Complejidad

Desde el punto de vista informático, se asocia la complejidad con los recursos de cómputos requeridos para resolver un problema, es decir la complejidad operativa. La teoría de la Complejidad Computacional se ocupa de estudiar estos recursos, que son normalmente tiempo y espacio.

Procedimiento y Algoritmo

Se denomina procedimiento a todo conjunto finito y ordenado de instrucciones, de forma tal que puedan ser ejecutadas por una persona o una máquina sin necesidad de conocimiento adicional a lo ya expresado en las propias sentencias. Un problema será considerado resuelto cuando se encuentre una forma mecánica de operar, paso a paso, a partir de sus datos hasta alcanzar un resultado.

Aquellos procedimientos que permiten alcanzar siempre una solución en un número finito de pasos se denominan algoritmos o procedimientos efectivos.

> Un procedimiento es una secuencia finita de instrucciones que pueden ser cumplidas en forma mecánica, como es el caso de un programa de computadora. Un procedimiento que siempre termina se denomina algoritmo.

Complejidad

El objetivo de la complejidad es establecer una escala que clasifique los problemas computacionales, permitiendo establecer medidas o “clases” de complejidad.

Se pretenderá buscar algún indicador de la complejidad intrínseca de los problemas, procurando prescindir de la habilidad de la persona que deba resolverlos, de los algoritmos que puedan ser utilizados y de la característica de los medios de cálculo disponibles.

Medidas y métricas de la complejidad

Lo que se pretende es determinar indirectamente la complejidad de un problema a partir de la medición de los recursos necesarios para resolverlo. Este enfoque es una determinación indirecta, ya que en realidad no se evalúa la complejidad del propio problema, sino más bien los recursos demandados correspondientes al procedimiento empleado.

Se seleccionan el tiempo y el espacio como los indicadores más representativos:

Complejidad Temporal

La complejidad temporal se refiere a la cantidad de intervalos o unidades elementales que demanda completar la ejecución de un proceso. La Complejidad Temporal se expresa en función del tamaño del problema o de sus datos.

Se procurará conocer la razón de crecimiento entre el indicador tiempo y la dimensión de los datos, que representa el parámetro medible. Como resultado se hablará de complejidad temporal lineal, polinómica, logarítmica, exponencial, etc. Según la naturaleza de la expresión que las vincula.

El límite de crecimiento de esta medida se denomina Complejidad Temporal Asintótica y es finalmente lo que determina el tamaño del problema que puede ser resuelto con cierto algoritmo. Tómese como ejemplo un algoritmo que procesa un conjunto de datos de entrada de tamaño “n” en un tiempo $C.n^2$, donde C representa una constante que es propia del problema: su complejidad temporal es de orden n^2 y esto se representa como

$O(n^2)$. En la forma general, se dice que una función $g(n)$ tiene un orden de complejidad temporal $O(f(n))$ si existe una constante C tal que:

$$g(n) \leq Cf(n)$$

para todos los posibles valores positivos de "n".

> **Se define la Complejidad Temporal como el número de unidades de tiempo requeridas para procesar una entrada de dimensión "n".**

Complejidad Espacial

La Complejidad Espacial es la cantidad de espacio de almacenamiento requerido para resolver un problema a través de un cierto algoritmo. Esto se apoya en la idea de que al aumentar la complejidad de un proceso aumenta también el espacio que demanda.

La Complejidad Espacial definirá la razón de crecimiento entre el espacio requerido y los datos del problema, y se representa por E(n).

El límite en el crecimiento de la Complejidad Espacial se denomina Complejidad Espacial Asintótica y determina finalmente el tamaño del problema que puede resolver cierto algoritmo.

Medición de la complejidad

Para determinar las complejidades temporales y espaciales sobre una aplicación en un computador real se deben considerar los tiempos requeridos por las instrucciones ejecutadas y los registros utilizados.

Una opción es el denominado "criterio de costo uniforme" que asigna a cada instrucción el consumo de una unidad de tiempo y a cada registro utilizado una unidad de espacio. T(x) es un número de instrucciones ejecutadas por el computador cuando lee cierta cadena de datos "x" y E(x) es la cantidad de registros de memoria utilizados.

A continuación se presentan algunos algoritmos y se muestran sus correspondientes complejidades teniendo en cuenta el criterio de costo uniforme:

- Ordenar un vector por selección

  $$T(n) = \frac{3}{4}n^2 + 3n = O(n^2)$$

  Complejidad temporal: polinomial (2º grado)

- Ordenamiento de un vector: método Shell Sort

  $$T(n) = O(n \log(n))$$

  Complejidad temporal: logarítmica

- Ordenamiento de un vector: método quick sort

  $$T(n) = O(n \log(n))$$

  Complejidad temporal: logarítmica

Lógica Difusa

La lógica binaria se basa en que la evaluación de sus premisas solo pueden ser verdaderas o falsas. En el caso de la lógica difusa los valores de verdad son no-deterministicos. El término “difuso” puede entenderse como la posibilidad de asignar más valores de verdad a los enunciados que los clásicos “verdaderos” o “falsos”.

La lógica difusa tiene como base los denominados conjuntos difusos y posee un sistema de inferencia basado en reglas de producción de la forma “SI antecedente ENTONCES consecuente”, donde los valores lingüísticos del antecedente y del consecuente están definidos por conjuntos difusos.

La idea fundamental que existe detrás de la teoría de los conjuntos difusos es una visión diferente de la noción de membresía aportada por la lógica tradicional, y consiste en que los elementos pueden pertenecer a más de un conjunto con cierto grado de pertenencia.

Función de Membresía

La **Función de Membresía o Pertenencia** es una función que representa en qué nivel o medida una variable de entrada pertenece al conjunto difuso (siempre entre 0 y 1). Esta es la que le dará forma a los conjuntos difusos.

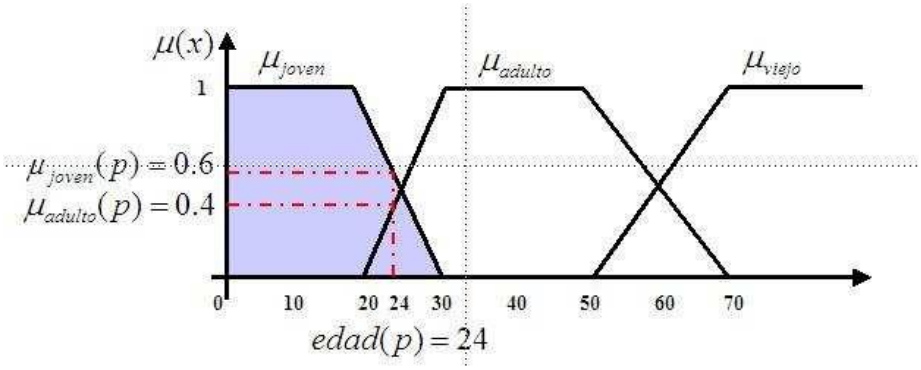
Por ejemplo, si consideramos el conjunto de personas jóvenes y considerando que pertenecen a este grupo los menores de 20 años. Según la lógica tradicional el conjunto se define de la siguiente manera:

Joven(x) = 1 si edad (x) <= 20
Joven(x) = 0 si edad (x) > 20

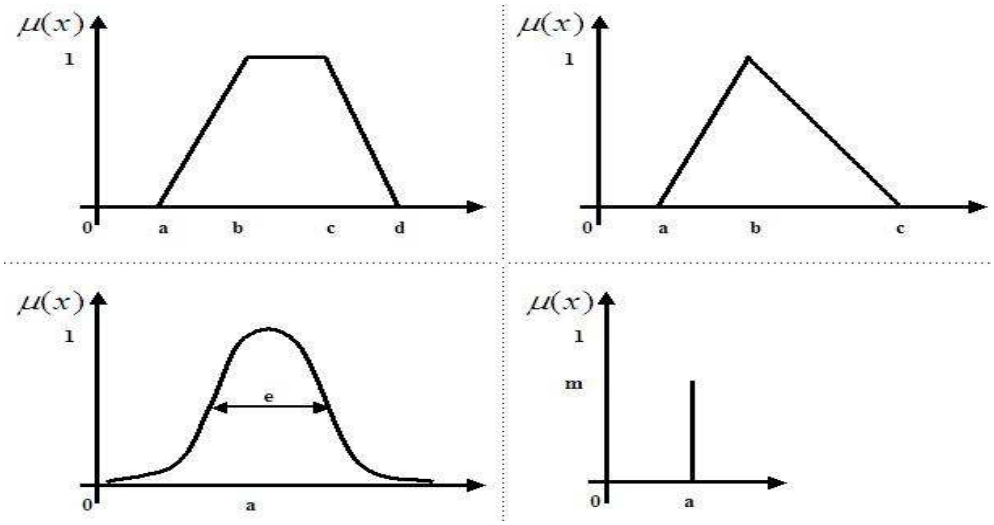
Para la lógica difusa, podemos decir que una persona de 21 años pertenece al conjunto de jóvenes pero con un grado de 0,9. Por lo tanto la función de membresía podría ser

μ joven(x) = 1 si edad(x) <= 20
1 – ((edad(x) -20)/10)

Ahora el conjunto *joven* contiene a personas entre 20 y 30 años con un grado de membresía que decrece linealmente



Las funciones de membresía están usualmente restringidas a ciertas clases de funciones. Las funciones pueden ser



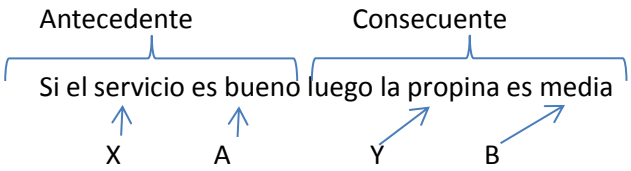
Trapezoidal, Triangular, Gausiana, Singleton

Regla Difuza – Si Entonces

Una regla difusa toma la forma:

Si X es A luego Y es B

Donde A y B son valores lingüísticos definidos por conjuntos difusos sobre los rangos X e Y respectivamente.
La primera parte de la regla es el antecedente o premisa, mientras que la parte final es el consecuente:



El antecedente es una interpretación que retorna un número entre cero y uno. Mientras que el consecuente es una sentencia que asigna el conjunto fuzzy a la variable de salida.

El antecedente de una regla puede estar compuesto de múltiples partes, en cuyo caso todas las partes del antecedente son calculadas simultáneamente y resueltas como un número simple empleando operadores lógicos. El consecuente también puede tener múltiples partes.

En cuyo caso todos los consecuentes son afectados igualmente por el resultado del antecedente.

Operadores Difusos

Si el antecedente de una regla ha sido dividido en más de una parte se debe aplicar un operador difuso para obtener un número que representa el resultado del antecedente.

Las entradas a un operador difuso son dos o más valores de pertenencia desde las variables de entrada fuzificadas. La salida es un único valor de verdad.

Existen diferentes métodos que en general son

- Conjunciones: mínimo, producto y truncamiento
- Disyunciones: máximo, amplificación y adición

- Negación.

Etapas de la lógica difusa

Etapas para la planificación de las reglas de lógica difusa:

1. **Paso I: Fuzzificación de las entradas**
   Tomar las entradas y determinar el grado en el cual ellas pertenecen a cada uno de los conjuntos difusos a través de funciones de membresía o pertenencia. La entrada es siempre un valor numérico limitado al universo de discurso de la variable de entrada y la salida es un grado difuso de pertenencia, un número en el intervalo entre 0 y 1.

   [Figura 8-6 Libro teórico]
   [Figura 8-7 Libro teórico]

2. **Paso II: Aplicación de los operadores difusos**
   Si el antecedente de una regla dada ha sido dividido en más de una parte se debe aplicar un operador difuso para obtener un número que representa el resultado del antecedente de esta regla.
   Las entradas del operador difuso son dos o más valores de pertenencia desde las variables de entrada fuzificadas. La salida es un único valor de esta regla.

   Existe distintos métodos que en general son conjunciones (mínimo, producto y truncamiento), disyunciones (máximo, amplificación y adición) y negación.

Operador	Método	Descripción
P y Q	Mínimo	min (XP,XQ)
P y Q	Producto	XP * XQ
P y Q	Truncamiento	max((XP+XQ-1),0)
P o Q	Máximo	max(XP,XQ)
P o Q	Amplificación	XP+XQ * (1-XP)
P o Q	Adición	min(XP+XQ,1)
no P	Complemento	1-XP

   Ver cuál es el mejor ejemplo
   [Figura 8-8 Libro teórico]
   [Figura 8-9 Libro teórico]

3. **Paso III: Aplicación del Método de Implementación**
   El método de implicación se define como la conformación del consecuente (un conjunto fuzzy) basado en el antecedente (un número). La entrada para la implicación es un número dado por el antecedente y la salida es un conjunto fuzzy. La salida se obtiene para cada regla por separado.

   [Figura 8-10 Libro teórico]

4. **Paso IV: Agregación**

La agregación es el proceso de unificar las salidas para cada regla uniendo los procesos paralelos. Es decir se combinan las salidas en un único conjunto difuso preparándolas para el paso final que es la Defuzificación.
El proceso de agregación consiste en tomar un único conjunto de salida para todo el sistema a partir de los conjuntos de salida de cada regla.
La entrada para la agregación es la lista de salidas truncadas (o modificadas por la implicación) retornadas por el proceso de implicación para cada regla. La salida de este proceso es un conjunto difuso para cada variable de salida.
La agregación es conmutativa, ósea que el orden de ejecución de las reglas no es relevante.

[Figura 8-11 Libro teórico]

5. **Paso V: Defuzzificación**
   La entrada para el proceso de Defuzificación es el conjunto difuso agregado (el conjunto difuso unificado de salida) y la salida debe ser un número.
   Entonces dado un conjunto difuso que engloba un rango de valores de salida se requiere obtener un único número.
   Un popular método de Defuzificación es el cálculo del centroide, el cual retorna el centro de un área bajo una curva.
   La defuzificación tiene lugar en dos pasos distintos. Primero las funciones de pertenencia son escaladas de acuerdo a sus posibles valores, luego estas son usadas para calcular el centroide de los conjuntos difusos asociados.

   [Figura 8-13 Libro teórico]

Un diagrama de inferencia difuzza completo se vería de la siguiente manera:

[Figura 8-12 Libro teórico]=ejemplo de resumen.

Búsqueda

En IA se entiende como búsqueda a un algoritmo que busca un dato en un conjunto, pero donde el objetivo es "encontrar un cierto camino" hasta encontrar el dato antes que corroborar la existencia del mismo.

Características de los Procesos de Búsqueda

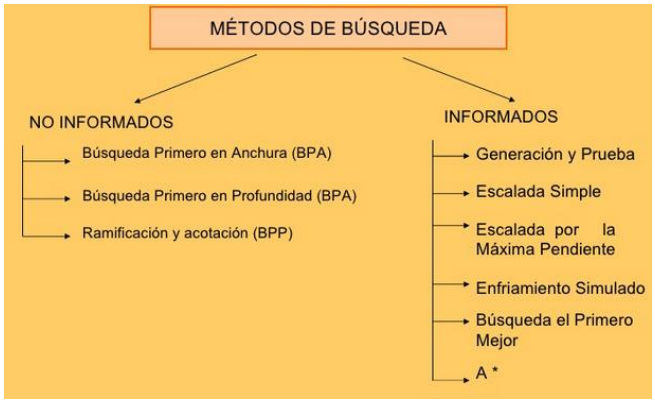
Los procesos de búsqueda tienen sentido en problemas que reúnen una serie de características:

- Cabe la posibilidad de asociar un *conjunto de estados* a las diferentes situaciones en que se puede encontrar el objetivo del *dominio* sobre el que se define el problema.
- Hay una serie de *estados iniciales* desde los que empezaría el proceso de búsqueda.
- Existen ciertos *operadores,* tal que un operador aplicado sobre un estado producirá otro estado.
- Existe al menos un *estado meta* o *estado solución.*

Cualquier proceso de búsqueda persigue, asociando nodos con estados y arcos con operadores, encontrar un camino que conduzca de un nodo inicial al otro final.

Se define *Espacio de Estados* como el conjunto de estados que podrían obtenerse si se aplicaran todos los operadores posibles a todos los estados que se fueran generando.

Métodos de Búsqueda



No informados (Búsqueda Exhaustiva)

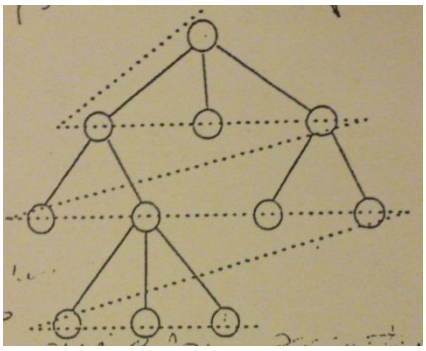
La búsqueda sin información del dominio pretende realizar una *exploración exhaustiva* del espacio de estados, dado que no hay conocimiento que pueda guiar la misma y que en principio, no deje ningún nodo sin ser examinado.

Búsqueda Primero en Anchura

La búsqueda Primero en Anchura recorre el árbol o grafo por niveles utilizando una estructura tipo cola (FIFO) en la que se van introduciendo los nodos generados.

Ventajas

- No queda atrapado explorando callejones sin salida
- Si existe una solución, garantiza que se logre encontrarla
- Si existen múltiples soluciones, se encuentra la solución mínima.



Algoritmo de búsqueda:

1. Crear una estructura dinámica tipo Cola
2. Mientras la cola no esté vacía hacer:
    a. Extraer el primer elemento de la Cola
    b. Generar nuevos estados derivados del actual

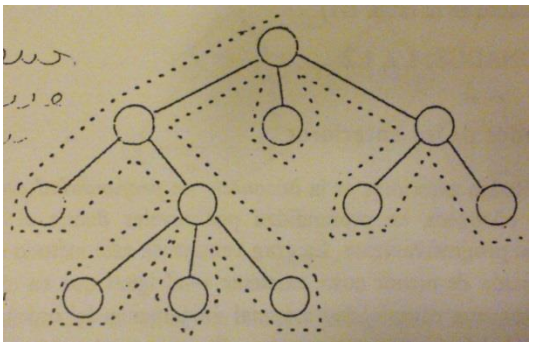
c. Explorar si los nuevos nodos son estados objetivos, si alguno lo es salir, en caso contrario añadir nuevos nodos a la Cola y volver a “a”.

Búsqueda Primero en Profundidad

La búsqueda en profundidad se centra en expandir un único camino desde la raíz. En el caso de llegar a un “callejón sin salida” se retrocede hasta el nodo más cercano desde donde se pueda tomar una rama alternativa para poder seguir avanzando.

Utiliza una estructura pila (LIFO) que va almacenando los nodos generados.

Suele establecerse el *límite de exploración*, que marca la máxima longitud que puede alcanzar cualquier camino desde la raíz durante los procesos de búsqueda.



Ventajas

- Necesita menos menoría ya que sólo se almacenan los nodos del camino que se sigue en ese instante.
- Si se tienen suerte este método, puede encontrar una solución sin tener que examinar gran parte del espacio de estados.

Algoritmo de búsqueda:

1. Crear una estructura dinámica tipo Pila
2. Mientras la pila no esté vacía hacer:
    a. Extraer el primer elemento de la Pila
    b. Si la profundidad del elemento es igual al límite de profundidad, regresar a “a”.
    c. Generar nuevos estados derivados del actual
    d. Explorar si los nuevos nodos son estados objetivos, si alguno lo es salir, en caso contrario añadir nuevos nodos a la Pila y volver a “a”.

Búsqueda por Ramificación y Acotación (Retroceso)

En lugar de generar todos los sucesores del estado actual, se genera un único sucesor en cada caso. La selección de un nodo a ser visitado no implica que sea sacado de la lista.

La característica de esta técnica es que el algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no están siendo óptimas, para «podar» (acotar) esa rama del árbol y no continuar malgastando recursos y procesos en casos que se alejan de la solución óptima.

Informados (Búsqueda Heurística)

Una búsqueda heurística es una técnica que aumenta la eficiencia de un proceso de búsqueda agregando información que permite reducir la cantidad de combinaciones y que permite seleccionar una ruta con menores costos.

El conocimiento dependiente del dominio puede ayudar a dirigir el proceso de búsqueda de manera que sean exploradas en primer lugar aquellas trayectorias que parecen más prometedoras a la hora de conducir a un estado solución.

El conocimiento Heurístico se incorpora en las reglas de producción o con una función heurística, determinando estados deseables y asignando pesos a ciertos aspectos del problema, permitiendo estimar que tan deseable es un nodo o un camino dado.

Las heurísticas son criterios, métodos o principios para decidir cuál de entre varias acciones promete ser la mejor para alcanzar una determinada meta.
El uso de heurísticas nos permite guiar nuestra búsqueda de una solución, lo que nos permitirá obtener una solución más rápidamente que si utilizásemos estrategias de búsqueda a ciegas.

En problemas de búsqueda:

- Una heurística será una función que utilizaremos para estimar cómo de cerca estamos de la solución.
- Cada heurística estará diseñada para un problema de búsqueda particular.

La familia de los algoritmos informados, frente a los desinformados o por fuerza bruta, son aquellos que poseen una información extra sobre la estructura a objeto de estudio, la cual explotan para alcanzar más rápidamente su objetivo final, con un camino de costo mínimo desde el punto inicial al final.

La búsqueda informada es aquella que utiliza el conocimiento específico del problema más allá de la definición del problema en sí mismo, la cual puede encontrar soluciones de una manera más eficiente que una estrategia no informada, increíblemente ineficiente en la mayoría de los casos.

Búsqueda Primero Mejor

Este algoritmo, combina las ventajas de los algoritmos primero en profundidad y primero en amplitud. Sigue un camino a la vez, pero puede cambiarse a otro camino que parece más prometedor que el que está siguiendo.

En este sentido, puede considerarse que es un algoritmo que realiza su proceso de búsqueda en un grafo de tipo O, ya que todos sus ramales representan una alternativa de solución.

Para su operación, el algoritmo necesita dos listas de nodos y una función heurística que estime los méritos de cada nodo que se genere:

1. **ABIERTOS**: Es una variable que contiene los nodos que han sido generados. La función heurística ha sido aplicada a ellos, pero todavía no han sido examinados, es decir no se han generado sus sucesores. **ABIERTOS** puede considerarse como una COLA DE PRIORIDADES en la que los elementos con mayor prioridad son los que tienen los valores más prometedores, dados por la función heurística.
2. **CERRADOS**: Es una variable que contiene los nodos que han sido examinados.

3. **FUNCIÓN HEURÍSTICA**: Permite que el algoritmo busque primero por senderos que son o parecen más prometedores.

$$f(n) = h(n)$$

Donde

$f(n) = función\ heuristica$

$h(n) = estimación\ del\ costo\ del\ camino\ óptimo\ desde\ \text{n}\ a\ una\ meta$

Estrategia

1. Se selecciona el nodo más prometedor, utilizando alguna *función heurística.*
2. Se expande el nodo elegido aplicando las reglas para generar a sus sucesores.
3. Si algún sucesor es solución el procedimiento termina.
   Si no:
   a. Se añaden los nodos a la lista.
   b. Se vuelven al punto inicial.

*Búsqueda A**

El problema de algunos algoritmos de búsqueda informados, es que se guían en exclusiva por la función heurística, la cual puede no indicar el camino de coste más bajo, o por el coste real de desplazarse de un nodo a otro, pudiéndose dar el caso de que sea necesario realizar un movimiento de coste mayor para alcanzar la solución. Es por ello que un buen algoritmo de búsqueda informada debería tener en cuenta ambos factores, el valor heurístico de los nodos y el coste real del recorrido.

Por este motivo el algoritmo de Búsqueda A* utiliza la siguiente Función de Heurística

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

Donde

$f(n) = función\ heuristica$

$h(n) = estimación\ del\ costo\ del\ camino\ óptimo\ desde\ \text{n}\ a\ una\ meta$

$g(n) =\ costo\ real\ del\ mejor\ camino\ encontrado\ en\ un\ determinado\ momento\ desde\ la\ raíz\ hasta\ “\ n\ “.$

Además, para su operación, el algoritmo necesita dos listas de nodos y una función heurística que estime los méritos de cada nodo que se genere:

- **ABIERTOS**: Es una variable que contiene los nodos que han sido generados. La función heurística ha sido aplicada a ellos, pero todavía no han sido examinados, es decir no se han generado sus sucesores. **ABIERTOS** puede considerarse como una COLA DE PRIORIDADES en la que los elementos con mayor prioridad son los que tienen los valores más prometedores, dados por la función heurística.
- **CERRADOS**: Es una variable que contiene los nodos que han sido examinados.

Estrategia

1. Comenzar con ABIERTOS conteniendo solo el estado inicial
2. Hasta que se llegue a un objetivo o no queden en ABIERTOS hacer:
   a. Tomar el mejor nodo de ABIERTOS
   b. Generar sus sucesores
   c. Para cada sucesor hacer:

i. Si no se ha generado con anterioridad, añadirlo a ABIERTOS y almacenar a su padre
ii. Si ya se ha generado antes:
    1. Si el nuevo camino es mejor que el anterior, cambiar padre.
    2. Actualizar el coste empleado para alcanzar el nodo y sus sucesores

Sistemas Expertos

Un sistema experto (SE) es un programa destinado a generar inferencias en un área específica del conocimiento en una forma similar a la que se espera de un experto humano.

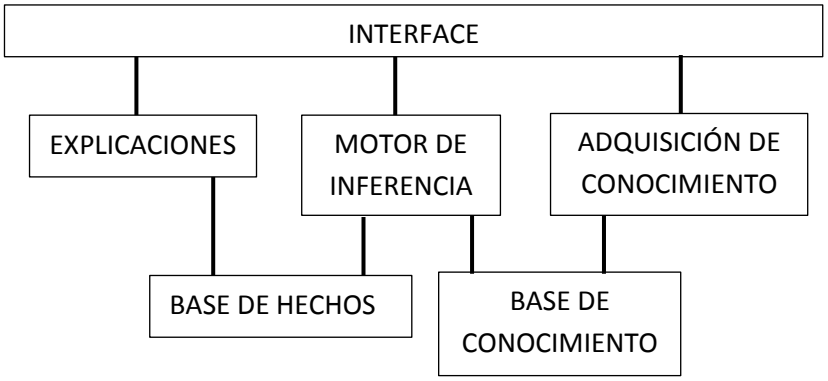
Deben resolver problemas por aplicación de conocimiento en un dominio específico. Este conocimiento es adquirido a través de la intervención de expertos humanos y almacenado en lo que se denomina Base de Datos de Conocimiento. Son aplicables a mundos reducidos, ya que no pueden operar en situaciones llamadas de sentido común por ser muy extenso el dominio de conocimiento que debe tener el sistema.

Se utiliza SE cuando:

- No existe algún algoritmo para resolver un problema
- El problema es resuelto satisfactoriamente por expertos humanos
- Existe algún experto humano que pueda colaborar en el desarrollo de un SE
- El conocimiento del dominio debe ser relativamente estático

El núcleo del sistema experto es el motor de inferencia.

Arquitectura



Los componentes básicos de un sistema experto son:

- **Símbolos:** Las diferentes definiciones que se van a utilizar en el SE, afirmaciones, elementos, etc.
- **Reglas:** son reglas a aplicar sobre los símbolos. Estas se deben obtener del conocimiento de los expertos humanos. Las reglas son consideradas heurísticas, dado que no se demuestra su validez general.
- **Hechos:** son predicados que se suponen verdaderos. Juntos constituyen la base de conocimiento.

Lógica de Predicados y Lógica Proposicional

El instrumento fundamental de la comunicación humana es el lenguaje, formado por frases del tipo:

- Interrogativo
- Imperativo
- Declarativo

Las frases declarativas constituyen el elemento básico de descripción del conocimiento.
La lógica es la disciplina que estudia los métodos de formalización del conocimiento humano. Por lo tanto la lógica estudia métodos de formalización de frases declarativas.

La **Lógica de Predicados** estudia las frases declarativas con mayor grado de detalle, considerando la estructura interna de las proposiciones. Toma como elementos básicos los **objetos** y sus **relaciones.** Es decir:

> **Que se afirma (predicado o relación)**
> **De quien se afirma (objeto)**
>
> *Sócrates es hombre y todos los hombres son mortales, entonces Sócrates es mortal.*

La **Lógica Proposicional** se caracteriza en relación a su composición por emplear muy pocos símbolos que pueden ser evaluados solamente como falsos o verdaderos (mediante clausulas y/o elementos atómicos), en contraposición al alfabeto lógico usado por la lógica de predicados para expresar sus formulaciones.

- La Lógica Proposicional está basada en constantes: $A \ \wedge B \ \rightarrow C$
- La lógica de predicados está basada en variables: $\forall x \ \forall y \ \ y \rightarrow x$

Lógica de Predicados y Sistemas Expertos

Los sistemas expertos se basan en la lógica de predicados. La palabra lógica hace referencia a la posibilidad de emplear un conjunto de reglas para gestionar inferencias creíbles.

En el caso de los sistemas expertos se pretende aplicar a mundos que solo conocen en profundidad los expertos humanos. Estos expresan su conocimiento del mundo que manejan, con sus propias palabras. Estas palabras no son otra cosa que los símbolos del lenguaje natural, cuyo significado dista mucho de estar bien establecidos. Por lo tanto para que exista una chance de que un sistema experto basado en el formalismo lógico produzca inferencias correctas, se requiere primero definir los símbolos con cuales se operará, y su significado preciso.

Los sistemas expertos se basan esencialmente en la lógica a la que incorporan algunas sencillas estrategias de programación.

La lógica de 1er orden es uno de los formalismos más utilizados para representar conocimiento en IA. La lógica cuenta con un lenguaje formal mediante el cual es posible representar formulas llamadas **axiomas,** que permiten describir fragmentos del conocimiento, y además consta de un conjunto de **reglas de inferencia** que aplicadas a los axiomas, permiten derivar nuevos conocimientos.

Planificación

Se entiende a la planificación como al proceso de computar varios pasos de un procedimiento de resolución de un problema, antes de ejecutar algunos de ellos.

También se considera a la planificación como una búsqueda en un espacio de estados con el agregado de que lo que interesa es el camino o recorrido entre los estados alcanzados hasta llegar al estado objetivo.

Para poder estudiar los problemas en forma sistemática, investigadores de IA propusieron un grupo de "mundos" en los cuales podrían proponer y probar mecanismos de planificación. El más famoso de esos mundos es el llamado "mundo de los bloques".

El problema de la planificación

Corresponde al área de resolución de problemas. Usualmente se ocupa de problemas "muy complejos" en el sentido de que no pueden explotarse todas las opciones.

Entradas:

- Descripción del estado del mundo
- Descripción del objeto
- Conjunto de acciones

Salida: Una secuencia de acciones que pueden ser aplicadas al estado inicial, hasta lanzar la descripción del estado final

En principio, se plantea el problema de la planificación como una búsqueda en un espacio de estados. La lógica nos ofrece la posibilidad de tener una manera de reducir nuestros pasos de búsqueda en planificación. Para esto se deben representar los estados mediante predicados, lo que permite pasar de uno a otro aplicando reglas lógicas.

El mundo de los Bloques

Para poder realizar un estudio sistemático de los métodos y poder compararlos, resulta útil trabajar en un único dominio lo suficientemente complejo para que permita observar todos los mecanismos.

Para esto se adopta el mundo de los bloques. En este mundo existe una superficie plana sobre la que se sitúan los bloques (por simplicidad cuadrados), los mismos se pueden colocar unos sobre otros, un manipulador puede tomar y reubicar los bloques.

Primero se definen las acciones que puede llevar a cabo el manipulador, que son los **operadores** asociados a las **reglas**. Luego se consideran los **predicados** que se emplearán para definir los **estados posibles** y las definiciones de los **estados final inicial y final.**

Acciones del Operador (Operadores):

- (Desapilar AB)
- (Apilar AB)
- (Levantar A)
- (Bajar A)

Predicados:

- (Apilando AB)
- (Desapilado AB)
- (Sobre la mesa A)
- (Despejado A)

- (Agarrado A).

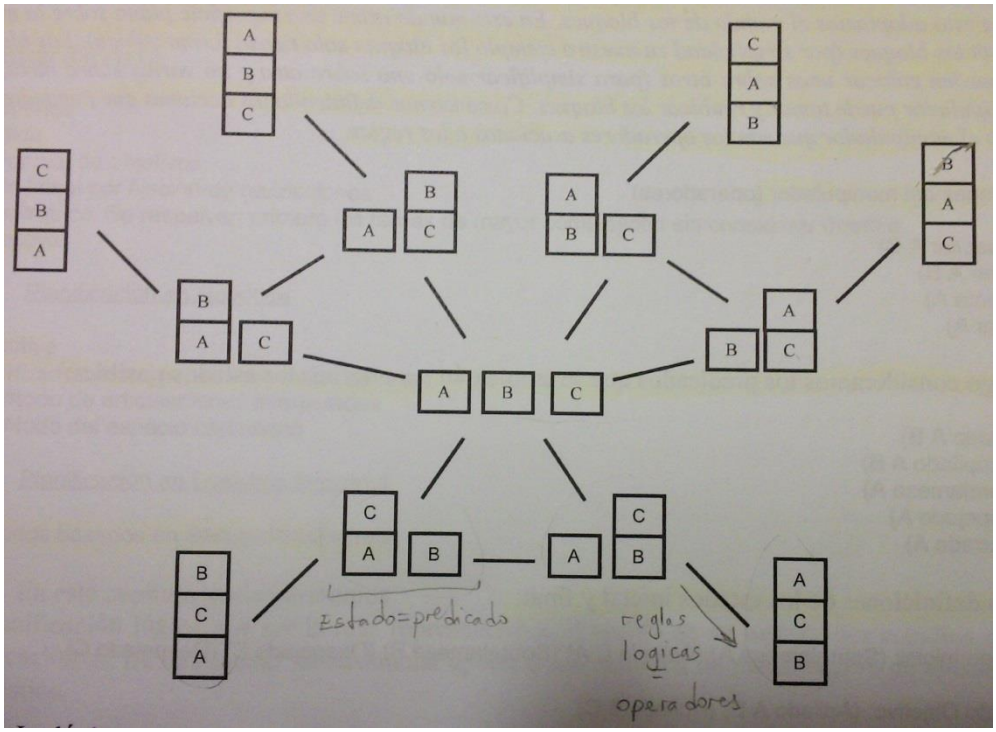
Estados inicial y final.

Estado Inicial: (sobre la mesa A)(Apilado CA)(Sobre la mesa B)(Despejado B)(Despejado C)

Estado Final: (Apilado AB)(Apilado BC)(Despejado A)(Sobre la mesa C)

Planificación como búsqueda en un espacio de estados

El siguiente gráfico muestra los estados posibles para el apilado de tres bloques. Dado que pasamos de un estado a otro mediante acciones (operadores) se observa que el problema de la planificación es básicamente un problema de búsqueda en un espacio de estados.



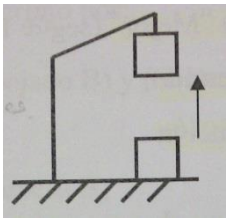
La lógica nos ofrece una manera de reducir nuestros pasos de búsqueda en planificación.

Para esto se deben representar los estados mediante **predicados** lo que permite pasar de un estado a otros aplicando reglas lógicas. La aplicación de la regla debería modificar el estado anterior definiendo un nuevo estado. Sin embargo, la regla en si solo nos dice que está permitido pasar de un estado particular a otro, pero no efectúa en un programa el cambio de predicados necesario para obtener el nuevo estado.

Entonces, para que en un programa el estado actual se modifique de acuerdo a la relación establecida en la regla es necesario aplicar procedimientos (operadores) que efectúen esa modificación. Con lo cual queda establecido que además de predicados y reglas se precisan operadores asociados alas reglas.

Codificación de operadores

Para poder pasar de un estado a otro, modificando predicados, usamos operadores. Cada operador puede describirse mediante una lista de nuevos predicados que el operador provoca que sean ciertos y una lista de viejos predicados que el operador provoca que sean falsos. Estas listas se denominan añadir y borrar.



Operador: (Desapilar A B)
Precondiciones: (Apilado A B) (Despejado A)
Borrar: (Apilado A B) (Despejado A)
Añadir (Pos condiciones): (Agarrado A) (Despejado B)

Planificación mediante aplicación de reglas

Para evitar realizar la búsqueda explorando completamente el árbol (o grafo), se plantea para las descripciones simbólicas el empleo de reglas. Los pasos básicos para la aplicación de estas reglas son:

- Elegir una regla cuyos antecedentes sean verdaderos en el estado actual. Si hay más de una regla elegir la "mejor" (según función heurística).
- Aplicar reglas para obtener nuevo estado.
- Detectar si se ha llegado a una solución.

Eventualmente estos procedimientos pueden requerir algún pos procedimiento tales como:

- Detectar callejones sin salida.
- Refinar soluciones "casi correctas".

Planificación mediante una pila de objetivos

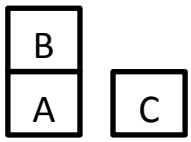
Los pasos son:

1. Se plantea una pila de objetivos.
2. Se trata a los objetivos como subproblemas separados a resolver.
3. Se pregunta si el 1er objetivo se cumple, si no se cumple se debería aplicar el operador que los transforme, lo que hace es reemplazar al predicado objetivo por el operador y luego agregar a la pila las precondiciones que deben cumplirse para ese operador.
4. Para resolver el objetivo que ha quedado arriba de la pila debemos emplear un operador y, como antes, reemplazar y agregar las precondiciones.
5. Posteriormente al ir extrayendo los operadores de esa pila, se incorpora a la lista PLAN.

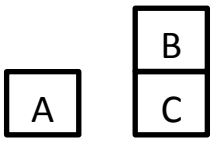
Ejemplo de Planificación mediante una pila de Objetivos

Operador	Precondición	Pos Condición
Desapilar (X Y)	Apilado (X Y), Despejado (X)	Agarrado (X), Despejado (Y)
Apilar (X Y)	Despejado (Y), Agarrado (X)	Apilado (X Y)
Levantar (X)	Despejado (X), SobreMesa (X)	Agarrado (X)
Bajar (X)	Agarrado (X)	SobreMesa (X), Despejado (X)

Estado Inicial



Estado Final



Estado Inicial: (Apilado B A) (SobreMesa A) (SobreMesa C) (Despejado B)

Estado Final: (Apiado B C) (SobreMesa A) (SobreMesa C)

Comenzamos agregando el estado final a la pila

(Apilado B C)
(SobreMesa A)
(SobreMesa C)

Pila de objetivos

Verificar si se cumplen los predicados comparando contra el estado inicial.
Como el primer objetivo *(Apilado B C)* no se cumple, lo reemplazamos por el operador que hace que se cumple y sus precondiciones, ósea:

Operador	Precondición	Pos Condición
Apilar (B C)	Despejado (C), Agarrado (B)	Apilado (B C)

Despejado (C)
Agarrado (B)
Apilar (B C)
(SobreMesa A)
(SobreMesa C)

Pila de objetivos

Como ahora el primer elemento es un predicado y si se cumple se saca de la pila:

Agarrado (B)
Apilar (B C)

(SobreMesa A)
(SobreMesa C)

Pila de objetivos

Como *Agarrado (B)* no se cumple lo reemplazamos por el operador que hace que se cumpla y sus precondiciones, ósea:

Operador	Precondición	Pos Condición
Desapilar (B A)	Apilado (B Z), Despejado (B)	Agarrado (B), Despejado (A)

Apilado (B A)
Despejado (B)
Desapilar (B A)
Apilar (B C)
(SobreMesa A)
(SobreMesa C)

Pila de objetivos

Como *Apilado (B A)* se cumple comparando con el estado inicial es extraído de la pila:

Despejado (B)
Desapilar (B A)
Apilar (B C)
(SobreMesa A)
(SobreMesa C)

Pila de objetivos

Se cumple *Despejado (B)* por lo que es extraído de la pila

Desapilar (B A)
Apilar (B C)
(SobreMesa A)
(SobreMesa C)

Pila de objetivos

Como el elemento a tomar de la pila **Desapilar (B A)** es un operador, esto es extraído y añadido a la pila PLAN. Con lo que tendremos:

Apilar (B C)
(SobreMesa A)
(SobreMesa C)

Pila de objetivos

Desapilar (B A)

Pila PLAN

El siguiente elemento a tomar de la pila **Apilar (B C)** es también un operador, que es extraído de la pila y añadido a la pila de PLAN

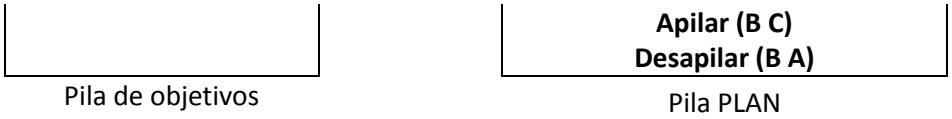
(SobreMesa A)
(SobreMesa C)

Pila de objetivos

Apilar (B C)
Desapilar (B A)

Pila PLAN

Como los predicados restantes de la pila se cumplen se quietan de la Pila de Objetivos



Pila de objetivos

Pila PLAN

Con este algoritmo los operadores de la Pila de PLAN quedan invertidos. Una vez que se quitan todos los elementos de la pila de objetivos se debe invertir el orden en la Pila de PLAN para obtener el resultado final, ósea, los pasos necesarios para llegar del estado inicial al estado final:

Operadores: **Desapilar (B A), Apilar (B C)**