∞
    Para cada s ∈ sucesores hacer
        β ← min(β, MaxValor(s))
    devolver(β)
    Fin {MinValor}
  
```

Figura 9. Algoritmo minimax con suspensión.

Ejemplo de las tres en raya:

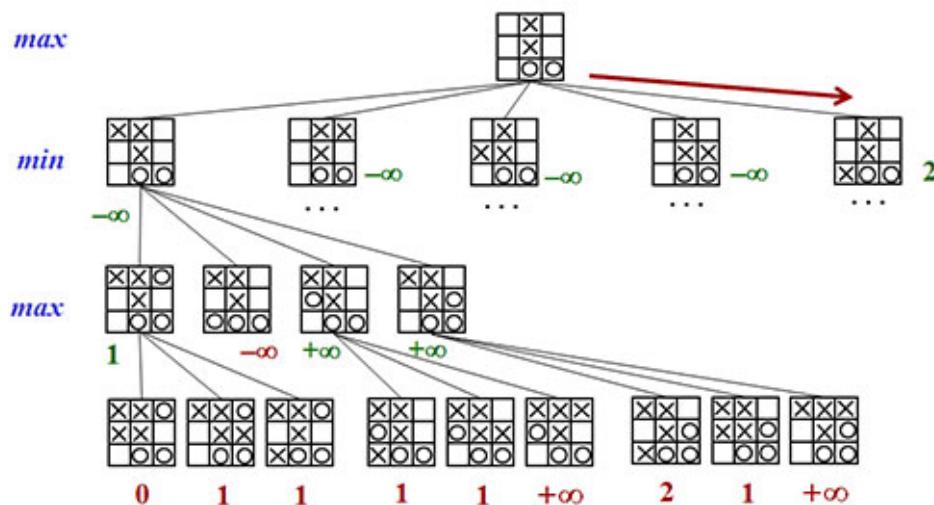
Suspensión en ply 3

Figura 10. Ejemplo de algoritmo minimax con suspensión para el juego tres en raya.

El algoritmo minimax tiene un problema de complejidad importante en el caso general, $O(b^d)$ (factor de ramificación b ; número de *de profundidad del árbol*).

Minimax genera el árbol de juego completo, en profundidad primero, hasta los nodos (o el nivel) de suspensión.

Existen extensiones de la estrategia minimax, algunas las veremos en las secciones siguientes:

- ▶ Mejorar la complejidad (poda alfa-beta: descartar nodos irrelevantes).
- ▶ Juegos con elementos de azar (expectiminimax: hay un nuevo jugador «azar»).
- ▶ Juegos con información parcial (búsqueda en estados de creencias).
- ▶ Mejorar las heurísticas: aprender funciones de evaluación y suspensión (por ejemplo, por el «efecto horizonte»).



Accede a los ejercicios de autoevaluación a través del aula virtual

6.4. La poda alfa-beta



Accede al vídeo «La poda alfa-beta» a través del aula virtual

El crecimiento exponencial de los nodos que se deben explorar en la estrategia minimax hace de él un algoritmo que presenta una complejidad poco deseable. Si bien este exponente no se puede eliminar, por lo menos es posible dividir a la mitad su valor, con la mejora de rendimiento que eso conlleva.

Es decir, que existe la posibilidad de tomar una decisión minimax correcta sin tener que mirar todos los nodos en el árbol. La poda alfa-beta permite eliminar partes del árbol.

La idea inicial es que, en muchas ocasiones, no es necesario evaluar todos los sucesores de un nodo para obtener el valor exacto de dicho nodo.

Por ejemplo:

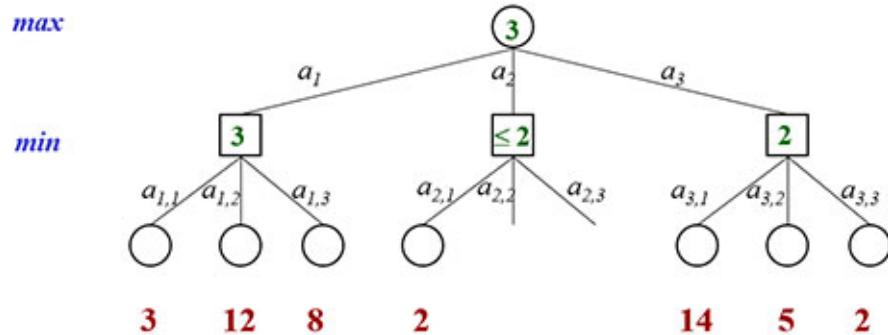


Figura 11. Ejemplo de poda alfa-beta.

En la figura 11, se muestra un árbol en el que se aplica la poda alfa-beta. Se tienen dos tipos de poda.

Poda alfa, sucede cuando es el turno de máx, y se mantienen los siguientes parámetros actualizados:

- ▶ *Alfa (α)*: evaluación más alta encontrada (de momento) en un nodo máx.
- ▶ *Beta (β)*: evaluación más baja encontrada (de momento) en su sucesor directo min.

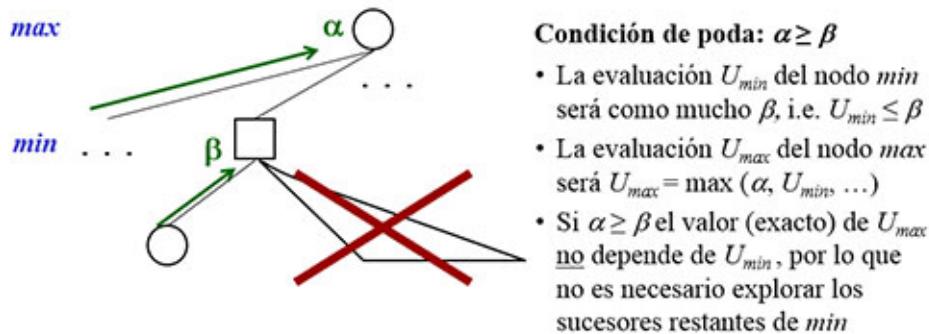
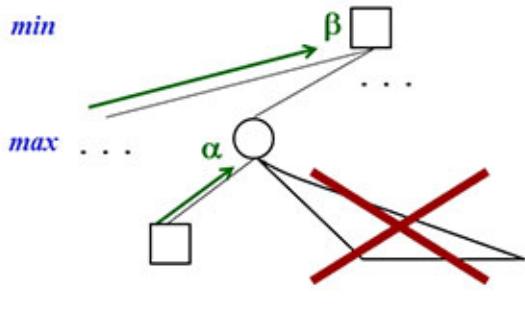


Figura 12. Poda alfa.

Poda beta, sucede cuando es el turno de min, y se mantienen los siguientes parámetros actualizados:

- ▶ *Beta (β)*: evaluación más baja encontrada (de momento) en un nodo min.
- ▶ *Alfa (α)*: evaluación más alta encontrada (de momento) en su sucesor directo máx.



Condición de poda: $\beta \leq \alpha$

- La evaluación U_{\max} del nodo *max* será al menos α , i.e. $U_{\max} \geq \alpha$
- La evaluación U_{\min} del nodo *min* será $U_{\min} = \min(\beta, U_{\max}, \dots)$
- Si $\beta \leq \alpha$ el valor (exacto) de U_{\min} no depende de U_{\max} , por lo que no es necesario explorar los sucesores restantes de *max*

Figura 13. Poda beta.

Estas dos podas se pueden presentar de manera profunda de la siguiente manera.

Poda alfa profunda:

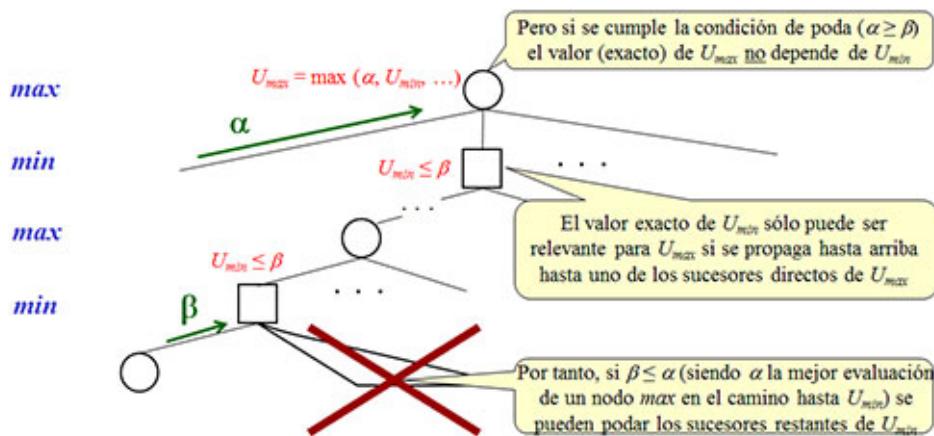


Figura 14. Poda alfa profunda.

La poda beta profunda funciona de forma dual. Emplearemos dos funciones mutuamente recursivas que irán creando las expansiones y podas de máx y min de modo alternado.

Minimax con poda alfa-beta:

Algoritmo:

- funciones mutuamente recursivas
- *estado* es el estado actual

- α es el mejor valor de evaluación para *max* en el camino hasta *estado*
- β es el mejor valor de evaluación para *min* en el camino hasta *estado*

{MaxValor: Minimax con poda $\alpha-\beta$ }Función MaxValor(*estado*, α,β)

```

    Si suspensión?(estado) entonces
        devolver(e(estado))
    sucesores ← expandir(max, estado)
    Para cada s ∈ sucesores hacer
         $\alpha \leftarrow \max(\alpha, \text{MinValor}(s, \alpha, \beta))$ 
        Si  $\alpha \geq \beta$  entonces devolver( $\alpha$ )
    devolver( $\alpha$ )
Fin {MaxValor}
  
```

{MinValor: Minimax con poda $\alpha-\beta$ }Función MinValor(*estado*, α,β)

```

    Si suspensión?(estado) entonces
        devolver(e(estado))
    sucesores ← expandir(min, estado)
    Para cada s ∈ sucesores hacer
         $\beta \leftarrow \min(\beta, \text{MaxValor}(s, \alpha, \beta))$ 
        Si  $\beta \leq \alpha$  entonces devolver( $\beta$ )
    devolver( $\beta$ )
Fin {MinValor}
  
```

Figura 15. Minimax con poda alfa-beta.

Ejemplo minimax con poda alfa-beta:

- **Negro / Rojo**: valores de α y β dentro de las funciones *MaxValor* y *MinValor*
- **Azul**: valores de los parámetros en las llamadas a *MaxValor* y *MinValor*
- **Amarillo**: valores devueltos por *MaxValor* o *MinValor*

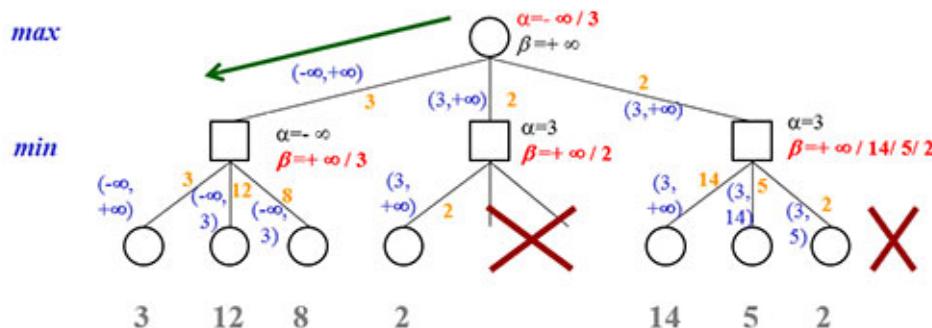


Figura 16. Ejemplo minimax con poda alfa-beta.

El algoritmo minimax con poda alfa-beta ($\alpha-\beta$) siempre produce el mismo resultado que sin la poda. Pero existe una relación directa en la mejora del rendimiento de la poda asociada al orden de exploración de los nodos. La eficiencia de minimax con poda $\alpha-\beta$ depende del orden en el que se exploran los nodos.

La Tabla 2 compara el tiempo computacional de la búsqueda minimax y la poda alfa-beta.

Ejemplo de complejidad para factor de ramificación b y profundidad d del árbol		$B=10, d=4$
Minimax sin poda	$O(b^d)$	$10^4 = 10\,000$ nodos
Minimax con poda (mejor caso)	$O(b^{d/2})$	$10^2 = 100$ nodos
Minimax con poda (caso medio)	$O(b^{3d/4})$	$10^3 = 1\,000$ nodos

Tabla 2. Análisis de complejidad del algoritmo minimax.



Accede a los ejercicios de autoevaluación a través del aula virtual

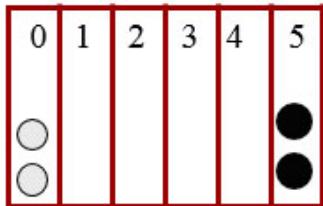


Accede al vídeo «Búsqueda expectiminimax» a través del aula virtual

Para los juegos bipersonales con elementos de azar se suele emplear el algoritmo expectiminimax (Billhardt, 2015). Cuando nos encontramos en un escenario donde aparecen elementos de azar, hay que evolucionar la idea del minimax. Para ello, añadimos un nuevo jugador «azar», que se incluye en el árbol siempre que haya un evento independiente de los jugadores y cuyo resultado es aleatorio.

Los nodos sucesores de estas jugadas de «azar» son los posibles valores resultantes de los elementos de azar. Por ejemplo, los distintos valores de una tirada de dado o el éxito o fracaso de una acción azarosa. Cada uno de los sucesores de un nodo «azar» tiene, por tanto, asociada una probabilidad de ocurrencia.

En un ejemplo como el del Backgammon simplificado en el que partimos de un estado inicial como este:



El backgammon es un juego de mesa para dos jugadores que une el azar con profundos conocimientos estratégicos. El objetivo es conseguir sacar fichas del tablero antes que el jugador rival.

Figura 17. Backgammon.

Tenemos el objetivo de mover nuestras fichas (por ejemplo, blanca para máx hasta el campo cinco y min lo tendrá que hacer hasta el campo cero), contando con que empieza máx y que en cada jugada primero se tira una moneda (donde obtener cara tiene el valor uno y cruz el valor dos).

Después, se mueve una de las fichas uno o dos campos en la dirección deseada (dependiendo del resultado de la tirada de la moneda), no siendo posible mover fichas a un campo donde el adversario tenga ya una. Y, en el caso de que el jugador no pueda mover, pierde el turno.



Figura 18. Árbol del juego. Representación de estados: (x_1, x_2, y_1, y_2) , donde x_1 y x_2 son posiciones de las fichas blancas y las y_1 e y_2 posiciones de las fichas negras.

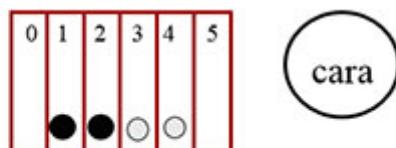
ExpectiMinimax tiene como objetivo, al igual que el minimax, elegir la mejor jugada para máx. Para ello debemos propagar los valores de utilidad/evaluación hacia los

nodos padres. La figura 19, muestra formalmente como se obtienen los valores de evaluación cuando el nodo es terminal (suspensión), máx, min o azar.

$$ExpectMinimax(n) = \begin{cases} e(n) & \text{si } n \text{ es nodo suspensión} \\ \max_{s \in \text{expandir}(n)} (ExpectMinimax(s)) & \text{si } n \text{ es nodo } \max \\ \min_{s \in \text{expandir}(n)} (ExpectMinimax(s)) & \text{si } n \text{ es nodo } \min \\ \sum_{s \in \text{expandir}(n)} (p(s) \cdot ExpectMinimax(s)) & \text{si } n \text{ es nodo azar} \end{cases}$$

Figura 19. Valor propagado al padre en función del tipo de nodo.

Por ejemplo, partiendo de la situación de partida siguiente y tocándole a máx:



(3,4,1,2); max tiene que mover una ficha (blanca) una posición

Figura 20. Situación de una partida de Backgammon.

Si el nivel de suspensión cinco y la función de evaluación que empleamos es la siguiente:

$$e(a,b,c,d) = a+b+c+d$$

Dado que los valores de a, b, c y d son mejores para máx cuanto más cerca del cinco se encuentren, tendríamos una evaluación para el estado anterior de:

$$e(3,4,1,2)=10$$

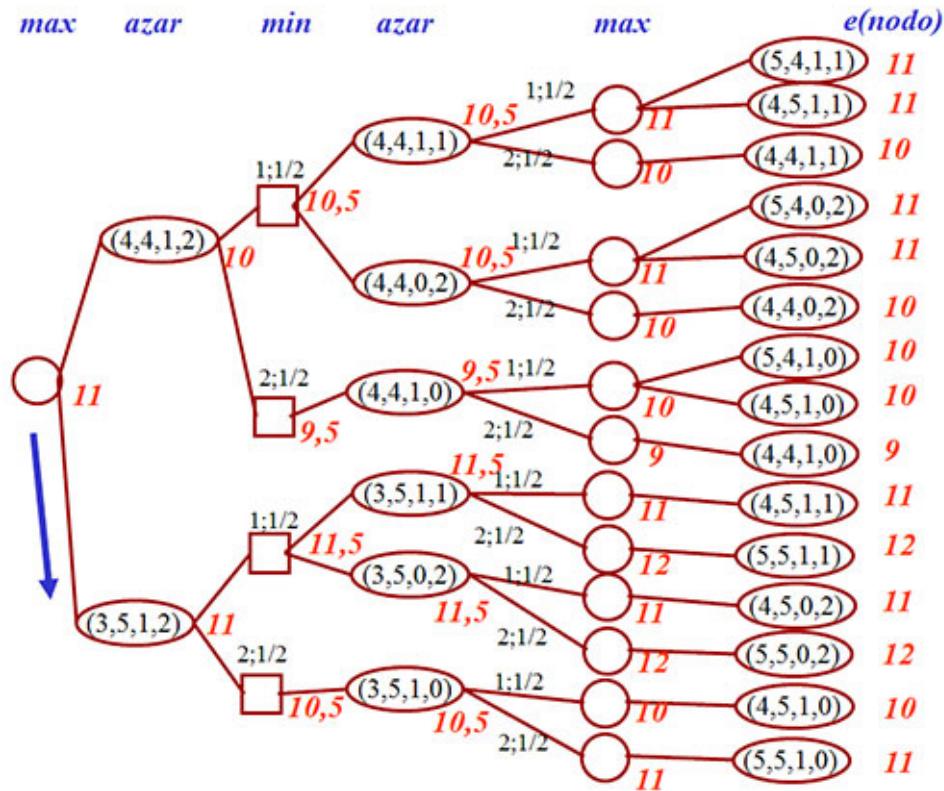


Figura 21. Expansión de nodos para un algoritmo expectiminimax para el backgammon.

Criterios para las funciones de evaluación/utilidad

Es importante tener en cuenta que la escala de los valores de dichas funciones sí importa (no como ocurría en el algoritmo minimax):

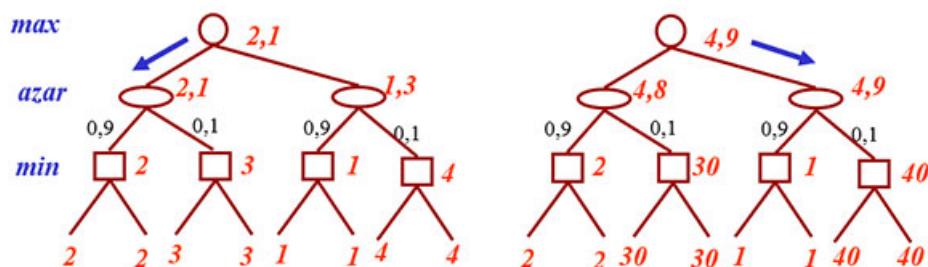


Figura 22. Dos ejemplos de árbol con distintos valores de escala generan distintos recorridos.

Por tanto, las funciones no deben devolver $+\infty$ o $-\infty$ porque reservaremos estos valores para los nodos de azar, por lo que es interesante escalar los valores a un intervalo finito (por ejemplo, entre 0 y 1).

La **función de evaluación** e debe estimar la probabilidad de ganar. En el caso ideal, e es una **transformación lineal positiva** de dicha probabilidad.

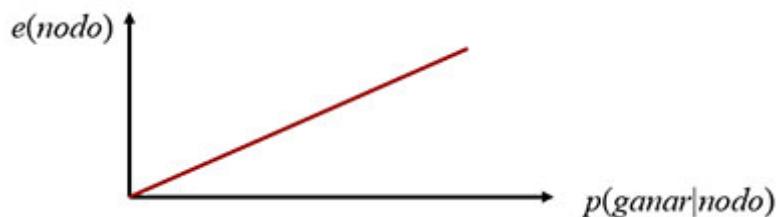


Figura 23. Estimación ideal de la probabilidad de victoria en función del valor $e(nodo)$.

Habitualmente, no es sencillo establecer una función e que cumpla este criterio para todos los estados del juego (nodos). Por lo que, normalmente, una función e es más «exacta» cuanto más cerca está el estado actual del juego de un estado terminal.

Es preferible aplicar la búsqueda *expectiminimax* en el máximo número de *plys* posibles y , solo al final, aplicar la función de evaluación e .

La complejidad *expectiminimax* es muy elevada y depende del nivel de profundidad d (*plys*), el factor de ramificación b (jugadas posibles) y los elementos de azar con n posibilidades, pero si incluimos los nodos de azar, nos encontramos con una complejidad de $O(b^d \cdot n^d)$.

Número de <i>plys</i> d anticipados	Número de nodos en el árbol
1	$20 * 21 = 420$
2	176.400
3	74.088.000

Tabla 3. Ejemplo Backgammon: $n=21$ (2 dados) y $b \gg 20$

Para más detalles sobre la aplicación de la poda α - β al algoritmo *expectiminimax*, puedes ver la sección 5.5.1. de (Russell, 2004).



Accede a los ejercicios de autoevaluación a través del aula virtual

6.6. Referencias bibliográficas

Billhardt, H. G. (2015). *Inteligencia artificial: Ejercicios resueltos*. Editorial Universitaria Ramon Areces.

Hazewinkel, M. (2013). *Minimax Principle*. En Hazewinkel, M. (Ed.), *Encyclopaedia of Mathematics*, (Vol. Volume 6). Dordrecht: Springer.

Russell, S. y. (2004). *Inteligencia Artificial: Un Enfoque Moderno*. Madrid: Pearson Educación.

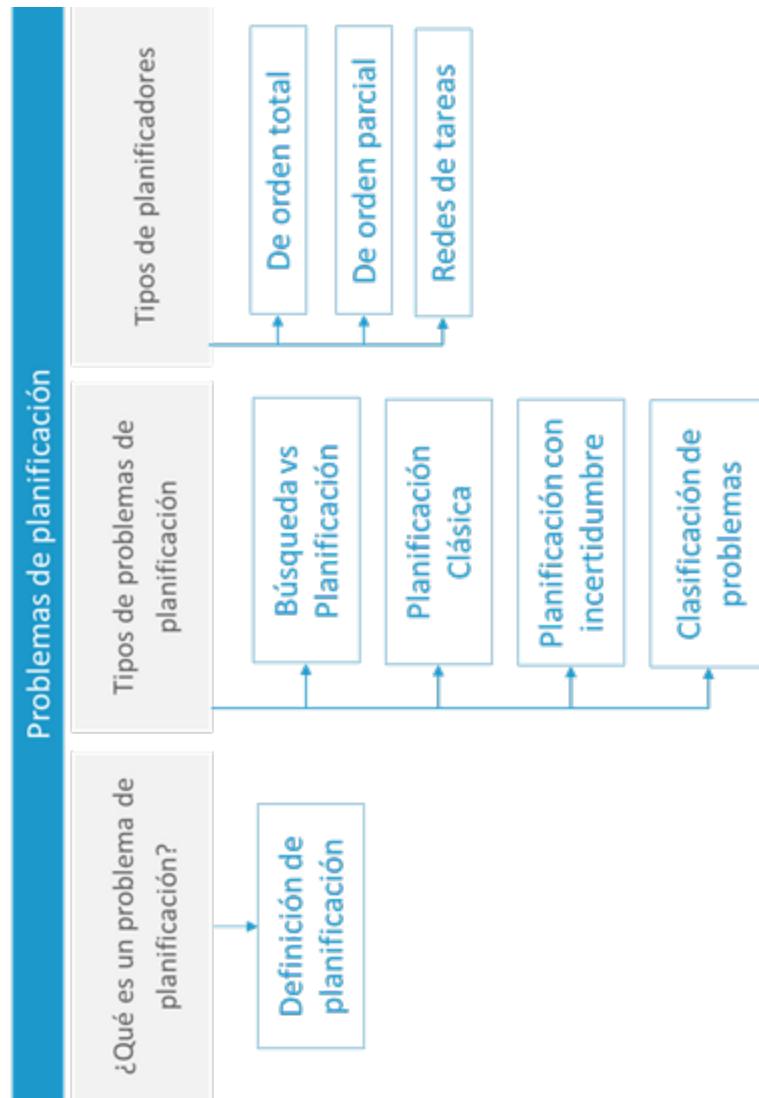


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Problemas de planificación

Esquema. Tema 7



Ideas clave. Tema 7

7.1. ¿Cómo estudiar este tema?

En este tema introduciremos los matices que definen un problema de planificación y sus diferencias principales respecto a los problemas de búsqueda generales.

Clasificaremos los problemas de planificación y presentaremos los conceptos generales que hacen necesaria la planificación en situaciones que requieren estrategias que construyan los planificadores de orden total y de orden parcial.

Expresaremos también el caso particular de los planificadores en un entorno específico como es el de los simuladores en tiempo real.

En el capítulo 11 (*Planificación*) del libro (Russell, 2004), tenemos muchos conceptos generales de la planificación clásica en inteligencia artificial.



Accede a los ejercicios de autoevaluación a través del aula virtual

7.2. ¿Qué es un problema de planificación?



Accede al vídeo «¿Qué es un problema de planificación?» a través del aula virtual

La planificación automática en inteligencia artificial apunta a secuencias ordenadas de acciones que alcanzan objetivos específicos, que definimos como **planes**. Los planes generados deben poder ser ejecutados por los agentes; de este modo, deben ser secuencias de acciones que un agente inteligente, robot o máquina pueda ejecutar.

«Desde principios de los años 70, la comunidad de IA especializada en planificación se ha ocupado del problema del diseño de agentes artificiales capaces de actuar en un entorno» (Vázquez-Salceda, 2011).

En los últimos años, se ha empezado a imponer el criterio de que los sistemas planificadores deberían ser una pieza primordial de gran parte de los agentes inteligentes artificiales, especialmente si queremos que usen estructuras cognitivas de razonamiento.

La idea principal que subyace a este concepto es proporcionar a los agentes inteligentes la capacidad de representar el objetivo a alcanzar, para lo cual formalizan las acciones que pueden realizar y generan un modelo simbólico del entorno.

Definición de planificación

Se define **planificación** como el **proceso formalizado de búsqueda de secuencias de acciones que partiendo del estado actual del entorno satisfacen una meta**. (Russell, 2004)

- ▶ Estado actual del entorno: es una representación estructura que crea el agente al momento de percibir su entorno. Como, por ejemplo, su posición actual, si llueve o no llueve, la posición de una mesa, etc.
- ▶ Meta: es cualquier condición que un agente quiera satisfacer. Un agente puede tener varias posibles metas, pero en un instante determinado **solo una puede estar activa**, controlando el comportamiento.
- ▶ Acción: es un paso simple y atómico dentro de un plan que hace que un agente haga algo (ir a un punto, activar un objeto, etc.)
- ▶ Plan: secuencia de acciones.
- ▶ Proceso de planificación: Un agente proporciona a un sistema (planificador) un estado actual del entorno, un conjunto de acciones y una meta que desea satisfacer, y el planificador busca un plan que con la ejecución de sus acciones consiga esta meta.

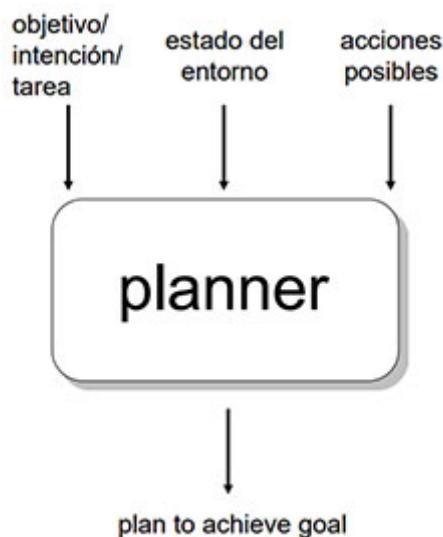


Figura 1. Esquema de *planner*. Fuente: (Russell, 2004)



Accede a los ejercicios de autoevaluación a través del aula virtual

7.3. Tipos de problemas de planificación



Accede al vídeo «Tipos de problemas de planificación» a través del aula virtual

Comparación entre problemas de búsqueda y de planificación

La principal diferencia entre búsqueda y planificación es la representación de estados. En la **búsqueda**, los estados se representan como una sola entidad (que puede ser un objeto bastante complejo, pero su estructura interna no es utilizada por el algoritmo de búsqueda). En la **planificación**, los estados tienen representaciones estructuradas (colecciones de propiedades) que son utilizadas por el algoritmo de planificación.

Si bien en el contexto de la planificación nos encontramos con la necesidad de analizar diversas opciones dentro de la exploración de los estados, en los problemas

de búsqueda deberemos determinar varios pasos por medio de la selección de una secuencia de acciones, con el objetivo de alcanzar un estado del mundo que sea lo mejor posible para el agente. Este proceso de **seleccionar acciones** se realiza por medio de búsqueda y lo único que debemos tener en cuenta es que el resultado de una acción debe ser predecible.

En resumen, la **planificación** es el proceso de calcular varios pasos de un procedimiento de resolución de problemas antes de ejecutar cualquiera de ellos. Este problema se puede resolver mediante la **búsqueda**.

Planificación clásica

La planificación es una tarea compleja y por esta razón la mayoría de los planificadores trabajan sobre un modelo restringido del entorno (**planificación clásica**). Este modelo es determinista, estático y totalmente observable.

Concretamente fija las siguientes asunciones:

1. Observable: considera que el entorno es totalmente observable y no existe información desconocida para el planificador.
2. Estático: considera que el entorno solo cambia al momento de aplicar una acción en el estado inicial completamente conocido. Es decir, no existen influencias externas que afecten el entorno. Además, considera que los objetivos son conocidos y no cambian durante la planificación.
3. Determinista: considera que los efectos de aplicar una acción en un estado son totalmente predecibles, conduciendo de forma determinista a un único nuevo estado.
4. Proposicional: considera un enfoque proposicional para representar el estado del entorno. Son variables del modelo de planificación que pertenecen al dominio lógico y toman valores de cierto o falso.

5. Duración de las acciones: considera que todas las acciones del modelo de planificación tienen la misma duración (1) y se ejecutan de manera atómica e instantánea.
6. Offline: considera que la tarea de planificación para construir un plan completo que satisface el estado meta se desarrolla antes de la ejecución de cualquier acción del plan en el estado del mundo.

Aun considerando estas simplificaciones, el problema a resolver por un planificador es PSPACE-Completo. Así bien, el empleo de técnicas clásicas de búsqueda o demostración de teoremas no resulta factible del todo.

Planificación con incertidumbre

Como un caso opuesto a la planificación clásica, encontramos la planificación con incertidumbre (Russell, 2004). Que se acerca más a lo que conocemos en el mundo real.

Concretamente, en una planificación con incertidumbre se fijan como importantes las siguientes asunciones:

1. Parcialmente Observable: considera que el entorno es parcialmente observable, por lo que puede existir información desconocida para el planificador.
2. Dinámico: considera que el entorno puede cambiar en cualquier momento. Es decir, existen influencias externas que afecten el entorno. Esto aplica también para los estados metas.
3. No Determinista: considera que los efectos de aplicar una acción en un estado no son predecibles, conduciendo de forma no determinista a uno o varios nuevos estados.

Por lo tanto, un agente puede no conocer todo el estado del mundo, entorno parcialmente observable. O tener acciones cuyos efectos son no deterministas. En

en estas situaciones, se requiere una planificación con incertidumbre donde el uso de la probabilidad o de los modelos ocultos de markov toman bastante valor.

Clasificación de métodos y problemas

Atendiendo a su modo de búsqueda en el espacio de estados para obtener el resultado u objetivo deseado, los sistemas de planificación pueden:

- ▶ Operar hacia delante (desde el estado inicial), buscando aquellas acciones que se pueden aplicar desde el estado en el que nos encontramos y realizando una búsqueda no informada para conseguir obtener las acciones precisas que alcancen el objetivo.
- ▶ Aplicar las acciones hacia atrás (Guzman, 2015) (desde el objetivo a obtener), con el fin de encontrar qué acciones pueden producir el resultado que deseo en último lugar e ir explorando desde la meta al inicio el espacio de estados.

Estos enfoques se emplean principalmente cuando los valores que definen el estado meta son independientes entre sí.

Cuando nos encontramos en escenarios en los que no podemos garantizar la independencia entre los elementos que componen un estado deseado, suele ser buena idea utilizar algoritmos que buscan dentro del espacio de los posibles planes sin intentar obtener de una sola vez una secuencia completa de acciones totalmente ordenada. Estos mecanismos se describen como **planificación de orden parcial (POP)**. Trabajan hacia atrás, desde el objetivo, y añaden acciones para planificar cómo alcanzar cada subobjetivo.

Otra alternativa, en caso de que los problemas sean demasiado largos, es subdividir el problema de forma jerárquica, ya sea mediante extensiones de los operadores de planificación general o mediante extensiones del lenguaje y modificación de la compartmentalización del problema, combinando de manera efectiva las soluciones

parciales en las que hay elementos comunes. Esta técnica se conoce como **redes de tareas jerárquicas**.

Planificación generalizada

«Un plan generalizado brinda una solución única para un conjunto de problemas de planificación. En muchos dominios, los planes generalizados solo pueden hacer cálculos en ciertas funciones de alto nivel, por ejemplo, funciones capaces de capturar conceptos clave o variables que distinguen cuidadosamente entre diferentes estados posibles. En la planificación generalizada, estas funciones de alto nivel se codificarán a mano». (Lotinac, 2016)



Accede a los ejercicios de autoevaluación a través del aula virtual

7.4. Planificadores de orden total y de orden parcial



Accede al vídeo «Planificadores de orden total y de orden parcial» a través del aula virtual

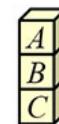
La planificación de orden total tiene principalmente una limitación cuando los objetivos que debemos alcanzar en un problema interactúan entre sí. En este caso, el método GPS puede deshacer un subobjetivo mientras intenta satisfacer a otro (Ghallab, 2004). La **anomalía de Sussman** describe esta situación perfectamente.

Anomalía de Sussman:

Estado inicial:



Estado meta:



Plan óptimo:

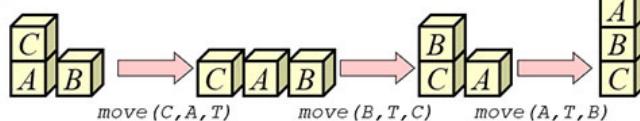


Figura 2. Anomalía de Sussman.

La figura 2, muestra un ejemplo de la Anomalía de Sussman en el dominio del mundo de bloques. En ella tenemos la posibilidad de completar un problema por medio de un plan, pero nos encontramos con la imposibilidad de encontrar un plan global óptimo a partir de concatenar subplanes parciales.

Como veremos, en muchos casos, las heurísticas de planificación global (como STRIPS) implican que las submetas son independientes, pero para conseguir alguna de ellas es necesario destruir otras metas ya conseguidas, degenerando en un bucle en el que los subobjetivos se anulen continuamente.

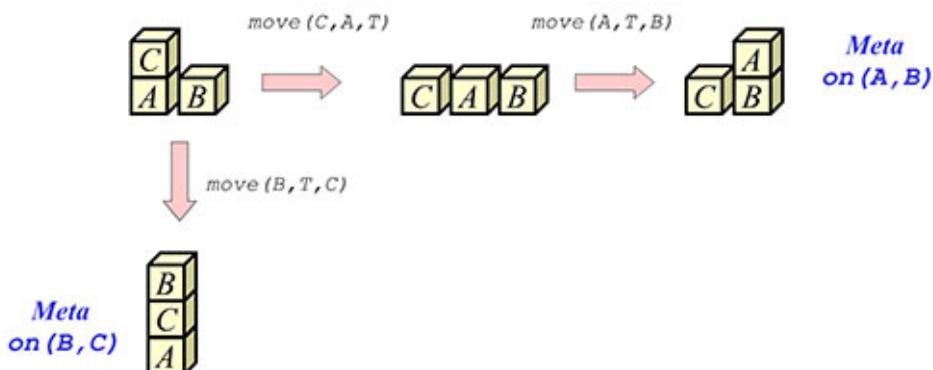


Figura 3. Un posible estado puede alcanzar dos subobjetivos por medio de acciones que se anulan continuamente.

Para resolver este tipo de problemas, veremos los **planificadores de orden parcial**. En ellos, en lugar de buscar en el espacio de estados u objetivos, buscamos en un

espacio de planes no completamente especificados (planes parcialmente ordenados).

Así, definiremos un **plan parcialmente ordenado** como un plan en el que solo se especifican algunas de las precedencias entre sus acciones y buscaremos los distintos planes que pueden concatenar acciones, de tal modo que podamos satisfacer las restricciones de orden entre las acciones de un plan (Russell, 2004).

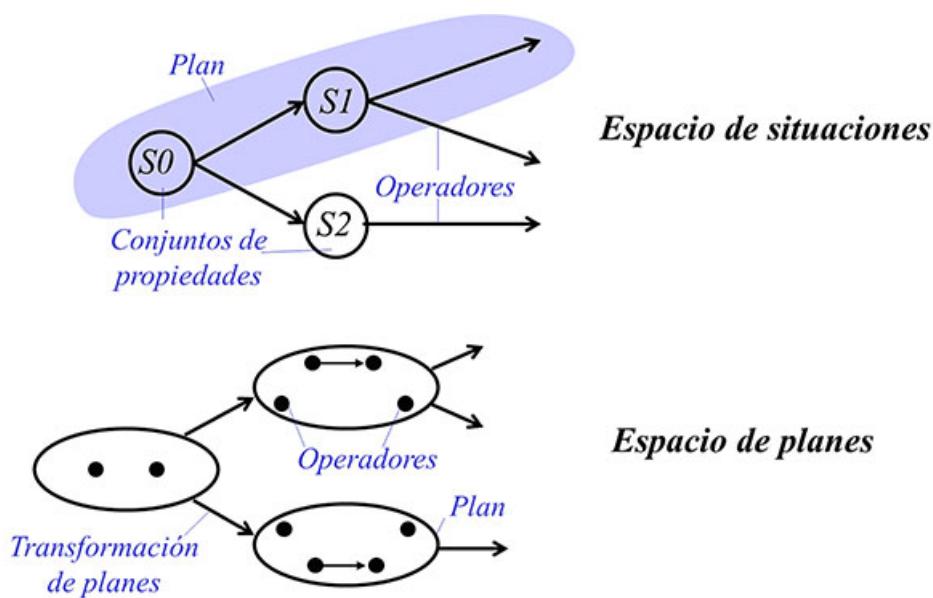


Figura 4. Búsqueda en el espacio de planes, idea inicial del POP.

Planificadores de orden parcial

Para resolver aquellos problemas en los que la consecución de una submeta conlleve, de modo endémico, una destrucción de otras metas conseguidas anteriormente y para solucionar problemas de modo general, disponemos del mecanismo basado en la búsqueda en el espacio de planes, en contraposición a la búsqueda en el espacio de estados en el que se basa STRIPS.

Así, buscaremos en el espacio de planes no completamente especificado, es decir, en el espacio de **planes de orden parcial**.

Definiremos **plan parcialmente ordenado** (*partial order plan*, POP) como un plan en el que solo se especifican algunas de las precedencias entre sus acciones.

Componentes

Un POP está compuesto por los siguientes componentes:

- ▶ Nodos/acciones: forman los pasos del plan, seleccionados de entre los operadores del agente. Cada uno con sus precondiciones y efectos.
 - Con dos acciones especiales: inicio (sin precondiciones y con el efecto de crear las propiedades del estado inicial) y final (sin efectos y que tiene como precondiciones todas las propiedades que conforman el estado meta).
- ▶ Arcos: son las restricciones de orden parcial entre dos acciones, $A \prec B$



Figura 5. Arcos.

- ▶ Arcos/enlaces causales: especifican la consecución de un efecto por parte de una acción, que es precondición para otra acción a la que se enlazan.



Figura 6. Arcos/enlaces causales.

- Precondiciones abiertas: representan aquellas propiedades que son precondición de alguna acción presente en el plan y que todavía no se han enlazado por medio de ningún arco causal.

Se define un **plan parcial inicial** como aquel que tiene, únicamente, las acciones de inicio y final. Con la restricción implícita de orden $INICIO \prec FINAL$ y que contiene todas las precondiciones de final abiertas.

El objetivo es conseguir un plan parcial final (de entre los posibles que existan) que **no tenga conflictos** entre los enlaces causales, **sin ciclos** entre sus restricciones de orden y **sin precondiciones abiertas**.

Donde definimos que la acción C entra en conflicto con el enlace causal entre $A \xrightarrow{p} B$ (que produce la propiedad **p**) si C tiene el efecto de eliminar **p** y, según las restricciones de orden, C podría ir antes que B y después de A. $A \prec C \prec B$

Para la obtención de una secuencia de planes parciales que nos permita alcanzar el plan parcial debemos colocar las acciones una detrás de otra, sin contradecir ninguna restricción de orden que se deduzca del plan parcial. Para ello deberemos tener en cuenta que cualquier secuenciación de un plan parcial solución supone una solución al problema original.

En este contexto elaboramos la idea de que debemos diseñar un algoritmo para encontrar planes parciales finales. La idea principal es la creación de un POP comenzando por el plan inicial e ir aplicando transformaciones u operadores a los planes parciales, refinándolos.

En esencia, consiste en resolver las precondiciones abiertas evitando las amenazas. En cada instante habrá varias alternativas para ir refinando el plan y no todas ellas conducen hacia el plan parcial final. Por tanto, el problema consiste en encontrar la secuencia de refinamiento que, partiendo del plan parcial inicial, llegue a un plan parcial solución (es decir, sin ciclos, sin amenazas y sin precondiciones abiertas).

Vuelve a ser una búsqueda en un espacio de estados, pero ahora los estados son los planes parciales y los operadores son los refinamientos sucesivos de los planes parciales.

Algoritmo

En consecuencia, un **algoritmo POP** simplemente emplea un algoritmo de búsqueda (en profundidad, por ejemplo) en el espacio de los planes parciales para encontrar el plan parcial final.

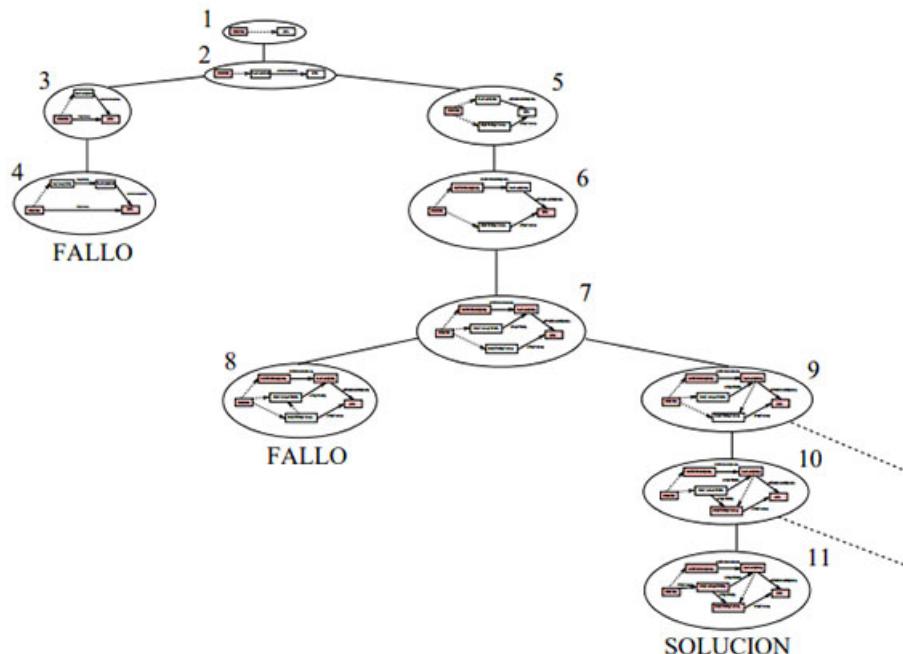


Figura 7. Búsqueda en profundidad en el espacio de planes.

Como todo algoritmo de búsqueda, contamos con dos componentes principales. Por un lado, los **estados**, que en este caso es un plan parcial creado hasta ese instante, con el conjunto de acciones enlazadas y precondiciones abiertas. Y, por otro lado, los **operadores o acciones** que se pueden crear desde cada estado.

En este caso, dispondremos de dos tipos de acciones en cada estado.

- ▶ Acciones de resolución de precondiciones abiertas: dada una precondición p abierta en una acción B dentro del plan del estado actual, por cada acción A que tiene como efecto producir p, podemos obtener un plan refinado aplicando uno de los pasos siguientes (siempre que no se produzcan ciclos):

- Establecimiento simple: si la acción A ya estaba en el plan, se añade la restricción $A \prec B$ y el enlace causal $A \xrightarrow{p} B$.
 - Añadir una nueva acción al plan: si la acción A no está en el plan actual del estado, se añade junto a las restricciones $A \prec B$, $INICIO \prec A$ y $A \prec FIN$ y el enlace causal $A \xrightarrow{p} B$.
- Resolución de conflictos dentro del plan actual: supuesto el caso de que exista un conflicto entre el enlace causal $A \xrightarrow{p} B$ y la acción C, se puede crear un plan sucesor (estado sucesor) aplicando alguno de los siguientes pasos, siempre que no creemos ciclos:
- Promoción de la acción B: añadiendo la restricción de orden $B \prec C$.
 - Degradar la acción A: añadiendo la restricción $C \prec A$.

Al realizar el proceso de creación del POP por medio de una técnica de búsqueda, tenemos que crear el árbol de búsqueda en el que:

- Los nodos son planes parciales.
- Creamos tantas ramas como las necesarias originadas por las acciones anteriores de resolución de precondiciones abiertas y conflictos.

No es necesario expandir todas las precondiciones en un nivel, basta con ir haciéndolo de modo ordenado a lo largo de la exploración de los siguientes niveles (debemos ser cuidadosos con este proceso, esta exploración no influye en la completitud del plan, pero sí puede ser muy determinante en la eficiencia).

Algunos mecanismos de búsqueda en árboles, como el de profundidad, pueden caer en iteraciones infinitas. Para evitarlo es necesario contar con una cota de profundidad de exploración en estos casos. Asimismo, existen heurísticas que permiten acelerar el proceso de obtención de planes.



Accede a los ejercicios de autoevaluación a través del aula virtual

7.5. Referencias bibliográficas

Ghallab, M. N. (2004). *Automated Planning: theory and practice*. San Francisco: Elsevier.

Guzman, C. C. (2015). Reactive execution for solving plan failures in planning control applications. *Integrated Computer-Aided Engineering*, 22(4), 343-360.

Lotinac, D. S.-A. (2016). Automatic Generation of High-Level State Features for Generalized Planning. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. Kambhampati, S.

Russell, S. y. (2004). *Inteligencia Artificial: Un Enfoque Moderno*. Madrid: Pearson Educación.

Vázquez-Salceda, J. (2011). *Agentes planificadores*. Obtenido de Agentes planificadores: <http://www.lsi.upc.edu/~jvazquez/teaching/iag/transpas/4-PL1-IntroPlanificaci%C3%B3n.pdf>

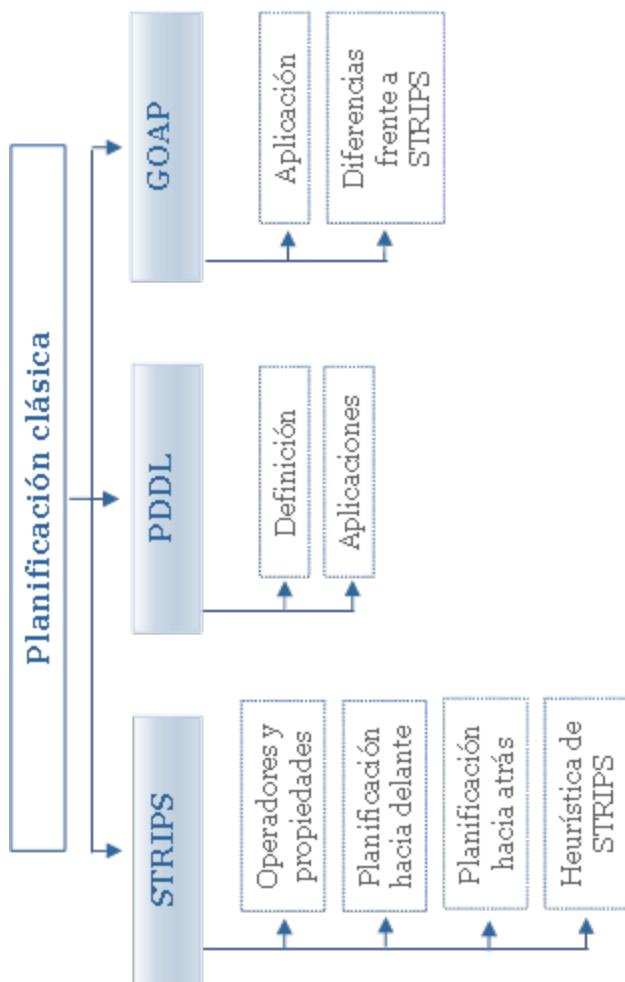


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Sistemas basados en STRIP

Esquema. Tema 8



Ideas clave. Tema 8

8.1. ¿Cómo estudiar este tema?

En este tema trataremos de definir los planificadores tradicionales basados en la descripción de STRIPS (Fikes y Nilsson, 1971) y su evolución en el lenguaje de definición de planes PDDL (McDermott, Ghallab, Howe, Knoblock, Ram y Veloso, 1998).

Por último, presentaremos una adaptación de los sistemas STRIPS pensada para sistemas con reacción en tiempo real, para poder crear sistemas que puedan reaccionar a entornos cambiantes de modo eficiente. El sistema GOAP, ideado por Jeff Orkin, adapta los estándares de STRIPS para estos entornos.



Accede a los ejercicios de autoevaluación a través del aula virtual

8.2. STRIPS



Accede al vídeo «STRIPS y PDDL (I)» a través del aula virtual

Definición y Base

Partiremos de una información que conoce *a priori* un agente y tenemos una representación de los conocimientos basada en la idea de los mecanismos de búsqueda. Por ello contaremos con:

- ▶ Un estado inicial: símbolo (s_0).

- ▶ Un conjunto de estados sucesores producidos por una función $\text{expandir}(s, a)$ que a partir de un estado s y una acción a genera un estado sucesor s' . Así, por ejemplo, a partir de un estado s_7 se pueden generar los estados sucesores ($s_7 \mapsto \{s_8, s_{10}, s_{12}, s_{27}, s_{112}\}$).
- ▶ Un estado meta G , que la función $\text{meta}(s)$ evalúa como final o no. Por ejemplo, dado el estado s_{112} , la función meta evalúa si G está contenido en s_{112} , en cuyo caso diría que s_{112} es un estado meta ($s_{112} \mapsto \text{true}$).
- ▶ Una función de coste c asociado a la aplicación de una acción a en un estado s . Por ejemplo, aplicar la acción a en el estado s_7 tendría un coste, $c(s_7, a) \mapsto 5$.
- ▶ Una o varias funciones heurísticas: $h^*(s_7 \mapsto 235)$.

Con esta información tenemos una abstracción completa de las características de los estados, pero en esta abstracción tenemos una pérdida de información acerca de la conformación de los estados (por ejemplo: «¿bloque C encima de bloque A?», «¿el agente se encuentra en Praga?»)

Esto nos genera varios problemas potenciales que derivan en un aumento de la complejidad para poder representar la información y el conocimiento (por ejemplo, para definir la función expandir), y hace más difícil el mantenimiento del conocimiento. Es por esto por lo que buscamos otros mecanismos para poder trabajar con la información en estos agentes inteligentes. Para ello, tendremos en cuenta que los estados presentan una «estructura» y definiremos un estado no por su nombre, sino por el conjunto de sus **propiedades o proposiciones**.

Elementos básicos de STRIPS

A principios de los años 70, se creó el lenguaje estandarizado para formalizar un problema de planificación (Fikes, 1971). STRIPS fue creado para describir los componentes necesarios para que un agente planificador pudiera resolver problemas complejos en los que la estructura de los estados era determinante para entender qué proceso sería necesario llevar a cabo para resolver el problema de modo racional.

Este estándar sigue muy vigente en la definición de paradigmas de planificación actual con la creación de nuevos modelos como son PDDL (Planning Domain Definition Language, de sus siglas en Ingles) y/o GOAP (*Goal-Oriented Action Planning, de sus siglas en Ingles*).

Para esta sección emplearemos el problema de apilado de bloques, ver figura 1, (dominio del mundo de los bloques), en el que, desde una configuración de partida de un conjunto de bloques, deseamos crear una configuración final de los mismos por medio de operaciones de apilado y desapilado.

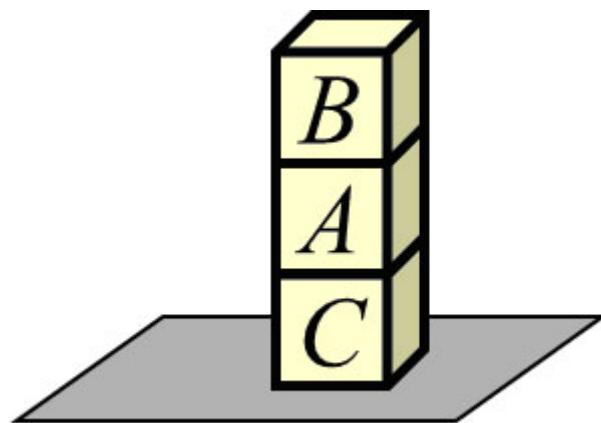


Figura 1. Bloques apilados.

Existen dos componentes básicos en una definición de problema de STRIPS:

- ▶ Proposiciones
- ▶ Operadores

Proposiciones

Para describir un estado, tendremos en cuenta el conjunto de proposiciones relevantes que lo caracteriza. Estas proposiciones, inicialmente, son expresadas por medio de valores *booleanos*, es decir, de existencia o no existencia en el estado actual.

Por ejemplo, para el problema del mundo de los bloques, con tres bloques tendríamos las siguientes proposiciones relevantes para caracterizar estados:



Figura 2. Proposiciones para el mundo de los bloques.

Para el caso del ejemplo de la figura 1, el estado representado en la imagen estaría formado por las siguientes proposiciones:

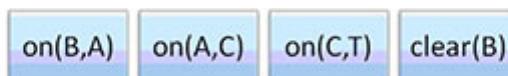


Figura 3. Ejemplo de proposiciones de un apilado.

Es importante tener en cuenta que no todas las combinaciones de proposiciones son **consistentes**. Por ejemplo, {on(C,B), on(B,C)} no representa un estado válido porque es inconsistente que B este sobre C y al mismo tiempo C este sobre B. A este tipo de situaciones se les conoce como **mutex**.

Operadores

Expresan las acciones que el agente contempla en el problema y por las cuales desea resolver el estado actual y transformar el entorno para alcanzar la meta o estado final.

Se definen por una **signatura del tipo**:

<nombre>(<PC>,<A>,>E>)

PC	Precondición	Conjunto de proposiciones que deben estar presentes en un estado para poder aplicar dicho operador.
A	Lista de añadir	Conjunto de proposiciones que se añadirán al estado tras aplicar el operador.
E	Lista de eliminar	Son aquellas proposiciones que se quitarán del estado tras la aplicación del operador. Debería ser un subconjunto de PC.

Tabla 1. Componentes de un operador de STRIPS.

move (A, B, C)

PC: On(A, B), Clear(A), Clear(C)
E: On(A, B), Clear(C)
A: On(A, C), Clear(B)

move (B, A, T)

PC: On(B, A), Clear(B)
E: On(B, A)
A: On(B, T), Clear(A)

Figura 4. Ejemplos de operadores en el mundo de los bloques.

En el esquema definido por STRIPS, los operadores se ejecutan teniendo en cuenta que un operador sea aplicable. Diremos que un operador es aplicable a un estado si las precondiciones del operador son un subconjunto del estado S : $PC(op) \subseteq S$.

Cuando aplicamos un operador a un estado S , produciremos un nuevo estado S' derivado en el cual deberemos eliminar las proposiciones descritas en E y añadiremos las presentes en A .

$$S' \leftarrow S - E(op) \cup A(op)$$

Y asumiremos que, todas aquellas proposiciones que no se especifican, no se dan y, aquellas que no se modifican, se quedan como estaban. Esto se define como la **asunción de mundo cerrado**.

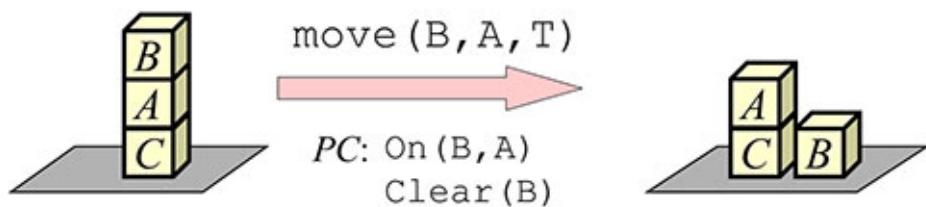


Figura 5. Esquema de ejecución de un operador STRIPS.

Planificación hacia adelante

La primera idea que nos surge para resolver un problema de planificación expresado por medio de estos operadores y propiedades STRIPS es crear una planificación que explore el espacio de búsqueda de estados, comenzando desde el estado inicial, y aplicando operadores como se hace en los problemas de búsqueda normales.

De este modo, y asumiendo el proceso de exploración y búsqueda que se basa en la existencia de un conjunto de acciones candidatas en un estado determinado, crearemos dicha función de expansión por medio de la identificación de todos aquellos operadores que sean aplicables en el estado. Una vez identificamos todos los operadores aplicamos la función de expansión (S, a) para todo a que pertenezcan a los operadores. Generando de esta manera nuevos estados S' que serán los nodos hijos del estado S .

Con esta idea, aplicamos algoritmos como los de búsqueda en amplitud, profundidad e incluso búsquedas heurísticas de A*.

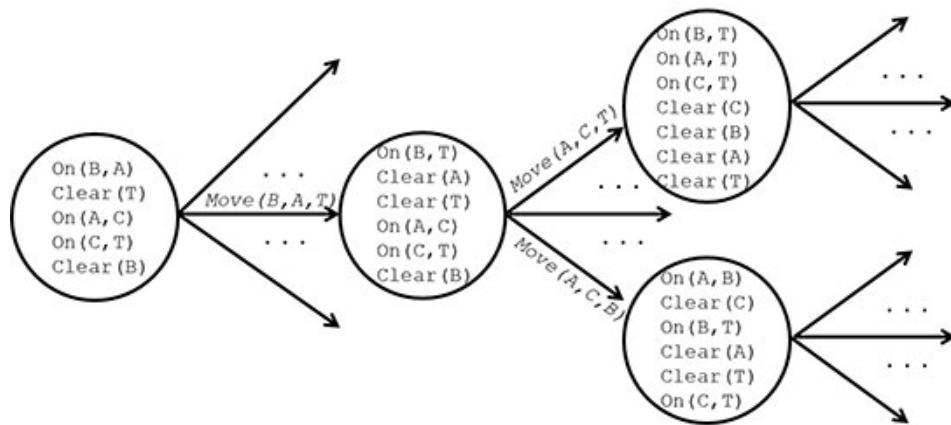


Figura 6. Ejemplo de planificación hacia adelante.

Pero rápidamente nos damos cuenta de que el factor de ramificación es muy elevado y supone un problema porque el estado inicial se describe por todas las proposiciones que podrían ser relevantes (por ejemplo, las posiciones de diez bloques) y la descripción del estado meta suele ser parcial, es decir, solo contiene lo que realmente importa (por ejemplo, las posiciones de tres bloques).

Planificación regresiva o hacia atrás

Dada la problemática de la planificación hacia adelante, surge la idea de aplicar operadores en orden inverso; es decir, para poder reducir el conjunto de propiedades no satisfechas de estado meta, aplicamos operadores de tal modo que podamos conseguir encontrar la secuencia de operadores que nos regresan desde el estado meta hasta el estado inicial.

Regresar un estado S' por un operador Op consiste en encontrar el conjunto de proposiciones menos restrictivo (más débil) que permitiría aplicar el operador al estado S . Para alcanzar dicho conjunto, lo que hacemos es eliminar del estado la lista de A del operador y añadir aquellos elementos que no tenemos en PC.

$$S \leftarrow S' - A(Op) \cup PC(Op)$$

Pero para conseguir la aplicabilidad de dicho operador, ningún elemento de S' es eliminado (*e. d.*: $S' \cap E(Op) = \emptyset$).

Hay que tener cuidado porque en este proceso de aplicación no todos los estados que se obtienen de la regresión son **consistentes**.

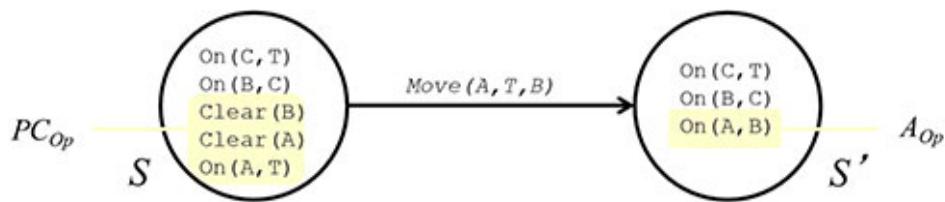


Figura 7. Regresión de un estado STRIPS de $S' \rightarrow S$.

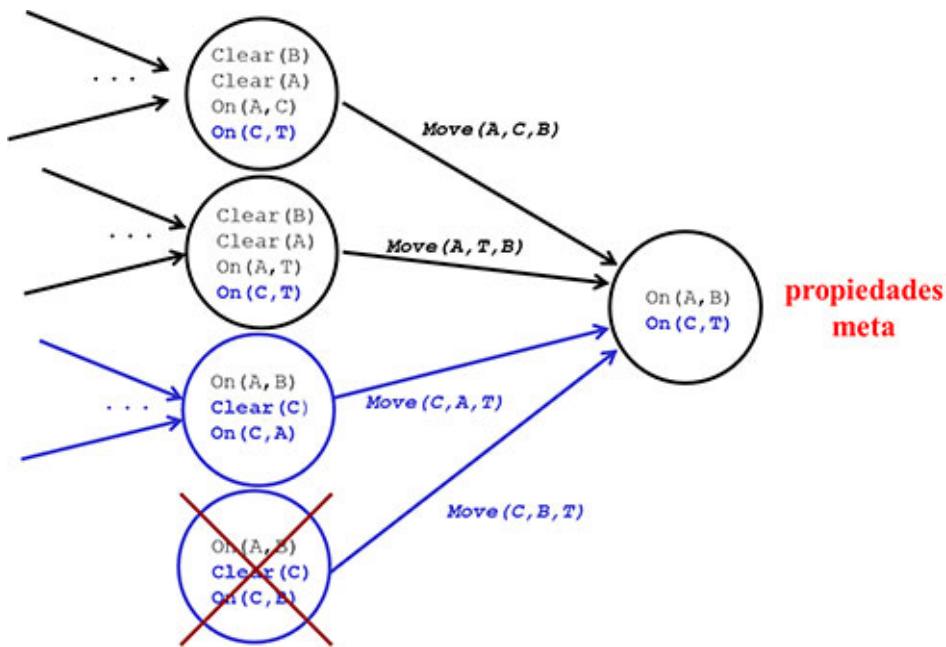


Figura 8. Ejemplo de regresión en el mundo de los bloques. Eliminación de los estados inconsistentes.

Heurística STRIPS

Aun contando con un factor de ramificación menor y un crecimiento de la exploración guiado por las metas, debemos combatir la complejidad de la planificación. Al igual

que en las búsquedas, la idea es buscar un conocimiento *a priori* que nos mejore la complejidad, es decir, una heurística.

La **heurística STRIPS** consiste en encontrar los planes parciales para alcanzar cada una de las proposiciones que se encuentran en el estado objetivo o meta. Para construir el plan final, concatenaremos todos los subplanes parciales que tengamos que construir.

Podemos definir la **planificación de tipo STRIPS** de la siguiente manera.

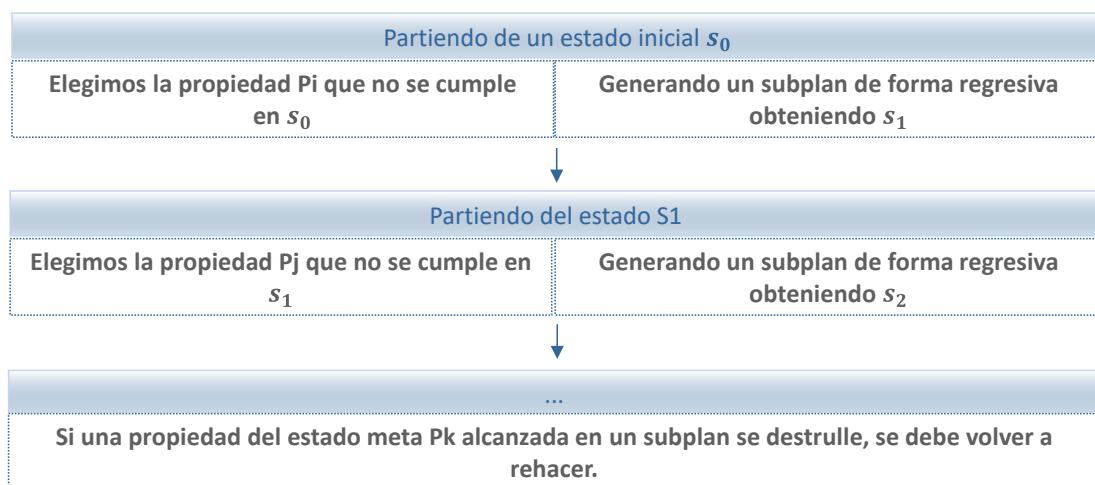


Figura 9. Planificación de tipo STRIPS.

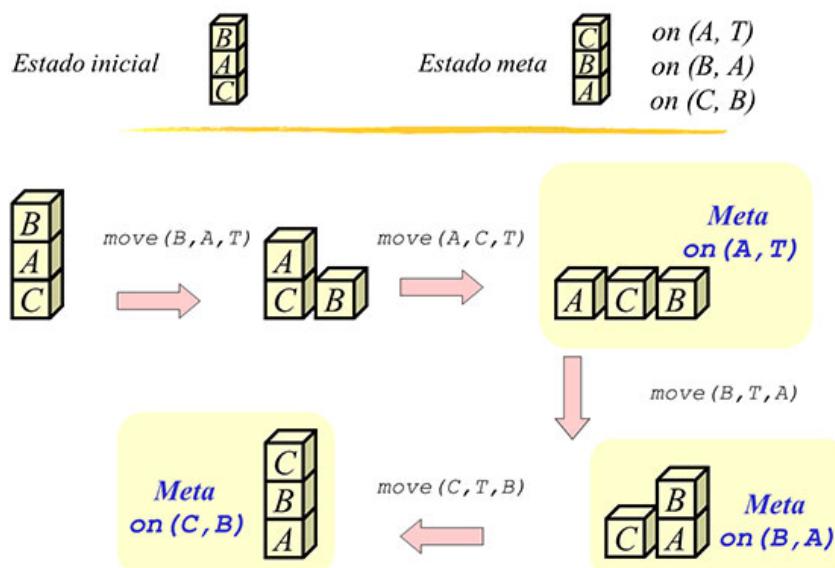


Figura 10. Uso de la heurística de STRIPS.

Pero pueden surgir problemas por el orden de exploración de las soluciones:

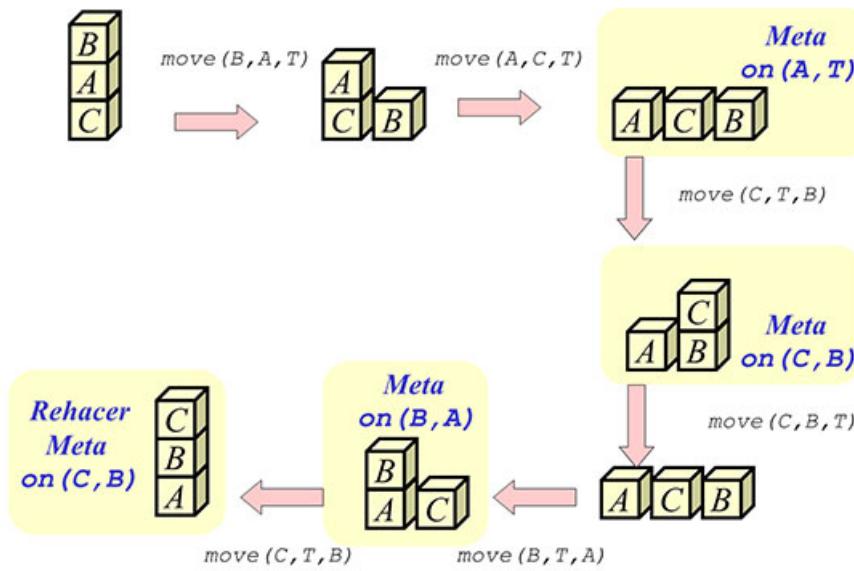


Figura 11. El orden de exploración puede suponer problemas.

El orden de elección de metas y/u operadores influye en los planes obtenidos.

```

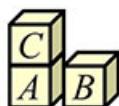
STRIPS(S,Metas,Plan) devuelve (SMeta, PlanMeta) ó false
Mientras que Metas ⊈ S Hacer % Mientras haya Metas no satisfechas en S
    (1) Elegir M∈Metas tal que M⊈S % M no está satisfecha en S
    (2) Elegir operador Op tal que M∈AOp % Op puede alcanzar M
    (3) (S,Plan) ← STRIPS(S,PCOp,Plan) % Alcanzar precond. de Op
    (4) Si STRIPS devuelve false Entonces falla
    (5) S ← S - EOp ∪ AOp % aplicar el Op al nuevo estado S (resultado de (3))
    (6) Plan ← Plan + Op % añadir el Op al final de nuevo Plan (resultado de (3))
Fin
Devolver(S,Plan)

```

Figura 12. Algoritmo no determinista de la heurística STRIPS.

La heurística STRIPS es una heurística fuerte que implica que las metas son independientes, pero para conseguir alguna propiedad o proposición se puede estar destruyendo de modo cíclico otra proposición, quedando atrapado en un bucle de destrucción de proposiciones. De este modo, no se podría alcanzar un plan global óptimo a partir de los subplanes parciales debido a que, por diseño, la linealización de la heurística de STRIPS no permite intercalar acciones de dos submetas independientes. Para poder solucionarlo, deberíamos poder intercalar operadores de los distintos subplanes.

Estado inicial:



Estado meta:

On (A, B)
On (B, C)



Plan óptimo:

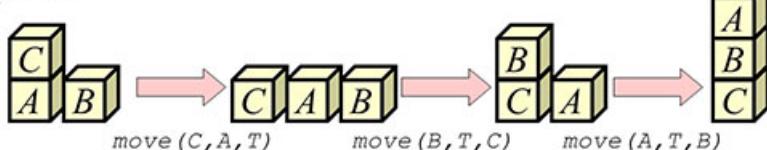


Figura 13. El plan óptimo para un caso del mundo de los cubos.

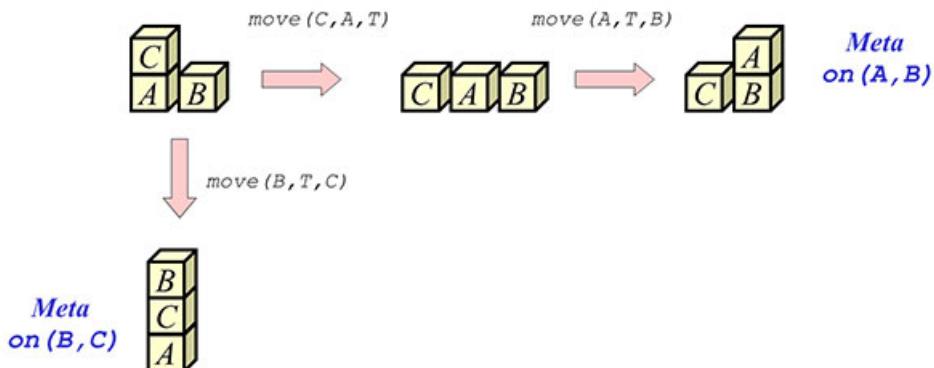


Figura 14. Anomalía de Sussman

Hay metas que suponen la destrucción de submetas obtenidas: anomalía de Sussman.



Accede a los ejercicios de autoevaluación a través del aula virtual

8.3. PDDL



Accede al vídeo «STRIPS y PDDL (II)» a través del aula virtual

El *Planning Domain Description Language* (PDDL) fue creado por Drew McDermott y su equipo. Se basaba en los estándares especificados de STRIPS y ADL (McDermott, 1998). El objetivo inicial era crear un lenguaje común y estándar de creación de planes para emplear como *benchmark* entre planificadores, y así poder comparar agentes de planificación en competiciones o estudios. En la actualidad, es un estándar que presenta varias versiones, desde la 1.0 a la 3.1, cada una de ellas con diferentes niveles de expresividad (si bien es cierto que actualmente no hay ningún planificador que soporte la versión 3.1 completa).

Se basa en una descripción de los componentes de un planificador en dos conjuntos: uno de **definición del dominio** y otro de **definición del problema**.

```
(define (domain DOMAIN_NAME)
  (:requirements [:strips] [:equality] [:typing] [:adl])
  (:predicates
    (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
    (PREDICATE_2_NAME [?A1 ?A2 ... ?AN]) ...)
  (:action ACTION_1_NAME
    (:parameters (?P1 ?P2 ... ?PN))
    (:precondition PRECOND_FORMULA)
    (:effect EFFECT_FORMULA) )
  (:action ACTION_2_NAME ...))
  ...)
```

Figura 15. Definición del dominio del planificador.

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA) )
```

Figura 16. Definición del problema.

En general, en las especificaciones de PDDL se representa el conjunto de objetos del mundo (**constantes**, en mayúsculas) y se usan **variables** para representar cualquier

objeto del universo. Para expresar proposiciones de los objetos se emplean **símbolos de predicado** y los operadores los representamos por medio de **símbolos de acciones**.

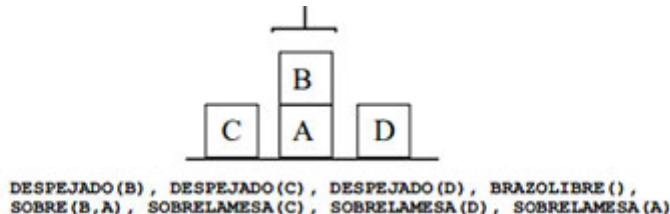
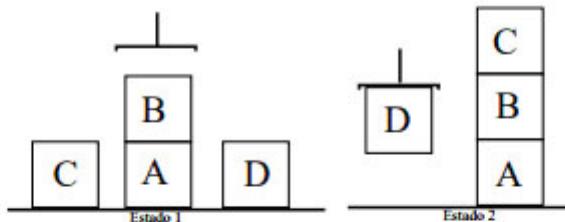


Figura 17. Ejemplo de descripción de proposiciones. Fuente: (Graciani, 2016)

A la hora de construir un problema completo, usamos elementos atómicos formar **literales** que pueden ser afirmativos o negativos. En nuestra definición de un problema pueden existir átomos y literales o fluents. Los fluents son átomos instanciados con objetos del estado del mundo. Por ejemplo, SOBRE (B, A) es un fluent que indica que el objeto B se encuentra sobre el objeto A. En toda la definición, el elemento clave son los **estados**, que definiremos por medio de conjuntos de fluents.



- **Ejemplos de objetivos:**

- SOBRE (B, A), SOBRELAMESA (A), -SOBRE (C, B) es satisfecho por el estado 1 y por el estado 2
- SOBRE (x, A), DESPEJADO (x), BRAZOLIBRE () es satisfecho por el estado 1 pero no por el estado 2
- SOBRE (x, A), SOBRE (y, x) no es satisfecho por el estado 1 pero sí por el estado 2
- El objetivo SOBRE (x, A), -SOBRE (C, x) es satisfecho por el estado 1 pero no por el estado 2

Figura 18. Ejemplo de objetivos en PDDL. Fuente: (Graciani, 2016).

Al igual que en la descripción de STRIPS, la hipótesis del mundo cerrado hace que los fluents que no sean nombrados explícitamente en una descripción sean considerados como falsos.



Accede a los ejercicios de autoevaluación a través del aula virtual

8.4. GOAP



Accede al vídeo «GOAP» a través del aula virtual

El sistema *Goal - Oriented Action Planning (GOAP)* fue creado por Jeff Orkin en 2004 (*Symbolic Representation of Game World State: Toward real-time planning in games*) con el fin de adaptar un sistema planificador como STRIPS a un entorno de simulación juegos. Fue implementado por primera vez en el videojuego F.E.A.R., de Monolith (2005) (Orkin, 2006).

Puedes ver más sobre F.E.A.R en el siguiente enlace:

<https://www.3driegos.com/juegos/analisis/750/0/f-e-a-r/>

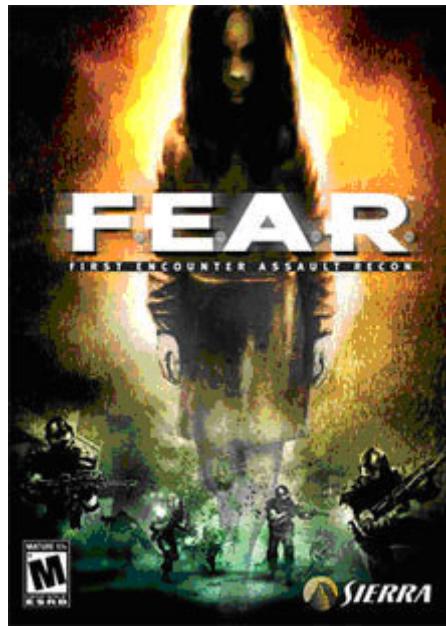


Figura 19. Portada del videojuego F.E.A.R., de Monolith (2005).

Fuente: <https://www.3djuegos.com/juegos/pc/750/f-e-a-r/>

Orkin se basó en la arquitectura de agentes C₄ del MIT y extendió ideas del planificador de STRIPS. Posee cuatro diferencias con STRIPS:

- ▶ Establece costes a las acciones con el fin de poder asignar prioridad a unas acciones frente a otras, que es una consideración necesaria para el diseño de agentes en videojuegos.
- ▶ Elimina las listas de añadir y eliminar objetos y las convierte en una única lista de modificaciones del estado en la que las proposiciones pueden ser modificadas de manera más flexible que por simple operación *booleana*.
- ▶ Añade precondiciones procedurales que permiten mayor flexibilidad a la hora de expresar condiciones que se deban dar en el entorno para poder aplicar un operador.
- ▶ Añade efectos procedurales con la misma filosofía de poder modificar el entorno con mayor flexibilidad.

La primera consideración es que representa las proposiciones de un estado dentro de un vector *multitipado*, en lugar de considerar las proposiciones como valores *booleanos* aislados.



Ejemplo:

(AtLocation, Wearing) = (Home, Tie)

Figura 20. Ejemplo de par estado valor que representa una propiedad GOAP.

El resto de elementos de definición del dominio del problema y del entorno siguen las especificaciones de STRIPS. Contamos con precondiciones necesarias en una acción (operador) que produce cambios en las proposiciones del entorno para alcanzar una meta.

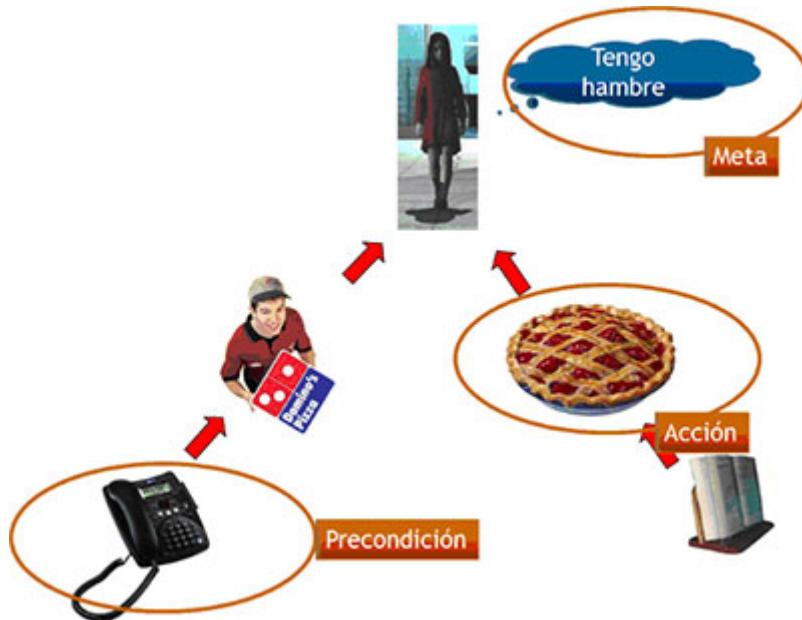


Figura 21. Ejemplo de problema análogo en STRIPS y GOAP.

Fuente: adaptada de Orkin (2006).

La idea es tener varios modelos de agente que puedan emplear el mismo motor de planificación con solo tener que definir distintos conjuntos de acciones, lo cual es de mucha importancia en entornos como el de los videojuegos, donde debemos diseñar varios tipos de agentes que resuelvan el problema con distintas capacidades.

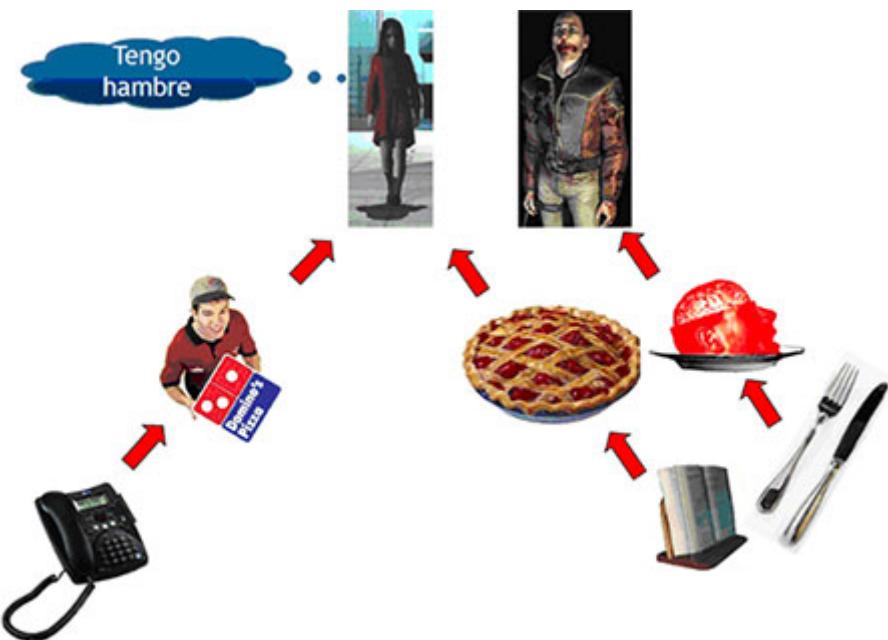


Figura 22. Varios agentes con el mismo problema, pero varios conjuntos de acciones distintas.

Fuente: adaptada de Orkin (2006).

En tiempo de diseño, en estos entornos es importante poder tener en cuenta distintas preferencias que caractericen a los personajes de un modo sencillo. Para ello, en GOAP se permite precisar un coste específico de cada acción y, en el proceso de búsqueda de planes, se puede aplicar un algoritmo como A* para hacer una búsqueda de costes mínimos en la consecución de un plan.

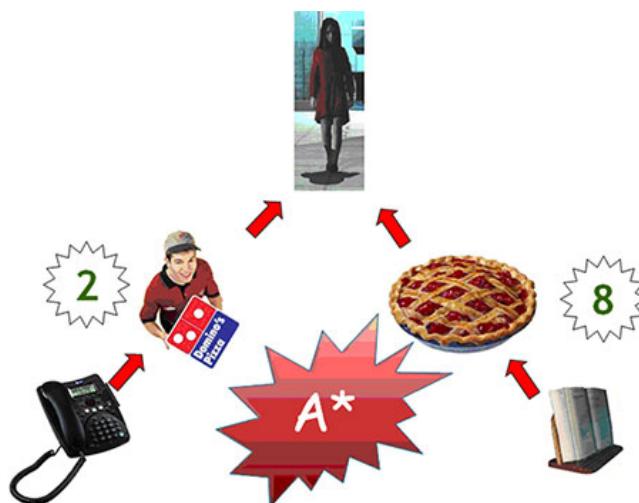


Figura 23. Aplicando distintos costes a las acciones en GOAP.

Fuente: adaptada de Orkin (2006).

Para resolver el problema de las cláusulas de mundo cerrado y permitir mecanismos de aceleración del cómputo dentro del motor, se suprimen las listas de añadir y eliminar y se crea una única lista de modificaciones.

State: (phone#, recipe, hungry?)



Action

Preconditions: ( , .. , ..)

Effects:

Delete List: Hungry(YES)

Add List: Hungry(NO)

Figura 24. Ejemplo de operador de STRIPS.

Fuente: adaptada de Orkin (2006).

State: [phone#, recipe, hungry?]



Action

Preconditions: [ , .. , ..]

Effects: [.. , .. , NO]

Figura 25. Ejemplo de operador de GOAP.

Fuente: adaptada de Orkin (2006).

Y un vector de proposiciones *multitipado* para almacenar los valores que alcanzarán las variables del estado tras la aplicación de un operador.

[-- , -- , --]	4-byte values
TargetDead	[bool]
WeaponLoaded	[bool]
OnVehicleType	[enum]
AtNode	[HANDLE]
- or -	
AtNode	[variable*]

Figura 26. Posible vector de proposiciones para un sistema GOAP.

Entenderemos en GOAP que las acciones son clases y, como tales, representan las precondiciones como un *array* de variables de estado del mundo. Estas precondiciones serán como una función para permitir filtros adicionales. Esta función permite la ejecución de cualquier trozo de código que fuera necesario.

```

class Action
{
    // [ -- , -- , -- ]
    WORLD_STATE m_Preconditions;
    WORLD_STATE m_Effects;

    bool CheckPreconditions();
};

```

Figura 27. Acciones con precondiciones procedurales.

Fuente: Orkin (2006).

Sucede lo mismo con las acciones que se podrán programar como un efecto procedural. Esto nos permite delegar el proceso de cambio en el entorno y gestionar interrupciones que se puedan producir del plan en tiempo real.

```

class Action
{
    // [ -- , -- , -- ]
    WORLD_STATE m_Preconditions;
    WORLD_STATE m_Effects;
    bool CheckPreconditions();
    void ActivateAction();
}

```

Figura 28. Acciones procedurales.

Fuente: Orkin (2006).

```

void ActionFlee::ActivateAction()
{
    // ...
}

```

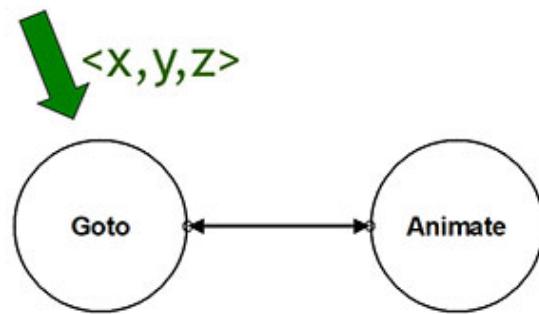


Figura 29. Una acción puede suponer varios ciclos de reloj, por lo que no son atómicas.



Accede a los ejercicios de autoevaluación a través del aula virtual

8.5. Referencias bibliográficas

Fikes, R. y. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4), 189-208.

Graciani, C. y. (2016). Curso de inteligencia artificial. . *Curso de inteligencia artificial*. Sevilla: Universidad de Sevilla.

McDermott, D. G. (1998). PDDL—the planning domain definition language—version 1.2. . *Yale Center for Computational Vision and Control, Tech. Rep.* . CVC TR-98-003/DCS TR-1165.

Orkin, J. (2004). Symbolic Representation of Game World State: Toward real-time planning in games. *Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence*, 5, 26-30.

Orkin, J. (2006). Three states and a plan: the AI of FEAR. *Proceedings of the 2006 Game Developers Conference (GDC '06)*. San José, California.

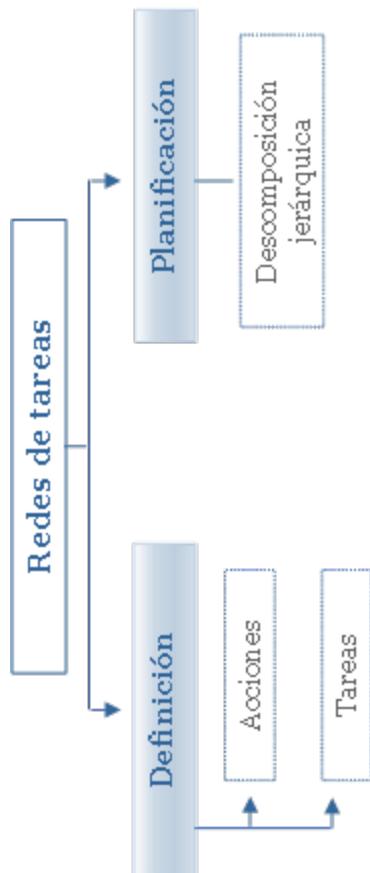


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Redes de tareas jerárquicas (HTN)

Esquema. Tema 9



Ideas clave. Tema 9

9.1. ¿Cómo estudiar este tema?

En este tema presentaremos una estrategia basada en el concepto de «divide y vencerás», por el que plantearemos un problema complejo y lo subdividiremos en tareas más simples. El sistema inteligente se encargará de combinar todas las subtareas disponibles del agente que sean relevantes para el problema, con el objetivo de alcanzar la meta que se le ha planteado.

En el tema trataremos:

- ▶ La definición de red de tareas jerárquicas, empleando ejemplos para poder entender cómo se diseñan.
- ▶ El mecanismo de planificación que emplean las tareas para construir un plan.

9.2. Definición



Accede al vídeo «Definición» a través del aula virtual

En la investigación de planificación de IA, la práctica de planificación (tal como se materializa en los sistemas de planificación implementados) tiende a ir muy por delante de las teorías que explican el comportamiento de esos sistemas. Existe un análisis muy reciente de las propiedades de los sistemas de planificación de orden total y parcial que utilizan operadores de planificación de estilo STRIPS. Los sistemas de planificación de estilo STRIPS, sin embargo, se desarrollaron hace más de veinte años y la mayor parte del trabajo práctico sobre sistemas de planificación de IA

durante los últimos quince años se ha basado en la descomposición jerárquica de la red de tareas (HTN).

La **planificación de la red jerárquica de tareas (HTA)** es una técnica de planificación de la IA que rompe con la tradición de la planificación (Ghallab, 2004). La idea básica detrás de esta técnica incluye:

- ▶ Una descripción de estado inicial,
- ▶ Una red de tareas inicial como un objetivo a alcanzar
- ▶ Y un conocimiento de dominio, que consiste en redes de tareas primitivas y compuestas.

Una **red de tareas** representa una jerarquía de tareas, cada una de las cuales puede ejecutarse, si la tarea es primitiva, o ser descompuesta en subtareas refinadas.

El **proceso de planificación** comienza descomponiendo la red de tarea inicial y continúa hasta que se descompongan todas las tareas compuestas, es decir, se encuentre una solución. La **solución** es un plan que equivale a un conjunto de tareas primitivas aplicables al estado mundial inicial.

Además de ser un factor que rompe la tradición, la planificación de la HTN también parece ser controvertida. La controversia radica en su requisito de conocimiento del dominio bien concebido y estructurado. Es probable que dicho conocimiento contenga abundante información y orientación sobre cómo resolver un problema de planificación, codificando así más la solución de lo que se previó para las técnicas de planificación clásicas. Este conocimiento estructurado y rico brinda una ventaja principal a los planificadores de HTN en términos de velocidad y escalabilidad cuando se aplica a problemas del mundo real y se compara con sus contrapartes en el mundo clásico (Georgievski, 2014).

La mayor contribución hacia la planificación de HTN ha surgido después de la propuesta del **planificador jerárquico simple (SHOP)** (Nau D. S., 1999) y sus

sucesores. SHOP es un planificador basado en HTN que muestra un rendimiento eficiente incluso en problemas complejos, pero a costa de proporcionar un conocimiento de dominio bien escrito y posiblemente algorítmico. La disputa sobre si proporcionar mucho conocimiento a un planificador debería considerarse una trampa en el mundo de la planificación de la inteligencia artificial sigue vigente (Nau D. S., 1999).



Accede a los ejercicios de autoevaluación a través del aula virtual

9.3. Planificación por medio de red de tareas



Accede al vídeo «Planificación por medio de red de tareas: Lenguajes de especificación HTN» a través del aula virtual

Idea principal: muchas tareas en la vida real ya tienen una estructura jerárquica incorporada. Por ejemplo: una tarea computacional, una misión militar o una tarea administrativa.

Sería una pérdida de tiempo construir planes desde los operadores individuales que forman las acciones propias del trabajo a realizar. Usar la jerarquía incorporada en el dominio ayuda a escapar de la explosión exponencial de las posibles combinaciones que tendrían las acciones atómicas.

Ejemplo de aplicación: la actividad de construir una casa consiste en obtener los permisos necesarios, encontrar un constructor, construir el exterior/interior, etc. En el enfoque de las HTN, se utilizan operadores abstractos al igual que operadores primitivos durante la generación del plan.

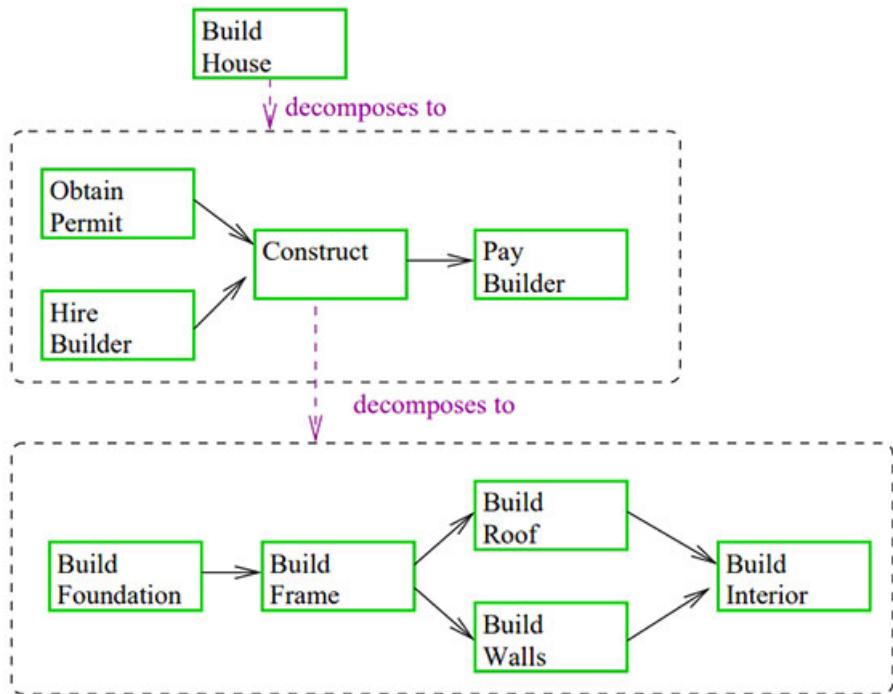


Figura 1. Ejemplo de descomposición de tareas.

Fuente: (Onder, 2020)

En general, los componentes de un problema que intentamos resolver por medio de un planificador HTN se basan en las propiedades que define el entorno y el dominio de las tareas que componen las redes que estableceremos para crear los planes. En el caso de los modelos basados en agentes, obtendremos las propiedades del entorno por medio de los sensores del agente y crearemos planes que iremos ejecutando para resolver los problemas a los que este deba hacer frente a lo largo del tiempo.

Asimismo, tendremos en cuenta las acciones que ejecutemos, de tal modo que apliquemos los efectos de dichas acciones sobre el mundo a la hora de evaluar la viabilidad de un plan.

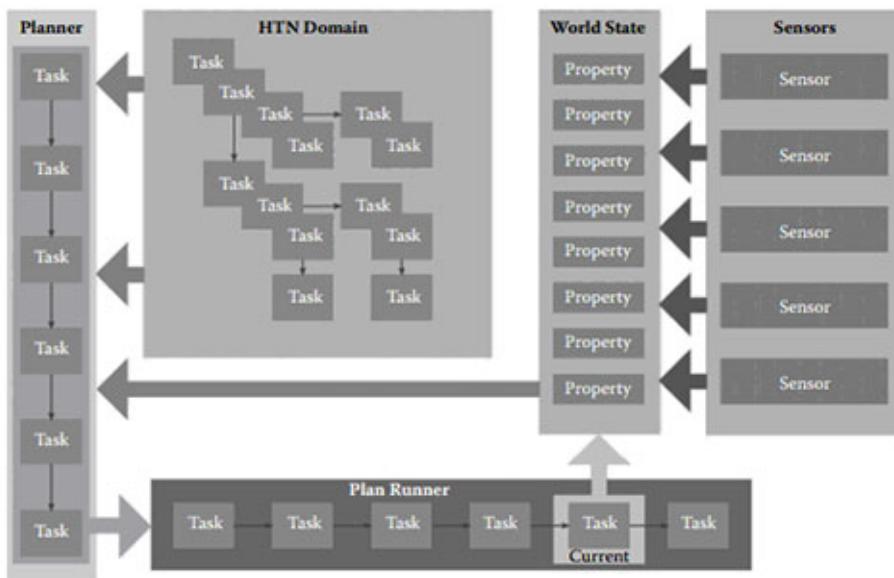


Figura 2. Vista general de un HTN.

Fuente: (Humphrey, 2015).



Accede a los ejercicios de autoevaluación a través del aula virtual

9.4. Descomposición jerárquica



Accede al vídeo «Descomposición jerárquica: planificadores HTN» a través del aula virtual

HTN es adecuado para dominios donde las tareas se organizan naturalmente en una jerarquía. Para construir estos modelos empleamos operadores abstractos para comenzar un plan. Utilizamos **técnicas de planificación de orden parcial** y **descomposición de acciones** para llegar al plan final.

Una vez que hemos terminado de descomponer el plan, nos encontramos con un conjunto ordenado de operadores primitivos.

No obstante, lo que debe considerarse primitivo es subjetivo: lo que un agente considera primitivo pueden ser los planes de otro agente. Por tanto, la

descomposición de un plan puede derivar en acciones complejas que, desde el punto de vista del plan, son consideradas operadores primitivos pero que, a la hora de implementar las acciones finales, debamos descomponerlas en otras por medio de controladores que en la jerarquía del agente se encarguen de detallar esas acciones. Recordemos los primeros modelos sobre controladores apilados para tareas complejas.

A la hora de descomponer las tareas tendremos una **librería del plan** que contendrá tanto las tareas primitivas como las que no lo son. Las no primitivas (o compuestas) tendrán un conjunto de **precondiciones** de ejecución y un conjunto de **efectos** producidos en el entorno.

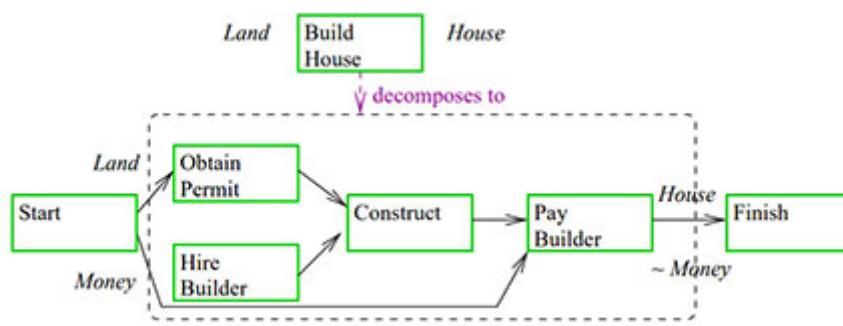


Figura 3. Ejemplo de descomposición de tareas.

Fuente: (Onder, 2020).

Pero podemos tener varias tareas que permitan construir un plan, como en la figura anterior. Por ejemplo, podríamos tener otra serie de tareas que nos permitan edificar una casa.

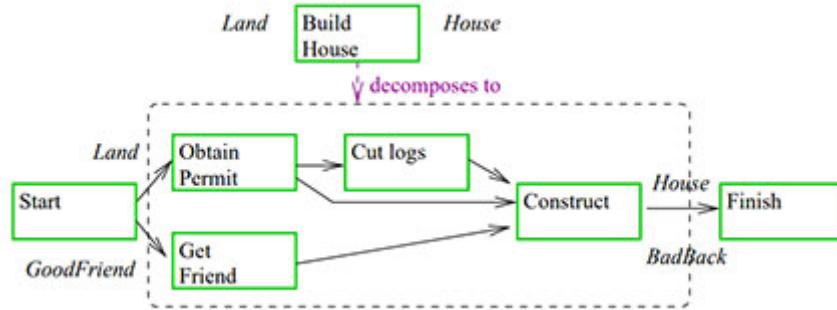


Figura 4. Ejemplo de descomposición de tareas.

Fuente: (Onder, 2020).

Ambos conjuntos de tareas estarían a disposición del planificador para poder crear distintos planes, teniendo en cuenta que ambos medios de creación del plan tienen distintos efectos en el entorno (tener menos dinero o tener la espalda mal) al margen de obtener el efecto deseado (tener la casa).

Definición de las tareas

Así, cuando definamos las tareas, tendremos que establecer dos elementos principales: las **precondiciones** necesarias que deben darse en el entorno para poder ejecutar una tarea y los **efectos** que se crean en el entorno.

Por ejemplo:

```

Action(BuyLand, PRECOND:Money, EFFECT: Land ∧¬ Money)
Action(GetLoan, PRECOND:GoodCredit, EFFECT: Money ∧ Mortgage)
Action(BuildHouse, PRECOND:Land, EFFECT: House)

Action(GetPermit, PRECOND:Land, EFFECT: Permit)
Action(HireBuilder, EFFECT: Contract)
Action(Construct, PRECOND:Permit ∧ Contract,
      EFFECT: HouseBuilt ∧¬ Permit)
Action(PayBuilder, PRECOND:Money ∧ HouseBuilt,
      EFFECT: ¬ Money ∧ House ∧¬ Contract)

```

Figura 5. Ejemplo de precondiciones y efectos.

Fuente: (Onder, 2020)

Cuando tengamos que desarrollar un plan, descompondremos las tareas de modo que empleemos las acciones del sistema. Por ejemplo, para el primer caso tendríamos que la descomposición del plan se produce de la siguiente manera:

Plan	Pasos (P1:GetPermit, P2: HireBuilder, P3: Construction, P4:PayBuilder)
Ordenación:	Inicio < P1 < P2 < P3 < P4 < Final Inicio < P2 < P3
	Con las propiedades del estado que se desprenden de los distintos efectos.

Tabla 1. Descomposición de un plan.

De esta manera, podremos descomponer varios planes que empleen distintas acciones en distintos contextos para obtener planes posibles que resuelvan el problema.

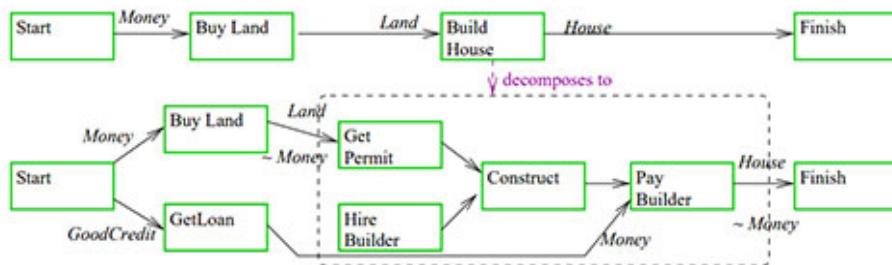


Figura 6. Ejemplo de descomposición de tareas.

Fuente: (Onder, 2020).



Accede a los ejercicios de autoevaluación a través del aula virtual

9.5. Referencias bibliográficas

Georgievski, I. y. (2014). An Overview of Hierarchical Task Network Planning. Groningen: Johann Bernoulli Institute for Mathematics and Computer Science University of Groningen. Obtenido de <https://arxiv.org/pdf/1403.7426v1.pdf>

Ghallab, M. N. (2004). *Automated Planning: theory and practice*. San Francisco: Elsevier.

Humphrey, T. (2015). Exploring HTN Planners through Example. *Game AIPro 2*, (pp. 149-167).

Nau, D. S. (1999). SHOP: Simple Hierarchical Ordered Planner. *IJCAI*, (págs. 968-975).

Nau, D. S. (2007). Current Trends in Automated Planning. *AI Magazine*, 28(4), 43-58.

Onder, N. (. (09 de 05 de 2020). *Computer Science Course*. Obtenido de Michigan Technological University: <https://pages.mtu.edu/~nilufer/classes/cs5811/2010-fall/lecture-slides-3e/cs5811-ch11b-htn>

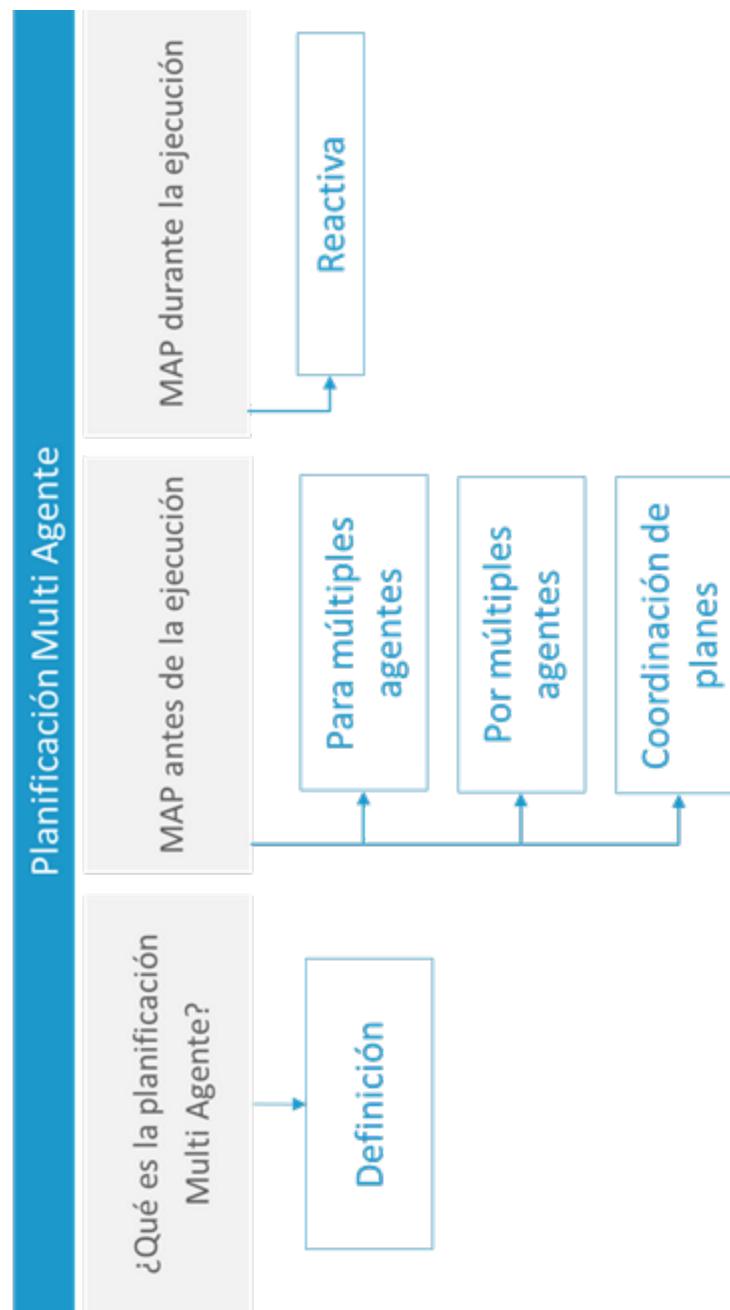


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Planificación multi agente

Esquema. Tema 10



Ideas clave. Tema 10

10.1. ¿Cómo estudiar este tema?

A lo largo de este tema profundizaremos en el concepto de planificación multi agente. Este concepto será abordado desde dos puntos claramente identificables, desde la planificación y la ejecución. Por un lado, profundizaremos en la planificación multi agente que se realiza **antes de enviar a ejecución cualquier acción de un plan**, y por otro lado profundizaremos en una línea de investigación relativamente nueva, en la que se realiza una planificación multi agente **durante la ejecución de las acciones de un plan**.

Considerando estos conceptos, ilustraremos el caso en el que deseamos construir **sistemas de planificación en entornos multi agentes**.



Accede a los ejercicios de autoevaluación a través del aula virtual

10.2. ¿Qué es la planificación multi agente?



Accede al vídeo «¿Qué es la planificación multi agente?» a través del aula virtual

Como lo hemos venido mencionando en temas anteriores los agentes se pueden clasificar en dos categorías según su arquitectura y las técnicas que emplean en su toma de decisiones:

- ▶ Agentes reactivos (Gúzman Álvarez, 2019): basan su toma de decisiones en forma rápida y teniendo en cuenta su estado actual, evitando siempre tener que realizar un proceso de búsqueda complejo.

- Agentes de deliberativos (Ghallab, 2004): basan su toma de decisiones con procesos de búsqueda complejos que pueden emplear grandes cantidades de tiempo. Durante la búsqueda simulan y predicen soluciones a situaciones futuras, posiblemente como un resultado de sus propias acciones. Todo esto para decidir el mejor plan de acciones (plan óptimo).

El decidir que un agente sea reactivo o deliberativo depende de la situación particular en la que se encuentre el agente. Por ejemplo, en el problema de planificar una ruta para ir de un lugar a otro. Un agente reactivo puede utilizar una brújula para guiar su camino, por el contrario, un agente deliberativo consultaría un mapa. Obviamente, el agente deliberativo generará un plan óptimo (la ruta más corta) en la mayoría de los casos porque de ante mano tomará decisiones de evitar colisiones, callejones sin salida, u otras situaciones que se puedan encontrar durante el camino.

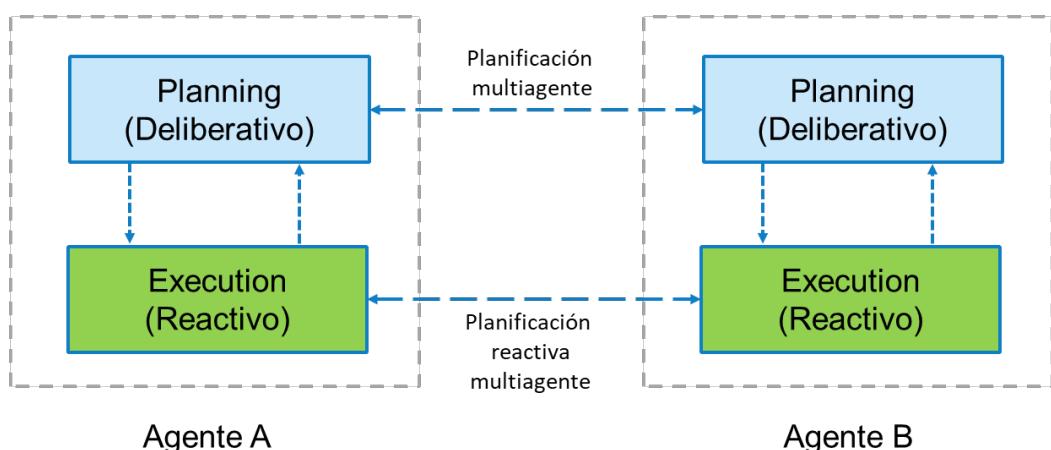


Figura 1. Situaciones donde se puede realizar planificación multi agente.

En la literatura (De Weerdt, 2009) (Guzman, 2015), los agentes reactivos son considerados más como agentes de ejecución. Y los agentes deliberativos como agentes de planificación. Ambos tipos de agentes pueden realizar una planificación, una menos profunda para los reactivos, y una más profunda para los deliberativos. Es decir, en el término más general, podemos tener una coordinación entre los agentes durante la ejecución o durante la planificación.

Para resolver un problema de planificación multi agente (durante la ejecución o planificación), en general, debemos resolver un conjunto de tareas (Durfee E. H., 2001), si bien algunas de estas se pueden suprimir o matizar:

1. Detallar y refinar la meta global.
2. Asignar tareas a los agentes.
3. Coordinar a los agentes antes de la planificación.
4. Resolver los planes individuales de cada agente.
5. Coordinar el plan resultante de todos los agentes antes de ejecutarlo.
6. Ejecutar el plan.
7. Resolver fallos en los planes de los agentes durante la ejecución

En estos escenarios multi agente, nos podemos encontrar con **entornos cooperativos, competitivos o parcialmente cooperativos**.

En el resto del tema, nos centraremos en la planificación multi agente que se realiza antes de la ejecución y durante la ejecución.



Accede a los ejercicios de autoevaluación a través del aula virtual

10.3. Planificación multi agente antes de la ejecución



Accede al vídeo «Planificación multi agente antes de la ejecución» a través del aula virtual

Es un tipo de planificación que sucede antes de enviar cualquier plan o acción a ejecución.

La planificación puede ser desarrollada por:

1. Un único agente de planificación centralizado;
2. Un número de agentes de planificación que interactúan entre sí para generar, entre todos, un plan libre de conflictos y en el que cada uno de los agentes involucra la acción que quiere ejecutar;
3. O un número de agentes de planificación que generan sus propios planes y luego realizan un proceso de coordinación de planes o plan merge (Torralba, 2019), con el fin de evitar conflictos entre los planes.

En cualquiera de los casos, las acciones del plan serán enviadas a los respectivos agentes de ejecución para que sean ellos los que ejecuten las acciones en el entorno.

Este tipo de planificación multi agente, donde múltiples agentes de planificación intervienen en el proceso de búsqueda de un plan solución, a menudo de forma distribuida, presenta dificultades adicionales sobre el ya difícil problema de la planificación en sí mismo:

- ▶ Necesidad adicional de protocolos de coordinación,
- ▶ Comunicación, a menudo, limitada;
- ▶ Y el resultado del plan final es poco óptimo.

Sin embargo, existen un número significativo de buenas razones para realizar la planificación por y/o para múltiples agentes:

1. Los agentes pueden ser entidades de la vida real que tienen principalmente sus propios intereses. Por lo tanto, aprecian mantener su privacidad y autonomía.
2. En muchos casos, un sistema distribuido (planificación por múltiples agentes) puede resolver problemas más eficientemente que un sistema centralizado (planificación para múltiples agentes).
3. Crear y mantener planes localmente permite una reacción más eficiente en caso de incidentes, especialmente cuando la comunicación es limitada.
4. Dividir el problema de planificación en subproblemas más pequeños (asignación de objetivos) y resolverlos en paralelo a veces puede ser más eficiente,

especialmente cuando los subproblemas de planificación individuales están poco vinculados.

Planificación para múltiples agentes

Es una técnica en la que se emplea un agente de planificación centralizado para encontrar un plan solución que consiga los objetivos. Cada acción o tarea es luego asignada (*task allocation* (Lee, 2017)) a su agente de ejecución para que se encargue de llevarla a cabo.

La ventaja de esta técnica es que puede utilizar cualquier planificador del estado del arte.

En este tipo de planificación se asume que:

- ▶ Los objetivos son comunes entre todos los agentes, por lo que buscan cooperar entre ellos para conseguirlos.
- ▶ Los agentes tienen una visión parcial del estado del mundo. Es decir, algunas proposiciones pueden ser desconocidas para algunos agentes. Por ejemplo, porque no están en su rango de visión.
- ▶ Cada agente tiene sus propias acciones, que pueden ser iguales (por ejemplo, moverse) o diferentes en capacidades (por ejemplo, en el dominio de los robots de Marte, un agente puede solo analizar rocas y otro agente puede solo comunicar los resultados a la tierra).

El agente de planificación centralizado mantiene una visión global del estado del mundo. Es un agente que genera y coordina los planes de otros agentes de ejecución y distribuye a estos agentes las tareas o acciones que definen sus propios planes mientras pueden negociar con otros sobre tareas o recursos (Clement B. , 2005).

Cada agente de ejecución debe sensorizar su parte del estado del mundo e informarla al agente de planificación, quien se encargará de realizar las siguientes tareas de manera centralizada:

1. **Unificar** los estados del mundo recibidos por cada agente. Como mencionamos anteriormente, los mismos pueden ser diferentes porque cada agente puede tener su propia visión del estado del mundo.
2. **Generar** un plan solución que a partir del estado del mundo global consiga los objetivos comunes de los agentes de ejecución.
3. **Asignar** las acciones del plan solución a cada agente de ejecución.

Un plan solución generado con planificación para múltiples agentes (PTMA, usando siglas en inglés) tiene la garantía de:

- ▶ no presentar conflicto entre sus acciones,
- ▶ ser óptimo (dependiendo del tipo de heurística que se utilice)
- ▶ y completo.

En resumen, un agente de planificación centralizado se orienta a tratar con problemas cuyos agentes tienen fuertes interacciones. En estos diseños se asume un entorno completamente cooperativo por parte de los agentes que comparten bases de conocimiento y mecanismos de comunicación y coordinación.

Planificación por múltiples agentes

El término **planificación por múltiples agentes** (PBMA, usando siglas en inglés) aborda el problema de la planificación en entornos donde varios agentes independientes planifican de forma incremental para generar un plan que en la mayoría de los casos tiene una representación centralizada del plan (Durfee E. H., 1999).

Notar que el término **representación centralizada del plan** se refiere a que cada agente tiene una visión global de cómo va el plan. Es como tener el plan en una pizarra que es común para todos los agentes. En ningún momento se refiere a planificar de forma centralizada.

PBMA puede referirse a la **planificación distribuida**, donde debemos tener en cuenta varios agentes independientes no centralizados.

Existen dos elementos principales que diferencian la planificación clásica de los modelos PBMA:

1. La coordinación de las actividades de planificación. Implica, por ejemplo, implementar un protocolo que permita planificar por turnos.
2. La distribución de la información entre agentes. Implica que cada agente tiene su propia visión del estado del mundo, y sus propias capacidades. Y que además pueden o no comunicar dicha información entre ellos.

El framework Partial Global Planning (PGP) (Durfee E. H., 1987), implementa una aproximación en la que cada agente tiene un conocimiento parcial de los planes de los otros agentes a través de una representación especializada del plan. En este método, la coordinación se logra de la siguiente manera:

Supongamos dos agentes de planificación A y B que desean resolver un problema.

- ▶ Si un agente A informa a otro agente B de una parte de su propio plan, B fusiona esta información en su propio plan global parcial.
- ▶ El agente B puede entonces intentar mejorar el plan global. Como, por ejemplo, eliminando acciones redundantes.
- ▶ El plan mejorado por el agente B se muestra a otros agentes, que pueden aceptar, rechazar, o modificarlo.
- ▶ Se supone que este proceso se ejecuta simultáneamente con la ejecución del plan local de cada agente.

PGP se aplicó en un problema de programación de pacientes de un hospital.

Otro enfoque para la coordinación de agentes es a través de modelos de actitud mental. El framework GRATE (Jennings, 1993) permite a los agentes coordinar su planificación individual razonando sobre sus creencias, deseos, intenciones e intenciones y compromisos conjuntos. La coordinación se intercala con la planificación, creando y revisando compromisos a través de un agente organizador. En resumen, una PBMA se orienta a tratar con problemas cuyos agentes tienen interacciones medias y/o fuertes. En estos diseños se asumen entornos cooperativos o parcialmente cooperativos. Los agentes en la mayoría de los casos tienen capacidades diferentes y/o recursos limitados. Además, comparten bases de conocimiento, protocolos de planificación y mecanismos de comunicación.

Coordinación de planes de múltiples agentes

Estas aproximaciones se centran en resolver problemas donde los agentes tienen escasas interacciones entre sí. Los agentes en estos modelos diseñan los planes de forma individual, de modo que el objetivo es coordinar *a posteriori* estos planes individuales (Durfee E. H., 1999) (Van Der Krogt, 2005). Es decir, integran la planificación individual con la coordinación de planes.

El esfuerzo se centra en cómo coordinar los planes que se construyeron por separado. Estos métodos son llamados fusión de planes (plan merge) y apuntan a la construcción de un plan conjunto dados los (sub) planes individuales de cada uno de los agentes participantes.

(Georgeff., 1983) fue uno de los primeros en proponer un proceso de sincronización de planes individuales. Definió un modelo para formalizar las acciones con precondiciones abiertas en el plan de un agente. La esencia de dicho modelo son las condiciones de corrección, que se definen en el estado del mundo y deben ser válidas antes de que la ejecución del plan se pueda llevar a cabo. Dos agentes pueden ayudarse mutuamente cambiando el estado del mundo de tal manera que se

cumplan las condiciones de corrección del otro agente. Por supuesto, cambiar el estado del mundo puede ayudar a un agente, pero también puede interferir con las condiciones de corrección de otro agente.

Otro enfoque de fusión de planes, realizado de manera centralizada, se puede observar en (Rosenschein, 1994). Aborda problemas de resolución de conflictos entre acciones y/o acciones redundantes. Lo resuelven mediante el algoritmo de búsqueda A*. Utilizan una heurística inteligente basada en costos. Llegan a demostrar que al dividir el trabajo de construir subplanes sobre varios agentes, se puede reducir la complejidad general del algoritmo de fusión (Rosenschein, 1994).

Otra técnica es intercalar la planificación y la coordinación de planes en múltiples niveles de abstracción (Clement B. J., 1999). La idea es que los agentes gradualmente vayan refinando sus planes a la vez que van coordinándolos. Los planes se van realizando básicamente en una manera abstracta. Este razonamiento abstracto también puede ser utilizado por los agentes para mantener la autonomía mientras se explotan los resultados de otros agentes para mejorar la eficiencia del plan y el rendimiento de la búsqueda. En esencia el algoritmo funciona de la siguiente manera.

Durante la realización del plan se van agregando dependencias condicionales al plan: si un agente logra la meta de otro, el agente puede ejecutar una rama más eficiente del plan; de lo contrario, se puede seguir el curso normal de acción. Esto funciona con éxito en un enfoque de agente único que utiliza una representación de red temporal simple condicional (STN) para fusionar acciones / subplanes redundantes en submetas (Tsamardinos, 2000).

En la vida real la fusión de planes se ha empleado de forma efectiva en los datos de planificación de una compañía de taxis.



Accede a los ejercicios de autoevaluación a través del aula virtual

10.4. Planificación multi agente durante ejecución



Accede al vídeo «Planificación multi agente durante la ejecución» a través del aula virtual

Una vez se ha generado un plan, estos se envían a ejecución. Los agentes de ejecución normalmente son los encargados de ejecutar y monitorizar la correcta ejecución de las acciones del plan.

La monitorización implica el poder detectar fallos que impidan la correcta ejecución de las acciones del plan. Cuando esto sucede los agentes de ejecución pueden intentar:

- ▶ Repararlo de forma individual,
- ▶ Repararlo de forma multi agente en el menor tiempo posible,
- ▶ Solicitar un nuevo plan a un agente de planificación.

La planificación multi agente durante la ejecución, es un tipo de planificación que sucede cuando se están ejecutando las acciones de un plan. Es desarrollada por:

- ▶ Un número de agentes de planificación reactiva que interactúan entre sí para reparar entre todos el plan de otro agente que ha fallado durante la ejecución.

Se orienta a tratar con problemas cuyos agentes tienen interacciones débiles y/o medias. En estos diseños se asumen entornos cooperativos o parcialmente cooperativos donde los agentes son independientes. Los agentes en la mayoría de los casos tienen capacidades diferentes y/o recursos limitados. No comparten bases de conocimiento, y cuentan con protocolos sencillos de planificación.

Es una línea de investigación relativamente nueva. En (Gúzman Álvarez, 2019), desarrollaron MARPE, un framework de planificación y ejecución reactiva para

ambientes multi agentes. Se profundizará más en este tipo de planificación más adelante.



Accede a los ejercicios de autoevaluación a través del aula virtual

10.5. Referencias bibliográficas

Clement, B. (2005). Workshop on Multiagent Planning and Scheduling. Taller llevado a cabo en el International Conference on Automated Planning and Scheduling ICAPS-05. Monterrey (California).

Clement, B. J. (1999). Top-down search for coordinating the hierarchical plans of multiple agents. *In Proceedings of the third annual conference on Autonomous Agents*, (pp. 252-259).

De Weerdt, M. &. (2009). Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4), 345-355.

Durfee, E. H. (1987). Planning coordinated actions in dynamic domains.

Durfee, E. H. (1999). A survey of research in distributed, continual planning. *Ai magazine*, 20(4), 13-13.

Durfee, E. H. (2001). Distributed problem solving and planning. *In ECCAI Advanced Course on Artificial Intelligence* (pp. 118-149). Berlin, Heidelberg: Springer.

Georgeff., M. P. (1983). Communication and interaction in multi-agent planning. *Third National Conference on Artificial Intelligence (AAAI-83)*, (pp. 125-129). Menlo Park, CA.

Ghallab, M. N. (2004). Automated Planning: theory and practice. San Francisco: Elsevier.

Gúzman Álvarez, C. A. (2019). *Reactive plan execution in multi-agent environments (Doctoral dissertation)*.

Guzman, C. C. (2015). Reactive execution for solving plan failures in planning control applications. *Integrated Computer-Aided Engineering*, 22(4), 343-360.

Jennings, N. R. (1993). Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. *International Journal of Intelligent and Cooperative Information Systems*, 2(03), 289-318.

Lee, B. H. (2017). Multi-UAV control testbed for persistent UAV presence: ROS GPS waypoint tracking package and centralized task allocation capability. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)* (pp. pp. 1742-1750). IEEE.

Rosenschein, E. E. (1994). Divide and conquer in multi–agent planning. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, (pp. 375-380). Menlo Park, CA.

Torralba, A. &. (2019). Merge-and-shrink task reformulation for classical planning. *HSDIP*, (p. 18).

Tsamardinos, I. P. (2000). Merging Plans with Quantitative Temporal Constraints, Temporally Extended Actions, and Conditional Branches. *AIPS*, (pp. 264-272).

Van Der Krogt, R. y. (2005). Plan repair as an extension of planning. *International Conference on Automated Planning and Scheduling ICAPS-05*. Monterrey (California).

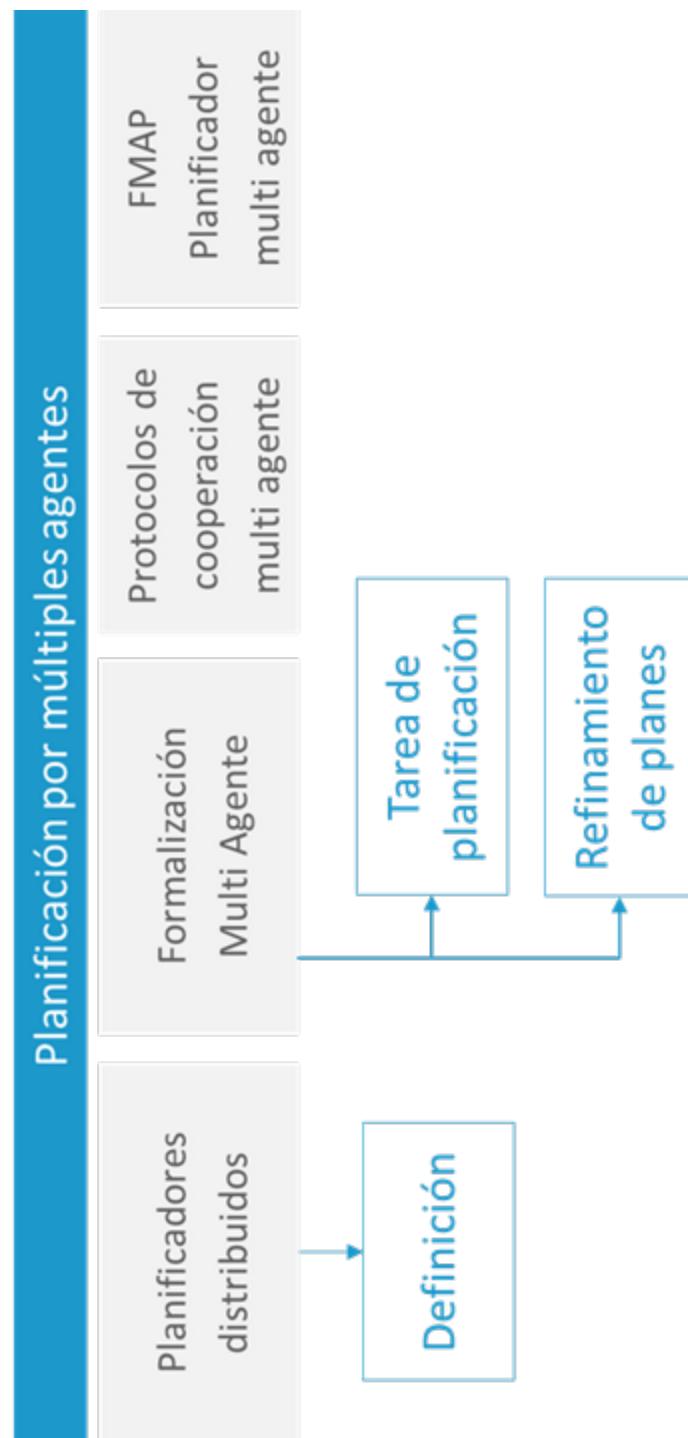


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Planificación por múltiples agentes

Esquema. Tema 11



Ideas clave. Tema 11

11.1. ¿Cómo estudiar este tema?

En este tema profundizaremos en los planificadores distribuidos o planificación por múltiples agentes. Definiremos formalmente este tipo de problemas y como se solucionan. Hablaremos de los protocolos utilizados en la planificación multi agente.

Considerando esta base de planificadores, ilustraremos el caso en el que deseamos construir **sistemas de planificación en entornos multi agentes**. Como sucedía en el caso de los algoritmos de búsqueda, los entornos donde varios agentes tienen que coexistir presentan problemas que deben ser tratados de modo específico. En nuestro caso, presentaremos un modelo de planificador por múltiples agentes diseñado por (Torreno Lerma A. , 2016).



Accede a los ejercicios de autoevaluación a través del aula virtual

11.2. Planificadores distribuidos



Accede al vídeo «Planificadores distribuidos» a través del aula virtual

Un posible diseño de un **planificador distribuido** lo encontramos en (Torreno Lerma A. , 2012), en el que definimos un sistema basado en componentes que se puede distribuir entre varios agentes.

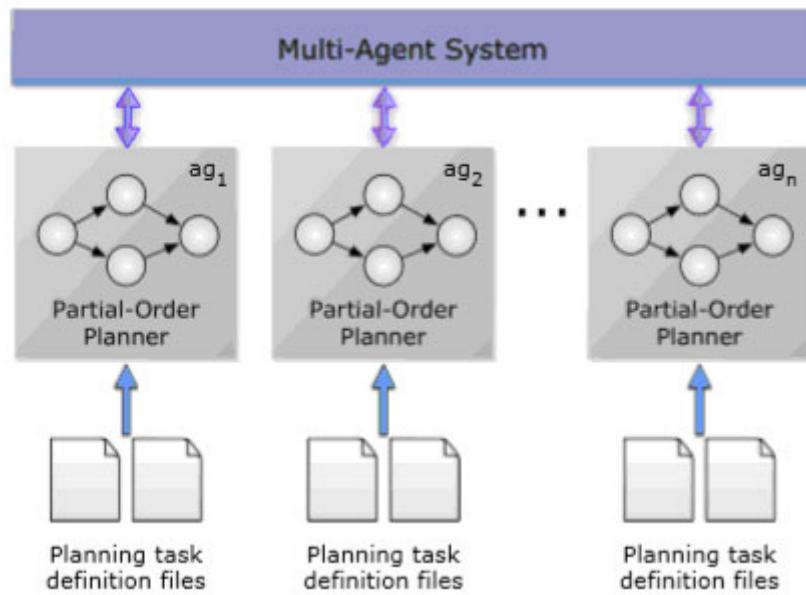


Figura 1. Estructura de un sistema PMA.

Fuente: (Torreno Lerma A. , 2012).

Este sistema se basa en tres tipos de componentes:

- ▶ Una descripción formal del dominio y el problema, especificada por medio de un lenguaje de planificación, aunque es necesario extender los estándares, dado que PDDL, en ninguna de sus versiones, cubre todas las necesidades del problema descentralizado.
- ▶ Un sistema multi agente distribuido para permitir la gestión y comunicación de varios agentes, el intercambio de los planes construidos y la toma de decisiones del plan inicial.
- ▶ Un POP (explicado en el Tema 7) que tendrá embebido cada uno de los agentes y que le permitirá refinar el plan global inicial aportando sus subplanes elaborados.

Para este tipo de arquitecturas debemos tener en cuenta que necesitaremos de:

- ▶ Información compartida por los agentes que deberá expresar aquellas variables instanciadas que son públicas y compartidas por estos. Así como los predicados que se presentan y asociar conjuntos de ellos a los distintos agentes o dejar compartido para todo el sistema elementos de esta información.

- ▶ Metas privadas y públicas (o globales): en esta diferenciación clave es donde tenemos la responsabilidad de crear planes consistentes en aquellas metas públicas y garantizar que todas las metas privadas son conseguidas en cada subplan.
- ▶ Permitir, en las variables instanciadas, información explícita de forma positiva y negativa, añadiendo un sistema que nos simplifica el procesado. Se añade el concepto de multifunciones que nos permite codificar parte de la información del problema mediante una notación más compacta.

Con esta información emplearemos un **algoritmo de planificación** que, en esencia, funciona de forma **distribuida**. En él, intercambiaremos inicialmente la información entre los agentes para configurar el problema de planificación. Posteriormente, procesaremos el plan por medio de refinamientos sucesivos. Para ello, cada agente, que tiene un POP embebido, refina de modo individual el plan actual e intercambia información con los otros agentes, enviando los nuevos planes refinados sobre el plan actual y seleccionando cada uno el plan que le resulta más prometedor para continuar con el refinamiento. Esto lo explicaremos en detalle más adelante.



Accede a los ejercicios de autoevaluación a través del aula virtual

11.3. Formalización



Accede al vídeo «Formalización» a través del aula virtual

Tarea en planificadores por múltiples agentes

En un PBMA (Torreno Lerma A. , 2016), definiremos una tarea como una tupla $\langle AG, V, I, G, A \rangle$ formada por:

- ▶ $AG = \{1, \dots, n\}$: Un conjunto finito no vacío de agentes de planificación.
- ▶ $V = \bigcup_{i \in AG} V^i$: Donde V^i es un conjunto de variables estado que definen un estado en el que se pueda encontrar un agente i en el entorno.
- ▶ $I = \bigcup_{i \in AG} I^i$: Es un estado inicial definido por un conjunto de variable instanciadas (fluents). Como en este tipo de problemas pueden existir agentes especializados, ellos pueden solo conocer un subconjunto de I . Como premisa, el estado inicial de dos agentes nunca se contradice.
- ▶ G : Conjunto de metas que deben alcanzar entre todos los agentes, teniendo en cuenta que se considerará cumplido el estado meta independientemente del agente que consiga el última fluent buscado. Definen los objetivos de la tarea de planificación multi agente.
- ▶ $A = \bigcup_{i \in AG} A^i$: Conjunto de acciones de planificación deterministas de los agentes.

A diferencia de STRIPS, en este tipo de diseños se admite la **hipótesis de mundo abierto**, que permite instanciar las variables y asumir que aquella información que no está definida de modo explícito es desconocida para el agente.

Un estado del mundo está definido a través de un conjunto finito de variables estado V , la cual está asociada a un dominio finito de valores Dv mutualmente exclusivo que se refiere a objetos del mundo*. Asignar un valor d a una variable v de V genera una variable instanciada o fluent, $\langle v, d \rangle$. Un estado está formado por un conjunto finito de fluents.

Una acción se compone de la forma $a = PRE(a) \rightarrow EFF(a)$, donde $PRE(a)$ y $EFF(a)$ son un conjunto finito de fluents que representan las precondiciones y los efectos de la acción a , respectivamente. Ejecutar una acción a en un estado del mundo S producirá un nuevo estado del mundo S' como resultado de aplicar los $EFF(a)$ sobre S . Un efecto de la forma $\langle v, d' \rangle$ actualiza el estado S reemplazando el fluente $\langle v, d \rangle \in S$ por $\langle v, d' \rangle$. Teniendo en cuenta que los valores de Dv son mutualmente exclusivos, incluir $\langle v, d' \rangle$ en S' implica que $\forall d \in Dv, d \neq d' \wedge \langle v, d \rangle \notin S'$.

En los esquemas PBMA, los agentes son heterogéneos, con lo que pueden tener conocimientos del entorno y habilidades distintas, con distintos sensores y actuadores específicos para cada agente y distinta información *a priori*.

La tarea de planificación vista desde un agente i está definida como $T^i = \langle V^i, I^i, G, A^i \rangle$:

- ▶ V^i : Conjunto de variables estado conocidas por el agente i .
- ▶ I^i : es el subconjunto de fluents conocidos por el agente i del estado inicial I .
- ▶ G : es el estado objetivo, común para todos los agentes.
- ▶ $A^i \subseteq A$: Es el conjunto de habilidades del agente i (acciones de planificación).

Finalmente, como podemos observar un agente i también puede tener una visión parcial sobre un dominio de valores D_v de una variable v . Normalmente, esto lo definimos como $D_v^i \subseteq D_v$ el subconjunto de valores de v conocidos por el agente i .

Así entonces, los mecanismos de comunicación implican, generalmente, la existencia de un subconjunto de variables instanciadas que son compartidas por los distintos agentes y que les permiten coordinar sus acciones sobre el entorno.

Los agentes pueden interactuar compartiendo información acerca de sus variables estado. Por ejemplo, $V^{ij} = V^{ji} = V^i \cap V^j$, lo que representa que el agente i y j comparten un subconjunto de variables estado que está definida como la intersección entre ambos subconjuntos V^i y V^j .

Existe también la posibilidad de tener acciones públicas, es decir, que varios agentes comparten y saben que comparten acciones con otros agentes. En consecuencia, pueden solicitar a otros agentes la ejecución de una acción de la cual son conocedores de sus precondiciones y efectos.

Refinamiento de planes

Un posible modelo de planificador distribuido es aquel que considera un esquema de refinamiento de planes en el que un agente propone un plan P (vacío) para solucionar un conjunto de metas abiertas en un problema global. A continuación, el resto de los agentes refinan P resolviendo alguna de las metas que quedan abiertas en el entorno, aportando con refinamientos sucesivos fragmentos de planes que completan las metas abiertas (Kambhampati, 1997).

Un sistema subyacente que permite este tipo de planificación sería la planificación de orden parcial. Tomando la definición que nos presentan los POP como grafos acíclicos dirigidos donde tenemos como nodos del grafo a las posibles acciones de un agente y como arcos a las restricciones y efectos que estas acciones necesitan y producen, podemos definir el **esquema de planificación de orden parcial multi-agente** (Torreño Lerma A. , 2016) como aquel en el que nuestros agentes cooperan para refinar un plan base P inicialmente vacío por medio de una serie de pasos de refinamiento en los que iremos resolviendo las metas abiertas que nos queden en el plan.

Así, un paso de refinamiento desarrollado por un agente supondrá la resolución de una meta abierta representada en una variable instanciada pública y de todas aquellas metas abiertas que tuvieran que ver con las variables privadas del agente. De este modo, garantizamos la consistencia de los planes, de forma que aquellas tareas que son propias de un agente y que solo este sabe los efectos que producen sus acciones privadas sobre el entorno son resueltas por el agente sin afectar al plan global de los demás agentes con los que se coopera.

Consideraremos que un plan es solución a un problema si es un plan concurrente entre varios agentes, si no existen metas abiertas en el entorno, no hay amenazas y cada par de acciones podemos decir que son mutuamente consistentes (Torreño, 2012).



Accede a los ejercicios de autoevaluación a través del aula virtual

11.4. Protocolos



Accede al vídeo «Protocolos» a través del aula virtual

Uno de los mayores retos para implementar una planificación multi agente es el desarrollo de los protocolos de comunicación entre los agentes.

Estos protocolos permitirán, entre muchas otras cosas, la correcta comunicación entre los agentes. Una estrategia muy atractiva es la utilizada en (Rosenschein, 1994), donde clasifica los diferentes dominios e identifica protocolos apropiados para cada tipo de dominio.

Otra idea es implementar un protocolo en el que la planificación o refinamiento se vaya realizando por turnos (Torreno Lerma A. , 2016).

En ambientes más reactivos (Gúzman Álvarez, 2019) se eliminan de los protocolos los conceptos de votación o turnos de espera demasiado largos. Por el contrario, se implementa un protocolo en el que el agente que falla solicita ayuda por medio de un broadcast a todos los posibles agentes que pueden contribuir para solucionar el fallo. De esta manera, el agente que falla recibe todas las posibles soluciones y se adapta a la más conveniente.

El protocolo es de vital importancia para la resolución de problemas multi agentes.



Accede a los ejercicios de autoevaluación a través del aula virtual

11.5. FMAP



Accede al vídeo «FMAP» a través del aula virtual

FMAP es un planificador multi agente que utiliza un POP y un algoritmo de búsqueda A* multi agente. Implementa una planificación hacia adelante (Forward). El algoritmo general de FMAP se divide en tres fases:

1. Intercambio de información entre los agentes
2. Refinamiento individual
3. Proceso de coordinación

Las fases 2 y 3 se repiten hasta que un plan solución es encontrado o el espacio de búsqueda es completamente explorado.

Intercambio de información entre los agentes

El algoritmo comienza con una etapa inicial de comunicación entre los agentes, donde intercambian la información declarada como compartida en el dominio de planificación. Por ejemplo, la siguiente sintaxis extiende el PDDL para representar la información compartida entre los agentes del dominio del transporte:

```
(:shared-data
  ((at ?t - truck) - city)
  ((pos ?p - package) - city)
)
```

Esto quiere decir que los agentes comparten los predicados relacionados con la posición de los camiones ((at ?t - truck) - city) y con la posición de los paquetes ((pos ?p - package) – city). Esto lo hacen con el fin de generar estructuras de datos (grafos de planificación relajados, grafos de transición de dominios, etc.) que luego les servirán durante el proceso de planificación.

Refinamiento individual

Inicialmente el plan se encuentra vacío con dos acciones ficticias α_i y α_f que no pertenecen a las acciones de ningún agente en particular. La acción α_i representa el estado inicial de la tarea de planificación, es decir, $PRE(\alpha_i) = \text{vacío}$ y $EFF(\alpha_i) = I$, mientras que α_f representa los objetivos G de la tarea de planificación, es decir, $PRE(\alpha_f) = G$ y $EFF(\alpha_f) = \text{vacío}$.

Cada agente realiza un refinamiento del plan. En el que coloca las acciones que consiguen precondiciones abiertas del actual plan Π_0 que contiene solo las dos acciones ficticias.

Las **precondiciones abiertas** son aquellas que:

1. No están presentes en el estado inicial.
2. O no se consiguen con ningún otro conjunto de acciones del plan.

La siguiente figura 2 muestra un ejemplo de refinamiento de nivel 1 para el dominio de transporte.

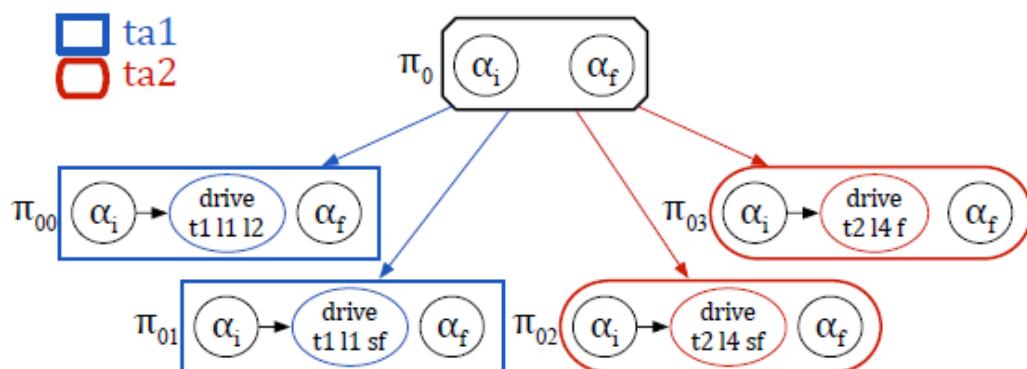


Figura 2: Ejemplo de un refinamiento de nivel 1.

Fuente: (Torreno Lerma A., 2016)

En este nivel, los agentes ta1 y ta2 proponen cada uno un refinamiento de dos planes, específicamente planes para conducir sus camiones dentro de sus áreas geográficas.

En cada uno de esos refinamientos los agentes agregan una acción con sus correspondientes enlaces **de orden** y **causales**.

Hay que recordar que, por definición (Russell, 2004), un **enlace de orden** es de la forma $a < b$, que se lee como «*a* antes de *b*» y significa que la acción *a* debe ser ejecutada en algún momento antes de la acción *b*, pero no necesariamente en el estado inmediatamente anterior. El enlace de orden describe un orden parcial apropiado. Cualquier ciclo (del tipo $A < B$ y $B > A$) representa una contradicción, que no podrá ser añadida a un plan.

Del mismo modo, un **enlace causal**, escrito como escrito como $a \xrightarrow{P} b$, y leído como la acción “*a*” alcanza la acción “*b*” a través del fluent *P*. Lo que expresa que *P* es un efecto de “*a*” y una precondición de “*b*”. También nos informa que *P* debe ser cierto durante el tiempo de acción que discurre desde la acción “*a*” a la acción “*b*”. Este sistema de enlace causal ayuda bastante durante el proceso de planificación. Porque obliga a no incluir entre las dos acciones “*a*” y “*b*” ninguna acción “*c*” que tenga como efecto la negación del fluente *P*.

En siguientes refinamientos (expansión de los siguientes nodos del árbol), los agentes pueden crear nuevos planes de refinamiento sobre el nuevo nodo seleccionado. Por ejemplo, en la figura 2, si el nodo Π_{00} es seleccionado, los agentes *ta1* y *ta2* pueden generar nuevos planes refinados sobre el nodo Π_{00} agregando una de sus acciones y manteniéndolas con los enlaces orden y causales necesarios.

Proceso de coordinación

Durante todo este proceso cada agente mantiene una copia del árbol de búsqueda multi agente, guardando la vista local que ellos tienen de cada uno de los planes que se encuentran en los nodos del árbol.

La búsqueda A* multi agente implica explorar el árbol multi agente de la siguiente manera:

1. El agente selecciona una de las hojas no exploradas del árbol;
2. El agente expande el plan seleccionado, generando todos los posibles planes refinados sobre ese nodo;
3. El agente genera y calcula el resultado de la función de evaluación de todos los nodos sucesores;
4. Y el agente comunica los resultados al resto de los agentes.

FMAP no utiliza un proceso de control por medio de mensajes broadcast. Por el contrario, mantiene un liderazgo democrático en el cual un rol de coordinador es planificado entre los agentes. Es decir, un agente en cualquier momento adopta el rol de coordinador en cada iteración, permitiéndole liderar el procedimiento de refinamiento. Inicialmente, el coordinador se elige de manera aleatoria.



Accede a los ejercicios de autoevaluación a través del aula virtual

11.6. Referencias bibliográficas

- Gúzman Álvarez, C. A. (2019). *Reactive plan execution in multi-agent environments (Doctoral dissertation)*.
- Rosenschein, J. S. (1994). *Rules of encounter: designing conventions for automated negotiation among computers*. MIT press.
- Russell, S. y. (2004). *Inteligencia Artificial: Un Enfoque Moderno*. Madrid: Pearson Educación.
- Torreno Lerma, A. (2012). *Diseño e implementación de un sistema de planificación distribuido (Trabajo fin de carrera)*. Valencia: Universidad Politécnica de Valencia.

Torreno Lerma, A. (2016). *Cooperative planning in multi-agent systems (Doctoral dissertation)*. Valencia: Universidad Politécnica de Valencia.

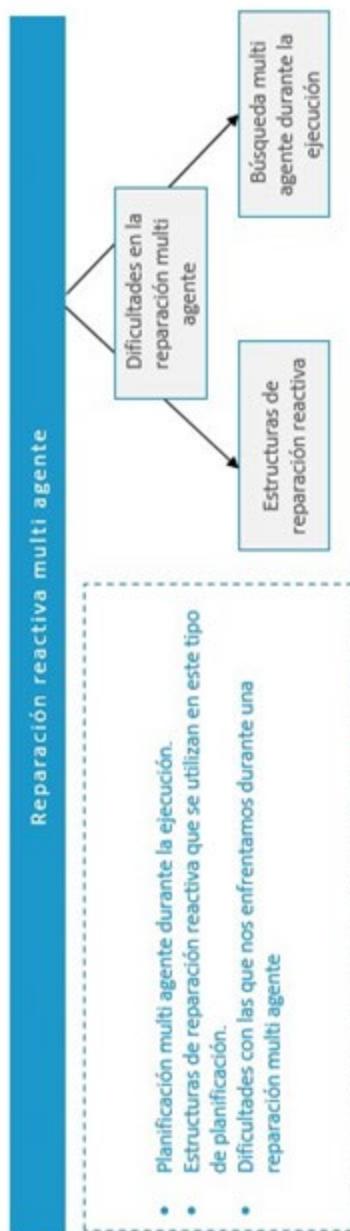


Accede al vídeo «Resumen del tema» a través del aula virtual

Razonamiento y Planificación Automática

Reparación reactiva multi agente

Esquema. Tema 12



Ideas clave. Tema 12

12.1. ¿Cómo estudiar este tema?

En este tema profundizaremos en la planificación multi agente durante la ejecución. Hablaremos sobre las estructuras de reparación reactiva que se utilizan en este tipo de planificación. Mencionaremos las dificultades con las que nos enfrentamos durante una reparación multi agente.

Finalmente, explicaremos la arquitectura y algoritmos de búsqueda utilizados durante la planificación en **entornos donde varios agentes están ejecutando sus planes**. En nuestro caso, presentaremos un modelo de reparación reactiva por múltiples agentes de ejecución diseñado por (Gúzman Álvarez, 2019).



Accede a los ejercicios de autoevaluación a través del aula virtual

12.2. Estructuras de reparación reactiva

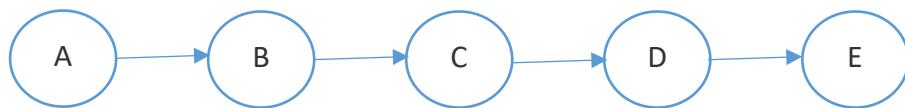


Accede al vídeo «Estructuras de reparación reactiva» a través del aula virtual

Nótese que en este tema hemos pasado de la planificación a la reparación de un plan. Y no por un solo agente, sino por varios agentes.

En la reparación partimos de un plan que se encuentra en ejecución por un agente. La filosofía consiste en intentar reparar el plan cambiando lo mínimo posible del mismo. O, en otras palabras, reutilizando al máximo posible las acciones del plan actual que ha fallado.

Por ejemplo, supongamos el siguiente plan compuesto por 5 acciones {A, B, C, D, E}:



Si durante la ejecución de este plan la acción B falla porque una de sus precondiciones no se cumple en el estado actual. Una posible reparación es generar la precondición de la acción B por medio de ejecutar nuevamente la acción A, o cualquier otra acción que genere con sus efectos la precondición de B.

Otras posibles reparaciones implican:

- ▶ Reutilización de las acciones {C, D, E}, consiguiendo en el estado del mundo las precondiciones de C.
- ▶ Reutilización de las acciones {D, E}, consiguiendo en el estado del mundo las precondiciones de D.
- ▶ Reutilización de la acción {E}, consiguiendo en el estado del mundo las precondiciones de E.

Si en cualquier caso no reutilizamos ninguna de las acciones, estaríamos haciendo una replanificación. Es decir, generaríamos un nuevo plan desde cero.

Para hacer la reparación, podemos usar las mismas técnicas de planificación (estudiadas en temas anteriores o las muchas disponibles en el estado del arte) con algunas pequeñas modificaciones.

Por ejemplo, en un POP, podríamos asumir que el nodo inicial es el plan actual que ha fallado. El mismo tendrá precondiciones abiertas que se tendrán que conseguir.

Otro ejemplo, puede ser LPG-Td (Gerevini, 2002), un planificador basado en búsqueda local y grafos de planificación que puede resolver reparación de planes. Utiliza estructuras basadas en grafos que representan planes parciales.

Las estructuras de datos basadas en árboles también son muy utilizadas para este tipo de problemas (Gúzman Álvarez, 2019). La particularidad radica en que se generan previamente después de la planificación y antes de la ejecución. Son estructuras donde cada nodo representa una posible situación del estado inicial del mundo. Desde cada nodo se puede llegar de una manera óptima al nodo raíz, que representa el objetivo de la tarea de planificación inicial. Así, el problema de reparar un plan se resume en buscar un nodo en el árbol que este contenido en el estado del mundo y a partir de este nodo generar el camino hasta el nodo raíz o nodo objetivo.

Otras estructuras son simplemente bases de datos con reglas condicionales del estilo:

SI X ENTONCES RETORNAR plan.

Lo que las hacen estructuras rápidas y eficientes para resolver problemas de reparación reactiva.



Accede a los ejercicios de autoevaluación a través del aula virtual

12.3. Dificultades en la reparación multi agente



Accede al vídeo «Dificultades en la reparación multi agente» a través del aula virtual

En muchas ocasiones un agente no es capaz de reparar un plan debido a diferentes situaciones. Por ejemplo, pierde una capacidad (no puede tomar imágenes, o comunicarse), o tiene escases de un recurso (tal como la gasolina).

En tales situaciones el agente siempre puede pedir ayuda a un agente de planificación especializado que trabaje de forma deliberativa. El problema es que puede tardar mucho tiempo en generar el plan o incluso en recibir el plan (por ejemplo, en el caso

del dominio de marte donde la comunicación de los robots con la tierra puede tardar 2 horas).

Una mejor solución radica en poder solicitar ayuda a agentes que se encuentren en el mismo entorno de ejecución.

Se deben buscar soluciones que:

- ▶ Se generen de forma rápida.
- ▶ Minimicen el costo de la comunicación entre los agentes.
- ▶ Y permitan reparar el plan del agente que falla a la vez que permite conseguir los objetivos de los demás agentes.

Está claro que la comunicación siempre es un hándicap que impactará de alguna forma en el tiempo de generación del plan solución. Por lo tanto, se deben implementar soluciones que en la medida de lo posible minimicen las comunicaciones entre los agentes.

En todo lo que concierne a reparación reactiva se evita el uso de las técnicas de planificación clásica que consumen tiempo y adicionalmente memoria. Se deben pensar en técnicas más eficientes, aunque se sacrifique la optimalidad del plan solución.



Accede a los ejercicios de autoevaluación a través del aula virtual

12.4. Búsqueda multi agente durante la ejecución



Accede al vídeo «Búsqueda multi agente durante la ejecución» a través del aula virtual

A continuación, vamos a explicar un modelo de planificación reactiva multi agente (Gúzman Álvarez, 2019). Como se mencionó anteriormente, esta es una línea de investigación relativamente nueva.

En (Gúzman Álvarez, 2019), proponen la siguiente arquitectura de planificación y ejecución multi agente:

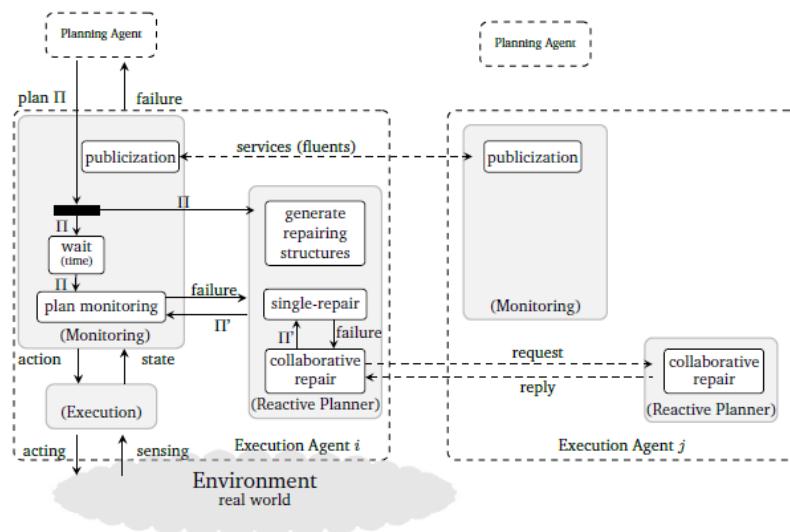


Figura 1. Arquitectura del modelo de planificación y ejecución multi agente.

Fuente: (Gúzman Álvarez, 2019)

El modelo describe dos agentes de ejecución, *i* y *j*. Ambos agentes tienen tres módulos principales, módulo de ejecución (Execution), módulo de Monitorización (Monitoring), y módulo de reparación reactiva (Reactive Planner). Adicionalmente, cada agente de ejecución tiene asociado un agente de planificación deliberativa. Que es el encargado de generar el plan inicial a ejecutar.

Veamos los pasos que se registran en este modelo al momento de ejecutar el sistema.

Pasos del modelo

Los agentes de ejecución solicitan el plan inicial a los agentes de planificación.

Mientras se genera el plan inicial, los agentes de ejecución publicitan entre ellos una información llamada servicios (caja *publicization* en la figura 1). Los servicios son los fluyentes que aparecen en los efectos de las acciones de cada agente de ejecución. De esta manera el agente que recibe información de un servicio puede saber de ante mano los fluyentes que puede conseguir otro agente. Por ejemplo, en el dominio de los robots de Marte que hemos venido tratando, si un agente *i* puede comunicar los resultados de una roca *r* analizada en una localización *w1*, enviará a los otros agentes el fluyente <comunicar-*r-w1*, true>. De esta manera los otros agentes saben que el agente *i* puede conseguir dicho fluyente, comunicar el análisis de la roca *r* desde la posición *w1*.

Una vez todos los agentes de ejecución reciben el plan inicial, comienza el proceso de monitorización y ejecución. Este es un proceso que se va desarrollando hasta que se ejecutan todas las acciones o la monitorización detecta un fallo, en cuyo caso se inicial el proceso de reparación.

En (Gúzman Álvarez, 2019), el proceso de reparación implica varios pasos:

- ▶ Reparar de manera individual (usando un planificador reactivo propio del agente).
Sino se encuentra un plan,
- ▶ Se activa la reparación reactiva multi agente. Si no se logra reparar con ayuda de los otros agentes, entonces finalmente,
- ▶ Se llama al agente de planificación deliberativo.

Nosotros nos centraremos en detallar brevemente cómo funciona la reparación reactiva multi agente. Temas relacionados con la reparación individual o con la planificación del agente deliberativo, se deben profundizar en (Gúzman Álvarez, 2019).

Reparación reactiva multi agente

requester agent i	helper agent j
plan window: $[G_0^i, \dots, G_n^i]$	$[G_0^j, \dots, G_m^j]$
1: for $G_t^i \in [G_0^i, \dots, G_n^i]$ do	
2: $F \leftarrow$ fluents that fail in G_t^i and i cannot achieve	
3: $AG \leftarrow$ agents that achieve all the fluents in F	
4: if $AG \neq \{\}$ then	
5: $\Pi^{AG} \leftarrow \{\}$ \triangleright solution plans	
6: for $j \in AG$ do	
7: send G_t^i to j	
8:	$G' \leftarrow$ fluents $\langle v, p \rangle \in \{G_t^i \cap eff(a) \forall a \in A^j\}$
9:	$G^{ji} \leftarrow$ Create_Joint_Partial_State(G_m^j, G')
10:	$T \leftarrow$ Helper_Joint_Search_Space(G^{ji}, A^j, S^j)
11:	$\Pi^j \leftarrow$ Iterative_Search_Plan($T, S^j, [G^{ji}]$)
12:	send Π^j to i \triangleright Otherwise, send reject message
13: $\Pi \leftarrow$ fluents $\langle v, p \rangle \in \Pi^j / v \in V^i$	
14: $\Pi^{AG} \leftarrow$ insert Π	
15: end for	
16: for $\Pi \in \Pi^{AG}$ do	
17: $\Pi^i \leftarrow$ Requester_Joint_Plan(Π, G_t^i, A^i, S^i)	
18: if $\Pi^i \neq \{\}$ then	
19: return form multi-reactive solution	
20: end for	
21: end for	

Figura 2. Algoritmo del flujo general del proceso de reparación colaborativa.

Fuente: (Gúzman Álvarez, 2019)

En la figura 2 se muestra el algoritmo utilizado una vez se detecta un fallo que no puede ser reparado de forma individual por el agente.

Los agentes de ejecución mantienen el plan de acciones que se está ejecutando representado en estados parciales. Por ejemplo, suponiendo que el plan este compuesto por las acciones [a1, a2, a3, a4]. Una representación del plan en estados parciales sería de la forma [G0, G1, G2, G3, G4]. Donde cada estado parcial Gi representa un conjunto de fluents necesarios para continuar ejecutando el resto de las acciones desde el estado parcial Gi. De esta manera, conseguir que el estado parcial G0 se cumpla en el estado del mundo nos garantiza que se pueden ejecutar las acciones [a1, a2, a3, a4]. Conseguir que el estado parcial G1 se cumpla en el estado del mundo nos garantiza poder ejecutar [a2, a3, a4].

Cuando una acción del plan falla es porque algún fluent de los estados parciales no se cumple en el estado del mundo. En este caso el algoritmo detecta el conjunto de fluentes que han fallado (línea 2 en la Figura 2). En la línea 3 obtiene los agentes que le pueden ayudar a conseguir dichos fluents (por medio de los servicios publicitados). El agente que falla pide ayuda a estos agentes, informando los literales que han fallado y desea conseguir (línea 7). Los posibles agentes colaboradores intentan reparar los fallos usando pequeños árboles de búsqueda que son generados en

tiempo de ejecución con un algoritmo en anchura con el ánimo de encontrar soluciones óptimas. Si una solución es encontrada, entonces la informan al agente que falla. Quien debe decidir si acepta o no la solución como válida (líneas 16-20).

En caso de que una solución se acepte como válida, se genera un compromiso entre los dos agentes que solo se puede romper, si existe un nuevo fallo durante la ejecución de la posible solución.



Accede a los ejercicios de autoevaluación a través del aula virtual

12.5. Referencias bibliográficas

Gerevini, A. &. (2002). LPG: A Planner Based on Local Search for Planning Graphs with Action Costs. *In AIPS*, (pp. Vol. 2, pp. 281-290).

Gúzman Álvarez, C. A. (2019). *Reactive plan execution in multi-agent environments (Doctoral dissertation)*.

Rosenschein, J. S. (1994). *Rules of encounter: designing conventions for automated negotiation among computers*. MIT press.

Russell, S. y. (2004). *Inteligencia Artificial: Un Enfoque Moderno*. Madrid: Pearson Educación.

Torreno Lerma, A. (2012). *Diseño e implementación de un sistema de planificación distribuido (Trabajo fin de carrera)*. Valencia: Universidad Politécnica de Valencia.

Torreno Lerma, A. (2016). *Cooperative planning in multi-agent systems (Doctoral dissertation)*. Valencia: Universidad Politécnica de Valencia.



Accede al vídeo «Resumen del tema» a través del aula virtual
