Fundamentos de Inteligencia Artificial

To begin with

Los problemas adecuados para tratar con técnicas de IA son aquellos para los cuales solo disponemos de descripciones poco claras, incompletas, imprecisas y con alto grado de duda. Cualquier problema que se pueda solucionar con un algoritmo no es IA.

Los objetivos de la IA consisten en formalizar este conocimiento impreciso traduciéndolo a descripciones claras, completas y precisas.

Técnicas de IA: representar y gestionar el conocimiento, diagnosticar, razonar, planificar y aprender.

Podemos definir informalmente la inteligencia artificial (IA) como aquella rama de la informática -ciencias de la computación- que estudia cómo hacer que las máquinas puedan presentar comportamientos que los humanos asociaríamos con la inteligencia humana como el aprendizaje o la solución de problemas. Esta definición provoca que conforme nos acostumbramos a que las máquinas sean capaces de desarrollar ciertas actividades, éstas en muchos casos dejen de ser consideradas como ‘‘inteligencia artificial’’.

Los objetivos tradicionales de la inteligencia artificial incluyen **razonamiento**, **conocimiento**, **planificación**, **aprendizaje**, **procesamiento de lenguaje natural**, **percepción** y la **capacidad de mover y manipular objetos**. Las aproximaciones a estos objetivos pueden ser variadas, desde manipulación simbólica a métodos estadísticos.

Es importante distinguir entre la inteligencia artificial débil de la inteligencia artificial general en cuanto a que se denomina normalmente inteligencia artificial débil a aquellos sistemas que muestran un comportamiento inteligente en una tarea concreta pero que dicho conocimiento no es posible aplicarlo a ningún otro problema. La gran mayoría de los sistemas existentes actualmente pertenecen a esta categoría. La inteligencia artificial general es un objetivo de investigación a muy largo plazo, aunque se está empezando a ver algún avance en la aplicación de un mismo sistema en la resolución de distintos problemas.

Paradigmas IA

- **SIMBOLICO**: Es un modelo descendente, que se basa en el razonamiento lógico y la búsqueda heurística (conjunto hechos y reglas) como pilares para la resolución de problemas. La IA simbólica opera con representación abstracta del mundo real.
- **SITUADO**: reactivo, basado en conductas. Inferencia mediante autómata de control.
- **CONEXIONISTA**: Los sistemas conexionistas no son incompatibles con la hipótesis simbolista (SSF) pero al contrario del simbólico, se trata de un modelo ascendente, ya que se basa en la hipótesis de que la inteligencia emerge a partir de la actividad distribuida de un gran número de unidades interconectadas que procesan información paralelamente. En la IA conexionista estas unidades son modelos muy aproximados de la actividad eléctrica de las neuronas biológicas.

Lógica proposicional (orden 0): también llamada lógica de enunciados, lógica de orden cero o cálculo proposicional, es un sistema formal cuyos elementos más simples representan proposiciones (expresión que es Verdadera o falsa), y cuyas constantes lógicas, llamadas conectivas lógicas, representan operaciones sobre proposiciones, capaces de formar otras proposiciones de mayor complejidad. Se puede usar para razonamientos sencillos, es una lógica de carácter finito que garantiza la decibilidad. No puede manejar componentes temporales ni espaciales.

Lógica predicados (orden 1): Una lógica de primer orden, también llamada lógica predicativa, lógica de predicados o cálculo de predicados, es un sistema formal diseñado para estudiar la inferencia en los lenguajes de primer orden. La inferencia es el proceso por el cual se derivan conclusiones a partir de premisas.

Todos los mamíferos son animales de sangre caliente. (Premisa mayor)
Todos los humanos son mamíferos. (Premisa menor)
Por tanto, todos los humanos son animales de sangre caliente. (Conclusión)

Introduce variables para denotar elementos del dominio, cuantificadores y predicados.

"David = {energía, ganas de aprender, actitud, aptitudes, atrevimiento, interés} ….. todo un activo para cualquier empresa"

$\forall$ *Empresa 'y'* $\exists$ *'x'* *(Activo y* $\rightarrow$ *Energía x, Ganas de aprender x, Actitud x, Aptitudes x, Atrevimiento x , Interés x)* | *x* $\rightarrow$ *David*

Para toda empresa 'Y' existe un 'X' donde si X tiene energía, ganas de aprender, actitud, aptitudes, atrevimiento, interés. X es un activo para Y. Siendo X = david.

Sistemas Basados en Reglas: Aplicación de los sistemas de deducción en lógica proposicional restringidos a cláusulas de Horn.

Redes semánticas: grafo orientado formado por nodos etiquetados (conceptos e instancias) y arcos unidireccionales etiquetados que representan relaciones entro los nodos.

Sistema de búsqueda (elementos)

Describir el espacio de búsqueda y luego elegir una estrategia para recorrerlo

- Estados: modelan el entorno.
- Operadores o reglas: modelan las acciones del agente.
- Estrategia de control: algoritmo de búsqueda.

Una descripción precisa de los estados y sus operadores (con sus constes) definen el espacio de búsqueda.

Un algoritmo de búsqueda es **completo si siempre encuentra una solución** y es **admisible si siempre encuentra la solución óptima.**

Ampliación Sistemas Inteligentes

Un algoritmo de búsqueda es **completo** si siempre encuentra solución, en el caso de que exista alguna.

Un algoritmo de búsqueda es **admisible**, o exacto, si siempre encuentra una solución óptima.

Los **heurísticos** son criterios, reglas o métodos que ayudan a decidir cuál es la mejor alternativa entre varias posibles para alcanzar un determinado objetivo.

TEMA 1: Búsqueda avanzada

Las técnicas de búsqueda avanzada, y en especial los algoritmos de búsqueda local, son un primer paso hacia los métodos de Computación Evolutiva

Algoritmos búsqueda informada

Permiten dirigir la búsqueda (usando heurísticas) a las regiones más prometedoras.

- **VORACES**: No admisibles y en general no completos. Nunca deshacen una decisión. Muy eficientes (sistemas tiempo real *e.g.* planificación del procesador u otros recursos de un SO). se aplica a problemas de optimización en los que la solución se puede construir paso a paso sin necesidad de reconsiderar decisiones ya tomadas. Si la búsqueda está guiada por un buen heurístico, la probabilidad de obtener una buena solución aumenta. En el peor caso, la complejidad es exponencial $O(2^n)$ [donde n máximo nivel espacio búsqueda].
  https://descubriendolaia.blogspot.com/2016/03/video-n-034-ia-grafos-busqueda-voraz.html
- **RAMIFICACION Y PODA**. Algoritmo Determinista (siempre devuelve la misma solución) y admisible si se recorre por completo todo el espacio de búsqueda, a excepción de las ramas que se podan, lo que no siempre es posible si se tiene un tiempo limitado. Como los algoritmos voraces, la búsqueda se plantea como un proceso en el que la solución se va construyendo paso a paso, de modo que un camino desde el inicial a un estado representa parte de la solución del problema y el estado representa un subproblema que se debe resolver para llegar a una solución del problema original. Muy usado en las tareas de optimización de recursos . Cada estado se interpreta como un subconjunto de soluciones del problema. Sus componentes fundamentales son:
  - Esq. Ramificación. Equivalente al cálculo de sucesores.
  - Calculo cota inferior. Mejor solución posible desde ese nodo. Se usa para podar ramas. Su valor es análogo a *f(n) = g(n) + h(n). g(n)* es el coste del camino recorrido.
  - Calculo cota superior. Mejor solución (real del problema) encontrada hasta el momento. Este algoritmo debe ser muy eficiente, por ejemplo, un algoritmo voraz.
  - Estrategia control. Gestión de la lista ABIERTA, que nodos entran en esta y cual es siguiente a ser expandido. Si un nodo tiene una cota inferior mayor que la mejor cota superior encontrada hasta el momento, se poda esa rama reduciendo el espacio de búsqueda, y el nodo no se inserta en ABIERTA (La lista ABIERTA contiene al principio de cada iteración los estados candidatos a ser desarrollados o expandidos y que estará ordenada con un determinado criterio en cada caso.).
- **MEJORA ITERATIVA, BUSQUEDA LOCAL**. Para problemas en los cuales el nodo solución contiene toda la información necesaria y el camino no es importante (problemas de optimización: TSP, N-reinas). Son algoritmos sencillos con coste espacial bajo y pueden encontrar soluciones en espacios infinitos, pasan de una solución a otra solución vecina según el criterio de aceptación.

Algoritmo 9.7 Esquema de un algoritmo de búsqueda local.

```
S = SolucionInicial;
mientras (no CriterioDeTerminación) hacer
   V = SolucionesVecinas(S);
   EvaluarSoluciones(V);
   S1 = Seleccion(V);
   si (CriterioDeAceptación) entonces
      S = S1;
   fin si
fin mientras
devolver S;
```

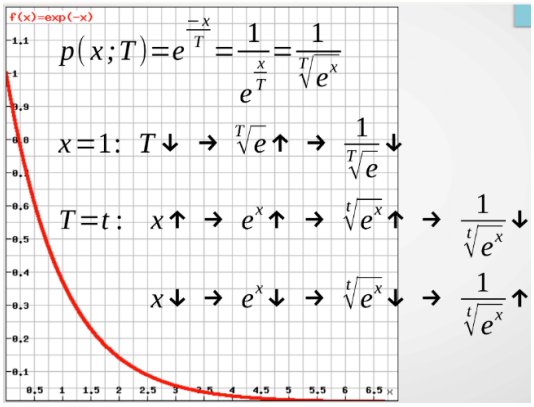
Pueden no encontrar la mejor solución y por lo tanto no son admisibles. Necesitan una heurística muy informada. Son un primer paso hacia los métodos de Computación Evolutiva.

https://descubriendolaia.blogspot.com/2016/05/video-n-055-ia-grafos-busqueda-local.html

- ESCALADA, MAXIMO GRADIENTE. Es una búsqueda en profundidad sin posibilidad de retroceso. El criterio de aceptación es que la solución vecina S1 sea mejor o igual que la solución actual S. Este tipo de búsqueda tiene los siguientes inconvenientes: **Óptimos locales**. Todos los vecinos son peores y la búsqueda termina sin encontrar óptimo global. | **Regiones planas**. Todos los vecinos tienen el mismo valor de S, por lo tanto, la búsqueda es aleatoria. | **Crestas**. Si la pendiente es muy marcada es fácil avanzar, pero si la pendiente es suave hacía el óptimo global resulta difícil guiar la búsqueda. En estos tres casos lo que ocurre es que la búsqueda se queda estancada en un óptimo local, que puede ser una solución razonable o no serlo. Lo que se puede hacer en estos casos es reiniciar la búsqueda a partir de otra situación de partida, lo que se suele denominar búsqueda **multiarranque**, con lo que se alcanzaría posiblemente otro óptimo local distinto. Así, después de un número de intentos se podría llegar a una solución aceptable,

  https://descubriendolaia.blogspot.com/2016/05/video-n-057-ia-grafos-ascenso-de.html

- TEMPLE SIMULADO. La idea es admitir, con una cierta probabilidad, algunas transiciones donde la solución actual empeora. De esta forma se puede salir de óptimos locales. El criterio de selección es elegir un vecino aleatorio, si es mejor se acepta y si es peor pero dentro de una determinada probabilidad se acepta también. Al inicio de la búsqueda la probabilidad de aceptar una solución peor es alta, pero a medida que la búsqueda progresa ve decreciendo y al final solo se aceptan soluciones mejores $T = \alpha(t,T)$, siendo t la iteración actual y α una función que decrece al aumentar t . El principal inconveniente que presenta este método es que requiere un ajuste de parámetros adecuado. Normalmente, este ajuste depende fuertemente del problema y hay que realizarlo de forma experimental.

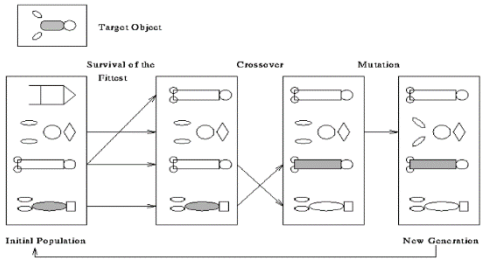
  https://descubriendolaia.blogspot.com/2016/06/video-n-061-ia-grafos-temple-simulado-01.html

- TABU. Se dispone de un mecanismo de memoria que se usa para evitar la generación de algunos vecinos dependiendo de la historia reciente. Puede ser una lista TABU donde se guardan las últimas operaciones realizadas para evitar repetirlas en el siguiente paso. Pueden existir excepciones en la lista (**criterio de aspiración**) que mejoren la solución actual. Al igual que en el temple simulado el mayor inconveniente es el ajuste de parámetros. En la práctica se realiza de forma experimental.

TEMA 2: Computación evolutiva

Desde el punto de vista de la algoritmia, los **A**lgoritmos **E**volutivos son **probabilistas** (toma decisiones aleatorias y pueden comportarse de forma distinta con el mismo conjunto de datos) y heurísticos. La heurística incorporada en un AE se inspira en la evolución de los seres vivos.

Son potentes técnicas de búsqueda y optimización que permiten resolver problemas propios de los procesos de aprendizaje, sin embargo, debido a fenómenos como la epítasis, decepción o *genetic drift*, la búsqueda puede caer en óptimos locales e impiden una localización diversificada de las soluciones del problema. Asimismo, permiten resolver problemas: De gran dimensionalidad, no lineales, en presencia de ruido, dependientes del tiempo



En general buscan soluciones sin considerar conocimiento específico del problema, pero presentan gran flexibilidad para incorporar este conocimiento.

Un AE debe contener los siguientes cinco componentes:

- Una representación apropiada de las soluciones del problema (saber cómo representar el problema).
- Una forma de crear una **población inicial** de soluciones potenciales(**individuos**). Usualmente se genera de forma **aleatoria** dentro del **espacio de búsqueda**, la incorporación de conocimiento puede ayudar a guiar la búsqueda.

Algoritmo 11.1 Algoritmo evolutivo.

```
1: t ← 0; /* generación inicial */
2: inicializar P(t);
3: evaluar P(t);
4: mientras (no se cumpla la condición de terminación) hacer
5:    seleccionar padres de P(t);
6:    recombinar padres y mutar ⇒ C(t);
7:    evaluar C(t);
8:    seleccionar supervivientes de P(t) ∪ C(t) ⇒ P(t + 1); /* sustitución generacional */
9:    t ← t + 1; /* siguiente generación */
10: fin mientras
```

- Una **función de evaluación** capaz de medir la adecuación de cualquier solución.
- Un conjunto de operadores evolutivos (**seleccionar**, **recombinar**, **modificar**)que actúan como reglas de transición probabilistas (no deterministas) para guiar la búsqueda, y que combinan entre sí las soluciones existentes.
- El valor de unos parámetros de entrada que el AE usa para guiar su evolución (tamaño de la población, número de iteraciones, probabilidades de aplicación de los operadores evolutivos, etc.).

https://descubriendolaia.blogspot.com/2016/06/video-n-067-ia-grafos-algoritmos.html
https://descubriendolaia.blogspot.com/2016/06/video-n-069-ia-grafos-algoritmos.html

Algoritmos genéticos (Pertenecen fam. AE)

Son métodos de resolución de problemas de búsqueda local y poda. Se utilizan en problemas donde no se pueden encontrar soluciones o estas no son satisfactorias. Los AG se deben adaptar específicamente a los problemas que van a resolver. No hay un marco teórico genérico para aplicarlo a todos los problemas.

Los AG han usado tradicionalmente una representación más independiente del dominio, normalmente, cadenas binarias. Sin embargo, pueden usar otras, tales como grafos, expresiones Lisp, listas ordenadas, o vectores de parámetros reales.

Se aplican sobre una población representada de forma abstracta como cromosomas, que son la codificación de soluciones candidatas a un problema.

Representación:

$P(t) = \{X_1^t, \ldots, X_m^t\}$

Población
Es un subconjunto de todas las soluciones posibles, codificadas, para un problema dado. Simplemente, este es el conjunto de individuos y cada individuo es una solución al problema

Cromosomas
Es una de esas soluciones al problema en cuestión.

Gen
Es la posición de un elemento de un cromosoma. Los individuos de una población se caracterizan por un conjunto de parámetros, variables y son particulares, denominados genes.

Alelo
Es el valor que toma un gen para un cromosoma en particular.

Se denomina '***locus***' a la posición de un *gen* en un *cromosoma*.

Al paquete genético total se le denomina *genotipo*, y a la interacción del *genotipo* con su entorno se le denomina $fenotipo$ (solución representada por la decodificación del *genotipo*).

El término **individuo** es usa para referirse al conjunto de información *genotipo*-*fenotipo*-*adecuación*. Así, podemos representar un individuo $\boldsymbol{X_i^t}$ en una generación *t*, como la terna:

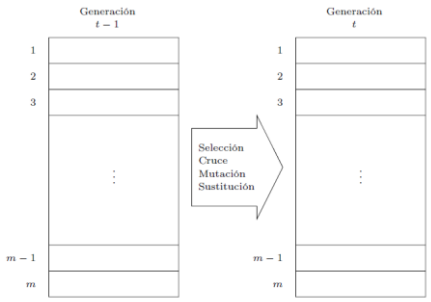
$$\boldsymbol{X_i^t} = \{c_i^t, x_i^t, f_i^t\}$$

donde x_i^t es la decodificación ($fenotipo$) del cromosoma c_i^t, y f_i^t es la adecuación (función heurística) de la solución al entorno o *fitness*.

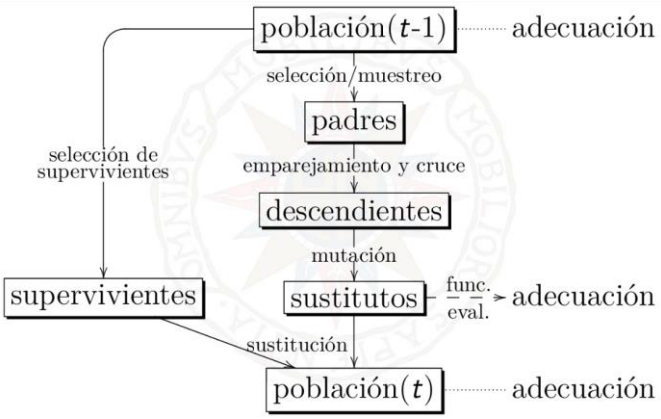
La función de evaluación *eval* se corresponde normalmente con la función objetivo *f* del problema, y entonces, dado un cromosoma c_i^t y su fenotipo x_i^t, podemos obtener su adecuación f_i^t como:

$$f_i^t = eval(c_i^t) = f(x_i^t)$$

Selección, muestreo, operadores genéticos y sustitución generacional:

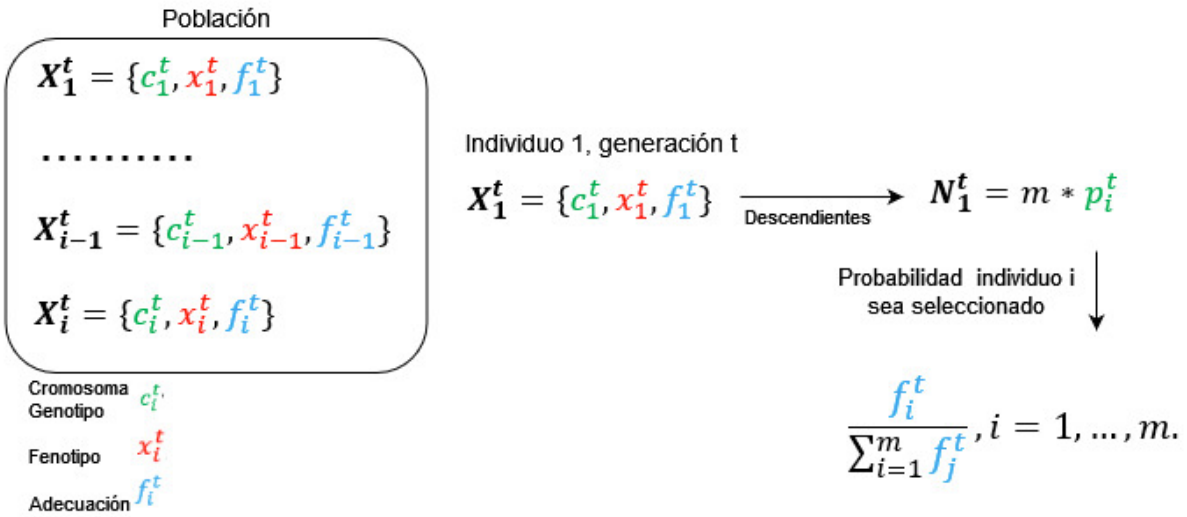


Una vez que se han evaluado todas las soluciones de la población en una generación, el proceso evoluciona hacia una nueva generación. La población en la nueva generación *t* sufrirá una transformación con respecto a la población en la generación anterior $t-1$. En el AG simple descrito en la sección anterior se han utilizado la selección proporcional a la función de adaptación, el muestreo estocástico con reemplazamiento (los estados sucesores son combinaciones de los estados actuales padres), y la sustitución generacional completa.



Selección y muestreo:

El esquema de selección asigna, a cada individuo *i* de una población en la generación *t*, el número esperado de descendientes $\boldsymbol{N_1^t}$**(**según su nivel de adaptación)



Asociado a un esq. de selección, un algoritmo de muestreo obtiene individuos de la población de acuerdo con el numero esperado de descendientes de los individuos.

- Calcular la probabilidad acumulada de cada cromosoma.

$$q_i^t = \sum_{j=1}^{i} p_j^t \, , i = 1, \dots, m.$$

- Repetir los siguientes pasos para cada individuo a seleccionar:
  - Extraer, con reemplazamiento, un número real aleatorio $r \in [0, 1]$.
  - Si $r \leq q1$ entonces se muestrea el primer cromosoma de la población en la generación t, y en otro caso se muestrea el *i-ésimo* cromosoma ($2 \leq i \leq m$) tal que $q_{i-1}^t < r < q_i^t$i

	F.Adecuación	P_i^t	q_i^t
individuo 1	0.026602	0.00414714	0.00414714
individuo 2	0.569424	0.08877091	0.09291805
individuo 3	1.442271	0.22484424	0.31776229
individuo 4	0.718466	0.11200595	0.42976824
individuo 5	1.373180	0.21407323	0.64384147
individuo 6	0.420877	0.06561303	0.7094545
individuo 7	0.913392	0.14239413	0.85184863
individuo 8	0.634017	0.0988407	0.95068933
individuo 9	0.316305	0.04931067	1

$\sum_{i\_1}^{m} f_i^t$ 6.414534

P_1^t= 0.026602 / 6.414534 q_1^t = 0.00415

P_2^t= 0.569424 / 6.414534 q_2^t = 0.00415 + 0.08877

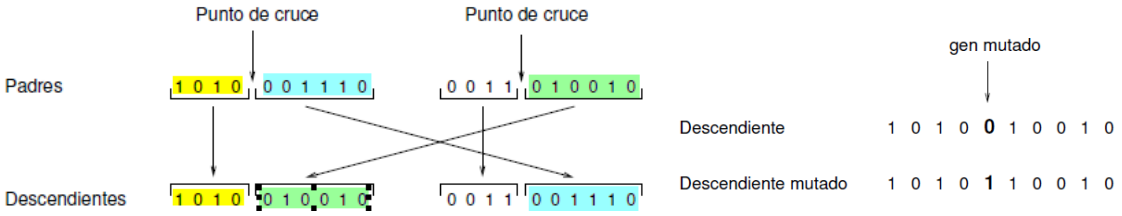
P_3^t= 1.442271 / 6.414534 q_3^t = 0.00415 + 0.08877 + 0.22484

Con el muestreo estocástico con reemplazamiento puede ocurrir que un mismo individuo sea seleccionado un número excesivo de veces (incluso podría completar la nueva población), lo que daría lugar a poblaciones muy uniformes.

Operadores genéticos (Combinación y modificación):

Una vez seleccionados un par de padres, el operador de cruce simple se aplica a éstos con probabilidad p_{cruc}, donde p_{cruc} es un parámetro de entrada el cual nos determina el número esperado de cromosomas a los cuales se les aplicará el operador de cruce en cada generación.

Seguidamente se aplica, a cada hijo, el operador de mutación. En cada gen del cromosoma se lleva a cabo un cambio de 0 a 1, o viceversa, con probabilidad $p_{mut.}$



El objetivo de la mutación es salir de los óptimos (máximos) locales.

sustitución generacional:

En el AG simple que estamos describiendo se usa el esquema de sustitución generacional completa, junto con una estrategia elitista. Con el elitismo, el mejor individuo de la población actual sobrevive en la nueva población. El resto de los individuos de la nueva población lo constituyen los descendientes generados vía *selección-cruce-mutación*, hasta formar una nueva población completa de m individuos.

La sustitución generacional completa puede producir un efecto negativo en la componente de explotación del AG.

En un algoritmo evolutivo en general, la nueva población se elige entre la población actual y los descendientes, $P(t) \cup C(t)$, pero si se trata de sustitución generacional completa en un algoritmo genético, lo más habitual es que toda la población (o casi toda, si es con elitismo) se sustituya solo por los descendientes obtenidos después de selección, cruce y mutación (no de la población actual). Esto es así, porque la población actual ya está representada en los seleccionados como padres y la tasa de cruce y mutación suele ser baja. Realmente podemos considerar que hay una gama de estrategias de sustitución generacional en algoritmos genéticos, desde la sustitución completa sin elitismo donde todos los m individuos se sustituyen por m descendientes, pasando por algo de elitismo donde uno o unos pocos de mejor adecuación se mantienen y se sustituye al resto, hasta la sustitución *steady-state* (estado estacionario) que se selecciona un número $n \in [1..m]$ de individuos los cuales serán sustituidos por n nuevos.

Ventajas del uso de los AG:

- Se adapta mucho a la programación funcional
- Es poco sensible a los mínimos locales, lo cual le confiere robustez.
- Asimismo, no depende de las condiciones iniciales, debido a que se usa búsqueda estocástica y ésta hace al principio un gran número de intentos aleatorios.
- El tiempo de convergencia de los AG es predecible por la naturaleza paralela de la búsqueda estocástica.
- Funciona de forma paralela, por lo que pueden usarse en sistemas distribuidos para mejorar más la velocidad de búsqueda.

Inconvenientes del uso de los AG:

- No hay un marco teórico genérico establecido.
- Si la población inicial es cercana a la solución óptima, los AG tardarán más que las técnicas de resolución tradicionales. (El AG perderá mucho tiempo comprobando soluciones sub-óptimas.)
- Hacen buenas estimaciones de la solución óptima, pero no la calculan exactamente.
- El usuario debe determinar cómo de cerca está la solución estimada de la solución real. (La proximidad a la solución real dependerá de la aplicación en concreto.)

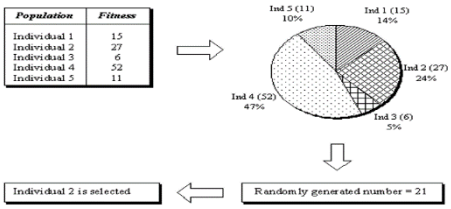
Diseño algoritmos evolutivos

Aunque la teoría sostiene que los AE convergen en soluciones optímales, en aplicaciones prácticas surgen inconvenientes que hacen que no siempre se cumplan los resultados teóricos. Una de las causas comunes que hacen que los AE se vean incapacitados para encontrar soluciones óptimas es el fenómeno de la convergencia prematura. Tal problema consiste en una convergencia demasiado rápida, posiblemente en un óptimo local. En CE, esto es debido a la presencia de **superindividuos** en la población, los cuales son mucho mejores que la adecuación media de la población, por lo que acarrean un gran número de descendientes e impiden que otros individuos contribuyan con su descendencia en la siguiente generación, con la consecuente pérdida de información, lo que lleva consigo la formación de poblaciones altamente uniformes e incapaces de evolucionar. Además, existen problemas que contienen restricciones no triviales, problemas con función multimodal, problemas con múltiples objetivos los cuales resultan difíciles o imposibles de resolver con el esquema simple de AG.

Todo esto ha llevado en los últimos años al estudio de mejoras y extensiones sobre los AE que proporcionan un mejor rendimiento:

- Representación: alfabetos binarios, números enteros y reales (especialmente útiles en problemas de optimización numérica) y una gran variedad de estructuras de datos diferentes como, por ejemplo, matrices, permutaciones, listas, árboles, grafos, etc.

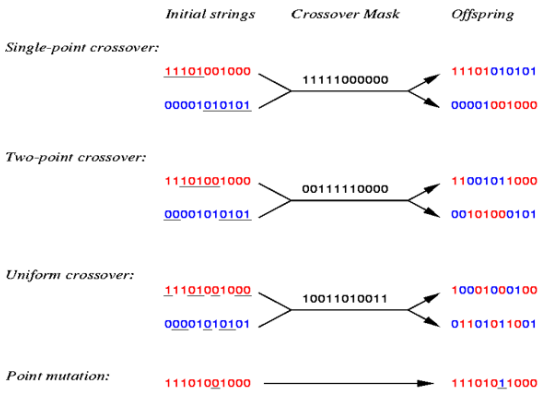
- Esquemas de selección, muestreo y sustitución generacional:
  - **Selección por ranking:** los individuos son ordenados en una lista de acuerdo con sus adecuaciones, y la probabilidad de selección de un individuo se obtiene según su posición en la lista ordenada.
  - **Selección por torneo:** Se selecciona el individuo con mejor adecuación de un grupo de individuos (2 en el torneo binario) elegidos aleatoriamente de la población. (Equivalente a un ranking lineal fijo)
  - **Muestreo estocástico universal**: La idea básica de este método es construir una rueda de ruleta con *m* punteros distribuidos equidistantemente, en lugar de un solo puntero como ocurre con la rueda de ruleta convencional. De esta forma, con un simple "giro" de la ruleta se seleccionan *m* individuos a la vez. Usa un numero aleatorio para seleccionar los *m* individuos a la vez.



  - **Sustitución steady-state:** se selecciona un número $n \in [1...m]$ de individuos los cuales serán sustituidos por *n* nuevos. Para la elección de los individuos que mueren suele utilizarse un ranking inverso, es decir, ordenando la lista de cromosomas de menor a mayor adecuación. Es una alternativa a la sustitución generacional completa (solo se sustituyen unos pocos de la población actual (los de peor adecuación) y el resto se mantienen)
  - **Modelo del factor del crowding:** Un individuo nuevo sustituye a uno viejo el cual es seleccionado de entre un subconjunto de *CF* miembros elegidos aleatoriamente de la población completa. En este subconjunto se selecciona para morir el individuo más parecido al nuevo, usando como medida un contador de semejanza bit-a-bit.

Los métodos de selección se ordenan en orden creciente de presión selectiva (para valores estándares de sus parámetros), del siguiente modo: selección proporcional, selección por ranking, selección por torneo.

- Operadores de variación:
  - **Cruce uniforme (rep. binarias y reales):** se utiliza una máscara de cruce generada aleatoriamente. Donde hay un 1 en la máscara, los genes en el primer hijo se toman del primer padre, y donde hay un 0 los genes se toman del segundo padre. Los genes del segundo hijo se establecen con las decisiones inversas.



  - **Cruce aritmético (rep. reales).** Combinación lineal complementarias de los padres.
  - **Cruce plano (rep. reales).** Un hijo, cada gen aleatorio en rango de padres.
  - **Cruce BLX-α (rep. reales).** Como anterior, pero rango extendido en frac. α.
  - **Cruce PMX, OX y CX.** Usados cuando se utilizan permutaciones como representación(TSP).
  - **Mutación uniforme (rep. reales).** Cambiar el gen por otro valor distinto.
  - **Mutación no uniforme (rep. reales).** Cambiar gen a valor aleatorio 'cercano'.
  - **Mutación por intercambio.** Se intercambian dos genes elegidos aleatoriamente.

- Manejo de restricciones: El manejo de restricciones no triviales no es fácil de implementar en CE. Básicamente, hay dos enfoques para manejar individuos no factibles, dependiendo de que se permita o no la presencia de individuos no factibles en la población:

  A. Métodos abortivos, eliminan los individuos ilegales que se generen.
  B. Métodos anticonceptivos, no permiten la generación de individuos ilegales.

  C. Funciones de penalización, degradan/penalizan la adecuación de las soluciones en relación con el grado de violación de las restricciones permitiendo la presencia de soluciones no factibles.

Optimización multiobjetivo: La mayor parte de los problemas de optimización del mundo real suelen tener dos o más funciones objetivo que deben satisfacerse simultáneamente y que posiblemente están en conflicto entre sí. Sin embargo, con la finalidad de simplificar su solución, muchos de estos problemas tienden a modelarse como mono objetivo. De esta forma, con la ayuda de algún conocimiento del problema, los problemas multiobjetivo se transforman usando sólo una de las funciones originales y manejando las adicionales como restricciones, o reduciendo el vector de objetivos a uno solo, el cual optimizar.

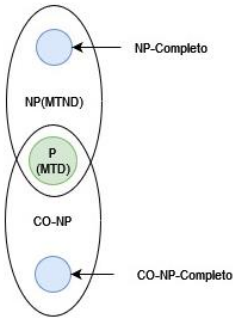
Una solución factible $\vec{x} \in \mathcal{F}$ para el problema representado por la Expresión 11.2 *domina* a otra solución $\vec{x}' \in \mathcal{F}$ si:

$$f_i(\vec{x}) \leq f_i(\vec{x}') \quad \textit{para todo } i = 1, \ldots, n$$
$$y \quad f_i(\vec{x}) < f_i(\vec{x}') \quad \textit{para al menos un } i = 1, \ldots, n$$

Una solución factible $\vec{x} \in \mathcal{F}$ es Pareto óptima o *no dominada* para el problema planteado si no existe ninguna solución $\vec{x}' \in \mathcal{F}$ tal que $\vec{x}'$ domine a $\vec{x}$. Dicho de otra forma, una solución es Pareto óptima si no existe ninguna otra solución que mejore en algún objetivo sin empeorar simultáneamente otro objetivo.

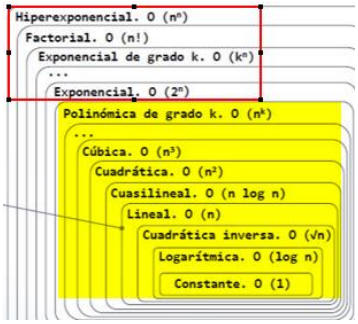
El viajante de comercio (TSP)

El TSP está entre los problemas denominados *NP-completos*, esto es, los problemas que no se pueden resolver en tiempo polinomial en función del tamaño de la entrada.



Los problemas *NP-Completos* son aquellos problemas que: Pertenecen a *NP* y no pueden ser reducidos a tiempo polinomial. Es decir, todos aquellos problemas que pertenecen a *NP* pero no a *P* (*P≠NP).*

Actualmente los investigadores piensan que las clases cumplen con el diagrama mostrado por lo que *P* y *NP-completo* tendrían intersección vacía. La importancia de la pregunta *P = NP* radica en que, de encontrarse un algoritmo en *P* para un problema *NPcompleto*, todos los problemas *NP-completos* (y por ende, todos los problemas de *NP*) tendrían soluciones en tiempo polinómico.



La línea divisoria entre los problemas tratables (se pueden resolver de forma eficiente) y los intratables (aquellos que se pueden resolver, pero no lo suficientemente rápido para ser útiles) se encuentra, generalmente, entre lo que se puede calcular en un tiempo polinómico y lo que requiere un tiempo mayor que un tiempo de ejecución polinómico.

TSP:Usando AE, un individuo: (3 1 4 6 9 8 5 2 7) representa la solución que, partiendo de una ciudad origen, realiza el camino 3 − 1 − 4 − 6 − 9 − 8 − 5 − 2 − 7, para retornar finalmente a la ciudad de origen.

La representación de las soluciones es por tanto una permutación de números enteros que representan ciudades, en la cual cada entero aparece una sola vez.

Establecida la representación de soluciones y su inicialización, el siguiente paso es diseñar operadores de variación que se adapten a la representación (e.g. Cruce PMX – Mutación intercambio o inversión).

- **Cruce-PMX**: Dados dos padres y los puntos de cruce (aleatorios), se producen dos hijos.

p1 = (1 2 3 | 4 5 6 7 | 8 9) p2 = (4 5 2 | 1 8 7 6 | 9 3)

h1 = (**1** 2 3 | **1** **8** 7 6 | **8** 9) h2 = (**4** **5** 2 | **4** **5** 6 7 | 9 3)

El algoritmo de reparo comienza estableciendo los siguientes mapeos: (1-4, 8-5, 7-6, 6-7)

h1 = (* * * | 1 8 7 6 | * *)

h2 = (* * * | 4 5 6 7 | * *)

Las ciudades que presentan un conflicto son entonces cambiadas siguiendo la tabla de mapeos.

h1 = (**1** 2 3 | **1 8** 7 6 | **8** 9) h2 = (**4 5** 2 | **4 5** 6 7 | 9 3)

h1' = (4 2 3 | 1 8 7 6 | 5 9) h2' = (1 8 2 | 4 5 6 7 | 9 3)

- **Mutación por intercambio recíproco**: se eligen dos ciudades aleatoriamente, y se intercambian,

h1' = (4 **2** 3 | 1 8 7 6 | 5 **9**) → h1'' = (4 **9** 3 | 1 8 7 6 | 5 **2**)

- **Mutación por Inversión:** se establecen dos puntos aleatorios entre los cuales el orden de las ciudades es invertido,

h1' = (4 9 |**3 1 8 7 6** | 5 2) → h1'' = (4 9 |**6 7 8 1 3** | 5 2)

Selección de variables en Minería de Datos

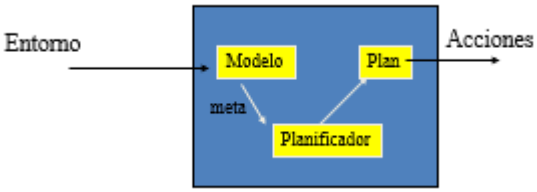
La Minería de Datos (MD) tiene como objetivo la criba de datos para revelar información útil por medio de una representación adecuada al usuario, comprimiendo enormes registros de datos. utilizamos el término Descubrimiento de Conocimiento en Bases de Datos (DCBD) o sólo Descubrimiento de Conocimiento (DC) para denotar el proceso completo de extraer conocimiento de alto nivel a partir de datos de bajo nivel, y el término MD como el acto concreto de extraer patrones o modelos de los datos.

A medida que intentamos resolver problemas del mundo real, nos damos cuenta de que son normalmente sistemas mal definidos, difíciles de modelar y con espacios de solución de gran escala. En estos casos, los modelos precisos son poco prácticos, demasiado caros o inexistentes. La información relevante disponible está normalmente en la forma de conocimiento empírico previo y datos del tipo entrada-salida representando instancias del comportamiento del sistema. Así pues, necesitamos sistemas de razonamiento aproximado capaces de manejar esa información imperfecta. *Soft Computing* es un término acuñado recientemente que describe el uso simbiótico de muchas disciplinas emergentes de computación que intentan manejar la información imperfecta. en contraste con lo tradicional, *soft computing* es tolerante a la imprecisión, a la incertidumbre y a la verdad parcial". Dentro de estas técnicas tenemos la Lógica Difusa, el Razonamiento Probabilístico, las Redes Neuronales y los Algoritmos Evolutivos.

Como sucede en la MD, si intentamos revelar información útil de los datos, un paso fundamental es el preprocesamiento de los datos para seleccionar las variables más adecuadas. En este proceso, la identificación de la estructura de un sistema tiene que encontrar las variables que representan los datos de un modo más preciso, de entre una colección de posibles variables. En este contexto tenemos que seleccionar un número finito de variables entre una colección finita de posibles candidatos (es un problema combinatorio).

TEMA 3: Planificación

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



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

- Modelo observable.
- Modelo estático (no posee dinámica interna por lo que permanecerá en cierto estado hasta que se aplique alguna acción.).
- Modelo determinista (Si una acción es aplicable a un estado, su aplicación lleva un sistema determinista a otro estado).
- Enfoque proposicional.
- Acciones instantáneas.
- Acciones no descomponibles.
- Planificación offline.

Elementos representados en un Sistema Planificación basado en Búsqueda:

- Acciones que generan descripciones de estados nuevos.
- Estados completamente representados, usados en heurísticas, generar sucesores, probar metas.
- Metas contra las cuales probar, como cajas negras.
- Solución o Plan como secuencia de acciones.

En planificación clásica existe la suposición de metras restringidas. ¿Cuál es esta restricción?, El planificador maneja sólo metas restringidas que son especificadas por un estado meta s_g o varios S_g.

¿Qué pasaría si se relaja? Al relajar esta suposición se podría trabajar con objetivos más complejos que un estado o estados finales, como alcanzar o evitar estados intermedios, funciones de utilidad, etc.

Diferencias que hay entre una **acción** y un **evento** desde el punto de vista de la planificación: A la evolución del sistema contribuyen tanto las acciones como los eventos. La diferencia yace en si el planificador tiene el control de ellas o no. Las acciones son transiciones controladas por el ejecutor del plan. Los eventos son transiciones contingentes: a pesar de ser controladas por el ejecutor del plan, los eventos corresponden a la dinámica interna del sistema, y por lo cual no se pueden seleccionar o disparar por parte del planificador.

La planificación lineal se caracteriza por:

- Es una planificación en el espacio de estados que se puede representar por un grafo cuyos nodos son estados posibles del sistema y los enlaces entre los nodos las acciones que se puede aplicar para pasar de unos estados posibles a otros
- Supone que los objetivos del problema a planificar son independientes entre sí.
- El resultado es un conjunto de acciones secuenciales y totalmente instanciadas.

La planificación no lineal se caracteriza por:

- No considera que los objetivos del problema son totalmente independientes entre sí.
- No es necesario establecer el plan de un determinado orden de ejecución. De esta forma un plan correcto en la planificación no lineal puede ser un plan parcialmente ordenado.

Problema de planificación (STRIPS)

Un problema de planificación se describe mediante un conjunto de acciones, un estado inicial del entorno y una descripción del objetivo a conseguir (Dominio del problema).

Un problema de planificación STRIPS se define como una tripleta *P = <I, G, O>* donde *I* es el estado inicial del problema, *G* es el objetivo(goal) a conseguir y *O* es el conjunto de operadores de planificación. En STRIPS se trabaja con literales de primer orden y las descripciones de estados se componen de literales positivos totalmente instanciados y sin dependencias funcionales.

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

- Los objetivos son conjunciones y los efectos son conjunciones.
- No soporta igualdades y no soporta tipos.
- Sólo literales positivos en estados.
- Hipótesis del mundo cerrado: los literales no mencionados son falsos.
- Sólo literales simples en objetivos.

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

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

- En cada llamada recursiva del algoritmo STRIPS, los estados objetivos predecesores se forman únicamente con las precondiciones de la última acción, permitiendo así reducir sustancialmente el factor de ramificación.
- Si en el estado actual del problema se satisfacen todas las precondiciones de una acción se dice que es APLICABLE, STRIPS ejecuta dicha acción en el estado actual del problema y actualiza el estado de planificación. Con este tipo de funcionamiento se consigue igualmente reducir una gran parte del espacio de búsqueda.

¿Qué es la suposición de STRIPS?, Permite evitar la complejidad del problema marco: los únicos cambios que se producen como resultado de la aplicación de una acción son aquellos que explícitamente se mencionan como efectos de esta. Es decir, se deduce automáticamente que el resto de las relaciones y predicados asociados con la situación en la que la acción ha sido aplicada se satisfacen también en la nueva situación. Introduce una sintaxis sencilla para definir operadores, en términos de precondiciones, efectos añadidos (efectos add) y efectos borrados (efectos delete).

Lenguaje de planificación PDDL

Actualmente existen varias versiones de PDDL, siendo la más reciente PDDL3 que engloba todas las características de sus predecesoras e incluye nuevas funcionalidades. En esta sección se apuntan sólo aquellas características de PDDL que introducen un cambio en el modelo semántico de las acciones de STRIPS.

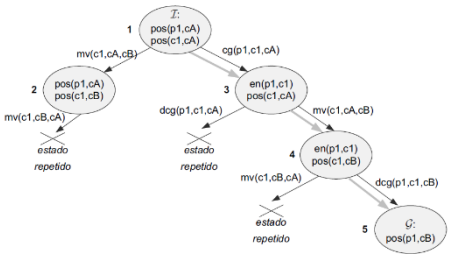
- Acciones con duración (acciones durativas), donde precondiciones y efectos se asocian a los puntos de inicio y final de la duración de la acción; además, las precondiciones también se pueden asociar a todo el intervalo de duración. De esta forma se puede modelar mejor la física del problema.
- Precondiciones y efectos con expresiones y variables numéricas. E.g. controlar el consumo de combustible (en litros) de un camión, comprobando que el camión dispone de cantidad suficiente para realizar un recorrido determinado (precondición numérica) y actualizando dicho valor al final de la ejecución de la acción (efecto numérico).
- Métricas del problema. Permiten definir funciones de optimización como una combinación lineal de uno o varios factores del problema. Por ejemplo, se puede definir una métrica para minimizar el consumo total de combustible de todos los camiones de un problema, ponderando cada factor con un porcentaje determinado.
- Ventanas temporales. Se utilizan como una forma restrictiva de expresar eventos (e.g. el almacén donde se guardan los paquetes está abierto desde las 8h hasta las 20h).
- Restricciones duras y blandas sobre el plan. Definen restricciones sobre la estructura de los planes que deben satisfacerse en los estados intermedios del plan. Estas restricciones se expresan mediante operadores modales como *always, sometime, at-most-once, at-end, always-within, hold-after*, etc.

Soportar estas características permite que PDDL sea un lenguaje de modelado útil para la definición de problemas más cercanos a la realidad, haciendo hincapié no sólo en los elementos clave para conseguir un plan, sino también en los necesarios para especificar los criterios de calidad del plan.

Características sintácticas de los problemas que se pueden describir con PDDL

- Efectos condicionales.
- Cuantificación universal sobre universos dinámicos (es decir, creación y destrucción de objetos).
- Axiomas del dominio.
- Especificación de restricciones de seguridad.
- Especificación de acciones jerárquicas compuestas de subacciones y subobjetivos.
- Manejo de problemas múltiples en múltiples dominios utilizando diferentes subconjuntos de características del lenguaje (para soportar la compartición de dominios entre diferentes planificadores que manejan distintos niveles de expresividad).

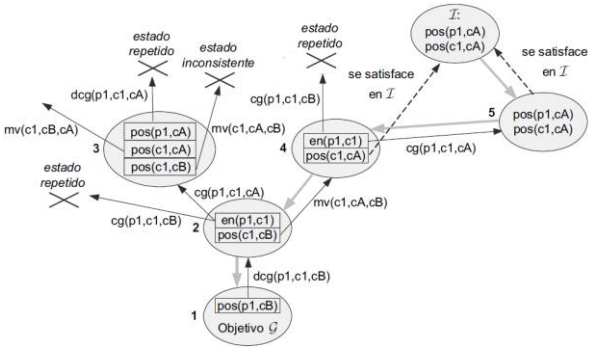
Planificación en un espacio de estados



- Búsqueda hacia delante (progresiva): Planificador de orden total. Los nodos del árbol representan estados del problema de planificación, siendo el nodo raíz del árbol el estado inicial del problema. Un nodo se expande a partir de todas las acciones del problema que son aplicables en dicho estado, generando así el nuevo conjunto de nodos (estados) sucesores, los cuales formarán parte de la lista ABIERTA del árbol.

  Tiene los inconvenientes; Se consideran todas las acciones aplicables en un estado sin desestimar aquellas acciones que son irrelevantes para la consecución de los objetivos, en dominios reversibles donde existen acciones inversas que conducen de nuevo al estado antecesor, se producen infinidad de ciclos en el árbol de búsqueda, el factor de ramificación para problemas de gran tamaño puede resultar en una explosión del espacio de búsqueda. Se consideró que la búsqueda hacia delante en un espacio de estados era una estrategia muy poco eficiente. Sin embargo, esta consideración ha cambiado recientemente gracias al empleo de excelentes funciones heurísticas.

- Búsqueda hacia atrás (regresiva): Planificador de orden total. Se parte del objetivo *G* del problema, el cual está constituido por el conjunto de literales a conseguir *G = {g1, g2, . . . gn}* y aplica inversamente los operadores del problema, generando así estados subobjetivos. El proceso terminará si se produce un conjunto de subobjetivos que se satisface en *I*. Una importante ventaja que presenta la búsqueda regresiva frente a la búsqueda progresiva es que permite considerar sólo acciones relevantes para la consecución de los objetivos, reduciendo de este modo el factor de ramificación del árbol de búsqueda en varios órdenes de magnitud.



¿De qué otros modos se pueden representar objetivos en un espacio de transición de estados?

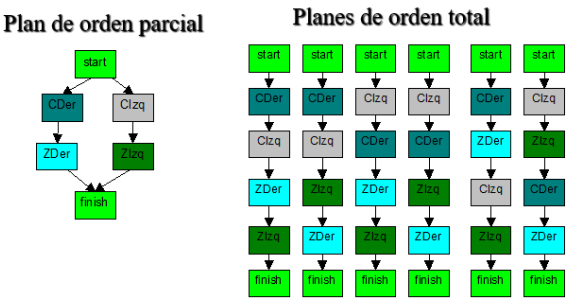
- La especificación más simple consiste en un estado meta o conjunto de estados. En este caso, se alcanza el objetivo gracias a una secuencia de transiciones de estado que terminan en uno de los estados meta.
- Más en general, el objetivo sería satisfacer alguna condición sobre la secuencia de estados. Por ejemplo, se podría querer que un estado se evitase o que se alcanzase en cierto punto de la secuencia, etc.
- Una especificación alternativa sería una función de utilidad sobre los estados. La meta sería optimizar algún componente de esa función de utilidades durante la secuencia de estados.
- Otra alternativa consiste en especificar los objetivos como tareas que el sistema debería realizar. Esas tareas pueden ser definidas como un conjunto de acciones y otras tareas.

Planificación de Orden Parcial

En la planificación de orden total las acciones del plan se obtienen en el mismo orden en que éstas se ejecutarían; esto es, el orden de planificación coincide con el orden de ejecución. De este modo, si un

planificador selecciona una acción equivocada, deberá incorporar una acción adicional para deshacer los efectos de la acción errónea y volver a un estado de planificación correcto.

En problemas complejos se suceden múltiples interacciones entre los objetivos del problema porque las acciones que se utilizan para resolver un objetivo pueden interferir en la solución de otro objetivo. Un planificador que tenga en cuenta esta característica generará un plan entrelazado en el que se trabaja simultáneamente con múltiples objetivos del problema. De este modo, el planificador tendrá que tomar decisiones sobre cómo las acciones que resuelven una parte del problema afectan a la resolución del resto del problema. A este tipo de aproximación se le denomina planificación de orden parcial (POP). La ventaja de la aproximación POP es la flexibilidad a la hora de establecer un orden entre las acciones que componen un plan. En POP se trabaja sobre los objetivos del problema simultáneamente y se mantiene un orden parcial entre las acciones sin tener que comprometer un orden concreto entre las mismas, hasta que la propia estructura del plan determina cuál debe ser este orden. A la estrategia general de retrasar una decisión durante el proceso de búsqueda se le conoce como **menor compromiso**. (Sólo hay que realizar elecciones de lo que interesa en cada momento, dejando las otras elecciones para más tarde).



Estructura:

- Pasos del plan. Un paso es una versión total o parcialmente instanciada de un operador del problema. Por consiguiente, cada paso de un plan se compone de un conjunto de precondiciones y efectos que están, a priori, total o parcialmente instanciados. Cuando finalmente se obtiene el plan que resuelve el problema, los pasos del plan serán instancias completas de los operadores del plan (acciones).
- Restricciones de orden: Una restricción de orden (precedencia) entre dos pasos del plan P_i, P_j se representa como $P_i < P_j$ y denota que el paso P_i debe ejecutarse antes que el paso P_j, aunque no necesariamente inmediatamente antes.
- Restricciones entre variables: Un POP también puede contener un conjunto de restricciones sobre variables como **restricciones de desigualdad** del tipo ***?x≠?y***, donde *?x* es una variable de un paso del plan e *?y* es una constante u otra variable. Otro tipo de restricciones que puede contener son las denominadas restricciones de **codesignación** o **unificación**. Éstas son restricciones del tipo ***?x=?y*** donde se fuerza que la variable *?x* tenga el mismo valor que *?y*.
- Enlaces causales y amenazas: Un enlace causal entre dos pasos de un plan P_i, P_j se escribe como $<P_i, l, P_j>$ y denota que P_i tiene un efecto que consigue el literal l para una precondición de P_j. Todo enlace causal establece por defecto una restricción de orden entre los dos pasos. Un paso P_k entra en conflicto o amenaza un enlace causal $<P_i, l, P_j>$ si P_k tiene un efecto $\neg l$ y P_k puede situarse entre P_i y P_j. Para resolver la amenaza el paso P_k debería situarse antes de P_i (método promoción) o después de P_j (método democión).

Búsqueda espacio de planes POP:

Un proceso de búsqueda en un espacio de planes realiza una exploración en un árbol donde cada nodo representa un plan parcial. se puede ver que las acciones aplicables en esta búsqueda no son acciones ejecutables en el mundo real sino acciones sobre los planes, como añadir un paso, establecer un orden entre acciones, etc.

El proceso termina cuando: 1) se selecciona un plan parcial que es correcto, es decir, que resuelve el problema, o 2) la lista de nodos a expandir del árbol se queda vacía, lo que indica que el problema no tiene solución

Algoritmo 13.2 Esquema básico de búsqueda en un espacio de planes para POP

```
1: Lista_planes ← {Plan vacío}
2: repetir
3:    Seleccionar no determinísticamente y extraer Π ∈ Lista_planes
4:    Pendiente ← prec_no_resueltas(Π) ⋃ amenazas(Π)
5:    si Pendiente = ∅ entonces
6:       Devuelve Π {Éxito}
7:    fin si
8:    Seleccionar y extraer Φ ∈ Pendiente
9:    Relevante ← {Π_r}∀ Π_r que resuelve Φ {Cada r es una forma (plan parcial) de resolver Φ}
10:   si Relevante ≠ ∅ entonces
11:      Lista_planes ← Lista_planes ⋃ Relevante
12:   fin si
13: hasta lista_planes = ∅
14: Devuelve fallo {No existe plan}
```

A pesar de las ventajas que las aproximaciones POP aportan sobre la búsqueda basada en estados, la complejidad de la resolución de un problema sigue siendo muy elevada. El número de nodos que se pueden generar en un proceso de búsqueda en un espacio de planes es:

$$((B_e + B_n) * m^a)^{pn}$$

- B_e, son las formas de resolver una precondición con pasos existentes en el plan.
- B_n, son las formas de resolver una precondición introduciendo pasos nuevos.
- *m,* son las formas de resolver una amenaza.
- *a,* es el número de amenazas de un plan.
- *P,* precondiciones.
- *n,* número de pasos del plan.

Heurísticas:

Los dos puntos claves en una búsqueda en un espacio de planes son la selección del plan de *Lista_Planes* y la selección, dentro de dicho plan, del objetivo pendiente a estudiar.

- Heurísticas para selección de planes: La estrategia habitual es ordenar los nodos frontera del árbol por el coste total de los operadores del plan (número de pasos) y seleccionar el plan con el mínimo coste. Para diseñar una buena estrategia heurística hay que determinar cómo afectan cada uno de los componentes de un plan a los factores *g(Π)* 'coste de la solución desde el nodo raíz hasta plan *Π*'y *h(Π)* 'coste de la solución desde el plan Π hasta el objetivo'.
    - Número de pasos N: se utiliza habitualmente como una medida del coste del camino (número de pasos que contiene).
    - Precondiciones sin resolver OP: dado que para cada precondición pendiente de resolver se debe introducir un paso, OP se puede incluir como medida heurística en *h(Π).*
    - Amenazas A: son un factor circunstancial de un plan que aparecen en un momento concreto y que pueden resolverse, o bien desaparecer, a medida que el plan se va construyendo.
- Heurísticas para selección de objetivo pendiente: las estrategias existentes priorizan entre escoger una amenaza o una precondición pendiente de resolución y entre ellas se establece otra prioridad para determinar el objetivo candidato a ser estudiado:
    - LIFO.

- Retrasar las amenazas potenciales. es preferible ignorar cualquier amenaza potencial que pueda haber en un plan y que los objetivos pendientes deben resolverse en el siguiente orden: 1) amenazas no resolubles, 2) amenazas que puedan resolverse con un solo método, 3) precondiciones sin resolver, y 4) amenazas que puedan resolverse con más de un método.
- Objetivo de menor coste. Esta estrategia selecciona la amenaza o precondición pendiente de menor coste de reparación, es decir, aquel objetivo pendiente con menos alternativas de solución.
- ZLIFO (Zero LIFO - heurística que mejores resultados). El objetivo de esta estrategia es dar mayor prioridad a aquellas precondiciones que puedan resolverse determinísticamente (0 ó 1 solución).

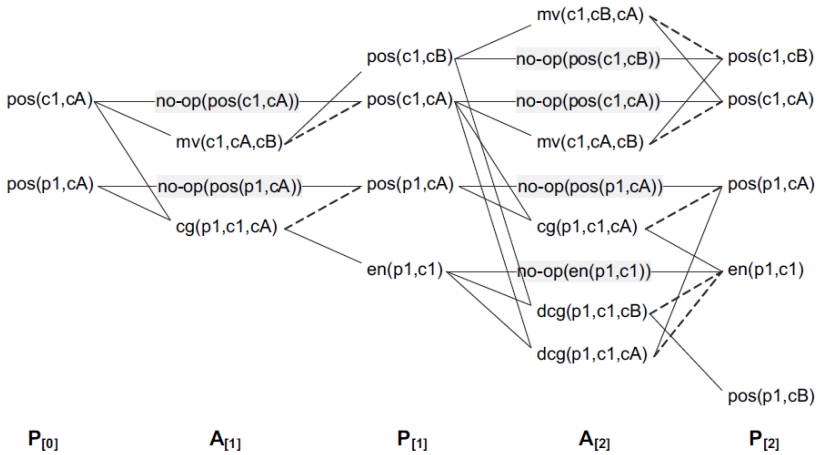
Planificación basada en grafos

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

Un grafo de planificación es una estructura de tamaño polinómico que modela la parte estática (proposiciones y acciones) y dinámica (relaciones de orden). La única restricción que impone el grafo de planificación es la de manejar un enfoque proposicional basado en STRIPS, es decir con literales positivos. Tanto las acciones como proposiciones (literales positivos) deben estar totalmente instanciadas (sin variables).

Es un grafo dirigido multinivel con dos tipos de nodos (proposición y acción) y tres tipos de aristas (*aristas-Pre, aristas-Efe$^+$ y aristas-Efe$^-$).* Si no contempla *Efe$^-$* se denomina ***grafo de planificación relajado*** (demasiado permisivo en cuanto a las acciones o proposiciones que pueden aparecer en un determinado nivel). Cada nivel *t* del grafo contiene, a su vez, un nivel de acción $A_{[t]}$ y uno de proposición $P_{[t]}$ con las acciones y proposiciones presentes, respectivamente, en dicho nivel.

Un nivel de un grafo de planificación no representa un estado individual válido, sino la unión de varios de ellos que no siempre pueden satisfacerse simultáneamente en el mundo real



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

- Mantiene una noción interna del tiempo mediante los niveles o pasos de planificación, y representa implícitamente muchas de las restricciones entre acciones y proposiciones.

- Codifica un plan que se encuentra a mitad camino entre un plan de orden parcial y uno de orden total.
- Complejidad temporal y espacial polinómica con respecto al tamaño del problema.
- Si existe un plan válido de t pasos para un problema de planificación, dicho plan existe como un subgrafo del grafo de planificación correspondiente de t niveles e lo contrario si no existe.

Graphplan:

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

Graphplan, amplía la construcción del grafo de planificación con un razonamiento sobre relaciones de exclusión mutua entre pares de nodos de un mismo nivel (acción-acción o proposición-proposición). En general, se dice que dos acciones o proposiciones de un mismo nivel son mutuamente excluyentes (*mutex*) si ningún plan válido puede contenerlas simultáneamente, es decir, no pueden coexistir en el mundo real. El cálculo de esta información de exclusión mutua produce un grafo de planificación más informado que permite detectar con mayor precisión el nivel desde el que comenzar la búsqueda y restringir el espacio de búsqueda.

Graphplan calcula y propaga estas relaciones mediante la aplicación estas reglas:

- Acciones; dos acciones a y b son *mutex* cuando: 1) a borra una precondición o efecto positivo de b (interferencia), o 2) una precondición de a y una de b se han marcado como mutuamente excluyentes en el nivel de proposición anterior (necesidades competitivas).
- Proposiciones; dos proposiciones p y q son *mutex* si todas las formas (acciones) de alcanzar p son excluyentes con todas las formas de alcanzar q.

Es importante decir que los *mutex* entre proposiciones son consecuencia directa de la propagación de los *mutex* entre acciones a lo largo del grafo de planificación.

Las reglas que aplica Graphplan no garantizan encontrar todas las relaciones de exclusión, sí son lo suficientemente potentes para determinar cuándo dos acciones o dos proposiciones no pueden darse simultáneamente.

Algoritmo 13.3 Esquema básico de Graphplan

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

El proceso de búsqueda llevado a cabo por Graphplan se puede considerar equivalente al de resolución de un CSP (Constraint satisfaction problem).

Los grafos de planificación proporcionan una estructura eficiente que permite calcular estimaciones heurísticas útiles para guiar la búsqueda en planificadores progresivos basados en estados.

Planificación basada en satisfactibilidad

La idea de abordar un problema de planificación como un problema basado en satisfacibilidad surge con el objetivo de reutilizar técnicas basadas en satisfacción de fórmulas proposicionales para su aplicación a planificación. Primero convierte el problema en fórmulas proposicionales y busca un modelo que las satisfaga.

Algoritmo 13.4 Esquema básico de planificación basada en satisfacibilidad

```
1: formulas ← Traducir_a_SAT(I + Acciones + G, longitud)
2: modelo ← Algoritmo_Resolucion_SAT(formulas)
3: si ∃ modelo entonces
4:     Devuelve Extraer_Plan(modelo) {Éxito, se ha encontrado un plan con la longitud dada}
5: si no
6:     Devuelve fallo {No existe solución con la longitud dada}
7: fin si
```

Planificación para el mundo real

- Control y navegación de robots autónomos.
- Planificación de rutas.
- Entornos simulados.
- Entornos de red.
- Situaciones catastróficas y manejo de crisis.
- Aplicaciones militares.

Son muchos los escenarios que se benefician de las técnicas de planificación, pero, para ello, es necesario ir más allá de la planificación clásica. Los problemas del mundo real requieren características que exceden del modelo clásico y de sus asunciones, como gestión de duraciones y recursos, optimización multicriterio, abstracción de acciones o trabajo con incertidumbre.

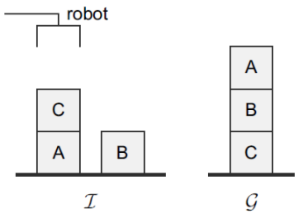
Planificación temporal: En los problemas reales de planificación no es aceptable que todas las acciones tengan la misma duración. Por lo tanto, la primera asunción de la planificación clásica que relajaremos es la A5: las acciones ahora tienen distinta duración y es el planificador el que debe gestionar las características temporales de forma explícita. En un contexto de planificación temporal el criterio de optimización de los planes se centra en minimizar la duración del plan, en lugar del número de pasos o acciones del mismo. Para amoldarse a la duración de la acción (Dur) y hacer frente a esta limitación, el modelo de acción se extiende teniendo en cuenta condiciones locales: precondiciones (Pre), postcondiciones (Post) y condiciones invariantes (Inv) que deben satisfacerse antes, después y durante la ejecución de la acción, respectivamente. Adicionalmente, los efectos de las acción también se dividen en efectos iniciales (EfeI) y finales (EfeF) que se generan al principio y final de la acción, respectivamente.

Problema: Anomalia de Sussman

Dominio Problema = <I, G, O> "se asume la hipótesis de mundo cerrado"

I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre

G: sobre(A,B) ∧ sobre(B,C)



O: conjunto de operadores de planificación

- recoger(?b)
  - Pre: libre(?b) ∧ en-mesa(?b) ∧ rob-libre
  - Efe: sujeto(?b) ∧¬libre(?b) ∧¬en-mesa(?b) ∧¬rob-libre
- soltar(?b)
  - Pre: sujeto(?b)
  - Efe: libre(?b) ∧ en-mesa(?b) ∧ rob-libre ∧¬sujeto(?b)

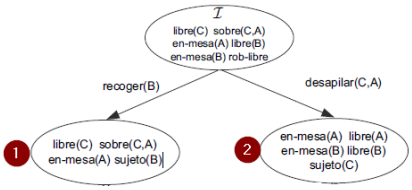
- apilar(?b1,?b2)
  - Pre: sujeto(?b1) ∧ libre(?b2)
  - Efe: sobre(?b1,?b2) ∧ libre(?b1) ∧ rob-libre ∧ ¬sujeto(?b1) ∧¬libre(?b2)
- desapilar(?b1,?b2)
  - Pre: sobre(?b1,?b2) ∧ libre(?b1) ∧ rob-libre
  - Efe: sujeto(?b1) ∧ libre(?b2) ∧ ¬sobre(?b1,?b2) ∧¬rob-libre ∧¬libre(?b1)

Búsqueda hacia adelante, usamos (Estrategia búsqueda DFS):

Paso 1: expandir I , dos acciones satisfacen las precondiciones: recoger(B) y desapilar(C,A). Se añaden nodo 1 y 2 a ABIERTA

ABIERTA: {I}

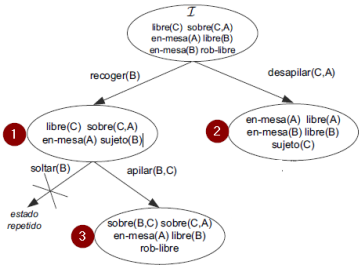
I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ ~~libre(B)~~ ∧ ~~en-mesa(B)~~ ∧ ~~rob-libre~~ ∧ sujeto (B)
2: ~~libre(C)~~ ∧ ~~sobre(C,A)~~ ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ ~~rob-libre~~ ∧ sujeto(C) ∧ libre(A)



Paso 2: expandir 1, dos acciones satisfacen las precondiciones: Soltar(B) y Apilar(B,C). Soltar(B) genera un nodo repetido 'I' por lo que no se ejecuta. Se añade nodo 3 a ABIERTA.

ABIERTA: {I,1,2}

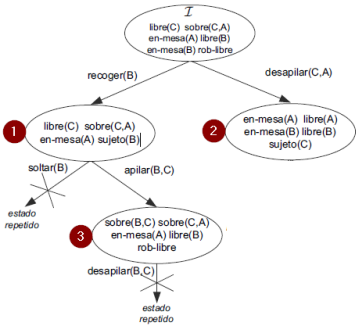
I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ sujeto (B)
2: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ sujeto(C) ∧ libre(A)
3: ~~libre(C)~~ ∧ sobre(C,A) ∧ en-mesa(A) ∧ ~~sujeto (B)~~ ∧ sobre(B,C) ∧ libre (B) ∧ rob-libre



Paso 3: expandir 3, solo se satisface Desapilar(B,C), genera un nodo repetido '2'por lo que no se ejecuta.

ABIERTA: {I,1,2,3}

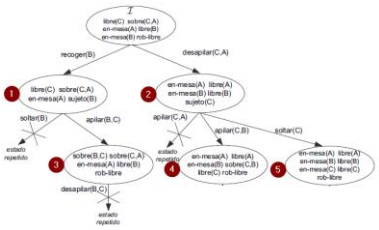
I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ sujeto (B)
2: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ sujeto(C) ∧ libre(A)
3: sobre(C,A) ∧ en-mesa(A) ∧ sobre(B,C) ∧ libre (B) ∧ rob-libre



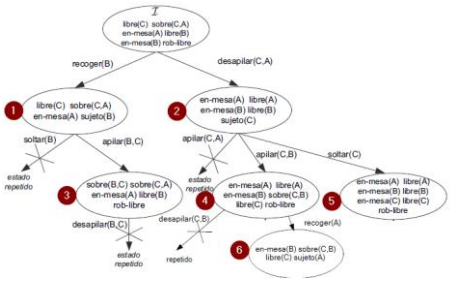
Paso 4: solo queda por expandir 2 en ABIERTA, se satisface soltar(C), apilar(C,A) genera un nodo repetido 'I' por lo que no se ejecuta. apilar(C,B) genera un nodo repetido '2'por lo que no se ejecuta.

ABIERTA: {I,1,2,3}

I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ sujeto (B)
2: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ sujeto(C) ∧ libre(A)
3: sobre(C,A) ∧ en-mesa(A) ∧ sobre(B,C) ∧ libre (B) ∧ rob-libre
4: en-mesa(A) ∧ ~~libre(B)~~ ∧ en-mesa(B) ∧ ~~sujeto(C)~~ ∧ libre(A) ∧sobre(C,B) ∧ libre (C) ∧ rob-libre
5: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ ~~sujeto(C)~~ ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ rob-libre



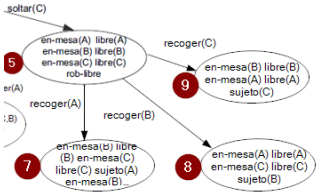
Paso 5: expandimos nodo 4 , se satisface recoger(A), desapilar(C,B) genera un nodo repetido '2'por lo que no se ejecuta.



ABIERTA: {I,1,2,3,4,5}

I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ sujeto (B)
2: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ sujeto(C) ∧ libre(A)
3: sobre(C,A) ∧ en-mesa(A) ∧ sobre(B,C) ∧ libre (B) ∧ rob-libre
4: en-mesa(A) ∧ en-mesa(B) ∧ libre(A) ∧sobre(C,B) ∧ libre (C) ∧ rob-libre
5: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ rob-libre
6: ~~en-mesa(A)~~ ∧ en-mesa(B) ∧ ~~libre(A)~~ ∧sobre(C,B) ∧ libre (C) ∧ ~~rob-libre~~ ∧ sujeto(A)

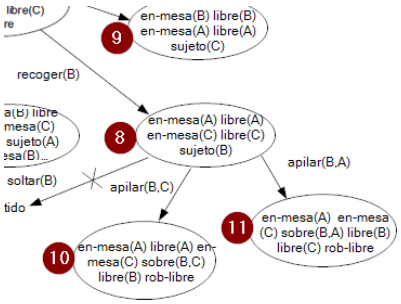
Paso 6: siguiendo la estrategia de búsqueda DFS deberíamos expandir el nodo 6 pero vamos a suponer que una posible Heurística nos indica que expandir el nodo 5 es más prometedor por lo que expandimos 5. Se satisface recoger(A), recoger(B) y recoger(C).



ABIERTA: {I,1,2,3,4,5,6}

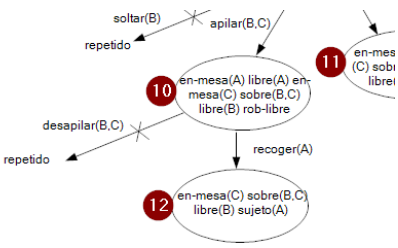
I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ sujeto (B)
2: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ sujeto(C) ∧ libre(A)
3: sobre(C,A) ∧ en-mesa(A) ∧ sobre(B,C) ∧ libre (B) ∧ rob-libre
4: en-mesa(A) ∧ en-mesa(B) ∧ libre(A) ∧sobre(C,B) ∧ libre (C) ∧ rob-libre
5: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ rob-libre
6: en-mesa(B) ∧sobre(C,B) ∧ libre (C) ∧ sujeto(A)
7: ~~en-mesa(A)~~ ∧ libre(B) ∧ en-mesa(B) ∧ ~~libre(A)~~ ∧ libre(C) ∧ en-mesa(C) ∧ ~~rob-libre~~ ∧ sujeto(A)
8: en-mesa(A) ∧ ~~libre(B)~~ ∧ ~~en-mesa(B)~~ ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ ~~rob-libre~~ ∧ sujeto(B)
9: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ ~~libre(C)~~ ∧ ~~en-mesa(C)~~ ∧ ~~rob-libre~~ ∧ sujeto(C)

Paso 7: una posible Heurística nos indica que expandir el nodo 8 es más prometedor. Se satisface soltar(B), apilar (B,C) y apilar(B,A). Soltar(B) no se ejecuta porque generaría un nodo repetido '5'.ABIERTA: {I,1,2,3,4,5,6,7,8,9}



I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ sujeto (B)
2: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ sujeto(C) ∧ libre(A)
3: sobre(C,A) ∧ en-mesa(A) ∧ sobre(B,C) ∧ libre (B) ∧ rob-libre
4: en-mesa(A) ∧ en-mesa(B) ∧ libre(A) ∧sobre(C,B) ∧ libre (C) ∧ rob-libre
5: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ rob-libre
6: en-mesa(B) ∧sobre(C,B) ∧ libre (C) ∧ sujeto(A)
7: libre(B) ∧ en-mesa(B) ∧ libre(C) ∧ en-mesa(C) ∧ sujeto(A)
8: en-mesa(A) ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ sujeto(B)
9: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ sujeto(C)
10: en-mesa(A) ∧ libre(A) ∧ ~~libre(C)~~ ∧ en-mesa(C) ∧ ~~sujeto(B)~~ ∧ sobre (B,C) ∧ libre(B) ∧ rob-libre
11: en-mesa(A) ∧ ~~libre(A)~~ ∧ libre(C) ∧ en-mesa(C) ∧ ~~sujeto(B)~~ ∧ sobre (B,A) ∧ libre(B) ∧ rob-libre

Paso 8: una posible Heurística nos indica que expandir el nodo 10 es más prometedor. Se satisface recoger(A), desapilar (B,C). desapilar (B,C) no se ejecuta porque generaría un nodo repetido '8'.



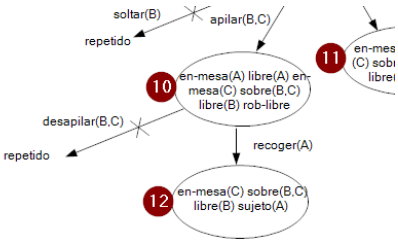
ABIERTA: {I,1,2,3,4,5,6,7,8,9,10,11}

I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ sujeto (B)
2: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ sujeto(C) ∧ libre(A)
3: sobre(C,A) ∧ en-mesa(A) ∧ sobre(B,C) ∧ libre (B) ∧ rob-libre
4: en-mesa(A) ∧ en-mesa(B) ∧ libre(A) ∧sobre(C,B) ∧ libre (C) ∧ rob-libre
5: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ rob-libre
6: en-mesa(B) ∧sobre(C,B) ∧ libre (C) ∧ sujeto(A)
7: libre(B) ∧ en-mesa(B) ∧ libre(C) ∧ en-mesa(C) ∧ sujeto(A)
8: en-mesa(A) ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ sujeto(B)
9: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ sujeto(C)
10: en-mesa(A) ∧ libre(A) ∧ en-mesa(C) ∧ sobre (B,C) ∧ libre(B) ∧ rob-libre

11: en-mesa(A) ∧ libre(C) ∧ en-mesa(C) ∧ sobre (B,A) ∧ libre(B) ∧ rob-libre
12: ~~en-mesa(A)~~ ∧ ~~libre(A)~~ ∧ en-mesa(C) ∧ sobre (B,C) ∧ libre(B) ∧ ~~rob-libre~~ ∧ sujeto(A)

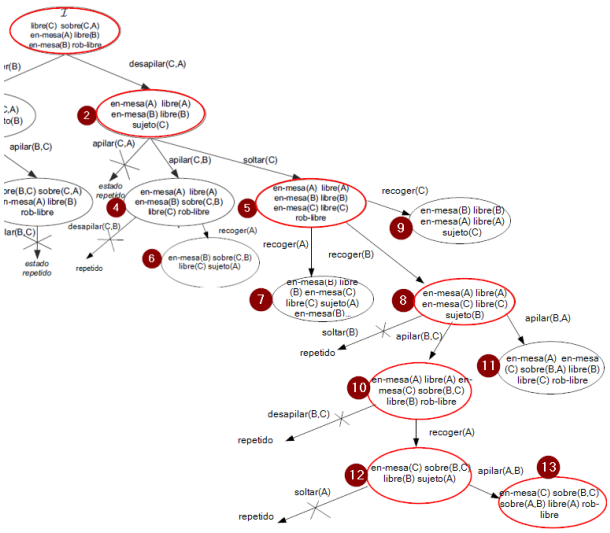
Paso 9: una posible Heurística nos indica que expandir el nodo 12 es más prometedor. Se satisface soltar(A), apilar (A,B). soltar(A) no se ejecuta porque generaría un nodo repetido '10'.

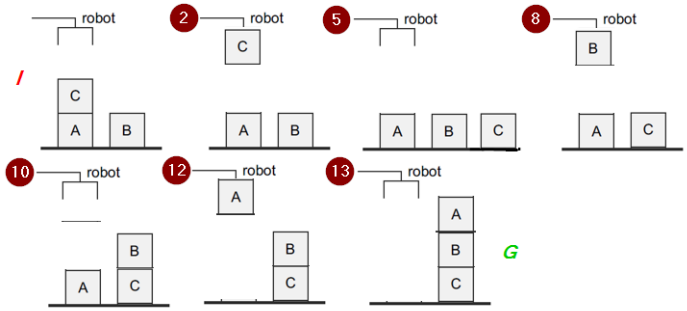
ABIERTA: {I,1,2,3,4,5,6,7,8,9,10,11}



I: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ rob-libre
1: libre(C) ∧ sobre(C,A) ∧ en-mesa(A) ∧ sujeto (B)
2: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ sujeto(C) ∧ libre(A)
3: sobre(C,A) ∧ en-mesa(A) ∧ sobre(B,C) ∧ libre (B) ∧ rob-libre
4: en-mesa(A) ∧ en-mesa(B) ∧ libre(A) ∧sobre(C,B) ∧ libre (C) ∧ rob-libre
5: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ rob-libre
6: en-mesa(B) ∧sobre(C,B) ∧ libre (C) ∧ sujeto(A)
7: libre(B) ∧ en-mesa(B) ∧ libre(C) ∧ en-mesa(C) ∧ sujeto(A)
8: en-mesa(A) ∧ libre(A) ∧ libre(C) ∧ en-mesa(C) ∧ sujeto(B)
9: en-mesa(A) ∧ libre(B) ∧ en-mesa(B) ∧ libre(A) ∧ sujeto(C)
10: en-mesa(A) ∧ libre(A) ∧ en-mesa(C) ∧ sobre (B,C) ∧ libre(B) ∧ rob-libre
11: en-mesa(A) ∧ libre(C) ∧ en-mesa(C) ∧ sobre (B,A) ∧ libre(B) ∧ rob-libre
12: en-mesa(C) ∧ sobre (B,C) ∧ libre(B) ∧ sujeto(A)
13: en-mesa(C) ∧ sobre (B,C) ∧ ~~libre(B)~~ ∧ ~~sujeto(A)~~ ∧ sobre (A,B) ∧ libre (A) ∧ rob-libre

El nodo 13 contiene a G: sobre(A,B) ∧ sobre(B,C) por lo que la búsqueda ha terminado y el camino es (I,2,5,8,10,12,13)





Búsqueda hacia atrás:

Los recursos utilizables (objetos) se denominan `A`, `B` y `C`. Para definir los estados de los recursos se utilizan las propiedades: `f(,)`, `h(,)`, aplicables a dos recursos *distintos*; `d()`, `u()`, aplicables a un recurso; y `p` aplicada al sistema en conjunto. Se usa el signo ¬ para indicar negación de una propiedad (o estado). Para abreviar la notación (ya que no hay disyunciones), indicamos la conjunción (∧) de predicados separándolos simplemente por un espacio.

El estado inicial es $\mathcal{I}$=`{h(A,C) f(B,A) d(B) d(C) u(A) p}` y el objetivo es $\mathcal{G}$=`{f(B,C) d(B) p}`.

Hay cuatro operaciones $\mathcal{O}$ sobre los recursos cuyos argumentos, precondiciones y efectos son los siguientes:

Operador `teq` entre un objeto `?x` y otro `?y`: `teq(?x,?y)`
- Pre: `f(?x,?y)`
- Efe: `h(?x,?y) u(?x) ¬f(?x,?y) ¬d(?y)`

Operador `lin` sobre un objeto `?x`: `lin(?x)`
- Pre: `u(?x)`
- Efe: `d(?x) ¬u(?x) ¬p`

Operador `mez` entre un objeto `?x` y otro `?y`: `mez(?x,?y)`
- Pre: `h(?x,?y) d(?x)`
- Efe: `f(?x,?y) ¬h(?x,?y)`

Operador `pas` entre un objeto `?x` y otro `?y`: `pas(?x,?y)`
- Pre: `u(?x) d(?y)`
- Efe: `h(?x,?y) ¬u(?x)`

Dominio Problema = <I, G, O> "se asume la hipótesis de mundo cerrado"

I: { h(A,C) f(B,A) d(B) d(C) u(A) p }

G: { f(B,C) d(B) p }

O: conjunto de operadores de planificación

- teq(?x ?y)
- lin(?x)
- mez(?x,?y)
- pas(?x,?y)

En la búsqueda hacia atrás se parte del objetivo y se aplican inversamente los operadores relevantes. Un operador es relevante si:

- Existe al menos un efecto + en G y no existe ningún efecto – en G

Paso 1: expandimos G

teq(?x ?y), no es relevante. No hay efe +
lin(?x), no es relevante por ¬p
mez(?x,?y), es relevante por f(B,C)
pas(?x,?y), no es relevante. No hay efe +

El operador mez(B,C) generará un nuevo estado G1 = G - Efe+ U (Pre) { **f(B,C)** d(B) p **-f(B,C)** } U { (h(B,C) d(B) } = { p h(B,C) d(B) } que se añade a ABIERTA.

G1 = { p h(B,C) d(B) }

ABIERTA {G, G1}

Paso 2: expandimos G1

teq(?x ?y), es relevante por h(B,C)
lin(?x), no es relevante por ¬p
mez(?x,?y), no es relevante. No hay efe +
pas(?x,?y), es relevante por h(B,C)

El operador teq(B,C) generará un nuevo estado G2 = G1 - Efe+ U (Pre) { p **h(B,C)** d(B) **-h(B,C) -u(B)**} U {f(B,C)} = {p d(B) f(B,C)} que es igual al nodo G por lo que está repetido y no se añade a ABIERTA.

El operador pas(B,C) generará un nuevo estado G3 = G1 - Efe+ U (Pre) { p **h(B,C)** d(B) **-h(B,C)** } U {u(B) d(C)} = {p d(B) u(B) d(C)} que se añade a ABIERTA.

G3 = {p d(B) u(B) d(C)}

ABIERTA {G, G1, G3}

Paso 3: expandimos G3

teq(?x ?y), para teq(B,A) es relevante por u(B),
para teq(B,C) no es relevante por d(C)
lin(?x), no es relevante por ¬p
mez(?x,?y), no es relevante. No hay efe +
pas(?x,?y), no es relevante. No hay efe +

El operador teq(B,A) generará un nuevo estado G4 = G3 - Efe+ U (Pre) {p d(B) u(B) d(C) **-u(B) -h(B,A)**} U {f(B,A)} = {p d(B) d(C) f(B,A)} que se añade a ABIERTA.

G4 = {p d(B) d(C) f(B,A)}

ABIERTA {G, G1, G3, G4}

G4 = {p d(B) d(C) f(B,A)} es un conjunto de subobjetivos de I: { h(A,C) f(B,A) d(B) d(C) u(A) p } por lo que el proceso termina.

El árbol de búsqueda es: teq(B,A) pas(B,C) mez(B,C)

TEMA 4: Sistemas multiagentes

El objetivo global del tema es describir cómo se pueden distribuir diversos sistemas inteligentes, o incluso tareas individuales, entre diferentes máquinas (físicas o virtuales) que se **comunican** para sacar beneficio de ese paralelismo de componentes

La ingeniería del software, aplicada al campo de la IA, se ha definido el concepto de **agente** (entidades que persiguen objetivos y tiene la capacidad de decidir por sí mismo qué tareas ejecutar o rechazar en función de los objetivos que quiera lograr) para referirse a los componentes distribuidos que forman estos sistemas. La visión de un sistema como un conjunto de agentes que colaboran entre sí para lograr unos objetivos comunes facilita la estructuración y gestión de la complejidad y la heterogeneidad. Se pueden ver los agentes como una extensión del paradigma de objetos distribuidos, donde las dos características distintivas fundamentales son autonomía y sociabilidad.

Los agentes son entidades sociales. Habitualmente los agentes requieren la **colaboración** y **coordinación** de otros agentes para llevar a cabo sus planes, así como, capacidades de **negociación**. Por ello, se habla de sistemas multiagentes, o SMA.

SMA: Conjunto de entidades individuales, capaces de actuar de forma independiente a favor de su usuario o propietario, o de si mismos, a la vez que son capaces de interactuar con el resto de agentes que componen el sistema pudiendo colaborar o competir con ellos.

El principal interés del paradigma de agente para desarrollo de software está en su capacidad para modelar sistemas con conceptos más cercanos a los que suelen utilizarse al tratar sistemas sociales y humanos.

SIRI o CORTANA son dos ejemplos de agente de interfaz.

Arquitecturas de agentes

Para construir agentes, primero hay que diseñarlos de acuerdo con ciertos principios arquitectónicos. Esto quiere decir que hay que definir una arquitectura, esto es, determinar y estructurar sus componentes y las relaciones entre ellos.

A continuación, se identifican propiedades que caracterizan a los agentes:

- **Autonomía**: Los agentes pueden trabajar sin la intervención directa del usuario y tienen cierto control sobre sus acciones y estado interno.
- **Reactividad**: Los agentes pueden percibir su entorno (que puede ser el mundo físico, un usuario detrás de una interfaz gráfica, aplicaciones en la red, u otros agentes) y responder oportunamente a cambios que se produzcan en el mismo.
- **Iniciativa**: El comportamiento de los agentes está determinado por los objetivos (metas) que persiguen y por tanto pueden producir acciones no sólo como respuesta al entorno.
- **Habilidad social**: Para satisfacer sus objetivos un agente puede requerir la colaboración de otros agentes en quienes delega o con quienes comparte la realización de tareas. Para ello necesita coordinarse, negociar, en definitiva, interactuar.
- **Razonamiento**: Un agente puede decidir qué objetivo perseguir o a qué evento reaccionar, cómo actuar para conseguir un objetivo, o suspender o abandonar un objetivo para dedicarse a otro.
- **Aprendizaje**: El agente puede adaptarse progresivamente a cambios en entornos dinámicos mediante técnicas de aprendizaje.
- **Movilidad**: En determinadas aplicaciones puede ser interesante permitir que los agentes puedan migrar de un nodo a otro en una red preservando su estado en los saltos entre nodos.

En una primera aproximación, todo agente es una entidad que percibe eventos de su entorno y que puede actuar sobre él. El agente tiene que decidir qué acciones tomar teniendo en cuenta el modelo del mundo que va construyéndose a partir de la información que consigue por los sensores.
Un agente puede fallar, bien en la acción que decide tomar o en el momento en que decide tomarla, por tanto, un agente debe observar su entorno una vez realizada una acción para considerar si ha tenido el efecto esperado o no, y en qué medida. Además, un agente debería utilizar estas observaciones para intentar mejorar sus decisiones en el futuro.

Agentes deliberativos (horizontales): Las arquitecturas deliberativas utilizan modelos de representación simbólica del conocimiento y suelen basarse en la teoría clásica de planificación. Los agentes tienen un conjunto de objetivos y, dependiendo del estado del mundo, generan planes para alcanzar los objetivos.

Cuando se decide implementar una arquitectura deliberativa hay que buscar una descripción simbólica adecuada del problema, e integrarla en el agente, para que éste pueda razonar y llevar a cabo las tareas encomendadas en el tiempo preestablecido.

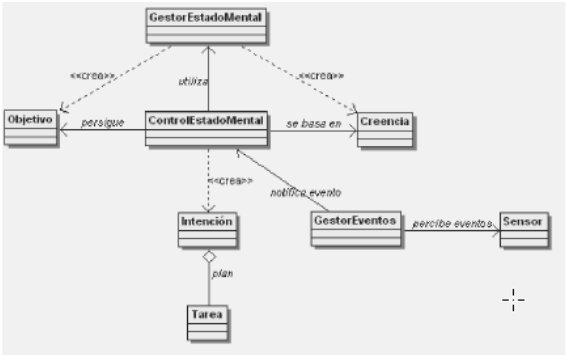
La arquitectura deliberativa más extendida es la denominada BDI, se caracteriza por el hecho de que los agentes están dotados cada uno de un estado mental (del inglés, Belief, Desire, Intention).

a) Las creencias representan el conocimiento que el agente tiene del entorno.

b) Los deseos son los objetivos que persigue el agente, su visión ideal del mundo. (El software convencional está orientado a la tarea en lugar de al objetivo, de forma que cada tarea u operación se ejecuta sin ningún recuerdo de por qué ha comenzado su ejecución).

c) Para alcanzar los objetivos propuestos, a partir de las creencias existentes es necesario definir un mecanismo de planificación que permita identificar las intenciones. Estos planes vinculados a la consecución de un objetivo constituyen las intenciones (conjunto de caminos de ejecución que pueden ser interrumpidos de una forma apropiada al recibir información acerca de cambios en el entorno) del agente.

La arquitectura del agente BDI debe contemplar entonces la gestión y el control del estado mental. El gestor de estado mental proporciona mecanismos para crear, modificar y destruir entidades del estado mental, manteniendo su consistencia.



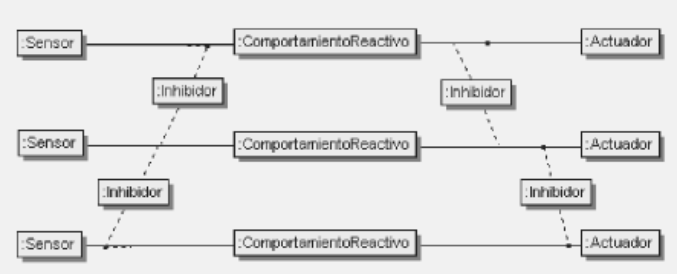
La recepción de un evento puede tener varios efectos:

- Un evento puede ser una evidencia de que un objetivo del agente se ha cumplido o que ha fallado. Esto modificará el estado del objetivo y puede ocasionar la necesidad de replanificar.
- El evento puede ser una información que el agente considere útil guardar como un hecho sobre el mundo, y por tanto puede incorporarse al modelo de creencias.
- También puede ocurrir que el evento no tenga significado para el agente en el contexto actual y decida simplemente ignorarlo.

Agentes reactivos (verticales): La utilización de una representación simbólica y mecanismos de toma de decisiones puede resultar costosa en tiempo y recursos. Si las acciones del agente se producen como respuesta directa a los eventos del entorno, sin requerir un proceso deliberativo complejo,

podemos eliminar esta complejidad. En estos casos, la inteligencia surgiría por la interacción de múltiples agentes, aunque cada uno tuviera un comportamiento muy sencillo. **hipótesis**: no es necesario que los agentes tengan una representación simbólica y que puedan razonar sobre ella para que el conjunto del SMA pueda ofrecer un comportamiento inteligente.

El ejemplo más clásico de estas arquitecturas es la arquitectura de **subsunción**; en una arquitectura de subsunción las tareas que definen un comportamiento están organizadas en jerarquías de capas, de menor a mayor nivel de abstracción.



Los comportamientos son sencillos, y se pueden implementar como autómatas simples o como reglas del tipo situación→ acción, que asocian acciones a la aparición de determinados eventos de entrada. Como en un momento dado podrían activarse varios comportamientos a la vez, para evitar acciones contradictorias, los comportamientos de los niveles inferiores pueden inhibir las entradas o salidas de los de niveles superiores.

Manejan jerarquías de tareas en función de niveles de abstracción.

Agentes híbridos: Para superar las limitaciones de las arquitecturas reactivas y deliberativas se han propuesto sistemas híbridos. Una primera propuesta puede ser construir un agente compuesto de dos subsistemas: uno deliberativo, que utilice un modelo simbólico y que genere planes en el sentido expuesto anteriormente, y otro reactivo, centrado en reaccionar ante los eventos que tengan lugar en el entorno, que no requiera un mecanismo de razonamiento complejo.

Estos subsistemas se pueden utilizar para definir una estructuración en dos ejes:

- Vertical: sólo una capa tiene acceso a los sensores y actuadores (capa más baja).
- Horizontal: todas las capas tienen acceso a los sensores y a los actuadores.

Estas capas se organizan jerárquicamente con información sobre el entorno a diferentes niveles de abstracción. La mayoría de las arquitecturas encuentran suficientes tres niveles:

1. Reactivo, o de más bajo nivel. En ésta se toman decisiones acerca de qué hacer con eventos recibidos del entorno en tiempo real. Suele estar implementado mediante arquitecturas de subsunción.
2. Conocimiento, o nivel intermedio. Trata el conocimiento que el agente posee del medio, normalmente con la ayuda de una representación simbólica del mismo.
3. Social : es la capa de más alto nivel. En ella se manejan los aspectos sociales del entorno, incluyendo tanto información de otros agentes, como los deseos, intenciones, etc.

El comportamiento global del agente queda definido por la interacción entre estos niveles.

En general, en un SMA suelen participar agentes de varios tipos. Algunos más simples de naturaleza reactiva que generalmente ofrecen servicios a otros más elaborados, de naturaleza híbrida o deliberativa. Depende de los objetivos del agente y la necesidad de razonar sobre su contexto que el diseñador decida aplicar un tipo de arquitectura u otro.

Sociedades de agentes

La sociabilidad es una característica fundamental de los agentes. Los agentes pueden colaborar o competir entre sí.

Organización: La organización juega un papel importante en la concepción de un SMA porque define el propósito general del sistema, su estructura, el contexto en el que los agentes desempeñan sus funciones específicas, las políticas de participación de los agentes, y las interacciones que permiten la colaboración entre los agentes.

Desde un punto de vista estructural, la organización es un conjunto de entidades asociadas por relaciones de agregación y herencia.

La estructura organizativa se define como una jerarquía de grupos. Un grupo se define identificando los roles y recursos que lo conforman. Un rol define una funcionalidad (objetivos) o las responsabilidades que tendrá que asumir el agente en un momento dado. Los recursos son utilizados por los agentes para realizar sus tareas. Las tareas en la organización están relacionadas en flujos de trabajo donde se determinan los responsables de la ejecución de las tareas, los recursos que se les asignan y los resultados que se esperan de cada una.

En la organización también se definen normas sociales que especifican cómo se establecen y desarrollan las relaciones sociales entre los agentes.

Coordinación: La coordinación (gestión de dependencias entre actividades) entre los agentes es necesaria para que puedan cooperar o colaborar en la consecución de los objetivos de una organización. Las dependencias se establecen entre tareas y no entre agentes. Esto tiene la ventaja de que permite modelar más fácilmente los efectos de reasignar tareas a varios agentes.

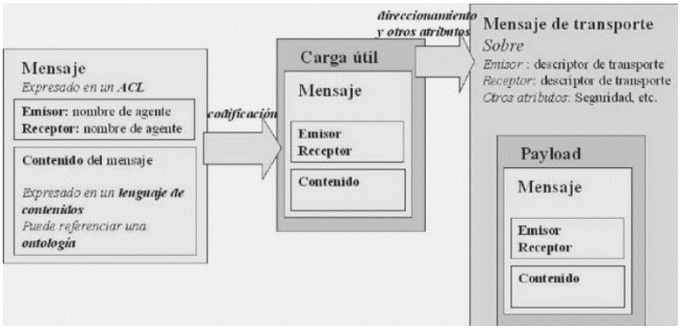
Dependencia	**Ejemplos de mecanismos de coordinación**
Recursos compartidos	FIFO, prioridades, autoridad central, puja
Asignación de tareas	(Similar a recursos compartidos)
Relaciones productor/consumidor	*
Prerrequisitos	Notificaciones, seguimiento
Transferencia	Gestión de inventarios
Usabilidad	Estandarización, participación de usuarios
Restricciones de simultaneidad	Planificación, sincronización
Tarea/Subtarea	Selección de objetivos, descomposición de tareas

Negociación, confianza y reputación: La autonomía de los agentes implica que éstos no tienen por qué aceptar toda petición que se les realice, especialmente cuando los agentes son autointeresados.

Un aspecto bastante relacionado con este ámbito es la confianza y la reputación de los agentes, esto requiere la existencia de métricas de confianza, que midan el grado de confianza de un miembro de una comunidad virtual. Una de las propiedades que deben cumplir estas métricas es la resistencia a ataques por agentes maliciosos que quisieran abusar de la presunción de confianza.

Comunicación entre agentes

La consideración de los agentes como entidades intencionales aporta una nueva dimensión a la comunicación entre agentes si se compara con el paradigma de objetos. Cuando un objeto (cliente) invoca un método en otro, el objeto que recibe solicitud no tiene control sobre la decisión de ejecutar dicho método: esto corresponde exclusivamente al cliente. Sin embargo, entre agentes, debido a su autonomía, dicha decisión tiene que ser acordada por ambos. Las intenciones de los dos agentes tienen que confluir para llevar a cabo el propósito de la comunicación.



Cada acto del habla tiene tres componentes: locución(modo de producción de frases), ilocución (acto realizado por el locutor sobre el destinatario) y perlocución (los efectos que puede tener el acto ilocutorio).

El lenguaje FIPA-ACL (Foundation for Intelligent Physical Agents – Agent Communication Languages) se estableció en 1996 como un consorcio cuyo objetivo era establecer estándares para la interoperabilidad de SMA.

```
( inform
    :sender     AgentePujador
    :receiver   AgenteSubastador
    :content    (precio libro  1000)
    :in-reply-to  ronda-4
    :reply-with puja04
    :language   sl
    :ontology   ontolibro
)
```

Los mensajes entre agentes son tuplas (clave, valor) escritos en ACL. El contenido está expresado con un lenguaje de contenidos que normalmente hará referencia a una ontología (especificación formal de una conceptualización compartida). Además, se deben incluir los nombres del emisor y el receptor. Un mensaje puede contener recursivamente otros mensajes.

Además del lenguaje utilizado para los mensajes, FIPA define protocolos de comunicación entre agentes. Esto es bastante útil porque los protocolos se pueden ofrecer en las plataformas de agentes como componentes reutilizables estándares, facilitando así la construcción de las aplicaciones basadas en agentes.

FIPA define un conjunto amplio de performativas, que se pueden agrupar en cuatro clases principales:

- Información: Para indicar que una proposición (el contenido) es verdadera, o preguntar si lo es, que responderá positivamente o no, o bien que no se ha entendido.
- Realización de Acciones: Un agente solicita a otro la realización de una acción, y el receptor puede indicar que la va a realizar, o que rehúsa.
- Negociación: Un agente puede iniciar una petición de propuestas, y cada propuesta recibida, puede ser aceptada o rechazada.
- Intermediación: sirve para permitir a un agente intermediario recibir solicitudes para reenviar mensajes.

Entre las plataformas que siguen el estándar de FIPA las dos más conocidas son FIPA Open Source (FIPA-OS) y Java Agent DEvelopment (JADE).

Además del lenguaje utilizado para los mensajes, FIPA define protocolos de comunicación entre agentes:

- Request

- Request when
- Query
- Contract net
- English auction
- Dutch auction
- Brokering
- Recruiting
- Propose
- Subscribe

JADE proporciona en primer lugar un conjunto de librerías para crear agentes que puedan comunicarse entre sí utilizando el lenguaje FIPA ACL y acceder a los servicios estándar de gestión de agentes FIPA (directorios y ciclo de vida de los agentes). Además, JADE proporciona varios agentes ya implementados:

- **Directory Facilitator (DF):** Proporciona los servicios de directorio.
- **Agent Management System (AMS):** uno por cada instancia de plataforma Jade en ejecución, ejerce de supervisor de acceso y uso de la plataforma.
- **Agente Sniffer:** ayuda en la depuración de las interacciones entre agentes.
- **El agente Introspector:** permite controlar el ciclo de vida de un agente en ejecución y los mensajes que intercambia.

Métodos, herramientas y plataformas

La construcción de SMA integra tecnologías de distintas áreas de conocimiento: técnicas de Ingeniería del Software para estructurar el proceso de desarrollo, técnicas de IA para dotar a los programas de capacidad de adaptación y toma de decisiones y técnicas de Programación Concurrente y Distribuida para tratar la coordinación de tareas ejecutadas en diferentes máquinas.

Casi todas las metodologías de desarrollo de SMA extienden el planteamiento de la orientación a objetos con elementos que caracterizan los SMA: aspectos sociales (organizaciones, interacciones, negociación, colaboración), de comportamiento (autonomía, estado mental, objetivos, tareas), de concurrencia y de distribución.

Aunque cada metodología define criterios propios, suele haber cierto consenso en considerar al menos los siguientes elementos:

- Agente: ofrece la visión micro del SMA, esto es, los aspectos relacionados con los agentes individuales. Puede tratarse de la definición de la arquitectura de los agentes, de identificar las entidades que configuran su estado mental, etc.
- Organización: permite describir la visión macro del SMA.
- Entorno: donde se sitúa y con el que interacciona el SMA. Las interacciones con otras aplicaciones se considerarían parte del entorno.
- Interacciones: entre los agentes o de los agentes con los usuarios (generalmente humanos).

Aplicaciones de los agentes

El concepto de agente permite modelar, de manera natural, una gama amplia de sistemas con requisitos diversos.

El tipo de sistemas más simples de realizar son los funcionales, que producen una salida a partir de una entrada aplicando una función o algoritmo más o menos complejo. Una característica de este tipo de sistemas es que no se asume que puedan producirse cambios en el entorno durante la ejecución de una función.

Los sistemas reactivos, por el contrario, mantienen una interacción continua con el entorno, el cual puede cambiar haciendo que las precondiciones válidas al principio de la ejecución de un proceso no lo sean durante la ejecución de este e incluso que el objetivo por el cual había comenzado su ejecución deje de tener sentido. En estas situaciones, las estructuras de control tradicional no son siempre válidas. Cuando se realiza una acción en el entorno no es posible determinar previamente el efecto que tendrá, sólo la intención de lo que se pretende. Todos los factores anteriores contribuyen a un indeterminismo de la dinámica del sistema, para lo cual los agentes tienen capacidades de adaptación y planificación. Los agentes pueden detectar si una tarea no se está llevando a cabo como estaba previsto, y pueden decidir cambiarla por otra tarea o colaborar con otros agentes.

Respecto a la mejora de la eficacia que puede suponer la adopción de un SMA, se proponen tres aspectos que serían adecuados:

- Distribución inherente de datos, control, experiencia y recursos (e.g. hospital, medico, enfermera).
- La noción de agente autónomo permite modelar de manera natural una funcionalidad específica (e.g. spam).
- Un sistema que tenga que coexistir con componentes heredados que deben interactuar entre sí y con nuevos componentes software.

Otra característica del entorno habitual en los SMA es la heterogeneidad.

Por último, hay que mencionar los aspectos, cada día más considerados, de usabilidad del software. Un ejemplo de este tipo de cuestiones es la personalización. La personalización permite que un mismo servicio se pueda ofrecer simultáneamente de manera distinta según las características de cada usuario.

Se utilizan principalmente:

- Para resolver problemas que son demasiado grandes para un agente centralizado.
- Para permitir la interconexión y la interoperabilidad de múltiples sistemas legados.
- Para ofrecer una solución a problemas inherentemente distribuidos.
- Para dar soluciones provenientes de fuentes de información distribuidas.
- Para proveer soluciones donde el conocimiento experto está distribuido.
- Para ofrecer claridad conceptual y simplicidad de diseño.
- Aumentar la modularidad (reduce complejidad), velocidad (paralelismo), fiabilidad, flexibilidad y la reutilización a nivel de conocimiento

En resumen, para decidir la conveniencia de realizar una aplicación como un SMA habrá que preguntarse si tiene características que hagan difícil su concepción con paradigmas tradicionales.

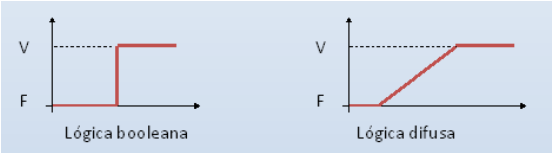
- ¿Se trata de un sistema distribuido abierto? ¿Pueden incorporarse dinámicamente nuevos tipos de entidades en el sistema? ¿Pueden cambiar las existentes?
- ¿Es necesario considerar una evolución del comportamiento independiente para cada uno de los componentes del sistema o para una parte significativa?

- ¿Hay incertidumbre? ¿Es posible para una entidad del sistema conocer su contexto suficientemente para poder decidir con certeza el efecto de las acciones que puede realizar?
- ¿Hace falta definir una organización de entidades que interactúan para resolver conjuntamente problemas globales? Las características sociales de los agentes permiten modelar muchos aspectos organizativos.
- ¿Hay interdependencias entre los elementos del sistema de las que pueden surgir comportamientos complejos, usualmente no predecibles?

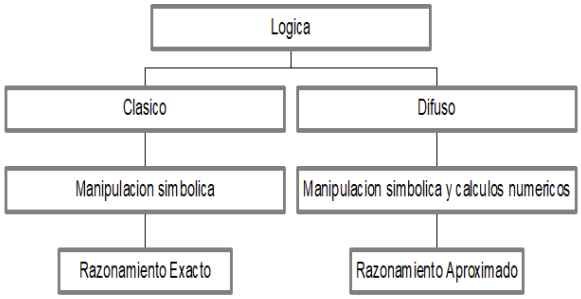
TEMA 5: Conjuntos borrosos

Históricamente, las ciencias formales han restringido el uso del lenguaje natural a aquellos predicados que, aplicados a cierta colección de objetos U, dividen ésta en dos subconjuntos. Así, si U es el conjunto de los números naturales U = {1, 2, 3, 4, ...}, el predicado 'impar' lo divide en dos subconjuntos (predicado preciso o clásico).

Tradicionalmente, han quedado excluidos de toda consideración formal otro tipo de predicados, que como 'alto', no permiten realizar una división satisfactoria de una población de individuos en dos subconjuntos, aquéllos para los cuales 'alto' es cierto, y aquéllos para los que es falso (predicado vago).



La publicación del trabajo de Lotfi A. Zadeh "Fuzzy Sets" (Conjuntos Borrosos) en 1965 supone un salto cualitativo en la relación entre la ciencia y el lenguaje. Si tradicionalmente el pensamiento científico occidental ha mantenido una relación difícil con el lenguaje ordinario, "Fuzzy Sets" plantea la necesidad de utilizar herramientas formales para su análisis, y así surge la Teoría de los Conjuntos Borrosos, como la teoría que ha de proporcionar dichas herramientas, y la **Lógica Borrosa** (permite representar conocimiento de tipo impreciso y subjetivo), como la ciencia de los principios formales del razonamiento aproximado.



Conjuntos borrosos

Todo predicado preciso *P*, aplicado a una colección *U*, tiene asociado un conjunto preciso que denotaremos de la misma forma $P \subset U$, y que puede describirse mediante una función $\mu P(u)$, cuyo valor viene determinado por la pertenencia a *P* de los distintos elementos $u \in U$.

$$\mu_P(u) = \begin{cases} 0 & \text{si } u \notin P \\ 1 & \text{si } u \in P \end{cases}$$

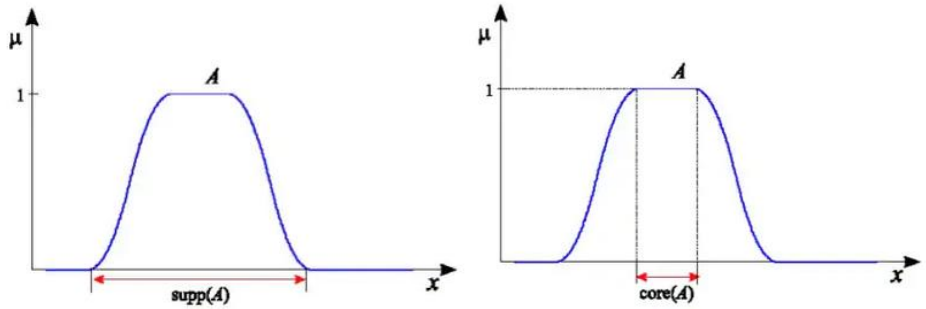
En cambio, un predicado vago *V* , aplicado a la misma colección *U*, tiene asociado un conjunto borroso que denotaremos de la misma forma $V \subset U$, y que puede describirse mediante una función $\mu V(u)$, cuya representación de la pertenencia a *V* de los distintos elementos de *U* es una cuestión de grado, de modo que:

$$\mu_V : U \rightarrow [0, 1]$$

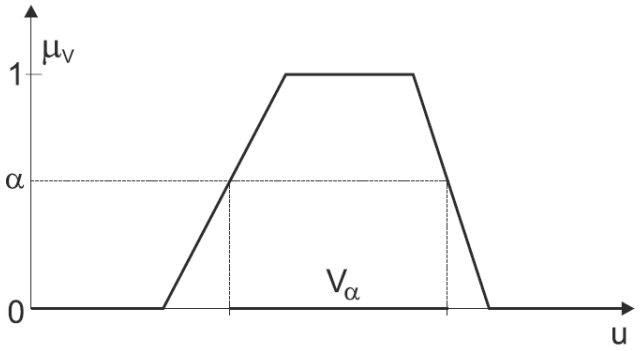
Además de aquellos elementos para los que el predicado *V* es cierto *($\mu V(u) = 1$)*, y aquéllos para los que *V* es falso *($\mu V(u) = 0$),* existe un conjunto de elementos para los que el predicado *V* es cierto en un determinado grado *$0 < \mu V(u) < 1$*.

Así definido, el concepto de conjunto borroso extiende la noción clásica de conjunto y a **función de pertenencia** encuentra su sentido en su capacidad para ordenar los elementos de U según su grado de pertenencia, **permitiendo representar conocimiento impreciso y subjetivo**.

Resulta útil definir los siguientes conjuntos asociados a una función de pertenencia μV : **soporte**, definido como *$\{u \in U : \mu V(u) > 0\}$*, y su **núcleo**, definido como *$\{u \in U : \mu V(u) = 1\}$*. Aquellos elementos que pertenecen al núcleo de V se consideran **prototipos** de *V* . El tamaño del soporte siempre es mayor o igual al tamaño de su núcleo.

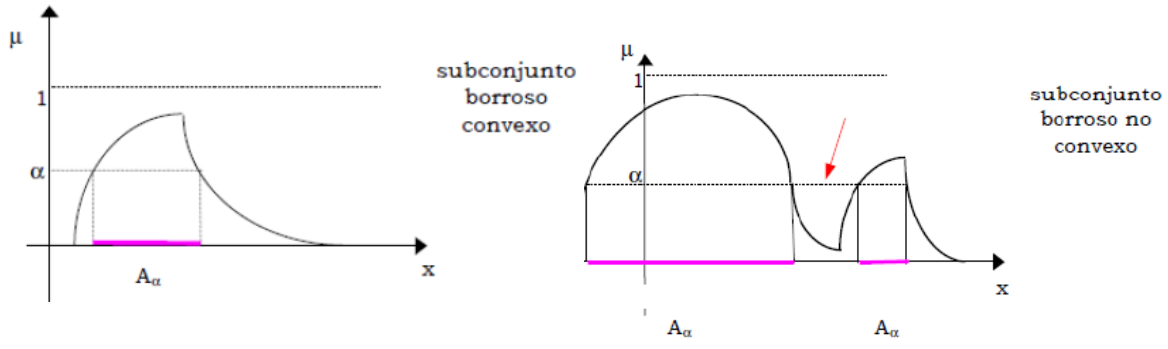


El ***α- corte V_α*** del conjunto borroso *V* se define como el conjunto *$\{u \in U : \mu V(u) \geq \alpha\}$,* para cualquier *$0 < \alpha \leq 1$.* El 1-corte de V coincide con su núcleo. Vemos cómo los miembros de un determinado α-corte son aquellos elementos cuyo grado de pertenencia es mayor o igual que α.

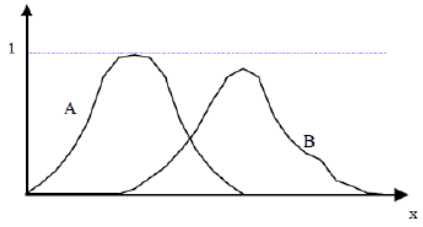


El conjunto V es convexo salvo si existe alguna tripleta de valores de x que no cumpla:

$$\forall u, u', u'' \in U, u' \in [u, u''], \ \mu_V(u') \geq \min\{\mu_V(u), \mu_V(u'')\}$$



Un conjunto V se dice normalizado si: $\exists u \in U$, tal que $\mu_V(u) = 1$



A es normal y B no.

Dados dos conjuntos borrosos *A,B* $\subset U$, decimos que "A está contenido en B", y escribimos $A \subseteq B$, si y sólo si, $\forall u \in U$: ***μA(u) ≤ μB(u)***

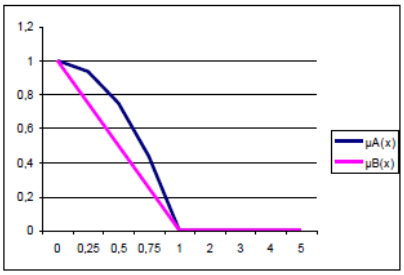
7. Si la *función de pertenencia* a un conjunto borroso A es: $\mu_A(x) = \{1 - x^2$ si $|x| < 1; 0$ en caso contrario$\}$, y la del conjunto borroso B es: $\mu_B(x) = \{1 - |x|$ si $|x| < 1; 0$ en caso contrario$\}$. Entonces **se cumple**:

(a) $B \subseteq A$.

(b) $A \subseteq B$.

(c) $A = B$.

(d) $A \nsubseteq B$ y $B \nsubseteq A$.



Para que "A está contenido en B", y escribimos **A ⊆ B**, si y sólo si, ∀x ∈ U:

μA(x) ≤ μB(x)

Calculamos y comparamos:

x	μA(x)	μB(x)
0	1	1
0,25	0,9375	0,75
0,5	0,75	0,5
0,75	0,4375	0,25
1	0	0
2	0	0
3	0	0

Para todos los valores de x se cumple μB(x) ≤ μA(x), por lo que → **B ⊆ A**

Semántica de los conjuntos borrosos

La idea de conjunto es por definición abstracta: una reunión de elementos que comparten alguna propiedad. Esta propiedad es la que determina la pertenencia de un elemento particular al conjunto. Si el conjunto es borroso, la pertenencia es una cuestión de grado. Sin embargo, el significado también depende del uso que se hace de dicho predicado. Tipos de predicados borrosos habituales:

- Similitud (tareas de clasificación, regresión o agrupamiento): Sea *V* un conjunto borroso. Si pensamos en los elementos del núcleo de *V* como en prototipos de *V*, entonces $\mu V(u)$ es el grado de proximidad de *u* al tipo de elementos prototipo de *V*.
- Preferencia (problemas de decisión): *V* reúne a un conjunto de elementos entre los que existe alguna preferencia y la función representa el grado de preferencia entre dichos elementos.
- Incertidumbre: Dada una variable *x* que toma variables en *U* un predicado (e.g. *x* toma un valor pequeño) se modela mediante un conjunto borroso $V \subset U$, con una función de pertenencia $uv=_{pequeño}$. Esta semántica está ligada a la presencia de incertidumbre en tareas de razonamiento.

Complementario: Así, dado un conjunto A ⊂ U, decimos que "u es no A", si u ∈ A, que se define como:

$$\overline{A} = \{u \in U : \ u \notin A\}$$

μAc(x) = 1 - μA(x)

Intersección: Dados dos conjuntos precisos A ⊂ U y B ⊂ U, decimos que "u es A y B" si u ∈ A∩B, que se define como:

$$A \cap B = \{u \in U : \ u \in A \ y \ u \in B\}$$

μA∩B(x) = Min(μA(x),μB(x))

Unión: Dados dos conjuntos A ⊂ U y B ⊂ U, decimos que "u es A o B" si u ∈ A ∪ B, que se define como:

$$A \cup B = \{u \in U : \ u \in A \ o \ u \in B\}$$

μAUB(x) = Max(μA(x),μB(x))

Es un resultado bien conocido en matemáticas que el conjunto *P(U)* de partes de *U*, esto es, el conjunto formado por todos aquellos subconjuntos precisos de un conjunto de referencia *U*, forma una estructura (P(U), ∪, ∩,−) denominada álgebra de Boole, dotada de un conjunto de propiedades (véase la Tabla) de entre las que destacamos la ley del tercero excluido A ∪ (¬A) = U, que nos dice que un predicado sólo puede ser cierto o falso, y el principio de no contradicción A ∩ (¬A) = ∅, que nos dice que un predicado no puede ser al mismo tiempo cierto y falso.

Idempotencia:	$A \cup A = A$	$A \cap A = A$
Conmutativa:	$A \cup B = B \cup A$	$A \cap B = B \cap A$
Asociativa:	$A \cup (B \cup C) = (A \cup B) \cup C$	$A \cap (B \cap C) = (A \cap B) \cap C$
Absorción:	$A \cup (A \cap B) = A$	$A \cap (A \cup B) = A$
Distributiva:	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Identidad:	$A \cup \emptyset = A$	$A \cap U = A$
Complemento:	$A \cup \overline{A} = U$	$A \cap \overline{A} = \emptyset$
De Morgan:	$(\overline{A \cup B}) = \overline{A} \cap \overline{B}$	$(\overline{A \cap B}) = \overline{A} \cup \overline{B}$
Involución:	$\overline{\overline{A}} = A$	

También podemos representar la composición de predicados precisos utilizando funciones de pertenencia, haciendo uso de las siguientes definiciones:

Complemento	$\mu_{\overline{A}}(u) = 1 - \mu_A(u)$
Intersección	$\mu_{A \cap B}(u) = \text{mín}\{\mu_A(u), \mu_B(u)\}$
Unión	$\mu_{A \cup B}(u) = \text{máx}\{\mu_A(u), \mu_B(u)\}$

Una forma de representar la composición en subconjuntos borrosos de U es mediante el uso de funciones: T (T : $[0, 1]^2 \rightarrow [0, 1]$) y S (S : $[0, 1]^2 \rightarrow [0, 1]$) podríamos pedir las siguientes propiedades, donde A y B son subconjuntos borrosos de U

1. Ley de identidad: ∀x ∈ [0, 1]

$$T(x,1) = x \quad (A \cap U = A)$$

$$S(x,0) = x \quad (A \cup \emptyset = A)$$

2. Ley conmutativa: ∀x, y ∈ [0, 1]

$$T(x,y) = T(y,x) \quad (A \cap B = B \cap A)$$

$$S(x,y) = S(y,x) \quad (A \cup B = B \cup A)$$

3. Ley asociativa: ∀x, y, z ∈ [0, 1]

$$T(x,T(y,z)) = T(T(x,y),z) \quad (A \cap (B \cap C) = (A \cap B) \cap C)$$

$$S(x,S(y,z)) = S(S(x,y),z) \quad (A \cup (B \cup C) = (A \cup B) \cup C)$$

4. Ley de monotonía: ∀x, y, u, v ∈ [0, 1], x ≤ u, y ≤ v

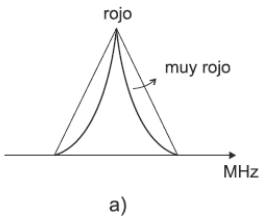
$$T(x,y) \leq T(u,v)$$

$$S(x,y) \leq S(u,v)$$

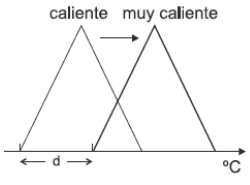
Aquellas funciones T que satisfacen las anteriores propiedades reciben el nombre de normas triangulares o **t-normas** y aquellas funciones S que también las satisfacen reciben el nombre de conormas triangulares o **t-conormas**

Variable lingüística

Una variable lingüística *V* toma valores en un conjunto *L = {L1, L2, ...,Ln}* de términos lingüísticos que realizan una descripción cualitativa de un conjunto de referencia *U*. (e.g. *U* es el conjunto de temperaturas, podríamos *LT = {frío, templado, caliente}*). Cada uno de estos términos $L_i \in L$ puede representarse mediante un conjunto borroso de *U*, donde normalmente *U* se corresponde con una escala de valores numéricos. Podemos añadir a estos términos lingüísticos el uso de modificadores lingüísticos, como son 'muy', 'bastante', 'más o menos', etc., que nos permiten enriquecer el lenguaje. Podemos modelar la acción de un modificador lingüístico M_i sobre la etiqueta L_i mediante dos funciones: una **función de modificación** (a) : $mM_i[0, 1] \rightarrow [0, 1]$, y una **función de desplazamiento** (b): $dM_i\, U \rightarrow U$, de modo que:



a)



b)

$$\mu_{M_i L_i} = m_{M_i} \circ \mu_{L_i} \circ d_{M_i}$$

donde ◦ representa la composición de funciones.

Principio de extensión

Sea una función $\varphi : U_1 \times ... \times U_n \rightarrow W$, que tiene como argumentos *n* valores precisos $(u_1, ..., u_n) \in U_1 \times ... \times U_n$, y tal que $w = \varphi(u1, ..., u_n)$. Este principio nos permite extender la función φ a un conjunto de argumentos borrosos $A_1 \subset U_1, ..., A_n \subset U_n$, e inferir un nuevo conjunto borroso B ⊂ W como B = $\Phi(A_1, ..., A_n)$, mediante la siguiente expresión:

$$\mu_B(w) = \sup_{w=\phi(u_1,u_2,\ldots,u_n)} \text{mín}\{\mu_{A_1}(u_1), \mu_{A_2}(u_2), \ldots, \mu_{A_n}(u_n)\} = 0 \quad si\ \phi^{-1}(w) = \emptyset.$$

siendo μA_i la función de pertenencia de A_i *(i = 1, ..., n).* De este modo, el principio de extensión nos permite hacer borroso cualquier dominio del razonamiento matemático basado en la teoría de conjuntos. Además, toda extensión borrosa ha de satisfacer siempre un axioma básico: ha de preservar el resultado clásico cuando opera sobre conjuntos precisos.

Aritmética borrosa

Podemos ilustrar el principio de extensión en el dominio de la aritmética. Definamos un número borroso como un conjunto borroso convexo y normalizado en el conjunto de los números reales R. Decimos que un número borroso A es positivo si μA(x) = 0, ∀x < 0. Del mismo modo, un número borroso A es negativo si μA(x) = 0, ∀x > 0.

Los números borrosos se pueden representar mediante:

- Intervalos de confianza [a1,a2]
- α-corte $A^{\alpha} = [a_1^{\alpha}, a_2^{\alpha}]$, ∀α ∈ [0,1]
- Función de pertenencia μA(x) = {1 si -1 <= x <= 1,5 ; 0 en caso contrario}

A continuación, se muestra las operaciones (+ - * y /) mediante funciones de pertenencia e intervalos de confianza.

- Sean A,B ⊂ R dos de estos números borrosos. El principio de extensión nos dice que podemos definir su suma borrosa, A ⊕ B, mediante la siguiente expresión:

$$\mu_{A\oplus B}(z) = \sup_{z=x+y} \text{mín}\{\mu_A(x), \mu_B(y)\}, \quad \forall x, y, z \in \mathbb{R}$$

Podemos descomponer la suma borrosa en una suma de $A_\alpha + B_\alpha = [a_\alpha^1 + b_\alpha^1, a_\alpha^2 + b_\alpha^2]$ intervalos

- Resta borrosa:

$$\mu_{A\ominus B}(z) = \sup_{z\ =\ x\ -\ y,} \text{mín}\{\mu_A(x), \mu_B(y)\}, \quad \forall x, y, z \in \mathbb{R}$$

Podemos descomponer la resta borrosa en una suma de $(A \ominus B)_\alpha = [a_\alpha^1 - b_\alpha^2, a_\alpha^2 - b_\alpha^1]$, intervalos:

- Multiplicación borrosa:

$$\mu_{A\otimes B}(z) = \sup_{z=x\times y} \text{mín}\{\mu_A(x), \mu_B(y)\}, \quad \forall x, y, z \in \mathbb{R}$$

La representación de esta operación mediante aritmética de intervalos adopta la forma:

$$(A \otimes B)_\alpha = [\text{mín}(a_\alpha^1 \times b_\alpha^1, a_\alpha^1 \times b_\alpha^2, a_\alpha^2 \times b_\alpha^1, a_\alpha^2 \times b_\alpha^2), \text{máx}(a_\alpha^1 \times b_\alpha^1, a_\alpha^1 \times b_\alpha^2, a_\alpha^2 \times b_\alpha^1, a_\alpha^2 \times b_\alpha^2)]$$

- División borrosa:

$$\mu_{A\oslash B}(z) = \sup_{z=x/y} \min\{\mu_A(x), \mu_B(y)\}, \quad \forall x, y, z \in \mathbb{R}^+$$

La representación de esta operación mediante aritmética de intervalos adopta la forma:

$$(A \oslash B)_\alpha = [a_\alpha^1 / b_\alpha^2, a_\alpha^2 / b_\alpha^1]$$

Ejemplos de operaciones mediante aritmética de intervalos.

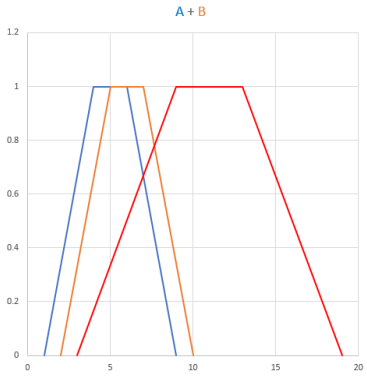
Suma de dos números borrosos: [a1,a2]+[b1,b2]=[a1+b1,a2+b2]

A(1,4,6,9) y B(2,5,7,10)

Empezamos sumando los soportes (1,9) y (2,10) = (3,19)

Ahora los núcleos (4,6) y (5,7)=(9,13)

Por lo tanto, A+B=(3,9,13,19)



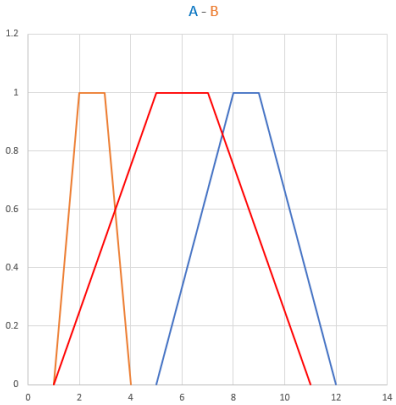
Resta de dos números borrosos: [a1,a2]+[b1,b2]=[a1-b2,a2-b1]

A(5,8,9,12) y B(1,2,3,4)

Empezamos restando los soportes (5,12) y (1,4) = (1,11)

Ahora los núcleos (8,9) y (2,3)=(5,7)

Por lo tanto, A-B=(1,5,7,11)



Multiplicación de dos números borrosos:

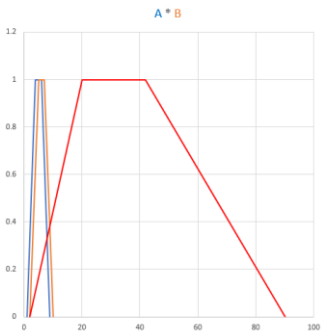
[a1,a2]*[b1,b2]=[min (a1*b1,a1*b2,a2*b1,a2*b2), max(a1*b1,a1*b2,a2*b1,a2*b2)]

A(1,4,6,9) y B(2,5,7,10)

Empezamos multiplicando los soportes (1,9) y (2,10) = min(2,10,18,90), max(2,10,18,90) = (2,90)

Ahora los núcleos (4,6) y (5,7)= min(20,28,30,42), max(20,28,30,42) = (20,42)

Por lo tanto, A*B=(2,20,42,90)



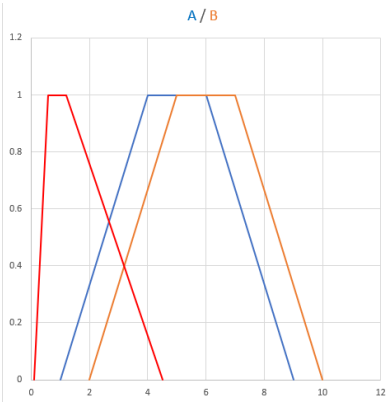
División de dos números borrosos: [a1,a2]/[b1,b2]=[a1/b2,a2/b1]

A(1,4,6,9) y B(2,5,7,10)

Igual, dividimos los soportes (1,9) y (2,10) = (0.1,4,5)

Ahora los núcleos (4/7,6/5)=(0.57,1.2)

Por lo tanto, A/B (0.1,0.57,1.2,4.5)



Relaciones borrosas

Las relaciones borrosas permiten modelar sentencias del tipo de "es mucho más alto que" o "es muy parecido a". Definimos una relación borrosa como un subconjunto borroso del producto cartesiano entre conjuntos.

Relaciones de similitud o equivalencia

Definidas mediante una función de pertenencia $\mu S : U \times U \longrightarrow [0, 1]$, donde U se corresponde habitualmente con una escala numérica. Todas las relaciones de similitud deben satisfacer tres propiedades:

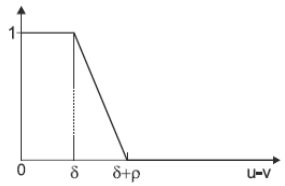
$$\textit{Reflexiva: } \forall u \in U,\ \mu_S(u,u) = 1.$$

$$\textit{Simétrica: } \forall u, v \in U,\ \mu_S(u,v) = \mu_S(v,u).$$

$$t\textit{-Transitiva: } \forall u, v, w \in U,\ T(\mu_S(u,v), \mu_S(v,w)) \leq \mu_S(u,w).$$

Donde *T* denota una *t-norma* cualquiera, haciendo depender la propiedad de transitividad de la *t-norma* escogida.

Una relación de similitud resulta una herramienta muy útil en la ingeniería. Pensemos que tareas como el reconocimiento de patrones, la clasificación, el agrupamiento o la recuperación de información, entre otras, utilizan alguna medida de similitud con respecto a un conjunto de prototipos, y su modelado borroso permite realizar una ordenación de los objetos del problema según un criterio de preferencia que viene determinado por la forma de la relación borrosa de similitud.



Relaciones de comparación

De un modo análogo podemos modelar relaciones de comparación, como pueden ser "mucho mayor que" o "menor o igual que", mediante una expresión del tipo de:

$$\mu_C = 1 - \mu_p \circ \max(0, u - v)$$

Proyección. Extensión cilíndrica

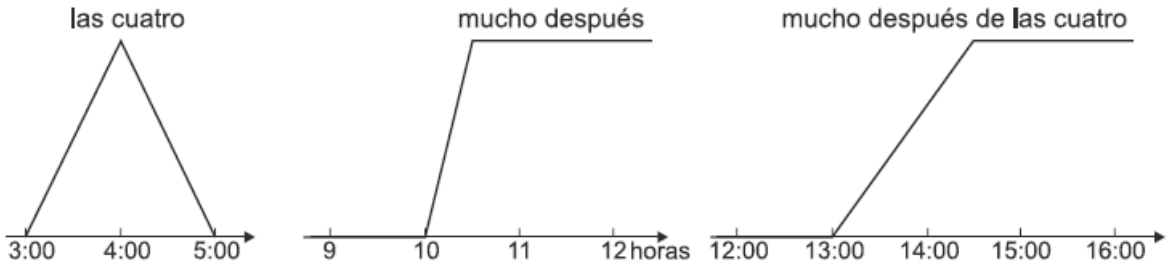
Dos conceptos matemáticos que nos serán de utilidad para la manipulación de información borrosa son los de proyección y extensión cilíndrica.

Composición de relaciones

Una operación propia de las relaciones que resulta de especial importancia es la composición de relaciones. Dada una relación *R* entre los conjuntos *U* y *V* , y otra relación *Q* entre los conjuntos *V* y *W,* definimos la composición de estas dos relaciones, y la denotamos como *R*◦*Q*, como una nueva relación entre los conjuntos *U* y *W.* Un ejemplo de esta composición lo encontramos en la sentencia "busco hotel cerca de alguna playa alejada de un centro urbano", donde la relación *R* se corresponde con la expresión "hotel cerca de alguna playa", y la relación *Q* se corresponde con la expresión "playa alejada de un centro urbano ". La composición entre *R* y *Q* define una relación entre $u \in U$ y $w \in W$ a partir de la relación que mantienen tanto *u* como *w* con cada uno de los elementos de *V* . Parece razonable que, dados tres elementos *u*, *v* y *w*, la relación que existe entre *u* y *w* no sea mejor que la peor de las relaciones que hay entre *u* y *v* y entre *v* y *w*, y dados dos elementos *u* y *w*, la relación que existe entre ellos debe ser la de aquel *v* con quien están mejor relacionados. Con esto en mente, Zadeh propone modelar este vínculo mediante la siguiente expresión:

$$\mu_{R \circ Q}(u, w) = \sup_{v} \min\{\mu_R(u, v), \mu_Q(v, w)\}$$

Esta expresión se conoce como ***composición sup-mín***, y generaliza la composición de relaciones precisas, según la cual *u* está relacionada con *w* si y sólo si existe un *v* tal que *u* está relacionada con *v* y *v* con *w*.



UN EJEMPLO DE COMPOSICIÓN BORROSA EN EL DOMINIO DE LOS NÚMEROS REALES

Supongamos las relaciones borrosas *R* en $U \times V$, *Q* en $V \times W$, *S* en $V \times W$ y *T* en $W \times X$. Se puede probar entonces el siguiente conjunto de propiedades:

$$\textit{Asociativa}\text{: } R \circ (Q \circ T) = (R \circ Q) \circ T.$$

$$\textit{Distributiva} \text{ respecto a la unión: } R \circ (Q \cup S) = (R \circ Q) \cup (R \circ S).$$

$$\textit{Distributiva débil} \text{ respecto a la intersección: } R \circ (Q \cap S) \subseteq (R \circ Q) \cap (R \circ S).$$

$$\textit{Monotonía}\text{: Si } Q \subseteq S \text{ entonces } R \circ Q \subseteq R \circ S.$$

El condicional

Un tipo de predicado de especial relevancia en la representación del conocimiento es el condicional, y aunque aparece en el lenguaje de múltiples maneras, como son: "no pienso salir hasta que te vayas", o "avísame en caso de que salgas", se suele estudiar bajo la forma común "si ... entonces ...".

Una expresión de la forma "Si x es A entonces y es B" puede adoptar múltiples significados en el lenguaje.

Como primera aproximación a la representación del condicional, nos interesa encontrar una implicación multivaluada, extensión de la implicación material clásica, y que permita modelar expresiones condicionales en las que los predicados A y B sean borrosos, como:

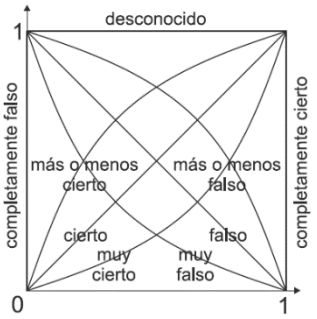
$$\mu_{A\to B}(u,v) = I(\mu_A(u), \mu_B(v))$$

Sustituyendo las distintas t-conormas, y la negación fuerte, obtenemos las siguientes expresiones para la relación de implicación obtenemos las siguientes expresiones para la relación de implicación:

$$\begin{aligned}
\mu_{A\to B}(u,v) &= \text{máx}\{1-\mu_A(u), \mu_B(v)\} \\
\mu_{A\to B}(u,v) &= 1-\mu_A(u)+\mu_A(u)\mu_B(v) \\
\mu_{A\to B}(u,v) &= \text{mín}\{1-\mu_A(u)+\mu_B(v), 1\}
\end{aligned}$$

Cualificación lingüística

De cualquier sentencia del lenguaje podemos decir si nos parece cierta o falsa. Y podemos añadir el uso de modificadores lingüísticos para dar lugar a apreciaciones del tipo de "muy cierta", "más o menos cierta" o "bastante falsa". A estas expresiones les podemos asignar un significado que viene determinado por un conjunto borroso en el intervalo [0, 1], y que por su sentido se corresponde con un valor de verdad borroso.



Razonamiento borroso

Hasta ahora hemos abordado el estudio de los conjuntos borrosos desde la perspectiva de su capacidad para representar aquel conocimiento que expresamos mediante el lenguaje natural. Nuestra preocupación se va a centrar ahora en la forma que adopta el razonamiento cuando involucra predicados vagos. Si la lógica clásica se ocupa del análisis de aquellas oraciones que pueden ser verdaderas o falsas, necesitamos una lógica borrosa que se ocupe del análisis de las oraciones cuya verdad sea una cuestión de grado.

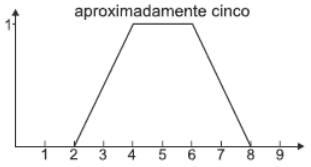
Una de las reglas de inferencia básicas en la lógica clásica es el Modus Ponens, ya que permite la deducción de nuevas observaciones y condiciones a partir de las observaciones y condiciones disponibles, y que podemos reducir al siguiente esquema:

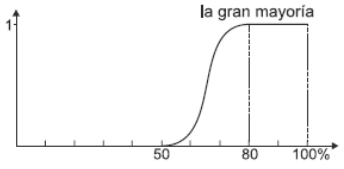
CONDICIÓN:	Si X es A	entonces	Y es B
OBSERVACIÓN:	X es A		
CONCLUSIÓN:			Y es B

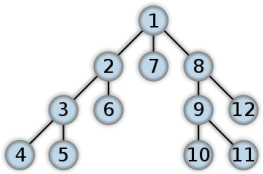
Cuantificación

Como hemos visto en el capítulo 2, la lógica de predicados de primer orden introduce dos cuantificadores presentes en el lenguaje: el cuantificador universal, representado matemáticamente por el símbolo ∀, y el cuantificador existencial, representado por ∃.

Sin embargo, el lenguaje natural admite una variedad de formas de cuantificar mucho más amplia, recorriendo múltiples situaciones intermedias entre los cuantificadores universal y existencial: mucho, pocos, bastante, la mayoría, casi, etc. Pensemos en sentencias comunes del tipo de "aproximadamente cinco personas miden más de 1'80m" o "la mayoría del congreso rechazó la moción". En estas sentencias el hablante cuantifica el número ("aproximadamente cinco") o proporción ("la gran mayoría") de individuos que satisfacen sendos predicados precisos ("miden más de 1'90m", "rechazó la moción"), y lo hace de una forma que no es obviamente universal ("todos") ni existencial ("al menos uno"), pero que indudablemente todos sabemos evaluar. Por otra parte, esta forma de cuantificar es inherentemente imprecisa, lo que nos sugiere realizar una definición borrosa de los cuantificadores que aparecen en el lenguaje, y que pasamos a llamar cuantificadores borrosos.







Glosario:

Recorrido en profundidad (depth-first search)

Una **Búsqueda en profundidad** (en inglés DFS o *Depth First Search*) es un algoritmo que permite recorrer todos los nodos de un grafo o árbol (teoría de grafos) de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa (Backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

Inicialmente se marcan todos los nodos como no visitados y se selecciona un nodo *u* como punto de partida. A continuación, se marca como visitado y se accede a un nodo no visitado *v* adyacente al nodo *u*. Se procede recursivamente con el nodo v. Al volver de la llamada o llamadas recursivas, si hay algún nodo adyacente que no se ha visitado, se toma como punto de partida y se vuelve a ejecutar el procedimiento recursivo. El recorrido termina cuando todos los nodos están marcados como visitados.

Este recorrido sería equivalente al recorrido en ***preorden*** de un árbol.

Se puede implementar el recorrido en profundidad de forma iterativa pero necesitaríamos una TAD tipo pila (LIFO)

```
tipo Vector = matriz[0..n] de booleano
fun RecorridoProfundidad(G = ⟨N,A⟩: grafo)
   var
      visitado: Vector
      v: nodo
   fvar
   para cada v ∈ N hacer
      visitado[v] ← falso
   fpara
   para cada v ∈ N hacer
      si ¬ visitado[v] entonces
         RecProfundidadRecursivo(v, visitado)
      fsi
   fpara
```