INTELIGENCIA ARTIFICIAL

RESOLUCIÓN AUTOMÁTICA DE PROBLEMAS

Structuralia

INDICE

1. INTRODUCCIÓN

La capacidad de resolver problemas de ámbito general es una cualidad inherente de la raza humana que proviene del proceso de razonamiento, el cual puede definirse cómo:

> *La facultad que permite resolver problemas, extraer conclusiones y aprender de manera consciente de los hechos, estableciendo conexiones causales y lógicas necesarias entre ellos.*

La capacidad de **razonar** nos ha permitido evolucionar hasta el punto de tener la capacidad de buscar diferentes formas de crear sistemas que sean capaces de imitar nuestro modelo de razonamiento o exhibir procesos de razonamiento similares a los nuestros a la hora de resolver los diferentes problemas que nos encontramos en el entorno o que generamos. Esta búsqueda, dio lugar a la aparición de la Inteligencia Artificial (IA) y del primer sistema de resolución general de problemas (GPS, General Problem Solver) creado por Herbert Simon y Allen Newell en 1957[1]. Este sistema tenía como objetivo resolver problemas de tipo general, que debían ser representados mediante un lenguaje de alto nivel basado en un modelo conceptual, por medio de un algoritmo de propósito general. De forma que el sistema de resolución era capaz de definir una secuencia de pasos o acciones que tras su ejecución permitían resolver el problema, partiendo de una representación inicial del entorno (estado inicial) hasta una representación final donde una serie de objetivos eran conseguidos (estado final), de forma que el problema era resuelto. Con el fin describir de manera sencilla el funcionamiento de un sistema general de resolución de problemas, vamos a utilizar uno de los problemas más utilizados por los investigadores en IA. El 8-Puzzle o puzzle deslizante, mostrado en la **Figura 1**, es un problema lógico que consiste en colocar un conjunto de piezas (8 piezas numéricas) en un determinado orden mediante la definición de una secuencia de movimiento individuales (arriba, abajo, izquierda y derecha) que se aplican sobre una determinada pieza utilizando la casilla libre que se denomina casilla blanca.

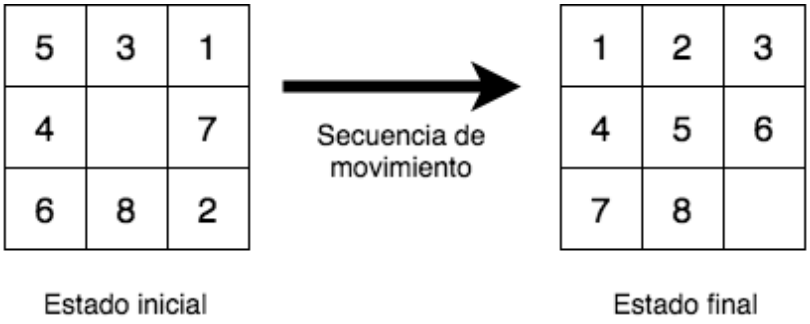

Figura 1: Ejemplo de resolución del 8-Puzzle

En el ejemplo mostrado en la **Figura 1**, se presenta una determinada configuración inicial del 8-puzzle que se considera incorrecta y que debe ser modificada mediante la aplicación de movimientos hasta conseguir llegar al estado final, que se denomina estado. Teniendo en cuanta esta definición del problema, la solución consiste en la secuencia de movimientos que nos permiten transitar del entado inicial al estado meta. Esta forma de resolver problemas de manera automática se denomina **Inteligencia Artificial Simbólica**[2] y fue el principal paradigma desde la aparición del concepto de IA hasta la década de los 90. La IA Simbólica se basa en el concepto de razonamiento simbólico que consiste en representar el conocimiento del entorno mediante la utilización de conceptos básicos denominados símbolos que deben ser manipulados de forma independiente mediante el uso de operaciones, reglas, acciones, etc.

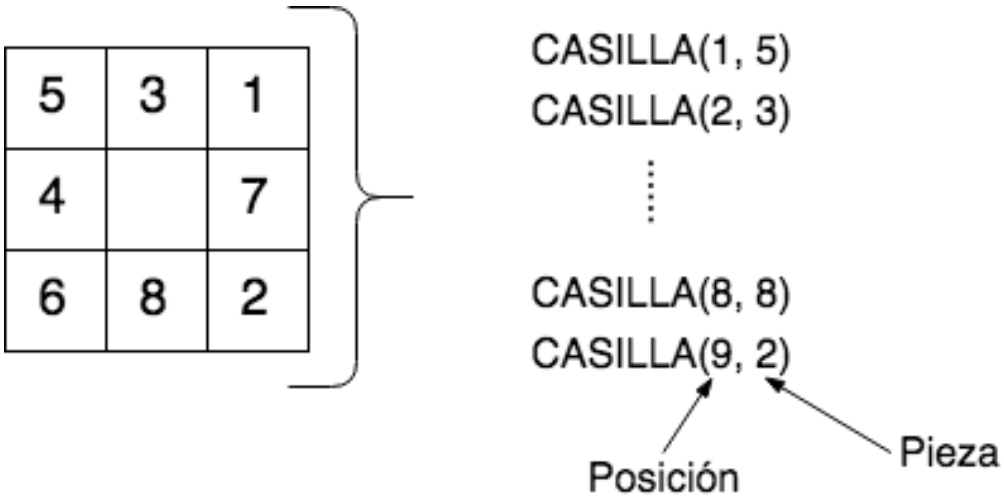

Figura 2: Ejemplo de representación de simbólica del 8-puzzle

Por ejemplo, si queremos representar el problema de la **Figura 2**, deberemos definir un símbolo para representar las diferentes posiciones del juego, de forma que cada casilla tiene que tener

una posición y una pieza. Este símbolo es una forma abstracta de representar el entorno. Para ello, introduciremos el símbolo **CASILLA** que se representa como un predicado en lógica proposicional, concepto que fue definido en el Tema 3 de este curso. Este predicado tiene dos parámetros: (1) la posición en la matriz que representa el puzzle; y (2) la pieza que se encuentra en esa posición. De forma que, si queremos mover una pieza, tendremos que eliminar dos símbolos y crear dos nuevos a continuación. Estos símbolos se corresponderán con el símbolo que representa la posición a mover y el que representa la posición donde se encuentre la pieza blanca. Teniendo en cuenta todo esto, se pueden definir una serie de características comunes a todos los sistemas de IA simbólica.

- Un **modelo conceptual**, es decir un conjunto de elementos básicos que permitían describir de forma general mediante la utilización de símbolos los diferentes problemas que pueden ser resueltos por el sistema. De forma general, el modelo conceptual debe ser capaz de representar el problema, las operaciones, así como la solución que resuelve el problema.
- Un **lenguaje de representación** de alto nivel que permita describir de forma general cualquier problema.
- Un **algoritmo** de propósito general que permita resolver cualquier problema que sea representado mediante el lenguaje de alto nivel siguiendo el modelo conceptual descrito. Este algoritmo deberá encontrar la solución que permite resolver el problema.

Durante el gran apogeo de la IA Simbólica se desarrollaron dos grandes familias de algoritmos y/o técnicas para la resolución automática de problemas: (1) el Razonamiento Automático basado en la construcción de sistemas expertos que intentan razonar de forma similar a como lo hacen los humanos y (2) la Búsqueda basada en la construcción de algoritmos que realizan una búsqueda en el espacio de estados o soluciones.

2. RAZONAMIENTO AUTOMÁTICO

El **Razonamiento Automático** puede definirse como el área de la IA que estudia cómo crear sistemas que sean capaces de resolver problemas que requieren razonamiento de forma similar a como lo haría un ser humano. Este tipo de sistemas son denominados “Sistemas expertos” [3] ya que intentan simular el proceso de toma de decisiones de un humano “experto” en la resolución de tipo específico de problema. Por ejemplo, uno de los primeros sistemas expertos a principios de la década de los 70 fue Mycin [4]. Este sistema permitía diagnosticar ciertas

enfermedades infecciosas de la sangre generando como solución un posible tratamiento en base a cierta información personal del paciente (estatura, peso, edad, etc). Este sistema recogía la información del paciente mediante una serie de preguntas y a continuación mediante la aplicación de sistema de inferencia y 500 reglas, que habían sido diseñadas por un conjunto de expertos, era capaz de identificar un conjunto posible de enfermedad y sugerir un tratamiento específico para el paciente. En base a esto podemos decir que los sistemas expertos están formado por un tres elementos básicos:

- Un **modelo conceptual** que permita definir todos los elementos básicos del sistema. Estos elementos de información son utilizados para el proceso de razonamiento, así como un conjunto de pruebas (reglas, operaciones, acciones, etc) que permiten al sistema explicar porque una conjetura es una consecuencia de un conjunto de axiomas e hipótesis. Todo esta información suelen ser definida por un conjunto de expertos que tiene un conocimiento muy amplio del problema y de la forma de solucionarlo.
- Un **lenguaje** de alto nivel que permita describir de forma sencilla los diferentes elementos del modelo conceptual. Comúnmente este lenguaje es lógica proposicional de primer orden (descrita en el tema 3 de este curso), aunque es posible utilizar lógicas de orden superior o incluso otro tipo de lenguaje basado en otros formalismos matemáticos. En este tema sólo vamos a tratar con representaciones basadas en lógica de primer orden.
- Un mecanismo de inferencia (**algoritmo**) que permita resolver el problema en base al modelo conceptual definido.

Una vez que han sido definidos y/o seleccionados todos estos elementos es posible construir un sistema experto específico para resolver un determinado tipo de problemas. Dependiendo de cómo sea definida este tipo de información y los algoritmos que son utilizados para resolver el problema es posible definir diferentes modelos de razonamiento:

- Basado en reglas lógicas.
- Basado en casos.
- Basado en Inferencia bayesiana.
- Basado en lógica borrosa o difusa.

En este tema sólo vamos a tratar de forma detallada los modelos de razonamiento más utilizados o conocidos: Razonamiento basado en reglas, Razonamiento basado en casos y el Razonamiento basado en lógica borrosa.

2.1 Razonamiento basado en reglas

Uno de los sistemas de razonamiento automático más extendido es el **Razonamiento basado en reglas** [6], también denominado sistemas de producción, el cual es una técnica de resolución automática de problemas basada en la utilización encadena de reglas en un proceso de inferencia. Este conjunto de reglas van siendo activadas a medida que sus condiciones son evaluadas de forma positiva dando lugar a nuevos hechos que permitan la aplicación de nuevas reglas. Este proceso deductivo se aplicará hasta que no exista ninguna regla que pueda ser aplicada, bien porque se ha resuelto el problema o porque no es posible resolverlo. Este tipo de técnicas están basadas en la utilización de tres elementos básicos que permiten modelar la información que describe el problema y definir las diferentes reglas que serán utilizadas para realizar el proceso de razonamiento:

- Una base de hechos acerca del mundo (entorno) que describen el conocimiento específico que el sistema puede manipular acerca del mundo. Donde un hecho puede definirse como una pieza básica de información utilizada para describir una característica del mundo que es cierta (evidencia). En el caso de lógica de primer orden el mundo está constituido de objetos y predicados que definen las propiedades de dichos objetos. Estos hechos son utilizados para definir el estado del mundo y pueden ser creados y destruido durante el proceso de razonamiento con el fin de obtener conclusiones lógicas.

Objeto	Valores
Tarjeta	Verificada, no_verificada
Fecha	Expirada, no_verificada
PIN	correcto, incorrecto
Intentos	Excedidos, no_excedidos
Límite	Excedido, no_excedido
Pago	Autorizado, no_autorizado

Tabla 1 - Ejemplo de objetos y predicados definidos para modelar el funcionamiento de un cajero automático mediante la utilización de lógica proposicional

En la Tabla 1 se presenta el ejemplo de los objetos y predicados para modelar el funcionamiento de un cajero automático. En este caso tenemos 6 objetos diferentes y dos valores o hechos para cada uno de los objetos. Por ejemplo una tarjeta puede estás o no verificada. Como se puede observar la definición de los hechos es de forma dual es decir la existencia de un predicado implica la inexistencia del otro.

- Una base de conocimiento que incluye un conjunto de reglas definidas de forma manual por al menos un experto que permiten extraer conclusiones de los hechos que definen el problema. Es decir, las reglas son el mecanismo de razonamiento que permite dado una situación inicial del problema llegar a una situación final. Estas reglas son definidas como proposiciones lógicas que relacionan dos o más objetos de la base de hechos a través de una serie de premisas (antecedentes) que dan lugar a una serie de conclusiones (consecuentes). Donde los antecedentes se corresponden con un conjunto de predicados conectados mediante operadores lógicos y los consecuentes son las diferentes conclusiones que se obtienes al aplicar una regla. En base a esto, la estructura básica de una regla sería:

  SI antecedente **ENTONCES** consequente1 **SINO** consequente2

  En base a los objetos y valores descritos en la Tabla 1 se podrían definir algunas reglas como las mostradas a continuación:

  SI Tarjeta = no_verificada **ENTONCES** Pago = no_autorizado

  SI Fecha = expirada **ENTONCES** Pago = no_autorizado

  SI PIN = incorrecto **ENTONCES** Pago = no_autorizado

- Un motor de inferencia es el algoritmo mediante el cual se produce el proceso de razonamiento a partir de la información almacenada en la hechos es capaz de inferir conclusiones y hechos mediante la utilización de las reglas almacenadas en la base de conocimiento. Es decir, el motor de inferencia va seleccionado reglas de la base de conocimiento en base a la información almacenada en la base de hechos de forma que

cuando el antecedente de una regla es cierto esta se aplica generando un conjunto de hechos y/o conclusiones que son añadidos a la base de hecho como resultado de la aplicación de la regla.

En base a estos tres elementos el sistema es capaz de razonar generando conclusiones que permiten al sistema de razonamiento resolver un problema que es configurado en base a la estructura de hechos y reglas utilizados para definir el problema. El proceso de resolución que se produce en el motor de inferencia es guiado mediante la utilización de una serie de reglas de inferencia, las cuales son definidas de modo general, que son aplicadas en base a una determinada estrategia.

2.1.1. Reglas para la generación de pruebas deductivas

Para poder inferir conclusiones durante un proceso de razonamiento, es necesario definir una serie de reglas especificas al problema las cuales normalmente está basadas en alguna reglas de tipo general: El modus ponens y el modus tollens.

Modus Ponens

El **modus ponens**, también denominada regla de separación, es la regla de inferencia en lógica proposicional más utilizada debido a que permite obtener una conclusión simple en base a una premisa o antecedente. Datos dos proposiciones P y Q donde P es el antecedente y Q es el consecuente, el modus ponens puede enunciar de la siguiente manera:

"Si **P** implica **Q**; y si **P** es verdad, entonces **Q** también es verdad"

$$(P \rightarrow Q) \wedge P \implies Q$$

De manera formal podemos definir que la regla del Modus Ponens como una regla formada por dos antecedentes que son la implicación de Q en base a P y la certeza de P. Este silogismo se suele representar de la siguiente manera:

$$\frac{\begin{array}{l} P \rightarrow Q \\ P \end{array}}{Q}$$

Modus Tollens

El **modus tollendo ponens**, también denominada regla de eliminación de la disyunción, es una regla de inferencia de lógica proposicional. Dados dos proposiciones P y Q donde P es el antecedente y Q es el consecuente, el modus tollendo ponens se puede enunciar de la siguiente manera:

"Si **P** o **Q** es verdad; y **P** es falso, entonces **Q** también es verdad"

$$(P \rightarrow Q) \wedge \neg Q \implies \neg P$$

El silogismo disyuntivo establece que, si se nos indica que al menos una de las dos proposiciones es verdadera; y también se indica que no es la primera proposición la que es verdadera; se puede inferir que debe ser la última la que es verdadera. Este silogismo se suele representar de la siguiente manera:

$$\frac{\begin{array}{r} P \rightarrow Q \\ \neg Q \end{array}}{\neg P}$$

2.1.2. Encadenamiento de reglas

Para poder obtener una solución a un determinado problema los sistemas de razonamiento basados en reglas realizan un proceso de selección de reglas en base a la información disponible en la base de hecho. Este proceso de selección consiste en un encadenamiento de reglas debido a que las reglas son encadenadas en base a las relaciones entre las premisas (antecedentes) y las conclusiones (consecuentes) de ciertas reglas. Este proceso de encadenamiento va generando nuevos hechos hasta que no es posible seleccionar más reglas o se alcanzan todas las conclusiones que resuelven el problema. Dependiendo de cómo se utiliza la información de la base de hechos se pueden diferenciar dos tipos de algoritmos de encadenamiento: Hacia delante y hacia atrás. La información puede ser utilizada para realizar búsqueda hacia delante comenzando por los hechos actuales o hacia atrás buscando desde los objetivos del problema hacia atrás.

Encadenamiento de reglas hacia delante

El algoritmo de encadenamiento de reglas hacia delante o dirigido por datos consiste en generar una secuencia de reglas encadenadas que comienzan a seleccionar reglas desde un conjunto de hechos conocidos que son considerados con el estado inicial desde los que se empieza a

resolver el problema. Este algoritmo va incluyendo nuevos hechos en la base de hechos según se van seleccionando las diferentes reglas partiendo de una base de hechos que incluye la información inicial del problema (estado inicial). Este tipo de algoritmo es muy útil para la resolución de problemas de planificación, configuración y diseño automático. Los motores de inferencia basados en este tipo de algoritmos están basados en un proceso secuencial compuesto por cuatro fases:

1. Inicialización: Este proceso sólo se ejecuta al inicio y consiste en inicializar las diferentes estructuras de datos: (1) La agenda que contiene todos los hechos activos de la base de hechos; y (2) la lista de reglas activas que contiene aquellas reglas que son disparadas o identificadas durante el proceso de identificación. La agenda hace las veces de estructura de control de algoritmo manteniendo información del proceso de ejecución que es relevante para la identificación de reglas.
2. Identificación (Matching): El proceso de identificación se encarga de analizar todas las reglas de la base de conocimiento identificando aquellas reglas relevantes en base a la información almacenada en la base de hechos. Es decir, identifica las reglas que se disparan en base a la base de hechos disponible. La relevancia de la reglas se basa en diferentes factores: satisfabilidad, referencias con el último conocimiento derivado, nuevo conocimiento para la base de hechos, etc.
3. Resolución de conflictos: El proceso de resolución de conflictos consiste en seleccionar aquella regla más importante de entre todas la seleccionados en el proceso de identificación. El mecanismo más utilizado para la resolución de conflictos se basa en el establecimiento de prioridades entre las reglas de forma que cuando se produzca un conflicto entre varias reglas, se resolverá hacia la de mayor prioridad.
4. Ejecución: El proceso de resolución consiste en ejecutar la regla resultante de la resolución de conflictos. El proceso de ejecución de una regla consiste en añadir los nuevos hechos a la base de hechos, lo que conlleva la activación de nuevas reglas que podrán ser seleccionadas en la siguiente iteración.
5. Reinicio (Reset): El proceso de reinicio es una fase opcional del algoritmo de encadenamiento y realiza una eliminación de las reglas que fueron consideras relevantes en la fase anterior con el fin de disminuir la aparición de conflictos. Una vez producido el proceso de eliminación se vuelve a la fase 2.

Encadenamiento de reglas hacia atrás

El algoritmo de encadenamiento de reglas hacia atrás o dirigido por objetivos consiste en generar una secuencia de reglas encadenadas que comienzan en la conclusión deseada hasta alcanzar un estado en el cual todos los hechos se corresponde con los iniciales. Este algoritmo utiliza una pila de objetivos (metas) que se utiliza como estructura de control para guiar el proceso de activación de las diferentes reglas. Este algoritmo comienza con una base de hechos vacía en la cual se van incluyendo nuevas hechos según se van disparando las diferentes reglas. Este tipo de algoritmo es muy útil para la resolución de problemas de diagnóstico. Los motores de inferencia basados en este tipo de algoritmos están basados en un proceso secuencial compuesto por tres fases o procesos:

1. Inicialización: Este proceso sólo se ejecuta al inicio y consiste en inicializar las diferentes estructuras de datos: (1) La agenda que se encuentra generalmente vacía; (2) la lista de reglas activas que contiene aquellas reglas que son disparadas o identificadas durante el proceso de identificación; y (3) la pila de objetivos que se inicializa con todos los objetivos iniciales del problema.
2. Selección de objetivos: El proceso de selección de objetivos consiste en seleccionar el objetivo que se encuentra en la cima de la pila y identifican todas las reglas que son capaces de satisfacer ese objetivos.
3. Análisis de premisas: El proceso de análisis realiza un análisis de todas las reglas seleccionadas en la fase anterior mediante el siguiente proceso:
   - Si todas las premisas (antecedentes) de una regla se pueden satisfacer, se ejecuta la regla derivándose sus conclusiones y se desapila el objetivo de la cima de la pila volviendo a continuación al paso 2.
   - Si alguna premisa (antecedentes) de una regla es desconocida pero existe al menos otra regla que la fije (es la conclusión de otra regla), se ejecuta la regla derivándose sus conclusiones y se apila la premisa desconocida como nuevo objetivo volviendo al paso 2.
   - Si alguna premisa (antecedentes) de una regla es desconocida y no existe ninguna regla que la fije se pregunta al usuario incluyéndose el valor introducido en la base de hechos, si la premisa introducida por el usuario satisface la regla, se pasa a la siguiente premisa y en caso contrario se descarta la reglas volviendo a continuación al paso 2.

4. Purgado: El proceso de purgado es opcional y sólo se ejecuta cuando ninguna de las reglas del paso 2 es verdadera de forma que el objetivo no puede ser satisfecho. Entonces el objetivo es desapilado de la pila de objetivos y etiquetado como desconocido en la base de hechos.

2.2 Razonamiento basado en casos

El **Razonamiento basado en casos** (CBR, en sus siglas en inglés) [6] es una técnica de resolución automática de problemas basada en la hipótesis de que problemas similares deben tener soluciones parecidas. Es decir, este tipo de técnicas utiliza un sistema de memoria, denominada base de casos, para almacenar información acerca de cada problema resuelto (experiencias) con el fin de que estas “experiencias pasadas” puedan ser utilizadas como guía para resolver problemas similares de mayor complejidad. Este tipo de técnicas puede verse como un proceso iterativo de aprendizaje y reutilización de casos.

2.2.1. Representación de la información en casos

En este tipo de técnicas, la información es representada mediante la utilización de casos o unidades de conocimiento. Un caso es una unidad de información previo que es almacenado en un sistema de memoria con el fin de poder ser utilizado por otros problemas con el fin de razonar en base a ellos. Es decir estos casos se pueden considerar como experiencias pasadas que pueden ser utilizadas para facilitar el proceso de razonamiento en problemas de mayor complejidad. De forma general un caso se puede definir como un conjunto de tres elementos:

- Representación del problema: Es la información que representa el problema que se quería resolver. Es necesario que está información sea modelada de manera genérica debido a que tiene que ser sencillo buscar similitudes entre la información almacenada en la base de casos y el problema que está intentando resolver.
- Solución: Es la solución obtenida al resolver el problema o el conjunto de acciones o pasos que se han realizado para obtener dicha solución. Dependiendo de cómo sea representada la solución al problema puede que la solución y los pasos sean exactamente lo mismo, como por ejemplo ocurre en los problemas de búsqueda, los cuales son descritos posteriormente en este documento.

- Conjunto de efectos: Es la representación del entorno una vez que se ha resuelto el problema. Este tipo de información es probablemente la más importante en las técnicas de razonamiento basado en casos ya que permite identificar problemas similares o problemas que pueden ser identificados como un subproblema de un problema mucho más complejo.

Este tipo de representación permite almacenar información de manera genérica produciendo un sistema de resolución independiente del domino. Por ejemplo se podría representar un problema binario de clasificación donde la representación del problema es el conjunto de variables de tipo atributo-valor, la solución es la clase en la que ha sido clasificado el ejemplo y los efectos podrías consistir en almacenar únicamente aquellos casos que pertenecen sólo a un tipo de clase. También sería posible representar un problema de búsqueda de forma similar, donde la representación del problema es el estado inicial, la solución es la secuencia de operadores que generar un estado meta y el conjunto de efectos se correspondería con la información que representa ese estado meta. Dependiendo de la forma en la que se modelan los casos se pueden distinguir dos tipos diferentes de sistemas de razonamiento basado en casos [7]:

- Clasificación: Este tipo de sistemas de razonamiento basado en casos se aplica a problemas donde la solución consiste en identificar una determinada clase (enfermedad, modelo, etc.) en base a un conjunto de valores de entrada. Este tipo de sistemas de razonamiento es muy útil para problema con información incompleta, como por ejemplo, los sistemas de diagnóstico, predicción, control de procesos, etc.
- Síntesis: Este tipo de sistemas de razonamiento basado en casos se aplica a problemas donde la solución debe ser construida en base a un conjunto de casos que deben ser combinados y adaptados. Este tipo de sistemas de razonamiento es muy útil para problemas de elevada complejidad, como por ejemplo, los sistemas de configuración, planificación automática o diseño de procesos.

2.2.2. Modo de razonamiento

El proceso de razonamiento basado en casos puede describirse como un ciclo continuo de aprendizaje en el cual se va mejorando el proceso de razonamiento en base a como se aumenta la experiencia del sistema. De forma general el proceso de funcionamiento de estos sistemas consta de 4 procesos, como se muestra en la **Figura 3**.

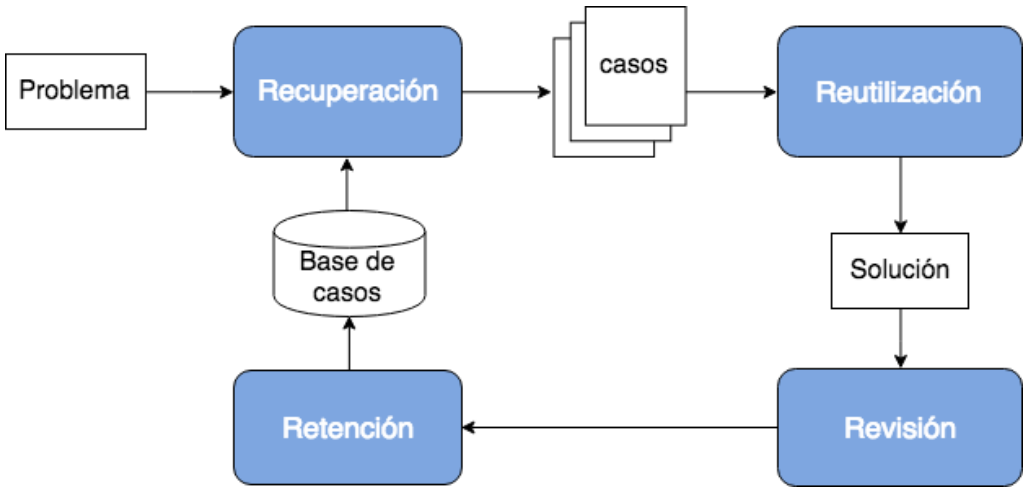

Figura 3: Proceso de funcionamiento del razonamiento basado en casos

Recuperación

El proceso de recuperación consiste en la extracción de información de la base de casos con respecto a su similitud al estado actual del problema. Para ello es necesario definir un medida de similitud que permita realizar consultas a la base de casos. El proceso de identificación de casos similares suele estar formado por cuatro etapas con el fin de minimizar al máximo el número de posibles casos a utilizar:

1. Identificación e características: Los problemas son almacenados con un número muy elevado de características, por lo que inicialmente en necesario identificar un conjunto de características que permitirán seleccionar aquellos casos similares entre sí. Esta fase identifica aquellas características que definen de la manera más completa el caso que se está analizando.
2. Comparación: Una vez identificadas aquellas características es necesarios realizar un proceso de “matching” con el fin de identificar los casos similares. Dependiendo de cómo este construida la base de casos este proceso de comparación se pueden realizar sobre toda la base de casos o sobre un subconjunto delimitado por los índices. Esta fase realiza un proceso de selección con el fin de seleccionar aquellos casos similares.
3. Ranking: Una vez identificados los casos similares, es necesario definir algún tipo de medida que permita ordenar los casos. Para ello es necesario utilizar un función de similitud que genera un valor en base a la distancia entre dos casos. Cuanto más similares sean dos casos mayor será el valor devuelto por esta función. Esta fase ordena los casos similares en fase al resultado de la función de similitud.

4. Selección: Esta es la última fase del proceso de recuperación y consiste en seleccionar los n casos con mayor similitud dependiendo de cómo se vayan a realizar las siguientes etapas del proceso de razonamiento. Lo más común es sólo seleccionar un único caso.

Reutilización

El proceso de reutilización consisten en utilizar la información que ha sido seleccionada en la fase previa con el fin de resolver el problema. La fase de reutilización será diferente dependiendo del tipo de problema de razonamiento que se esté resolviendo (clasificación o síntesis). En el caso de problemas de clasificación se seleccionará directamente el caso más similar; y en el caso de problemas de síntesis se deberá realizar un proceso de adaptación de los casos seleccionados en la fase de recuperación que puede realizarse de dos maneras diferentes:

1. Transformación: El proceso de transformación consiste en cambiar estructuralmente el caso seleccionado conservando aquellas partes que son similares y modificando aquellas que no se adaptan. Este proceso de transformación puede realizarse de diferentes formas (sustitución, reglas de transformación, ajuste de parámetros, etc).
2. Derivación: El proceso de derivación consiste en generación una nueva solución mediante la utilización de las mismas operaciones (acciones, reglas, inferencias, etc.) usadas para resolver el problema descrito en el caso seleccionado. Es decir, este proceso intenta transferir por analogía el proceso de razonamiento.

Normalmente estos dos procesos se utilizan de forma independiente, aunque existen algunos enfoques híbridos que utilizan ambos procesos. Realizan una re-instanciación del caso mediante un proceso de transformación y luego se utiliza un proceso de derivación.

Revisión

El proceso de revisión analiza la solución generada con el fin de comprobar si la solución es correcta y puede resolver de forma correcta el problema.

Retención

El proceso de retención es la fase final del razonamiento basado en casos y probablemente uno de los más importantes ya que se encarga del mantenimiento de la base de casos mediante la información extraída por el nuevo problema resuelto. Dependiendo de cómo este construido el sistema de razonamiento se pueden definir 4 fases en el proceso de retención:

1. Evaluación: El proceso de evaluación analiza el nuevo problema e identifica si es posible incluir nuevos casos en la base de casos.
2. Combinación: El proceso de combinación realiza una unión de uno o más casos similares mediante algún tipo de técnica de generalización. El proceso de mezcla suele ser definido en la fase de evaluación y normalmente depende la similitud de los casos y del ratio de crecimiento que se le permita a la base de casos.
3. Indexación: El proceso de indexación inserta el nuevo caso en la base de casos.
4. Eliminación: El proceso de eliminación realiza ciertas acciones de mantenimiento determinando si alguno de los casos almacenados ya no es válido.

2.3 Razonamiento borroso o difuso

Los dos modelos de razonamiento descritos previamente se basan en el concepto de lógica clásica o bivaluada donde la información del mundo es definida mediante proposiciones binarias, normalmente predicados, cuya existencia implica que la proposición es verdadera y sino implica que es falsa. Pero muchas veces la información del mundo real no puede ser modelada mediante proposiciones bivaluadas debido a que no es posible definir muchos de los conceptos del entorno en base a una verdad absoluta o una completa falsedad, por lo que es necesario incluir más valores para representar una proposición extendiendo la lógica clásica a un tipo de lógica superior denominada multivaluada. Debido a esta necesidad, en 1965 se definió el concepto de **lógica difusa** [8] que es un tipo superior a la lógica de primer orden (clásica) que permite definir infinitos valores reales comprendidos entre 0 y 1 para una proposición. Es decir, este tipo de lógica permite la utilización de premisas imprecisas para modelar el conocimiento mediante un grupo de conjuntos interrelacionados entre sí.

2.3.1. Conjuntos borrosos

En la lógica clásica la información es representada mediante conjunto clásicos (crips) donde se asigna el valor 0 o 1 a cada elemento de para indicar si pertenece o no al conjunto. Pero, este concepto de pertenencia se puede generalizar de forma que los valores asignados a los elementos del conjunto caigan en un determinado rango que permita indicar el grado de pertenencia de los elementos al conjunto [9]. Esta función se denomina función de pertenencia μA(x) y el conjunto definido mediante ella se denominada conjunto borroso A. Esta función devolverá un valor entre 0 y 1 para el elemento x que indicará el grado de pertenencia a dicho conjunto. La suma de los grados de pertenencia a todos los conjuntos borrosos relacionados

debe ser siempre 1. La **Figura 4** presenta un conjunto borroso para la definición de la temperatura del entorno formado por 5 posibles conjuntos representados por un variable lingüística (Helado, Frío, Tibio, Templado y Cálido) cada uno de los cuales tienen una función de pertenencia acotada donde la altura representa en la figura representa el grado de pertenencia, siendo los valores posibles entre 0 y 1. Es decir, existen elementos que pueden pertenecer a varios conjuntos a la vez con un grado diferente de pertenencia. Por ejemplo un elemento con un valor de -7 grados puede pertenecer al conjunto Helado y al conjunto Frio con diferentes grados de pertenencia, donde la suma de ambos valores debe ser 1.

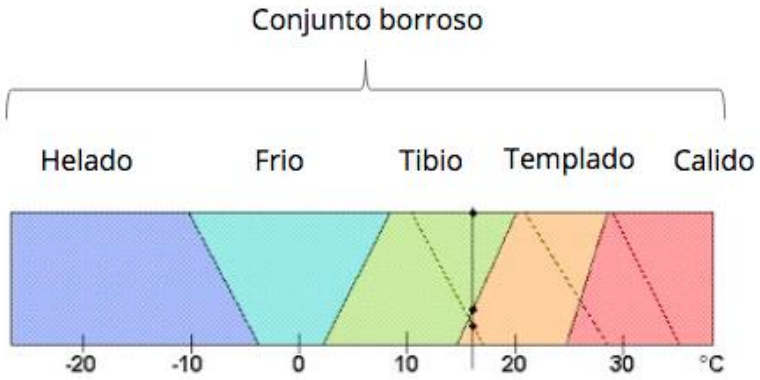

Figura 4: Ejemplo de un conjunto borroso para la definición del temperatura en grados centígrados.

La gran ventaja de los conjuntos borrosos es que son capaces de modelar la información del entorno de forma mucho más precisa ya que permite representar la información mediante una serie infinita no numerable de conjuntos clásicos. Esta forma de representar el conocimiento permite representar concepto subjetivos que hacen más fácil representar el conocimiento humano.

2.3.2. Modo de razonamiento aproximado

El razonamiento aproximado[8] es una técnica de resolución automática de problemas basada en la obtención de conclusiones a partir de información incompleta o con cierto grado de incertidumbre que es modelada mediante la utilización de conjuntos borrosos. Los diferentes sistema basados en razonamiento aproximado funcionan de forma similar a los sistemas basados en reglas con la salvedad de que las reglas de la base de conocimiento pueden utilizar conjuntos borrosos para representar los antecedentes y consecuentes. En este tipo de sistemas es posible inferir conclusiones en situaciones en las cuales aunque los antecedentes no verifiquen de forma plena la regla. Por ejemplo, en el caso de que estemos creando reglas en

las que intervenga como antecedente el conjunto borroso definido en la Figura con un valor de -6 grados centígrados. De forma que el resultado para las funciones de pertenencia sería μHelado(-6) = 0,58 y μFrio(-6) = 0,42. Dados estos valores se podrían disparar reglas que esperasen el valor Helado y Frio. Dependiendo de cómo se modelen este tipo de elementos podemos definir dos tipos de reglas borrosas:

- Reglas de tipo Mamdani: Son reglas borrosas puras, es decir tienen un antecendente definido mediante un conjunto borroso y una consecuente definido como un conjunto borroso.

SI x es Helado **ENTONCES** y es Congelación severa
SI x es Frio **ENTONCES** y es Congelación lebe

- Reglas de tipo Sugeno: Son reglas borrosas parciales, es decir el antecedente es definido mediante un conjunto borroso y el consecuente son valores numéricos lo que implica que que no es necesaria una fase de desborrosificación.

SI x es Helado **ENTONCES** y es F(x)

Los motores de inferencia borrosos (FIS, es sus siglas en inglés) están basados en un proceso secuencial compuesto por cinco fases o procesos: (1) borrosificación de hechos; (2) aplicación del operador borroso en el antecedente; (3) implicación del antecedente al consecuente; (4) la agregación de los consecuentes a través de las normas; y (5) la desborrosificación (opcional) de hechos. Estas etapas son similares a las definidas para un sistema no borroso donde las fases 3, 4 y 5 se podrían corresponder con las fases de identificación, resolución de conflictos y ejecución. Las únicas fases diferentes son aquellas que permiten transformar la información del entorno en conjunto borrosos y viceversas, las cuales son descritas a continuación.

Borrosificador

El borrosificador es un sistema que transforma la información no borrosa almacenada en la base de hechos a sus correspondientes conjuntos borrosos, es decir, se calcula el valor de pertenencia a cada conjunto con el fin de poder aplicar las diferentes reglas. Existen dos tipos de estrategias de borrosificación:

- Singleton. Es el método de borrosificación más utilizado, principalmente en sistemas de control, y consiste en considerar los propios valores discretos con conjuntos borrosos.
- No singleton. En este método de borrosificacion se utiliza una función exponencial para transformar los valores.

El borrosificador transforma la información de la base de hechos para poder disparar las diferentes reglas en las siguientes fases del proceso.

Desborrosificador

El desborrosificador es un sistema que transforma los conjuntos borrosos en valores no borrosos y que son almacenados en la base de hechos. Este proceso de desborrosificación se puede realizar por diferentes métodos:

- Desborrosificador por máximo (Mamdani).
- Desborrosificador por media de centros (Mamdani).
- Desborrosificador por centro de área (Mamdani).
- Desborrosificador por media ponderada (Sujeno). En este caso no se aplica realmente un proceso de borrosificación sino que la propia regla genera el valor numérico.

2.4 Otros tipos de razonamiento

Además de los tres sistemas de razonamiento automático descritos en este documento existe otras formas de razonamiento, como por ejemplo: (1) Los sistemas de razonamiento basados en inferencia bayesiana que guardan ciertas similitudes con los sistema de razonamiento borroso; (2) Los sistemas de razonamiento basados en lógica temporal lineal que incluyen el concepto de tiempo en el proceso de razonamiento; o (3) Los sistemas de razonamiento analógico que intentan razonar mediante analogías sobre las premisas de las reglas.

3. BÚSQUEDA

La **Búsqueda** puede ser definida como el conjunto de algoritmos utilizados para resolver problemas mediante la búsqueda en un espacio de estados por medio de la aplicación de un conjunto de operaciones (acciones). Esta familia de técnicas o algoritmos se basa en **encontrar** una secuencia de acciones, que puede no ser la única, que nos permita transitar desde un estado inicial hasta una estado meta. En la **Figura 5** se presenta un ejemplo de una posible solución

generada por un algoritmo de búsqueda, que consiste en una secuencia de acciones que modifican el estado inicial para “transformarlo” en el estado final.

Figura 5: Representación de la solución a un proceso de Búsqueda

Actualmente existen diferentes tipo de familias de algoritmos de búsqueda que se diferencian entre sí en como la información que representa el entorno es definida y utilizada para conseguir encontrar una solución. En base a estos criterios podemos diferenciar entre diferentes tipo de técnica de búsqueda:

- Búsqueda no informada.
- Búsqueda informada.
- Búsqueda local.
- Búsqueda en tiempo real.
- Búsqueda con contrincantes .

3.1 Conceptos básicos

Antes de comenzar a describir cada uno de los diferentes algoritmos de búsqueda es necesario describir una serie de conceptos básicos comunes a cada uno de las diferentes familias de algoritmos de búsqueda:

- Instancia: Las instancias son las entidades básicas de información que describen el problema. Estos están a su vez compuestas por elementos atómicos que describen los elementos básicos del problema de forma más detallada. Es decir, una instancia es un conjunto finito de elementos atómicos. Las instancias pueden ser utilizadas para definir diferentes facetas del problema, las dos facetas del problemas que son
    - Estado: Las instancias son modeladas como estados, que representan un instante especifico del entorno, al cual se puede llegar tras la aplicación de un operador que genera una variación en los elementos atómicos que componen el estado.

Los estados suelen varias su tamaño tras la aplicaciones de los operadores, ya que estos añaden o eliminan elementos.

- Solución: Las instancias son modeladas como soluciones al problema que se quiere resolver, cada una de estas soluciones puede ser definida mediante la aplicación de una operación que genera variaciones en la instancia. Un problema puede tener un conjunto de posibles soluciones, cada una de las cuales puede ser clasificada en base a algún tipo de concepto de calidad, que nos permita comparar la calidad de dos posibles soluciones. Las soluciones normalmente suelen tener siempre el mismo tamaño, ya que sus operadores sólo modifican el valor de los elementos atómicos que componen la solución, aunque existen técnicas que permiten variar la estructura de la solución.

- Espacio de instancias: El espacio de instancias se corresponde con el conjunto finito de todas las posibles instancias del problema. Los algoritmos de búsqueda de pueden dividir en dos grandes familias dependiendo de que tipo de instancias que se utilicen para modelar el problema:
  - Búsqueda en el espacio de estados: Consiste es buscar una solución en el espacio de estados, donde la solución se corresponde con una secuencia de acciones que modifican la estructura del estado. Comenzando en un estado inicial hasta encontrar un estado meta. El espacio de estado se modela normalmente como un grafo explícito donde los estados son los nodos y los operadores son las arcos que conectan los nodos.
  - Búsqueda en el espacio de soluciones: Consiste en buscar una solución en el espacio de estados, donde la instancia representa una solución problema que debe ser mejorara en base a algún objetivo.
- Operadores: Los operadores son cada una de las operaciones básicas que pueden aplicarse sobre el espacio de instancias para generar nuevas instancias. Para los algoritmos de búsqueda en espacio de estados pueden ser denominados como acciones, ya que son acciones que cambian el entorno.
- Espacio de operadores: El espacio de operadores se corresponde con el conjunto finito de todos los posibles operadores que pueden aplicarse sobre cada una de las instancias con el fin de generar nuevas instancias.

- Algoritmo: El algoritmo se corresponde con una técnica de resolución mediante la cual se van generando las diferentes instancias hasta encontrar una de las posibles soluciones al problema.

3.2 Búsqueda no informada

La búsqueda ciega [10] o no informada es un tipo de búsqueda en el **espacio de estados** donde **no se utiliza ningún tipo de información** del problema para resolverlo. Es decir se utilizan algoritmos de forma para el recorrido de árboles de búsqueda. Debido a que el grafo que representa el espacio de estados se puede transformar en un árbol si el proceso de búsqueda comienza en un determinado estado denominado estado inicial. Este tipo de búsqueda se basa en la aplicación de estrategias sistemáticas para la exploración del espacio de búsqueda, es decir, se aplica una estrategia fija para definir como los diferentes nodos (estados) del árbol son visitados. El proceso de exploración de árbol de búsqueda se basa en dos procesos básicos:

- Generación: Este proceso se corresponde con la generación de un estado tras la aplicación de un operador mediante la inserción y/o eliminación de ciertos elementos atómicos.
- Expansión: Este proceso se corresponde con la expansión de un estado generado previamente. Es decir, consiste en la aplicación de todos los posible operadores aplicables sobre el estado, de forma que se genere un conjunto de estados igual al número de operadores aplicables. Los estados generados por el proceso de expansión se denominan sucesores (hijos), ya que son los estados generados a partir de un estado que es denominado padre. Este estado expandido se considera como un estado visitado del espacio de búsqueda.

El proceso de exploración consiste en una ejecución secuencial de los dos procesos (expansión y generación) hasta que se expanda un estado meta. El proceso de expansión se ejecuta tras la comprobación de que el estado a expandir no es una meta del problema. De forma que este tipo de algoritmos se pueden considerar como **algoritmos exhaustivos**, ya que van explorando el espacio de estados de forma exhaustiva hasta encontrar una solución. Esto, puede dar lugar a situaciones en el cual se tengan que generar todos los estado del problema para encontrar una solución, lo que puede hacer que el coste de encontrar una solución sea prohibitivo para problemas muy complejos. Existen, básicamente, dos estrategias para explorar el espacio de estados, en profundidad y en amplitud.

En la Figura, se presentan un ejemplo del árbol de búsqueda generado para resolver un problema del 8-puzzle. En este caso, se ha aplica un algoritmo de búsqueda en amplitud, donde el espacio de estados se van explorando por niveles de profundidad. Es decir, dado el nivel 0, compuesto por el estado inicial, se generan cada uno de sus sucesores generando el nivel 1 del árbol de búsqueda. A continuación se realiza la expansión de cada uno de los nodos (estados) del nivel 1 siguiendo el orden en el cual han sido generados. En este caso la solución se encuentra tras expandir uno de los nodos del nivel 5 del árbol de búsqueda.

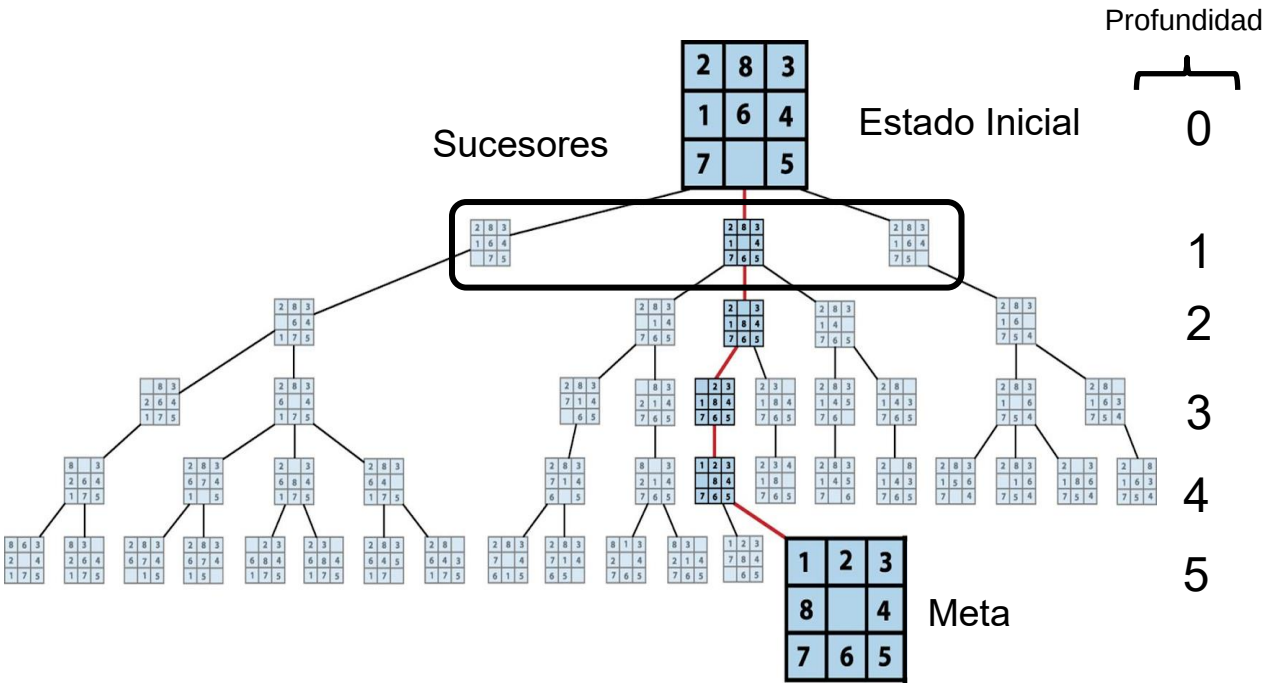

Figura 6: Ejemplo de búsqueda en amplitud para el 8-Puzzle

Los diferentes algoritmos de búsqueda no informada utilizan ciertas estructuras de datos para conocer que nodos deben ser expandidos a continuación. Estas estructuras de datos son diferentes dependiendo de si los diferentes nodos (estados) del espacio de estados pueden ser generados a diferentes niveles. Es decir, dependiendo si el algoritmo de búsqueda esta recorriendo un grafo o un árbol. En base a estos, los algoritmos de búsqueda no informada pueden utilizar dos estructuras de datos:

- Lista abierta: Esta estructura se corresponde con una pila o cola de estados dependiendo de la estrategia que se utilice para expandir los nodos que son introducidos. Los nodos

insertados en esta estructura se denominan **abiertos**, debido a que han sido generados, pero no expandidos.

- Lista cerrada: Esta estructura se corresponde con una lista enladada un una tabla hash, donde se almancenan los nodos que han sido expandidos con el fin de comprobar si un nodo ha sido generado previamente. Los nodos insertados en esta estructura se denomina **cerrados**, debido a que han sido visitados o expandido previamente. Es decir, sus sucesores ya han sido generados por el proceso de búsqueda.

3.2.1. Algoritmo de búsqueda en amplitud

El algoritmo de búsqueda en amplitud (Breadth First Search, BFS, en sus siglas en inglés)[11] se basa en expandir los nodos por niveles de profundidad. Es decir, cada nodo tiene un valor que se corresponde con su nivel de profundidad en el árbol de búsqueda, de forma que su nivel se corresponde con el nivel del padre (nodo generador) más 1. Este tipo de proceso de exploración viene definido en parte por la estructura de datos utilizada para definir la lista abierta. En este caso la lista abierta del algoritmo de búsqueda en amplitud se corresponde con una Cola (First in, First Out, FIFO en sus siglas en inglés) de forma que los sucesores del siguiente nivel son extraídos de la cola (expandidos) en el orden en el que fueron insertados.

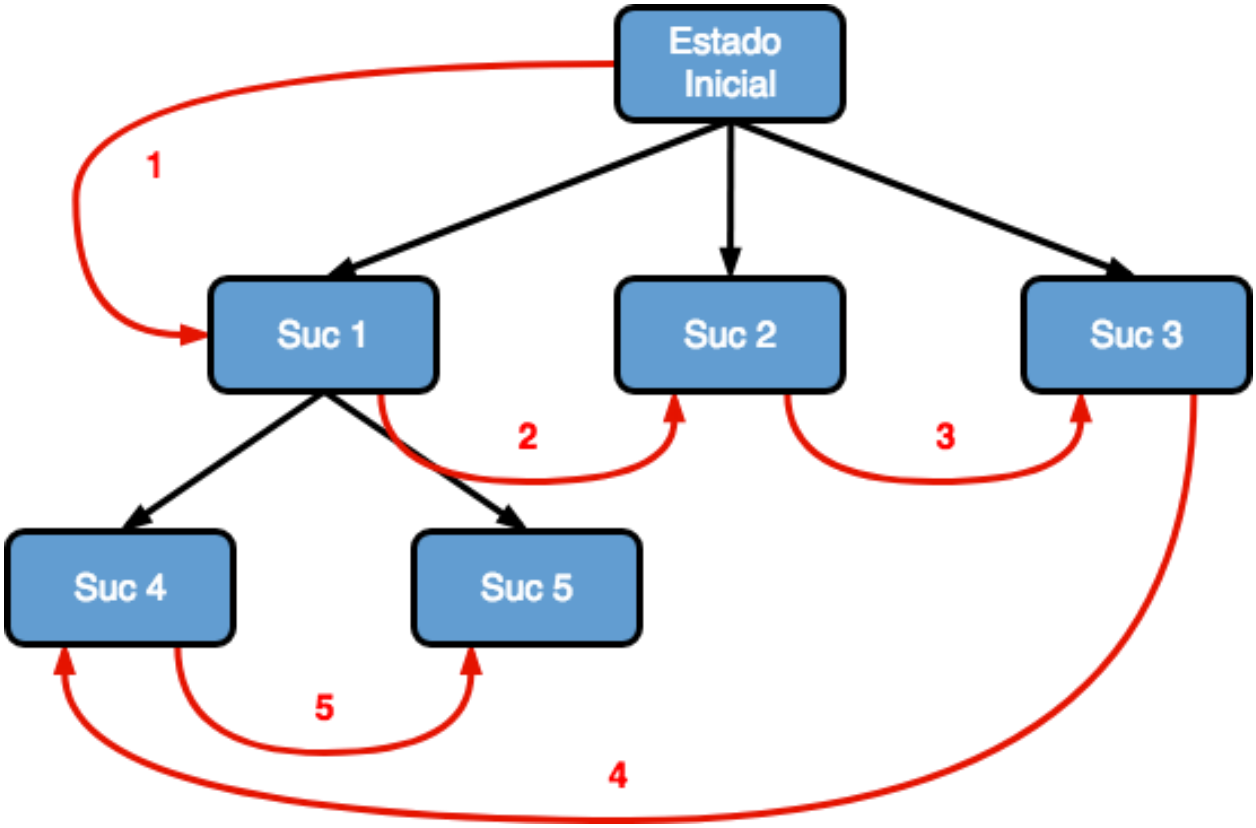

Figura 7: Ejemplo del funcionamiento de un algoritmo de búsqueda en amplitud

La **Figura 7** presenta un ejemplo del proceso de funcionamiento de un algoritmo en amplitud. Como se puede observar el proceso de generación se corresponde con las flechas negras de izquierda a derecha y el proceso de expansión con las flechas rojas. Este algoritmo presenta una **gran ventaja a nivel de tiempo**, si la solución se encuentra niveles de profundidad bajos, pero presenta una **gran desventaja a nivel de memoria**, ya que en situaciones donde cada nodo tiene un máximo de dos sucesores, el número de nodos en la lista para cada nuevo nivel crece exponencialmente, lo que puede suponer que el algoritmo no tenga memoria suficiente para encontrar la solución.

3.2.2. Algoritmo de búsqueda en profundidad

El algoritmo de búsqueda en profundidad (Deep First Search, DFS, en sus siglas en inglés)[11] se basa en expandir los nodos por niveles de profundidad. Es decir, tras expandir un nodo (estado) del espacio de estados, se expande el primero de sus sucesores. De forma que el proceso de exploración va generando nodos en profundidad hasta que no es posible generar más sucesores de un nodo. Este tipo de proceso de exploración viene definido en parte por la

estructura de datos utilizada para definir la lista abierta. En esta caso la lista abierta del algoritmo de búsqueda en profundidad se corresponde con un Pila (Last in, First Out, LIFO en sus siglas en inglés) de forma que los sucesores de un nodo son insertados en orden inverso de generación. Siendo el último de los sucesores generados el siguiente nodo a expandir.

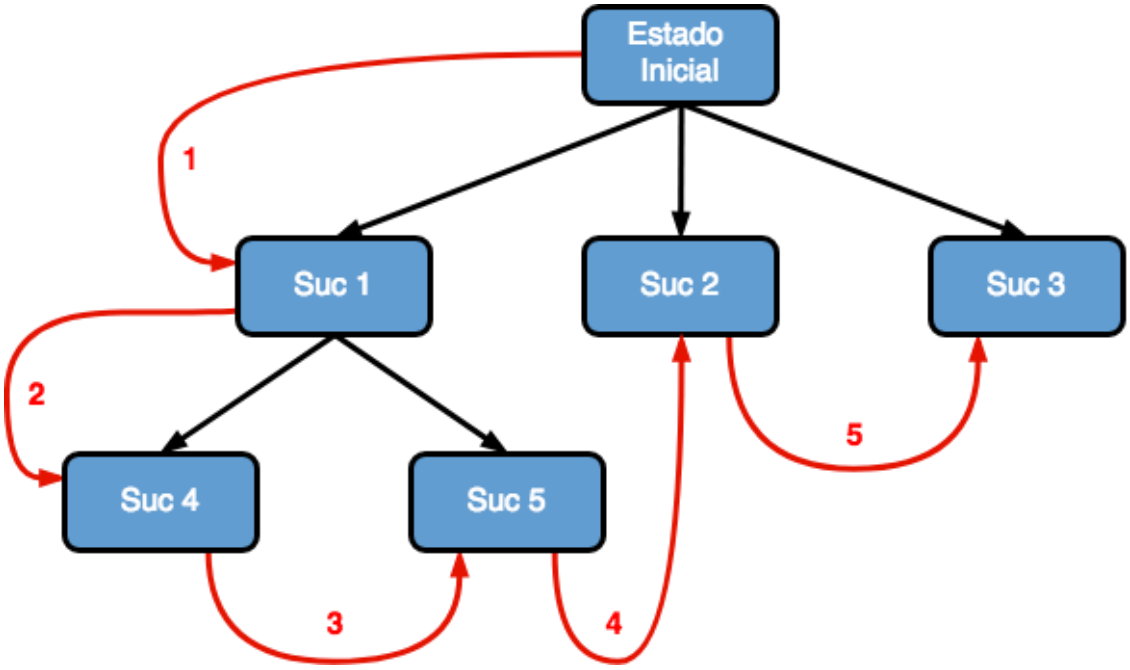

Figura 8: Ejemplo de funcionamiento de un algoritmo de búsqueda en profundidad

La **Figura 8** presenta un ejemplo del proceso de funcionamiento de un algoritmo en profundidad. Como se puede observar en la figura, el proceso de generación se corresponde con las flechas negras de derecha a izquierda y el proceso de expansión con las flechas rojas. En este algoritmo el proceso de expansión profundiza en el árbol de búsqueda de forma recursiva hasta que no se pueden generar más sucesores. A continuación el algoritmo vuelve hacia atrás (Backtraking) de modo que se repita el mismo proceso para cada uno de los nodos hermanos del último nodo expandido. Este algoritmo presenta una **gran ventaja a nivel de memoria** ya que necesita una cantidad mínima de espacio para el almacenamiento de los nodos generados aunque presenta una **desventaja a nivel de tiempo**, ya que si la solución se encuentra a baja profundidad muy a la derecha del árbol de búsqueda se tardará mucho si la profundidad del árbol del búsqueda es muy elevada. Debido a su naturaleza recursiva, sólo es necesario almacenar los nodos generados durante el camino de búsqueda que aún no han sido expandidos, lo cual disminuye el espacio de almacenamiento, pero incrementa la complejidad del proceso de búsqueda de repetidos, haciéndolo imposible en algunas situaciones.

Con el fin de disminuir el coste temporal del proceso de búsqueda, este algoritmo puede ser modificado incluyendo un límite de profundidad iterativo, de forma que el algoritmo sólo busca hasta una determinada profundidad máxima. De forma que sólo se puedan expandir nodos hasta una determinada profundidad. Aunque esta variación puede disminuir el tiempo de búsqueda, puede llevar a situaciones en las cuales no se encuentre una solución al problema, debido a que esta se encuentre a mayor profundidad.

3.2.3. Algoritmos de búsqueda iterativa

Los dos algoritmos descritos anteriormente pueden ser combinados con el fin de unificar la ventajas de ambos y evitar algunos de sus problemas que presenta. Su combinación da lugar a dos nuevos algoritmos de búsqueda ciega o no informada:

- El algoritmo de profundidad iterativa (Iterative Deep First Search, IDFS en sus siglas en inglés) consiste en realizar búsqueda en profundidad repetidamente sobre un árbol de profundidad incremental. Es decir primero se realiza una búsqueda a profundidad 1, luego a profundidad 2, luego en profundidad 3 y así sucesivamente hasta que se encuentra una solución. El incremento en el nivel de profundidad no tiene por que ser lineal, se puede hacer por ejemplo un incremento exponencial.
- El algoritmo de profundidad iterativa (Iterative Breath First Search, IBFS en sus siglas en inglés) consiste en realizar búsqueda en profundidad repetidamente sobre un árbol de anchura incremental. Es decir primero se realiza una búsqueda a profundidad con anchura 1, luego con anchura 2, luego con anchura 3 y así sucesivamente hasta que se encuentra una solución. Al igual que en el caso anterior, el incremento en el número de succesores (anchura) no tiene por que ser lineal, se puede hacer por ejemplo un incremento exponencial.

3.2.4. Algoritmo de búsqueda bidireccional

La búsqueda bidireccional (Bidirectional Search, BS en sus siglas en inglés)[12] consiste en realizar dos búsqueda simultaneas. Una desde el estado inicial hasta el estado meta y otra desde el estado meta hasta el estado inicial. En este algoritmo, el proceso de búsqueda finaliza cuando ambos procesos expanden el mismo estado intermedio. A pesar de que parece un algoritmo muy útil, presenta dos importante problemas:

- Definición del estado meta: En algunos problemas es imposible definir un único estado meta, debido a que existen un conjunto de posibles estados metas. Esto hace que sea

necesario definir una función de generación de estados metas y otra que elija que estado meta utilizar en base a algún criterio.

- Definición de la función de identificación: Para la utilización de este algoritmo es necesario definir un función de identificación que permita identificar aquellos estados que han sido generados por ambos algoritmo con el fin de parar el algoritmo de búsqueda y construir la solución. Este proceso implica nuevas estructuras de datos y procesos de comprobación de estados en ambos algoritmo. Por ejemplo, si los algoritmos son ejecutados de forma simultanea en diferentes hilos de ejecución, estos deben esperarse mutuamente cuando se hacen las comprobaciones entre los nuevos estados expandidos. En algunos casos, este proceso puede aumentar la complejidad del algoritmo.

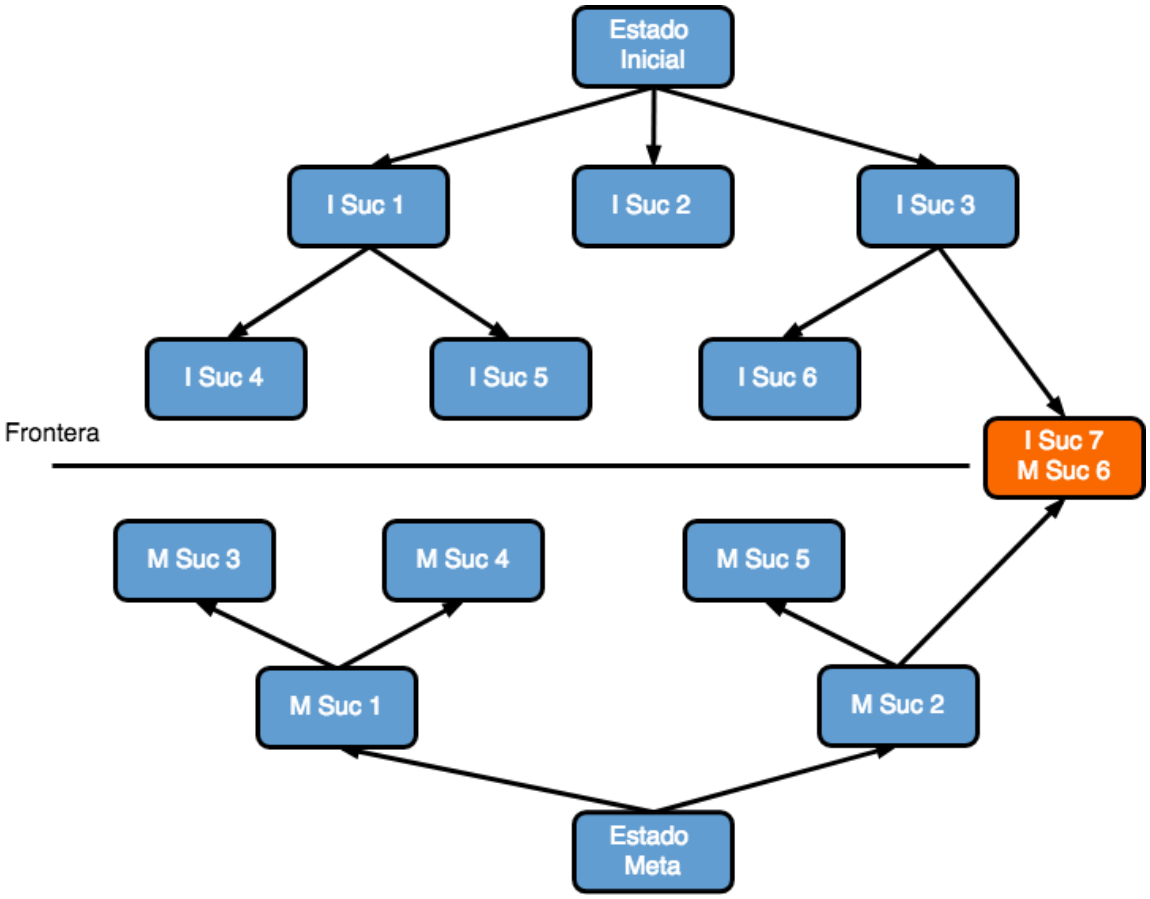

Figura 9: Ejemplo de funcionamiento de un algoritmo de búsqueda bidireccional

En la **Figura 9** se presenta un ejemplo del funcionamiento de un algoritmo de búsqueda bidireccional. En este caso tenemos dos algoritmos que van generado nodos de forma simultanea hasta que ambos generan el mismo nodo. En este ejemplo, se puede observar que el séptimo sucesor de la búsqueda iniciada en el estado inicial se corresponde con el sexto sucesor de la búsqueda comenzada en la meta. Cada vez que se genera un sucesor en uno de los dos nodos se debe comprobar si esta nodo ha sido generado previamente por la búsqueda contraria. Normalmente estos algoritmos utilizan una lista cerrada compartida con el fin de detectar nodos repetidos que se corresponde con la frontera de ambos algoritmos y que pueden ser utilizados para generar una solución. Normalmente los dos algoritmos utilizados en la búsqueda suelen ser del mismo tipo, pero existen aproximaciones que utilizan diferentes tipo de algoritmos.

Los cinco algoritmos de búsqueda no informada descritos en este capítulo son los ejemplos más importante de esta familia de algoritmos de búsqueda, se les considera como los algoritmos clásicos de búsqueda no informada. Aunque existe muchas otras versiones que incluyen mejoras para optimizar su funcionamiento y resolver algunos de los problemas que presentan. Algunos ejemplos son la búsqueda mediante islas, que es una mejora sobre la búsqueda bidireccional; la búsqueda de coste uniforme, donde los nodos son expandidos en base al coste de las acciones (las acciones tienen coste no unitario) que se usan para generar los nodos.

3.3 Búsqueda informada

La búsqueda informada o heurística [13] es un tipo de búsqueda en el **espacio de estados** donde se utiliza algún tipo de información del problema con el fin de minimizar el coste de encontrar una solución mediante la introducción de una función que guía el proceso de búsqueda. Esta función guía, denominada **función heurística**[14], calcula de forma estimada el coste que supone alcanzar el nodo meta (solución). Esta función sirve para definir el orden en que los nodos generados son almacenados en la lista abierta, de forma que los nodos con menor coste serán los primeros en ser expandidos. De forma que el orden de expansión de los estados ya no se realizará de forma sistemática en base a su posición en el árbol de búsqueda, sino mediante el coste calculado por la función heurística. Este tipo de algoritmos fueron creados con el fin de minimizar el coste de búsqueda de los algoritmos no informados en problemas con un espacio de estados extremadamente grande, así como para asegurar la obtención de la solución óptima en el caso de que existieran múltiples soluciones a un problema. Por ejemplo, imaginemos que

queremos resolver un problema en el juego del Ajedrez, el tamaño el espacio de estados es de 10^{47} estados. En el hipotético caso de que pudiéramos generar 3 billones de nodos por segundo y teniendo un espacio de memoria infinito, se tardarían 1030 años en expandir de forma completa el espacio de búsqueda. Probablemente encontrar un solución, no implicaría expandir de forma compleja el espacio de búsqueda, pero tal vez supondría alrededor de 100 años en el caso de que tuviéramos muchas suerte. Pero, el proceso de búsqueda se puede complicar mucho más, si queremos obtener la **solución óptima** de entre las diferentes posibles soluciones al problema.

> *La **solución óptima** a un problema es una **solución** factible que da el valor más favorable de la función objetivo. En el caso de la búsqueda informada se puede definir la solución óptima cómo aquella solución con el menor coste.*

Para poder diferenciar la calidad de las distintas soluciones de un problema y conseguir encontrar la solución óptima, es necesario introducir el concepto de coste. Para poder utilizar el concepto de coste, es necesario introducir un coste asociado a cada uno de los operadores generando dos posibles tipos de problemas de búsqueda informada:

- Búsqueda con costes unitarios: Este es el tipo de búsqueda más sencilla y asume que el coste de todas las acciones es igual a 1.
- Búsqueda con costes no unitarios: Este es el tipo más complicado y cada uno de los operadores tiene un coste diferente. Este coste puede ser global al tipo de operador o depender de algún tipo de símbolo que representa alguna característica del entorno y que puede ir variando en base a los operadores seleccionado durante el proceso de búsqueda incrementando la complejidad del proceso de búsqueda.

Como describimos anteriormente, la función heurística h(n) se puede definir como una función que permite estimar el coste (comúnmente el número de acciones que deben ser ejecutadas) desde un determinado estado hasta un estado meta (solución). La calidad de esta función es muy importante, ya que puede suponer la diferente entre encontrar o no una solución a un problema. Una función heurística con poco conocimiento del problema o con un conocimiento erróneo podría guiar la búsqueda a regiones del espacio de búsqueda donde no existe una solución, produciendo un proceso de búsqueda peor que si estuviéramos resolviendo le problema mediante búsqueda no informada. En cambio, si tuviéramos conocimiento perfecto del

problema, podríamos guiar la búsqueda hacia la solución descartando todos aquellos caminos con mayor coste, eligiendo en cada expansión el estado de menor coste, y encontrando finalmente la solución óptima en tiempo lineal.

El proceso de definición de funciones heurísticas es muy complejo incluso cuando se tiene conocimiento perfecto del problema y más aún si se intenta realizar de forma automática. Una de las propiedades más importantes a la hora de definir una función heurística es la admisibilidad. Se dice que una función heurística h(n) es admisible si y sólo si el coste para cada uno de los estados del espacio de estados es siempre menor o igual al coste real. Es decir, una función heurística es admisible si nunca sobreestima el coste de alcanzar un estado meta. Algunos ejemplos de funciones heurísticas ampliamente utilizadas son:

- Distancia euclidea: Es la distancia "ordinaria" entre dos puntos de un espacio euclídeo, la cual se deduce a partir del teorema de Pitágoras. Este tipo de función heurística es muy utilizada en problemas en los que hay que encontrar el camino entre dos puntos en un espacio bidemensional.
- Distancia de Manhattan: Es la distancia entre dos puntos calculada como la suma de las diferencias (absolutas) de sus coordenadas. Al igual que la anterior, este tipo de función heurística es muy utilizada en problemas en los que hay que encontrar el camino entre dos puntos en un espacio bidimensional.

Figura 10: Ejemplo de cálculo del coste mediante la utilización de diferentes funciones heurísticas

En la **Figura 10**, se presenta un ejemplo real sobre el callejero de la ciudad de Madrid. De forma que queremos estimar el coste desde el punto en la parte inferior hasta el punto en la parte superior del mapa. Para ellos vamos a utilizar los dos ejemplos de funciones heurísticas descritas anteriormente. En este caso, la distancia euclídea se corresponde con la línea roja, cuyo coste es admisible ya que es menor que el coste real. Mientras que la distancia de Manhattan puede ofrecer varios posibles valores debido a que existen múltiples maneras de conectar los dos puntos del mapa. En este caso la distancia de Manhattan viene dada por los caminos azul y verde. Como podemos observar en la imagen, la distancia euclidea parece que calcula un valor menor, pero tal vez la distancia de Manhattan ofrece un valor más cercano a la realidad, que puede mejorar el proceso de búsqueda.

3.3.1. Algoritmo de búsqueda Voraz

El algoritmo Voraz o Avaricioso (Gready Best First Search, GBFS en sus siglas en inglés)[11] se basa en la ordenación de los nodos generados en base al valor calculado por la función heurística h(n). En este caso, la función h(n) calcula el coste de aplicar la siguiente acción sobre el nodo n, de forma que aquellos nodos que tengan un coste menor se colocan al principio de la lista abierta.

Se podrías decir que este algoritmo explora siempre primero aquellos nodos cercanos al nodo n que tienen menor coste.

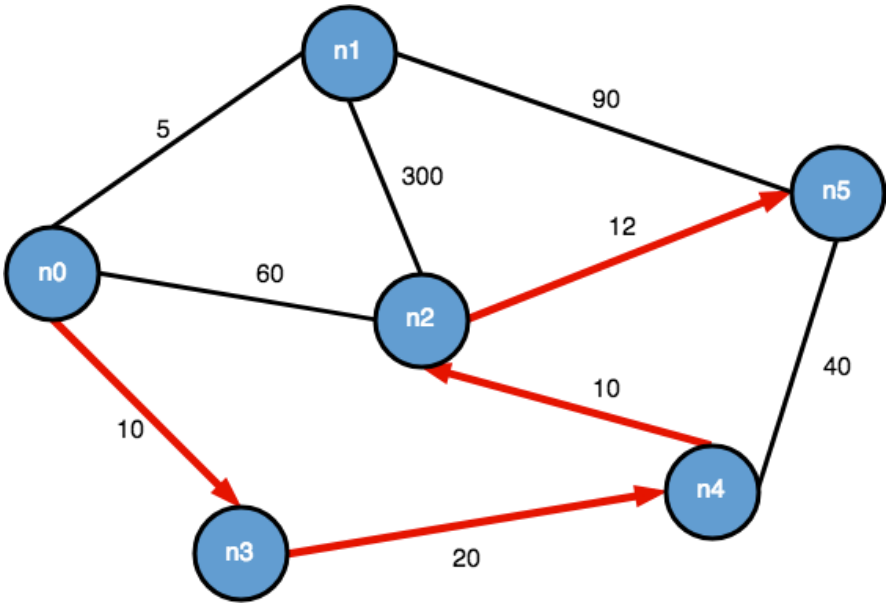

Figura 11 - Ejemplo de aplicación del Algoritmo Voraz

En la Figura 11, se presenta un ejemplo de cómo se aplicaría el algoritmo voraz. En este caso, el algoritmo utilizaría el valor inmediato de los sucesores. Para el nodo n0, se generarían tres nodos sucesores (n1, n2, y n3) cuyos valores heurísticos serían respectivamente (5, 60 y 10). De forma que el algoritmo voraz elegiría el nodo n1 debido a que su valor heurístico es h(n1) = 5. A continuación, el algoritmo descartaría el nodo n1, debido a que su sucesores tienen valores heurísticos muy grandes, de forma que el siguiente nodo a expandir sería el nodo n3, que nos llevaría directamente a la solución (n0 → n3 → n4 → n2 → n5).

3.3.2. Algoritmo de búsqueda Mejor Primero

El algoritmo de Mejor Primero (Best First Search, BFS en sus siglas en inglés) [12] se basa en la ordenación de los nodos generados en base al valor calculado por la función heurística h(n) que calcula una estimación del coste entre el estado n y uno de los posibles estados meta. Los nodos generados durante el proceso de expansión son insertados en una lista ordenada por coste de menor a mayor, donde los nodos con mejor coste se encuentran en la cabecera de la lista y los nodos de mayor coste se encuentran en la cola. Este algoritmo funciona de forma similar a un algoritmo en amplitud (BFS) pero incluyendo un proceso de ordenación de la lista abierta tras

cada inserción. De forma que este algoritmo siempre va a expandir en primer lugar aquellos estados con menor coste.

3.3.3. Algoritmo de búsqueda A*

El algoritmo A*[15] se basa en la ordenación de los nodos generados en base al valor calculado con la función de evaluación denominada f(n), es decir funciona de forma similar a una algoritmo de Mejor Primero pero utilizando otro tipo de función de evaluación. Los algoritmos descritos anteriormente sólo utilizan diferentes tipos de función heurística para evaluar los nodos generados y elegir aquel con menor coste con el fin de acercarse a la solución. Pero en algunos casos es necesario elegir un nodo con mayor coste con el fin de conseguir una solución mejor o encontrar una solución que puede que no está disponible eligiendo el nodo de menor coste. El algoritmo A* introduce un nuevo tipo de función de evaluación con el fin de calcular de forma más eficiente el coste real de la solución cuando un determinado nodo. Esta función de evaluación combina dos funciones:

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

- Una función de coste g(n) que se corresponde con el coste del camino explorado, que se corresponde con el coste real hasta el nodo n que ha sigo generado y que consiste el sumar el coste del operador que ha generado el nodo al coste del nodo padre. Este valor es un valor perfecto y se calcula mediante un función del tipo g(n).
- Una función heurística h(n) que calcula el coste del camino inexplorado, que se corresponde con el coste desde el nodo n hasta el nodo meta. Este coste debe ser calculado utilizando algún tipo de información específica del problema a resolver. Este valor se corresponde con una estimación del coste futuro para resolver el problema.

3.4 Búsqueda local

La búsqueda local es un tipo de búsqueda en el **espacio de soluciones** donde **se utiliza una función de evaluación o fitness** que nos permite evaluar la calidad de cada solución. Es decir, este tipo de algoritmos generan una solución inicial de forma aleatoria e intentan optimizarla mediante la aplicación de una serie de operaciones con el fin de mejorar la calidad de la solución con respecto a un conjunto de objetivos que tienen que ser maximizados o minimizados (existen

algoritmos que optimizan múltiples soluciones de forma simultaneas). Estas operaciones dan lugar a la generación de nuevas soluciones que espacialmente se encuentran muy próximas a la solución utilizada para generarlas (las diferencias desde el punto de vista de la representación de la solución son muy pequeñas).

El proceso de funcionamiento de este tipo de algoritmos se puede definir como un proceso de **optimización matemática** donde los mejores individuos (soluciones) son seleccionados de un conjunto de posibles individuos en base a un determinado criterio. En este caso los algoritmos de búsqueda local utilizan esos individuos (población) para generar individuos de mejor calidad con el fin de encontrar la solución óptima o acercarse a ella lo más posible.

3.4.1. Algoritmos genéticos

Los algoritmos genéticos[16][17] es un método de optimización matemática basado en el proceso evolutivo de los organismos vivos. Este proceso se basa en los principios de la selección natural enunciados por Darwin en 1859, donde los diferentes individuos de una especie mejoran sus características mediante un proceso continuo de evolución a través del cruzamiento de individuos y la aparición de mutaciones aleatorias. Este proceso puede ser modelado como un algoritmo de búsqueda local mediante una población de individuos (instancias), cada uno de los cuales representa una solución factible del problema a resolver con un determinado factor de calidad. En base a este factor de calidad los individuos son combinados (cruzados) entre sí para generar una nueva población de individuos que remplace a la actual. De forma que aquellos individuos con mayor calidad tendrán una mayor probabilidad de ser elegidos y aquellos con menor calidad no serán elegidos para el proceso de combinación. Además, con el fin de evitar la falta de variabilidad de los nuevos individuos se introducen variaciones aleatorias sobre los individuos mediante mutaciones. El proceso de búsqueda es guiado por la calidad de los individuos favoreciendo el cruzamiento de los mejores individuos de cada población produciendo una exploración de las zonas más prometedores del espacio de búsqueda siempre convergiendo hasta la solución óptima del problema siempre y cuando la función que calcula la calidad de un individuo sea capaz de calcular este valor con precisión.

El funcionamiento de los algoritmos genéticos se puede describir mediante el más simple de sus variaciones denominado Algoritmo Genético Canónico cuyo pseudocódigo se presenta en la **Figura 12**. Este algoritmo consiste en un bucle de evolución que es iniciado con la generación de la población aleatoria de individuos cuya calidad es calculada mediante la función de fitness. A continuación, se ejecutarán un conjunto de episodios de evolución donde los individuos con

mejor calidad son seleccionados de forma aleatoria de dos en dos para ser cruzados generando dos nuevos individuos, sobre cada uno de los cuales se aplicará un operador de mutación. El resultado de cada uno de los episodios de evolución consistirá en una nueva población de individuos que será utilizada en el siguiente episodio de evolución. El problema global de evolución finalizará cuando se consiga alcanzar el criterio de parada. El objetivo del algoritmo es conseguir la solución óptima, pero los algoritmos genéticos no aseguran la obtención de la solución óptima. El objetivo suele ser ejecutado un número máximo de episodios de evolución.

```
BEGIN
        Generar una población inicial de individuos.
        Computar la funcion de evaluacion de cada individuo.
        WHILE NOT terminar DO
                BEGIN
                FOR i := 0 HASTA i = poblacion/2 DO
                        BEGIN
                        Seleccionar dos individuos.
                        Cruzar los dos generando dos nuevos individuos.
                        Mutar los nuevos individuos.
                        Calcular el fitness de los dos nuevos individuos.
                        Insertar los nuevos individuos en la nueva población.
                        END
        IF parada THEN
                terminar := TRUE
        END
END
```

Figura 12: Pseudocódigo de un algoritmo genético canónico

Definición del individuo

Un elemento muy importante a la hora de utilizar algoritmos genéticos consiste en definir la codificación de los individuos que representan las posibles soluciones del problema. La codificación de los individuos es la estructura mediante la cual se representan las soluciones al problema y que toma su estructura y nomenclatura de los componentes básicos mediante los cuales están construidos los seres vivos. Las soluciones (denominadas cromosomas) son representado por una secuencia de parámetros o variables (que son denominados genes). Los valores de los parámetros son definidos mediante un alfabeto que normalmente está formado por dos elementos (0 y 1) aunque también es común ver alfabetos con más de dos elementos. Pero en ambos casos los elementos del alfabeto son numéricos. Cada uno de estos individuos es evaluado mediante una función de **fitness** que calcula la calidad del individuo con respecto a un determinado objetivo. Esta función es especifica para cada problema y siempre devuelve un valor decimal, normalmente entre 0 y 100.

Operaciones

Las operaciones se corresponde con acciones que se aplican para generar nuevos individuos. Estas operaciones se basan, al igual que los demás conceptos de los algoritmos genéticos, en la teoría genética. Las dos operaciones básicas son cruzamiento y mutación:

Cruzamiento

Es la operación mediante la cual se combinan las variables que componen los cromosomas de dos individuos para generar uno o dos nuevos individuos. El proceso de cruzamiento permite combinar las variables de los cromosomas de diferente manera. La más común es el cruzamiento simple que consiste en cortar los cromosomas de ambos individuos por un único punto y los grupos de genes se intercambian generando así dos nuevos individuos, como se muestra en la **Figura 13**. Este tipo de proceso puede modificarse utilizando múltiples puntos de corte.

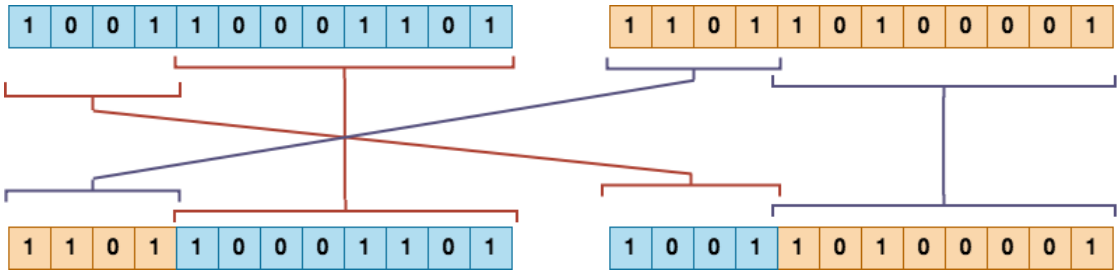

Figura 13: Ejemplo de proceso de cruzamiento

Aunque es posible realizar multiples cortes o incluso en algunos casos incluir más de dos individuos para el proceso de cruzamiento.

Mutación

Es la operación mediante la cual se aplicación algún tipo de variación en el valor de ciertas variables del cromosoma de forma aleatoria. Comumente, el operador de mutación modifica el valor de n variables elegidas de manera aleatoria como se puede observar en la **Figura 14**, donde las variables 3, 5, 8, 9 y 10 has sido modificadas.

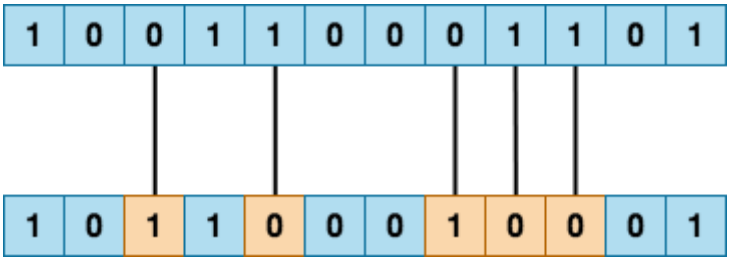

Figura 14: Ejemplo de proceso de mutación

Existen múltiples variaciones de los algoritmos genéticos, los cuales incluyen otros tipo de operadores, así como diferentes mecanismos de selección de individuos o generación de mutaciones. Además existen diferentes tipos de algoritmos multi-objetivo basados en algoritmos genéticos los cuales intentan buscar una solución optimizando múltiples objetivos que tienes que ser maximizados o minimizados. Esto implica que tendremos múltiples funciones de fitness, una para cada objetivo.

3.4.2. Búsqueda tabú

El principal problema de los algoritmos de búsqueda local es su facilidad para queda atrapado en ciertas zonas del espacio de búsqueda debido a que están continuamente generando soluciones similares de forma cíclica. Estas zonas son partes del espacio de búsqueda donde existe una solución de muy buena calidad, pero alejada de la solución óptima, que una vez alcanzada no permiten al algoritmo escapar mediante la generación de nuevas soluciones. Se denominan máximos o mínimos locales dependiendo del tratamiento que se les estén dando a los objetivos del problema (maximizar o minimizar). En la **Figura 15** se presenta un ejemplo de superficie de una función objetivo para un problema de optimización donde existe un máximo local donde el proceso de búsqueda local puede quedar atrapado antes de acercarse a la solución óptima. Con el fin de evitar este tipo de situaciones surgieron los algoritmos de búsqueda tabú.

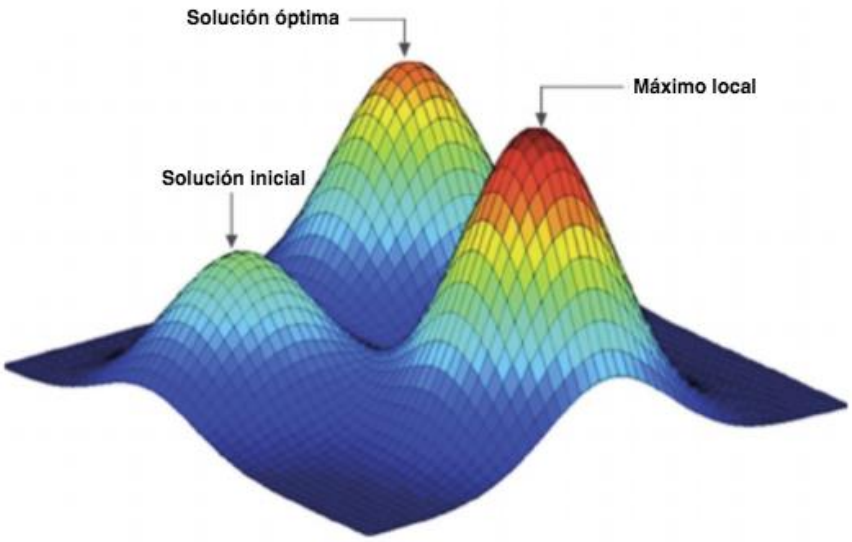

Figura 15: Superficie de una función objetivo de un problema de optimización combinatoria

Este tipo de algoritmos son un método de optimización matemática basados en la utilización de una estructura de memoria que almacena las soluciones generadas previamente con el fin de que el algoritmo de búsqueda no vuelva a visitar o generar estas soluciones evitando quedarse atrapados en un mínimo o máximo local. Para ello se utiliza un procedimiento de búsqueda por vecindades que permite generar nuevas soluciones mediante la utilización de una función de vecindad. Esta función permite generar soluciones que pertenecen a la misma vecindad a partir de otra solución, de forma que cada solución puede pertenecer a múltiples vecindades. Con el fin de poder explorar la mayor cantidad de regiones del espacio de búsqueda se utiliza la estructura memoria que modifica el resultado de la función de vecindad para cada solución a medida que la búsqueda progresa. Es decir, elimina soluciones que ya han sido exploradas previamente con el fin de no quedar atrapado en una zona del espacio de búsqueda. La estructura de memoria puede utilizarse de dos maneras diferentes:

- Memoria a corto plazo: La información insertada en la lista tabú solo se mantiene almacenada durante n iteraciones, es decir para cada solución se indica el número de iteración en la que fue insertado eliminándose al pasar n iteraciones. De esta manera el tamaño de la lista tabú está siempre controlada.
- Memoria a largo plazo: Todas las soluciones generadas son almacenadas por la lista tabú, lo que implica un gasto excesivo de memoria y aumenta el tiempo a la hora de detectar soluciones previamente insertadas cuando el tamaño de la lista es muy elevado.

3.5 Otros tipos de técnicas basadas en búsqueda

3.5.1. Además de las diferentes técnicas de búsqueda que hemos descrito, existen otros tipos de técnicas basadas en búsqueda cuya complejidad es superior debido a ciertas simplificaciones aplicadas en base a ciertos factores como un tiempo de búsqueda limitado, un mayor número de agentes de búsqueda, etc.

3.5.2. Búsqueda en tiempo real

Los algoritmos de búsqueda informada descritos en la sección 3.3 de este documento tienen ciertas limitaciones cuando el espacio de búsqueda es extremadamente grandes como por ejemplo en problemas de navegación o los videojuegos donde es necesario obtener una solución en tiempo limitado. Actualmente existen dos grupos de algoritmos que son capaces de resolver problemas con estas características.

- Algoritmos incrementales: Son algoritmos de búsqueda que reutilizan información de búsquedas previas para encontrar soluciones de forma mucho más rápida a problemas similares de mayor complejidad que resolviendo del problema desde el estado inicial [19].
- Algoritmos en tiempo real: Son algoritmos de búsqueda tradicionalmente utilizados para resolver problemas relacionados de complejidad muy elevada que deben interactuar con el entorno en tiempo real, para ello combinan una fase de planificación (búsqueda) y una fase de ejecución. En la fase planificación realizan una búsqueda en profundidad limitada para definir los siguiente n movimientos (acciones) mientras que en la fase de ejecución se realiza la ejecución del primer movimiento de la secuencia de acciones recogiendo información real del entorno que es utilizada en la siguiente fase de planificación. Los algoritmos más utilizados de este tipo son: (1) Real-Time A* (RTA* es sus siglas en inglés)[20]; y (2) Learning Real-Time A* (LRTA* en sus siglas en inglés)[20].

3.5.3. Búsqueda con adversario

Los algoritmos de búsqueda con adversario se utilizan para generar soluciones en entornos donde el proceso de búsqueda implica al menos dos agentes cuyas acciones producen variaciones en el entorno. Los algoritmos más comunes están basados en dos jugadores donde el proceso de búsqueda simula los turnos de los jugadores mediante los niveles del árbol de búsqueda. Los algoritmos más utilizados de este tipo son:

- MinMax: Es un algoritmo de búsqueda con adversario para dos jugadores (Max o Min) que busca minimizar la pérdida máxima esperada utilizando información perfecta del entorno. El algoritmo utiliza dos jugadores (Min o Max) cada uno de los cuales realiza movimiento independientes en un nivel del árbol, de que para el jugador Max se seleccionará la jugada con la mejor puntuación posible y para el jugador Min se seleccionará la jugada con la menor puntuación. El jugador Max siempre escogerá la mejor jugada suponiendo que el adversario, el jugador Min escogerá la peor [21].
- Alfabeta: Es un algoritmo de búsqueda que mejora el funcionamiento del anterior mediante la introducción de un tipo de poda, denominada poda alfabeta, que reduce el significativamente el número de nodos el árbol de búsqueda (número de estados a explorar es exponencial al número de movimientos) [22].

4. CONCLUSIONES

En este tema se han presentado dos grandes grupos de técnicas de Inteligencia Artificial utilizadas para la resolución automática de problemas. Como se puede observar el proceso de resolución no es completamente automático ya que para que ambas técnicas funcionen correctamente es necesario definir una serie de elementos básicos que describen y acotan el problema que queremos resolver. La correcta descripción de estos elementos puede ser la diferencia a la hora de resolver o no el problema.

Por un lado hemos descrito el funcionamiento de diferentes técnicas de razonamiento automático las cuales nos permiten construir sistemas expertos que intentan simular el proceso de razonamiento utilizadodo por los humanos para resolver problemas. Pero para permitir que estos sistemas expertos funcionen es necesario definir como se va a modelar la información del entorno y cuales son las reglas que se van a utilizar para inferir conclusiones. Una vez definida toda esta información será posible construir un sistema experto basado en alguno de los modelos de razonamiento descrito que sea capaz de resolver problemas de un ambito específico, es decir este sistema no podrá ser utilizado para nada más. En caso de que queramos aumentar su complejidad podríamos incluir nuevos hechos o reglas pero **no** podríamos moficiar el dominio sobre el cual trabaja el sistema experto.

Por otro lado hemos descrito el funcionamiento de diferentes técnicas de búsqueda (no informada, informada, local, etc) las cuales nos permiten generar soluciones a cierto tipos de

problemas mediante la utilización de una definición detallada de las características del dominio a utilizar. En este caso, los algoritmos están más orientados a la resolución de problemas de ambito general de forma que se puedan definir las características básicas del problema y las posibles acciones que pueden aplicar y cualquiera de los algoritmos será capaz de resolver el problema. Esto no es del todo cierto, ya que dependiendo del tipo de algoritmo habrá que modelar la información de una manera especifica en base a su funcionamiento. Pero los algoritmos podrán ser utilizados de forma general. Un ejemplo de la versatilidad de los algoritmos de búsqueda es la planificación automática, la cual es un área de la inteligencia artificial que utiliza búsqueda heurística para resolución de problemas con independencia del dominio. Para ello, los investigadores en esta área han desarrollado diferentes tipo de heurísticas genéricas que pueden aplicarse a cualquier tipo de problema convirtiendo a los sistemas de planificiación en sistemas independientes del dominio ya que para resolver un problema sólo necesitan una definición especifica del dominio (tipos y acciones) y una definición del problema (objetos y metas).

5. REFERENCIAS

[1] G. Ernst and A. Newell (1969), “Gps: A case study in generality and problem solving”, ACM Monograph Series. Academic Press, New York, NY, United State of America.

[2] Haugeland, John (1985). Artificial Intelligence: The Very Idea, Cambridge, Mass: MIT Press, ISBN: 0-262-08153-9

[3] Jackson, Peter (1998). Introduction To Expert Systems (3rd edition), Addison Wesley, p. 2, ISBN: 978-0-201-87686-4

[4] Buchanan, Bruce G. and Shortliffe, Edward H. (1984). Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project (The Addison-Wesley series in artificial intelligence). Boston, MA, USA: Addison-Wesley Longman Publishing Co.

[5] Antoni L. (2006). Logical Foundations for Rule-Based Systems (Studies in Computational Intelligence) (Studies in Computational Intelligence), Springer-Verlag Berlin, Heidelberg ISBN: 3540291172

[6] Kolodner, J. (1993). Case-Based Reasoning. Morgan Kaufmann.

[7] Watson, I. (1997). Applying Case-Based Reasoning: Techniques for Enterprise

Systems. Morgan Kaufmann Publisher.

[8] Bellman R.E. and Zadeh L.A. (1970), Decision-Making in a Fuzzy Environment. Management Science, 17.

[9] Zadeh, L.A. (1965). Fuzzy Sets. Information Control, volume 8 pp. 338-353.

[10] Russell, Stuart J.; Norvig, Peter (2002), "3.4 Uninformed search strategies", Artificial Intelligence: A Modern Approach (2nd ed.), Prentice Hall.

[11] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill. pp. 531?539. ISBN:0-262-03293-7.

[12] de Champeaux, Dennis (1983), Bidirectional heuristic search again, Journal of the ACM, Volume 30(1) pp:22?32.

[13] Edelkamp, S. and Schroedl S. (2011). Heuristic Search: Theory and Applications Stefan (1st Edition). ISBN: 9780080919737

[14] Pearl, J. (1984). Heuristics: intelligent search strategies for computer problem solving. United States: Addison-Wesley Pub. Co., Inc., Reading, MA.

[15] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4. volume 4(2), pp 100?107.

[16] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. Cambridge, MA:

MIT Press.

[17] Banzhaf, W.; Nordin, P.; Keller, R.; Francone, F. (1998). Genetic Programming: An Introduction. San Francisco, CA: Morgan Kaufmann. ISBN:978-1558605107.

[18] Fred Glover (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research. volume 13(5). pp 533?549.

[19] Koenig S.-, Likhachev M.; Liu Y.; Furci D. (2004). Incremental Heuristic Search in Artificial Intelligence. Artificial Intelligence Magazine.

[20] Richard K. (1990). Real-time heuristic search. Artificial Intelligence. volume 42(2-3). pp 189?211.

[21] Nilsson, N.J. (1980). Principles of Artificial Intelligence. Tioga Publishing Company, Palo Alto, CA

[22] Edwards, D.J.; Hart, T.P. (1961). The Alpha-beta Heuristic. Massachusetts Institute of Technology.