TEMA 1

Test de Turing: la maquina debe hacerse pasar por humana en un chat con una persona. Si la persona no se da cuenta de que es una maquina entonces la maquina es inteligente.(Utilizado para el control de spam,captcha)

-**General Problem Solved GPS**:para resolver problemas lenguage IPL

Russel y tipos de sistemas

-Sistemas que piensan como humanos(redes neuronales)

-S. Actúan como humanos(robótica)

-S. Piensan racionalmente(con lógica)

-S. Actúan racionalmente(emular comportamiento humano)

-IA y sus grupos

-IA fuerte:emulan comportamiento humano , pensar,aprender,resolver,crear

-IA debil:maquinas para resolver problemas especificos

Agente Inteligente:Cosa que percibe del entorno y actua en el.

-Funcion:define el comportamiento

-Programa:hace la funcion

-Agente: hace la acción

-Autonomia:comportamiento en base a experiencia

Entornos

-**Totalmente/parcialmente observable**: Los sensores ¿acceden al estado completo del entorno?(problemas de juguete/Conducir)

-**Determinista/estocástico**: El siguiente estado ¿está totalmente determinado por la acción?

-**Episódico/secuencial**: ¿Se puede dividir la experiencia del agente en episodios atómicos, tales que la elección de acción en cada episodio depende solo de éste?(Clasificación/Ajedrez)

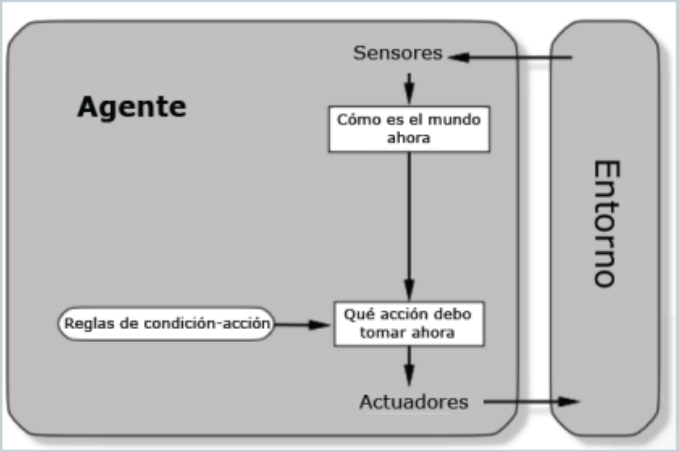
-**Estático/dinámico**: ¿Puede cambiar el entorno mientras el agente elige una acción?(Resolver Crucigrama/Conducir)

-**Discreto/continuo**: ¿Cómo se maneja el tiempo, las percepciones, las acciones?(Ajedrez/Conducir)

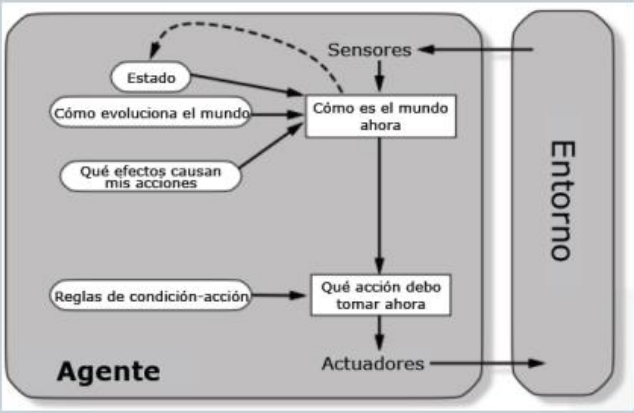
-**Individual/Multiagente**: ¿Hay un solo agente?(Busca ruta en mapa/Competi:Parchis,Cooperativo:robots industria)

Estructura de Agentes

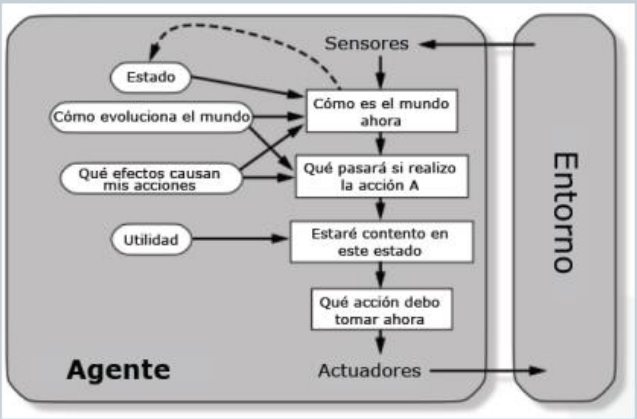
Agentes reactivos simples.



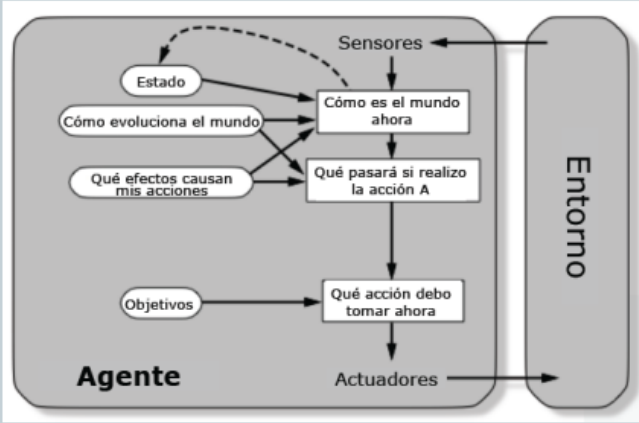
Agentes reactivos basados en modelos.

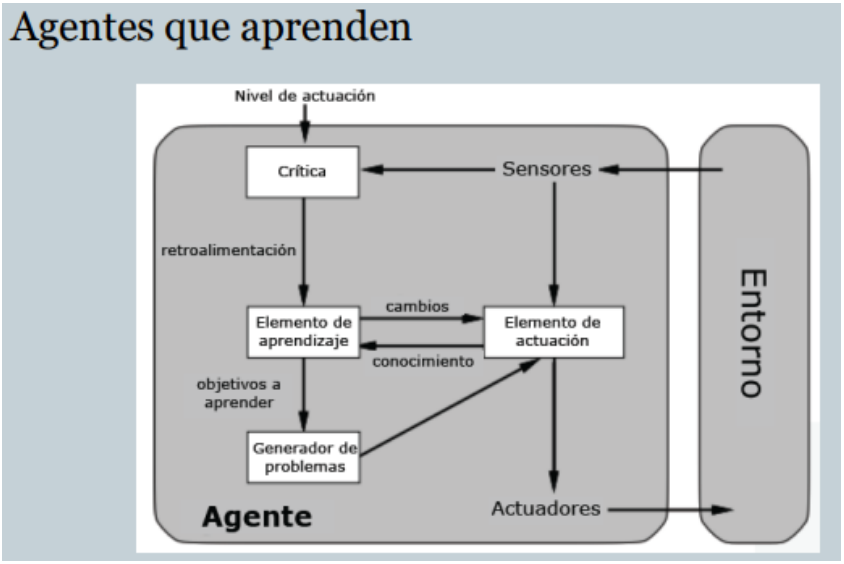


Agentes basados en utilidad



Agentes basados en objetivos





TEMA 2.1

Resolución de problemas

Etapas:

1º Proceso con datos iniciales

2º Utilizando un conjunto de procedimientos escogidos

3º Determinar el conjunto de pasos a hacer

4º Solución

Como resolver un problema

-Encontrar un lenguaje común para definir el problema

-Definir algoritmos o estrategias para resolverlo

Pasos para resolver un problema

-Establecer un punto de partida con las características del problema

-Establecer un objetivo a alcanzar

-Establecer acciones para resolverlo

-Establecer que características tiene que tener la solución

-Establecer los conocimientos para resolver el problema

Representar problemas para resolverlos de forma automática

-Problemas generales

Espacio de estados: conjunto de pasos para la resolución

Reducción a subproblemas

-Problemas especificos

Resolución de juegos: Competición entre dos agentes

Satisfacción de restricciones: cumplir requisitos

Estado: representación de los elementos y características de un problema en un momento X

Operadores de transformación(Mover la ficha): Para cambiar de estado y definen a su vez una **Relación de accesibilidad** entre estados

Espacio de estados: Son todos los caminos que hay entre los estados(posibilidades) la solución es un camino.

Coste de la solución: la suma de cada coste de cada estado hasta llegar a la solución

Tamaño de espacios: cantidad de estados que puede tomar el problema

Definir un problema con espacio de estados

-Definir conjunto de estados

-Definir estado inicial

-Definir estado final y condiciones

-Definir operadores de cambio de estado

-Especificar solución: estado final ,el camino con menor coste….

Resolver un problema con espacio de estados

-Definir representación del conjunto de estados:

Estructura de datos:

Estados=Nodos

Arboles: Solo un camino lleva a un nodo

Grafos: varios caminos pueden llevar a un nodo

Grafo: Se construye mientras de hace ,Puede ser infinito

Función: Búsqueda en espacio de estados().
Datos: El estado inicial.
Resultado: Una solución.
Seleccionar el primer estado como el estado actual
mientras estado actual != estado final **hacer**
    Generar y guardar sucesores del estado actual (expansión)
    Escoger el siguiente estado entre los pendientes (selección)
fin

Selección del siguiente nodo

-Orden de expansión: orden en el que generamos los sucesores

-Orden de selección: Explorar nodos que no se han visitado

Tipos de grafo

-Nodos abiertos: generados pero no visitados

Nodos cerrados: estados ya visitados y expandidos

Nuestro Algoritmo dispondrá de una lista de nodos abiertos que se insertaran FIFO o QUEQUE dependiendo de el tipo de búsqueda

```
Algoritmo: Búsqueda General
Est_abiertos.insertar(Estado inicial)
Actual ← Est_abiertos.primero()
mientras no es_final?(Actual) y no Est_abiertos.vacia?() hacer
        Est_abiertos.borrar_primero()
        Est_cerrados.insertar(Actual)
        Hijos ← generar_sucesores(Actual)
        Hijos ← tratar_repetidos(Hijos, Est_cerrados, Est_abiertos)
        Est_abiertos.insertar(Hijos)
        Actual ← Est_abiertos.primero()
fin
```

Variando la estructura de abiertos variamos el comportamiento del algoritmo (orden de visita de los nodos).

La función generar sucesores seguirá el orden de generación de sucesores definido en el problema.

El tratamiento de repetidos dependerá de cómo se visiten los nodos.

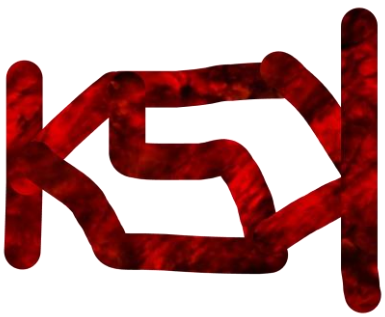
Características del algoritmo

-**Completitud:** ¿Encontrará una solución?

-**Complejidad temporal**: ¿Cuanto tardará?

- **Complejidad espacial**: ¿Cuanta memoria gastará?

-**Optimalidad:** ¿Encontrará la solución óptima?



El algoritmo se puede clasificar en:

Algoritmos de búsqueda ciega

- No tienen en cuenta el coste de la solución en la búsqueda.

- Su funcionamiento es sistemático, siguen un orden de visitas y generación de nodos establecido por la estructura del espacio de búsqueda.

EJ: Anchura prioritaria, Profundidad prioritaria, Profundidad iterativa

Algoritmos de búsqueda heurística.

-Utilizan una estimación del coste de la solución para guiar la búsqueda.

-No siempre garantizan el óptimo, ni una solución.

EJ: Hill-climbing, Branch and Bound, A*, IDA*

TEMA 2.2

Algoritmos de búsqueda no informada o búsqueda ciega: no dependen de información propia del problema a la hora de resolverlo. Se basan en espacios de estados y su búsqueda recorriéndolos.

-2 Tipos de recorridos: en **Anchura** y en **Profundidad**

-solo serán aplicables para problemas pequeños por lo sistemático y exhaustivo

-No hace falta tener ningún conocimiento del problema

Ej para el ajedrez serian 10^30 años

Búsqueda en Anchura prioritaria (recorrido z)

-Arriba de abajo de izquierda a derecha(por niveles) la lista va->

Un nodo se visita cunado sus padres y hermanos anteriores ya sean visitado

Los nodos abiertos se almacenan en cola **FIFO**

Completitud: el algoritmo tarda pero encuentra solución.

Complejidad temporal: tiempo que tarda en encontrar la solución

Complejidad espacial : almacenaje de datos(nodos pendientes Ramificación^Profundidad)

Optimalidad: la solución es optima respecto al numero de pasos tomados en total

Búsqueda en Profundidad Prioritaria.(recorrido en N)

-Na:Criterio de expansion de cada nodo cerrado <-

-Nc:cojemos en primer nodo de la pila y ponemossus hijos en NA

Los nodos abiertos se guardan en una pila **LIFO**

-Hay caminos (infinitos)

-Variante->**Profundidad limitada** para poner un máximo de profundidad para acceder , no garantiza encontrar solución

Algoritmo de profundidad limitada

```
Procedimiento: Búsqueda en profundidad limitada (limite: entero)
Est_abiertos.insertar(Estado inicial)
Actual ← Est_abiertos.primero()
mientras no es_final?(Actual) y no Est_abiertos.vacia?() hacer
        Est_abiertos.borrar_primero()
        Est_cerrados.insertar(Actual)
        si profundidad(Actual) <= limite entonces
                Hijos←generar_sucesores (Actual)
                Hijos←tratar_repetidos (Hijos, Est_cerrados, Est_abiertos)
                Est_abiertos.insertar(Hijos)
        fin
        Actual←Est_abiertos.primero()
fin
```

Completitud: si se pone profundidad limite y existe en ese limite si encuentra sol,si no inf

Complejidad temporal: Ramificación^Profundidad

Complejidad espacial : Ramificación^Profundidad si es recursivo el coste es solo profundidad(uno a uno)

Optimalidad: no garantiza sol optima

Búsqueda en Profundidad Iterativa. (IDA)

Ventajas de profundidad y anchura

Repetir el algoritmos de profundidad limitada aumentando la profundidad máxima a cada iteración.

Algoritmo de profundidad iterativa (Iterative Deepening)

```
Procedimiento: Búsqueda en profundidad iterativa (limite: entero)
        prof ←1
        Actual←Estado inicial
        mientras no es_final?(Actual) y prof < limite hacer
                Est_abiertos.inicializar()
                Est_abiertos.insertar(Estado inicial)
                Actual ←Est_abiertos.primero()
                mientras no es_final?(Actual) y  no Est_abiertos.vacia?() hacer
                        Est_abiertos.borrar_primero()
                        Est_cerrados.insertar(Actual)
                        si profundidad(Actual)<=prof  entonces
                                Hijos ←generar_sucesores (Actual)
                                Hijos ←tratar_repetidos (Hijos, Est_cerrados, Est_abiertos)
                                Est_abiertos.insertar(Hijos)
                        fin
                        Actual←Est_abiertos.primero()
                fin
                prof ←prof+1
        fin
```

Completitud: El algoritmo siempre encontrará la solución de existir

Complejidad temporal: La misma que la búsqueda en anchura. El regenerar el árbol en cada iteración solo añade un factor constante a la función de coste. Ramificación^Profundidad

Complejidad espacial: Igual que en la búsqueda en profundidad Ramificación^Profundidad

Optimalidad: La solución es óptima

TEMA 2.3

Los algoritmos de búsqueda vistos tienen un coste temporal que es una función exponencial del tamaño de entrada.

Búsqueda Informada

Para reducir el coste temporal hay que tener un conocimiento del problema especifico del problema perdiendo la generalidad de los otros algoritmos.

Los algoritmos calcularan el coste de cada camino para saber cual merece la pena explorar

Búsqueda del primera mejor

Se selecciona un nodo para la expansión basada en una **función de evaluación f(x)** (mas baja)

Método venerable pero inexacto

Búsqueda Heurística o Búsqueda Voraz el primero Mejor

función heurística h(n)= coste estimado del camino más barato desde el nodo n a un nodo objetivo.

Elegiremos el camino en el que los nodos estén mas cerca de la solución.

Se almacenan los nodos abiertos en una cola con prioridad ,marcada por el coste del nodo hasta la solución

```
Est_abiertos.insertar(Estado inicial)
Actual←Est_abiertos.primero()
mientras no es_final?(Actual) y no Est_abiertos.vacía?() hacer
        Est_abiertos.borrar_primero()
        Est_cerrados.insertar(Actual)
        hijos←generar_sucesores (Actual)
        hijos←tratar_repetidos (Hijos, Est_cerrados, Est_abiertos)
        Est_abiertos.insertar(Hijos)
        Actual←Est_abiertos.primero()
fin
```

Al tener en cuenta el nodo mas cercano a la solución no evaluamos su coste por lo que estamos ignorando el coste del camino y la solución optima.

A* Algoritmo

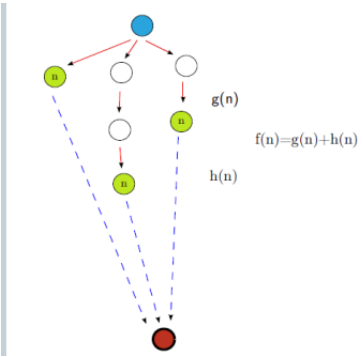
Costo de un arco: coste entre dos nodos

Costo de un camino: costo total en dos nodos conectados por otros nodos

Función de evaluación: g(n)+h(n)

G(n): coste del camino mas corto conocido hasta n

H(n):valor estimado de lo que falta para llegar desde n a la solución.



h (n)= 0 sera un algoritmo de Branch & Bound

Si la h(n) va decreciendo encontraremos el camino optimo

Algoritmo: A*

```
Est_abiertos.insertar(Estado inicial)
Actual←Est_abiertos.primero()
mientras no es_final?(Actual) y no Est_abiertos.vacía?() hacer
        Est_abiertos.borrar_primero()
        Est_cerrados.insertar(Actual)
        hijos←generar_sucesores (Actual)
        hijos←tratar_repetidos (Hijos, Est_cerrados, Est_abiertos)
        Est_abiertos.insertar(Hijos)
        Actual←Est_abiertos.primero()
fin
```

La cola de **nodos abiertos** tiene la prioridad marcada por la función **f(n)= g(n)+h(n)**

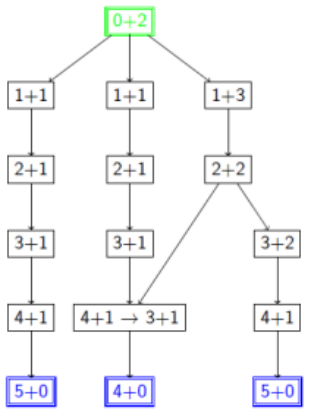
En cada iteración se coje el mejor camino(el primero de la cola)

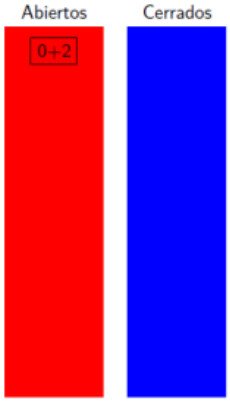
Si hay nodos repetidos:

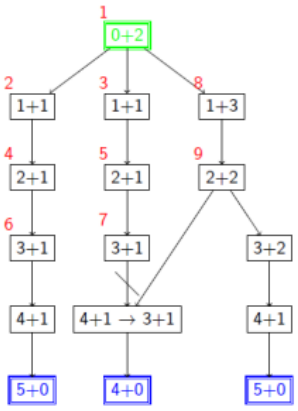
-En la cola de nodos abiertos: si el coste es menor lo sustituimos cambiándolo,

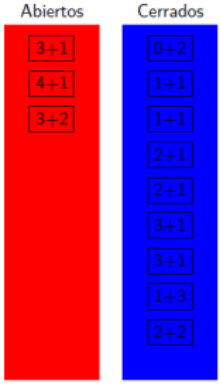
si es igual o mayor nada

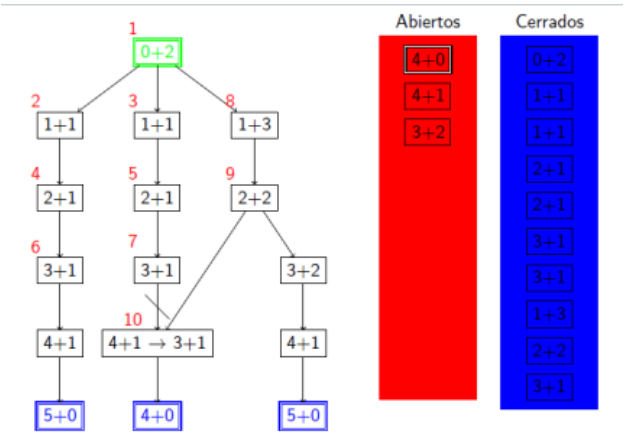
- En la cola de nodos cerrados: si el coste es menor reabrimos el nodo volviéndolo a insertar en la cola de abiertos , si es igual o mayor nada

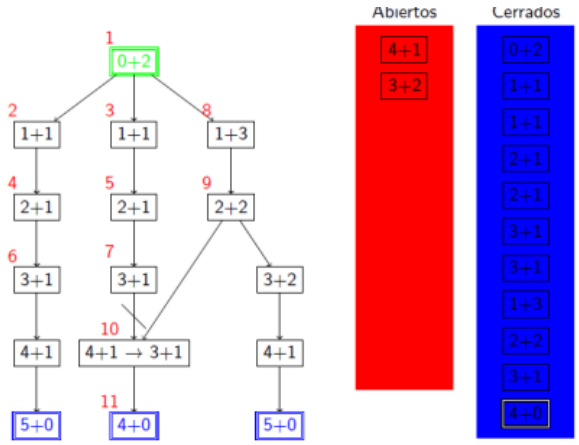










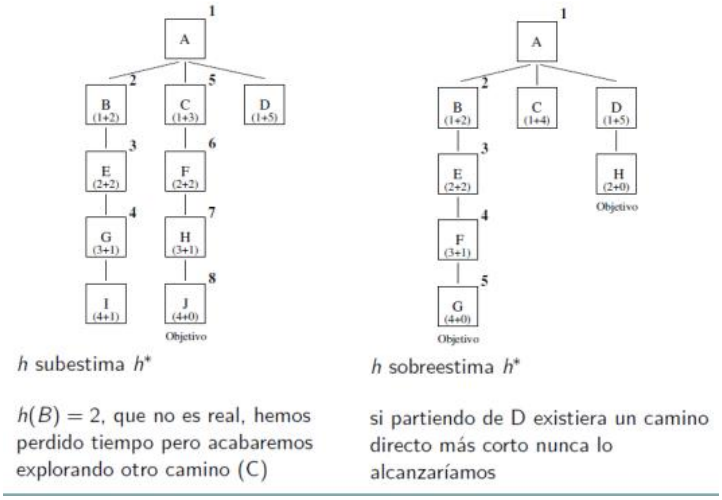


Para que sea optimo 0<=h(n)<=h*(n)

h*(n) es la siguiente (debe ser optimista)

Existen mejores algoritmos para

almacenar nodos en memoria



Utilidad de consistencia.

- h(n) consistente ➔ g(n) = k(s, n) (camino óptimo a n)
- Si g(n) = k(s, n), dado que h(n) siempre es constante ➔ f (n) es mínima
- En este caso **no es necesario tratar los nodos duplicados cerrados** ya que los nodos expandidos ya no se podrán reexpandir (hemos llegado a ellos por el camino mínimo)

WTF

IDA* Algoritmo

-Ventajas y desventajas de A* pero el coste espacial es mejor

-Se le da un conocimiento parcial del grafo(con un limite , establecido por la función heurística) incrementándolo en cada iteración

-Si no encuentra solución se actualiza el limite

```
Algoritmo: IDA* (limite entero)
Prof←f (Estado inicial)
Actual←Estado inicial
mientras no es_final?(Actual) y prof<limite hacer
        Est_abiertos.incializa()
        Est_abiertos.insertar(Estado inicial)
        Actual ←Est_abiertos.primero()
        mientras no es_final?(Actual) y no Est_abiertos.vacía?() hacer
                Est_abiertos.borrar_primero()
                Est_cerrados.insertar(Actual)
                Hijos←generar_sucesores (Actual, prof)
                Hijos←tratar_repetidos (Hijos, Est_cerrados, Est_abiertos)
                Est_abiertos.insertar(Hijos)
                Actual ←Est_abiertos.primero()
        fin
        prof←prof+1
fin
```

TEMA 2.4

Búsqueda con adversario para resolver problemas de juego (ajedrez)

-Jugador MAX: inicia el juego, el objetivo es que MAX gane

-Jugador MIN

El acceso a los estados dependen de la elección de estado del rival

Existen dos tipos de soluciones distintas una para cada jugador

No hay noción de optimalidad (todas las soluciones son iguales, no importa la longitud del camino)

-General un árbol de jugadas gana MAX=+1 gana MIN=-1

-Búsqueda en profundidad (para juegos sencillos)

Algoritmo MiniMax

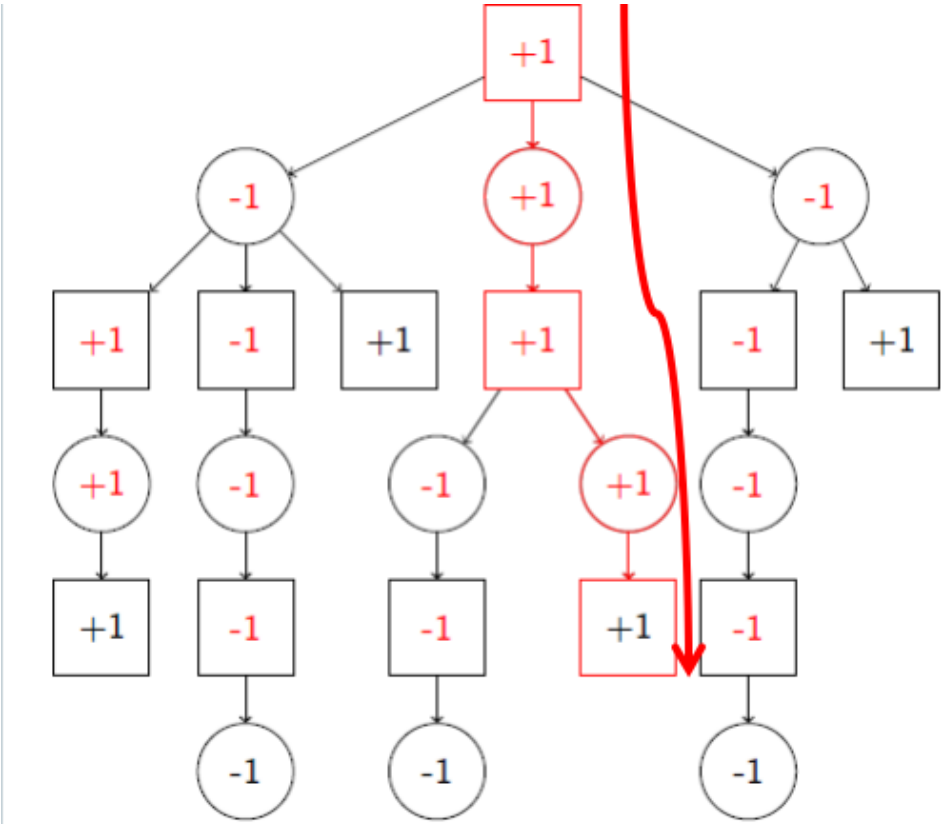
Búsqueda con adversario usando Función Heurísticas :Para saber que tan cerca estamos de ganar MAX

Las jugadas ganadoras=+inf y las perdedoras=-inf

Busca en profundidad limitada (cuanto mas profundidad mejor)

Consumen mucha memoria (ineficiente , su mejora es la poda AlfaBeta)

Se empieza de abajo a arriba y se extiende el valor de las hojas a los padres Min=-1=circulo Max=+1=cuadrado y se elige el camino con todo +1. Si el hijo es +1 y su hermano -1 pero el padre es circulo=-1 si es cuadrado +1 si es hijo único hereda el valor



Poda AlfaBeta

https://www.youtube.com/watch?v=I0y-TGehf-4&ab_channel=BitBoss

Descarta ramas que no sean factibles.

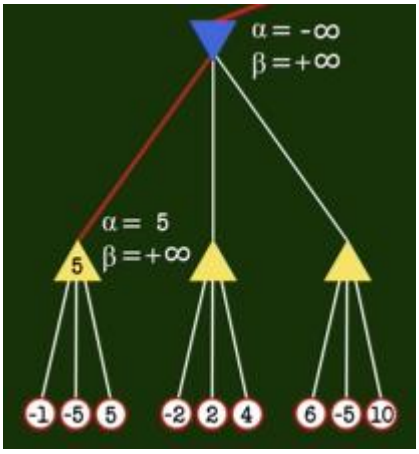
Se inicializa todos los nodos del árbol a Alfa=-inf y Beta=+inf, menos las hojas

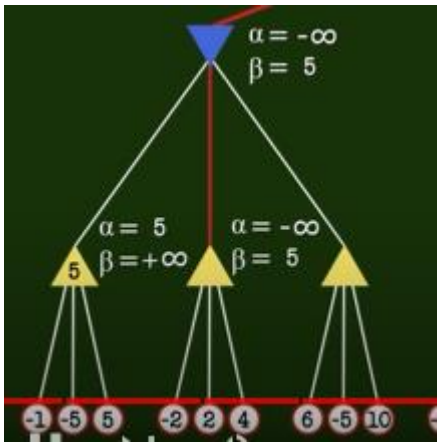
Si el padre es MAX se actualiza el valor de alfa si es mayor

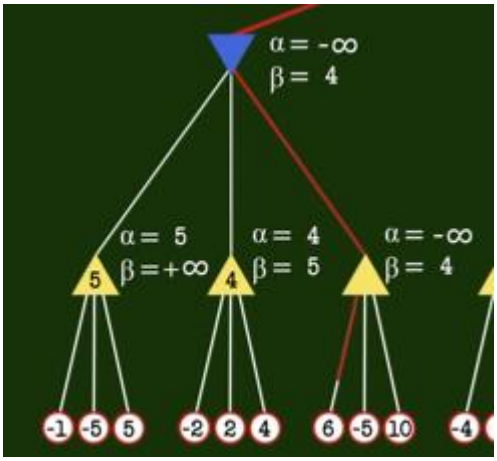
Si el padre Min se actualiza el valor de beta si es menor

Se actualiza el padre del padre y comprobamos que Alfa >= Beta , si es así no se podrá mejorar y entonces se poda la rama, si no se sigue y se pasa el valor del padre al hijo

Azul Min Amarillo max







En la ultima imagen cuando inspeccionamos el 6 se actualiza el valor de Alfa=6 y como **Alfa>=Beta** no hace falta explorar mas podamos el -5 y -10

Complejidad Ramificacion^Profundidad

Para hacer el algoritmo mas eficiente, explorar primero las ramas donde se encuentren las mejores jugadas y ordenar nodos creciente para MIN y decreciente para MAX hace que la complejidad sea Ramificacion^Profundidad/2

TEMA 2.5

Navegar dentro del espacio de posibles soluciones haciendo operaciones para mejorar las soluciones

Elementos:

-Estado inicial: una solución

-Operadores de cambio de solución: manipulan las soluciones (sin coste)

-Función de calidad: para saber que tan buena es una solución

Tamaños de espacios de búsqueda inmensos entonces tendremos que delimitarlo y saber cuales no explorar=ramificación y poda =(branch and bound)

Algoritmos de ramificación y poda

Algoritmo Hill-Climbing

2 tipos :

-Ascenso de colinas simple: elegir el primer nodo que este mejor que el actual(mas rápido ,pero inexacto)

-Ascenso de colinas por máxima pendiente: Expande los descendientes del nodo y coje el mejor(mas utilizado, pero arriesgado)

Solo se consideran los descendientes cuya función de estimación es mejor que la del padre.

Es posible acceder a óptimos locales, dependerá del punto de inicio.(no garantiza solución optima)

Posible solución:

-Reiniciar la búsqueda en otro punto buscando mejorar la solución actual (Random Restarting Hill Climbing)

-Uso de una población de soluciones BeamSearch

Swarm intelligence

Algortimos inspirados en comportamientos de grupos de pájaros,hormigas,abejas

Algoritmo Simulated Annealing(templado simulado)

Inspirado en un fenómeno físico que se observa en el templado de metales y en la cristalización de disoluciones (variando la temperatura)

El siguiente nodo a explorar no será siempre el mejor descendiente, sino que lo elegiremos aleatoriamente, en función de los valores de unos parámetros, de entre todos los descendientes

Minimizar la funcion de coste **Heurística** (energía) pero en funcion de otros parámetros(temperatura) nos podremos mover a soluciones peores ,evitando óptimos locales

Además de otra funcion que dependerá de la temperatura y la diferencia de calidad entre el nodo actual y el sucesor.

El numero de iteraciones es fijo y cuando avanza(cada x tiempo) la temperatura bajara ,llegando al final a 0.

No hace falta generar todos los sucesores , solo uno al azar y elegir continuamos por el.

Temp alta=random Temp baja=el mejor vecino Hill climbing

1. Inicialización. Solución aleatoria. 'Temperatura 'muy alta.
2. Movimiento. Perturbar la solución con algún tipo de movimiento.
3. Evaluar. Calcular la variación de la puntuación (energía).
4. Elegir. Dependiendo del resultado de la evaluación aceptar o rechazar. La probabilidad de aceptación depende de la 'temperatura'.
5. Aceptar una solución mala con probabilidad variante.
6. Actualizar y repetir. Reducir el valor de la temperatura. Volver al paso 2 hasta que alcancemos el 'punto de enfriamiento'.

Para que sirve

Problemas de optimización combinatoria, problemas grandes con muchos óptimos locales

Para el problema del viajante de comercio

Desventaja: Determinar los valores de los parámetros requiere experimentación.

TEMA 3

METAHEURISTICA

Los algoritmos metaheurísticos son algoritmos aproximados de optimización y búsqueda de propósito general.

Ventajas:

Algoritmos de propósito general

Gran éxito en la práctica

Fácilmente implementables

Fácilmente paralelizables

Inconvenientes:

Son algoritmos aproximados, no exactos

Son no determinísticos (probabilísticos)

No siempre existe una base teórica establecida

Intensificación: cantidad de esfuerzo empleado en la búsqueda en la región actual (explotación del espacio)

Diversificación: cantidad de esfuerzo empleado en la búsqueda en regiones distantes del espacio (exploración)

Buscar un equilibrio entre Intensidad y diversificación para saber un buen espacio de soluciones y no perder el tiempo en regiones sin futuro.

Clasificación de las metaheurísticas

-Basadas en métodos constructivos: Colonias de hormigas

-Basadas en trayectorias :enfrentamiento simulado

-Basadas en poblaciones: algoritmo genético

TEMA 3.2

ALGORITMOS DE BÚSQUEDA

Para problemas de gran complejidad computacional que resuelven con buenas soluciones(no tiene que ser la optima)en un tiempo razonable.

Teniendo una Función Objetivo: con las variables de decisión , y Espacio de búsqueda: valores de las variables de decisión

Ej con variable binaria: Problema de la mochila

Ej con variable continua: Viajante de comercio: red de nodos(ciudades) que tienen que unirse sin repetirse es un grafo cerrado (volver al inicio como un circulo)

ALGORITMOS APROXIMADOS

Aportan soluciones cercanas a la óptima en problemas complejos (NPduros) en un tiempo razonable.

Cuando se usa

-Cuando no hay un método exacto o es demasiado costoso

-Cuando se necesita solución de buena calidad(no optima) en tiempo aceptable

Características

Entorno: soluciones cercanas

Movimiento: transformación de solución actual en otra

Evaluación: para evaluar la solución+

TEMA 3.3

Computación evolutiva: modelos de evolución de población donde cada individuo es una solución

EVOLUCIÓN		RESOLUCIÓN DE PROBLEMAS
Individuo	⟷	Solución Candidata
Adaptación	⟷	Calidad
Entorno	⟷	Problema

Cada cromosoma se trocea en n partes las cuales son recombinadas

La mutación ocurreo con probabilidad en cada gen

EDA: Algoritmos basados en

estimación de distribuciones

Algoritmos Genéticos que utilizan operadores genéticos sobre cromosomas. 1975, Michigan University

Estrategias de Evolución que enfatizan los cambios de comportamiento al nivel de los individuos. 1964, Technische Universität Berlin

Hans-Paul Schwefel
Universität Dortmund

Programación Evolutiva que enfatizan los cambios de comportamiento al nivel de las especies. 1960-1966, Florida

Programación Genética que evoluciona expresiones representadas como árboles. 1989, Stanford University

Aplicaciones

Generación de trayectorias,Aprendizaje,Clasificación

Conclusiones

Ventajas

Buena actuación a costo aceptable

Paralelismo intrínseco

Superior a otras técnicas como muchos óptimos (locales)

Sin restricciones de espacio de soluciones

Bajo coste de desarrollo

Soluciones interpretables

Fácil de mezclar con otras técnicas

Desventajas

No garantiza solución optima en tiempo finito

Débil base teórica

Tiene muchos parámetros para ajustar

Computación costosa

TEMA 3.4

Algoritmos Genéticos que utilizan operadores genéticos sobre cromosomas

Estrategias de Evolución que enfatizan los cambios de comportamiento al nivel de los individuos

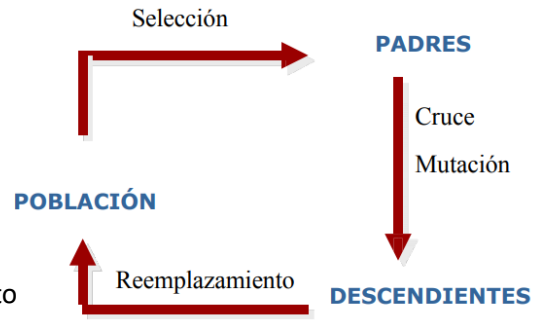
Programación Evolutiva que enfatizan los cambios de comportamiento al nivel de las especies

Programación Genética que evoluciona expresiones representadas como árboles

Algoritmos Genéticos

son algoritmos de optimización, búsqueda y aprendizaje inspirados en los procesos de evolución natural y genética.

Muy potentes ,altas prestaciones a un bajo costo



Como hacerlo

-Diseñar representación del individuo

-Inicializar población(valores aleatorios siguiendo una reglas)

-Operadores de mutación

-Operadores de cruce

-Operadores de selección de individuos

-Decidir como evaluar a los individuos y evaluarlos

-Saber cuando un individuo es solución y parar

Cromosoma: representación de un individuo formado por genes

Genotipo: representación del individuo (genes para programar)

Representación binaria: bits

Representación real: vectores

Representación de orden: permutación

Fenotipo: la parte visible lo que es (un sudoku)

Representación binaria: un numero

Representación real: valor real

Evaluación del Individuo : Para saber que tan bueno es un individuo

Si hay restricciones se suele penalizar en la fitness , y aciertos con aciertos

Operador de mutación: Modifica cualquier individuo por probabilidad generando uno nuevo

-Representación binaria :cambia un bit

-Representación real: permutando valores

-Representación de orden: selección aleatoria de dos genes y se cambian los lugares

Operador de Cruce: manera para que los padres produzcan hijos con las características de ambos

-Representación binaria :cortar cromosomas padres y recombinarlos

-Representación real: recombinación aleatoriamente AbCdfgHI

-Representación de orden: cruce aritmético

Estrategia de Selección: para garantizar que los mejores individuos tengan mas probabilidad de reproducirse (sin descartar a los menos buenos que pueden ser valiosos) **Presión selectiva**

-Selección proporcional ruleta(mejor es mas espacio en la ruleta)

Desventajas: peligro de convergencia prematura, baja presión selectiva

Solución->Escalado del fitnes:ajustar la fitnes de 0 a 1 o Normalizar: que la suma sea=1

-Selección por torneo: se cogen n individuos aleatoriamente y se elije el mejor

-Selección basada en orden rank: método de selección de individuos. Este método utiliza la probabilidad de selección de un individuo es inversamente proporcional a la posición que ocupa tras ordenar todos los individuos de mayor a menor fitness

Estrategia de remplazamiento de los individuos

-Se puede reemplazar toda la población actual con la de descendientes (esquema **generacional**) o sólo una parte de ella (esquema **estacionario** :es elitista convergencia rápida)

Criterio de parada: cuando se alcanza el optimo la solución, después de x iteraciones

Utilizar Algoritmos genéticos

-Nunca sacar conclusiones de una ejecución: utilizar varias y estadística

-No trabajar con casos simple , si no con casos reales

Doble enfoque y diferente diseño

- Encontrar una solución muy buena al menos una vez
- Encontrar al menos una solución muy buena en cada ejecución

Diversidad genetica

Falta de diversidad ->convergencia al vecino más cercano

Alta presión selectiva-> falta de diversidad

Soluciones: mecanismos de diversidad o reinicio cuando hay convergencia prematura

Exploración = muestrear regiones desconocidas(si es excesiva =búsqueda aleatoria)

Explotación = trata de mejorar el mejor individuo(si es excesiva =búsqueda local)

LEER TEMA OPERADORES GENETICOS

TEMA 3.5

Swarm Intelligence

Swarm Intelligence: La inteligencia colectiva emergente de un grupo de agentes simples

Sociedades de Insectos ,sistemas de enjambre: actuaciones colectivas, sin divisiones efectivas(sin inteligencia) , ni garantistas .Computación distribuida.

Toma de decisión colectiva, comunicación directa por contacto y indirecta por comportamiento.

Enjambre

-Agentes simples(auto-organizadas)

-Descentralizados(no hay único supervisor ,lider)

-Si plan global

-Robusto(aunque uno falle se completa)

-Flexible al entorno

Particle Swarm Optimization – PSO: técnicas inspiradas en el comportamiento de las bandadas de aves o bancos de peces: problemas de optimización ,cada solución una ave, en un espacio de búsqueda móvil

Ant Colony Optimization – ACO: para rutas de grafos ,dejando feromonas en los caminos

reflexión: Partes tontas/mudas, conectadas adecuadamente en un enjambre, producen resultados elegantes/inteligentes

TEMA 3.6

Algoritmos memeticos

Introducción:

Todos los problemas pueden ser representados como un Problema de Optimización que es la búsqueda del máximo (o el mínimo) de una función objetivo dada.

Los métodos determinísticos pueden fallar porque pueden converger hacia el óptimo local.

Los algoritmos evolutivos pueden fallar porque podrían converger hacia una solución subóptima

El **Meme** es una unidad de "transmisión cultural"

La combinación de Algoritmos Evolutivos con Operadores Locales de Búsqueda que trabajan dentro del bucle EA se ha denominado "**Algoritmos Memeticos**":que son mas rápidos y precisos que los EA en algunos problemas

Local Searcher (LS): un método determinístico capaz de encontrar el óptimo local más cercano

Order zero(búsqueda directa) Order uno (primera derivada) Order dos(segunda derivada)

Steepest Ascent Pivot Rule: el LS explora todo el vecindario

Greedy Pivot Rule:utiliza la primera dirección de busqueda

Profundidad: criterio de parada

Como combinar EA y LS

- Población inicial: por regla heurística (vectores ortogonales) aumenta la aptitud pero disminuye la diversidad

-**Intelligent Crossover**: encuentra la mejor combinación entre los padres para generar la descendencia con mayor rendimiento

-Intelligent Mutation : prueba varios individuos mutados posibles para obtener la mutación más "afortunada"

Algoritmos Meméticos: Búsqueda local actuando sobre la descendencia

LS al vástago de Krasnogor:existen ventajas teóricas al usar una búsqueda local con un operador de movimiento pero depende del problema

Lamarckian: los rasgos adquiridos por un individuo a lo largo de su vida pueden transmitirse a su descendencia

Baldwinian: los rasgos adquiridos por el individuo no pueden ser transmitidos a su descendencia

Para mejorar la eficiencia y la robustez de una EM:

-Adaptativo: los memes se controlan durante la evolución mediante unas reglas en función del estado de la población.

-Auto adaptativo: las reglas adaptativas se codifican en el genotipo de cada individuo

Sistemas Multi-Meméticos: emplea un conjunto (una lista) de LSs y estrategias adaptativas para seleccionar la LS que mas convenga en la etapa

Diferentes LSs deberían "competir" y "cooperar" trabajando para resolver el clásico problema, en EAs, del equilibrio entre "exploración" y "explotación".

TEMA 4.1

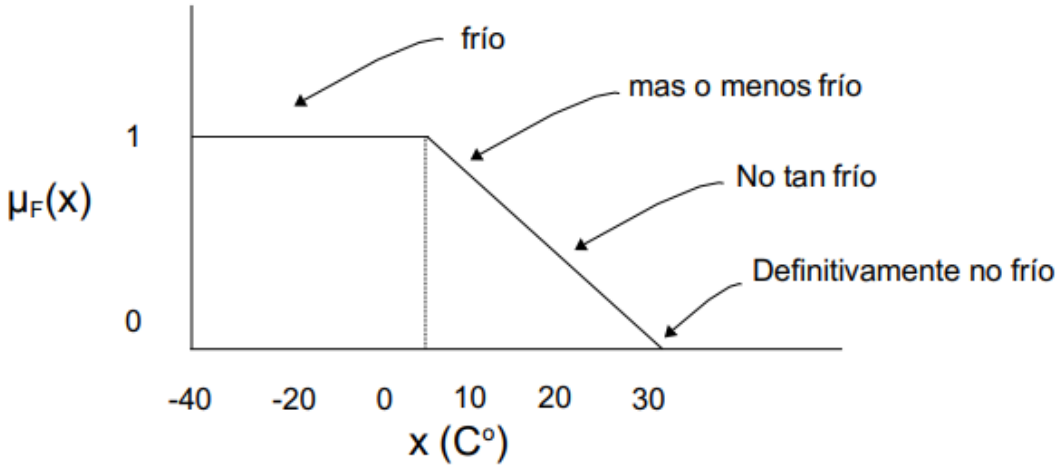
La lógica difusa

La lógica difusa :no usa valores exactos como 1 o 0 pero usa valores entre 1 y 0 para que algo no sea blanco o negro

Set difuso: Asumiendo que X es un set, un set difuso A en X es asociado con una función característica o **función de pertenencia o menbresia**: μA(x) X=un numero entre 0 y 1

La función μF(x) nos indica cual es el grado de pertenencia de x al atributo F

Ej: $\mu_F(x)$ corresponde al nivel de frío medido en la variable x



Un set exacto: μS : X -> {0,1} puede tomar 1 o 0 {}

μS(x) = 1 si x es un miembro de S

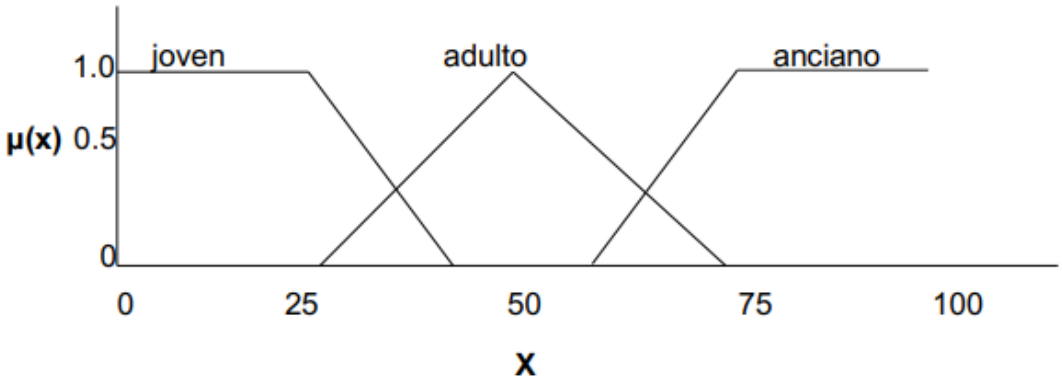
Un set difuso: μS : X -> [0,1], 0 ≤ μS(x) ≤ 1 valores entre 0 y 1

μS(x) = 0.20 si x es 20% miembro de S

Variables Lingüísticas

Se usan variables lingüísticas para analizar y modelar un sistemas:

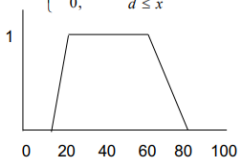
Ej: Supongamos que X = "edad", se pueden definir set difusos: "joven", "adulto", "anciano"



- Función de pertenencia trapezoidal:

$$\mu_{trapesoidd}(x;a,b,c,d)=\begin{cases}0, & x\le a\\ \frac{x-a}{b-a}, & a\le x\le b\\ 1, & b\le x\le c\\ \frac{d-x}{d-c}, & c\le x\le d\\ 0, & d\le x\end{cases}$$

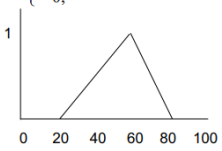
Ej: $\mu_{trapesoid}$ (x; 10, 20, 60, 95)



- Función de pertenencia triangular:

$$\mu_{triángulo}(x;a,b,c)=\begin{cases}0, & x\le a\\ \frac{x-a}{b-a}, & a\le x\le b\\ \frac{c-x}{c-b}, & b\le x\le c\\ 0, & c\le x\end{cases}$$

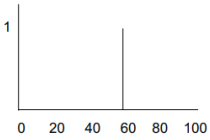
Ej: $\mu_{triangle}$ (x; 20, 60, 80)



- Función de pertenencia llamada un singleton, tiene un valor único cuando x = a (es como una función delta de Dirac):

$$\mu_s(x;a)=\begin{cases}0, & x=a\\ 1, & x\ne a\end{cases}$$

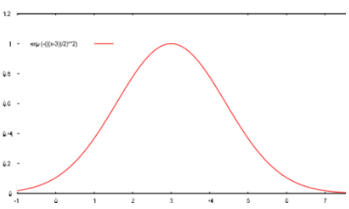
Ej: $\mu_{singleton}$ (x; 60)



- Gausiana

$$\mu_{Gausiana}(x;c,\sigma)=e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$$

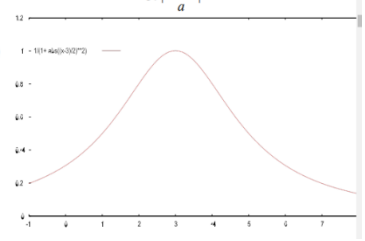
j: $\mu_{Gausiana}$ (x; 3, 1)



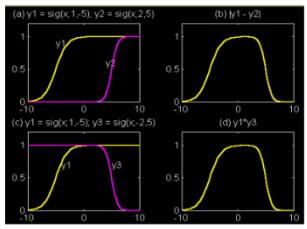
- Bell

$$\mu_{Bell}(x;a,b,c)=\frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}}$$

μ_{Bell} (x; 2, 1, 3)



- Sigmoide: $sigmf(x;a,b,c)=\frac{1}{1+e^{-a(x-c)}}$



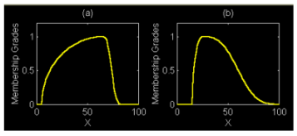
disp_sig

- LR:

$$LR(x;c,\alpha,\beta)=\begin{cases}F_L\left(\frac{c-x}{\alpha}\right), x<c\\ F_R\left(\frac{x-c}{\beta}\right), x\ge c\end{cases}$$

$F_L(x)=\sqrt{\max(0,1-x^2)}$ $F_R(x)=\exp(-|x|^3)$

c=65
a=60
b=10



c=25
a=10
b=40

difflr.m

Pueden tener mas de 2 dimensiones

Definiciones de sets difusos

-**Set difuso normal:** si hay un algún elemento que A(x) = 1

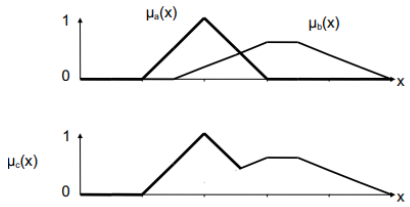
-**La altura** : de un set difuso A es el miembro mas grande en A

- **El soporte/support**: los sets que su valor de pertenencia sea >0

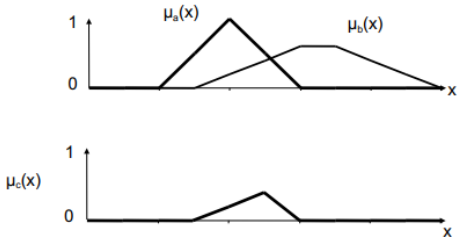
-**La medula/core :** los elementos con valor de pertenencia = 1

Operaciones

Unión: C=A+B, C(x) = Max[A(x), B(x)]



Intersección: C(x) = Min[A(x), B(x)]



Reglas IF-THEN difusas: si se cumple A entonces B

If presión es alta, then volumen es pequeño

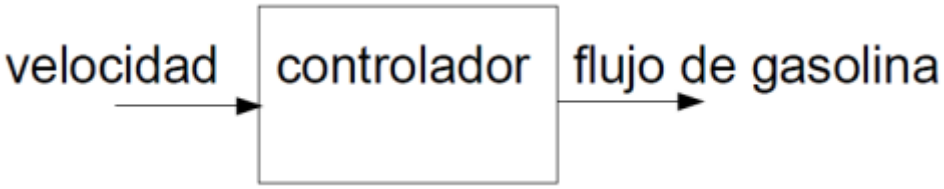
Inferencia usando lógica difusa:

El sistema de inferencia difuso consiste de tres componentes conceptuales:

– reglas difusas,

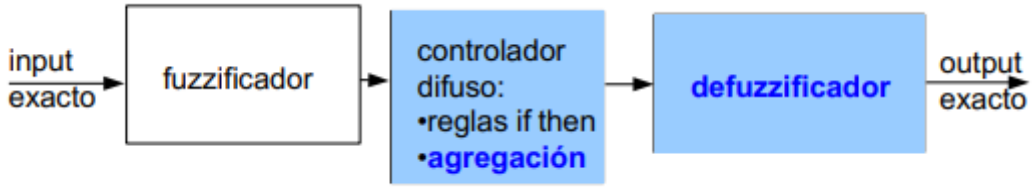
– diccionario (con funciones de pertenencia),

– mecanismo de raciocinio



Tipicamente los controladores se relacionan con el mundo externo a traves de valores exactos

Si el controlador usa logica difusa va a ser necesario alguna conversión usando un fuzzificacion y defuzzificacion



Modelo Mamdani FLC : método de control difuso ,defuzificacion usando el centroide

Modelo Sugeno: no es necesaria la defuzificacion, ya que cada regla tiene un output exacto

– If x is A and y is B then z = f(x, y)

TEMA 4.2

Redes neuronales

Una red neuronal artificiales : una herramienta diseñada para emular la forma en que el cerebro humano funciona.

Redes neuronales

• Propiedades:

-Formadas por conjuntos de neuronas individuales activadas por umbral

-Muchas conexiones con pesos entre las neuronas

-Procesamiento distribuido y paralelo

-El aprendizaje consistente en ajustar los pesos de las conexiones entre las neuronas

• Tipos de problemas adecuados

-Entradas formadas por vectores de valores discretos o reales

-Pueden ser problemas de clasificación o regresión

-Puede producir vectores de valores

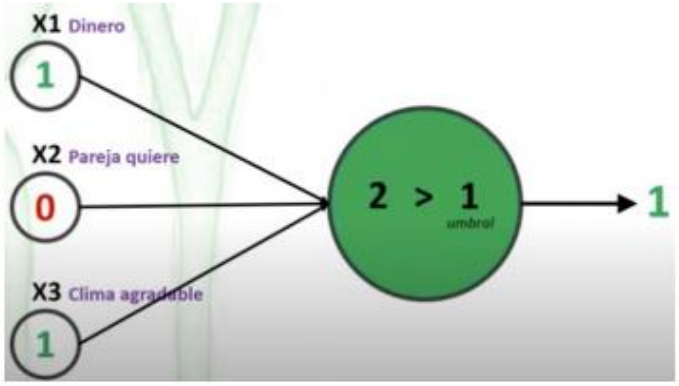
-Los usuarios finales no quieren obtener explicaciones: no producen conocimiento inteligible (caja negra)

-Los datos de entrenamiento pueden contener errores

4.2.2 PERCEPTRÓN

- Ejemplos de activación:
- Si tengo dinero y mi pareja quiere
- Si tengo dinero y el clima es agradable
- Si mi pareja quiere y el clima es agradable

- Los humanos no tomamos decisiones tan simples. Nosotros incorporamos importancia a ciertos factores, lo que en redes neuronales se conoce como peso.



La neurona se puede actvivar=1=ir de viaje o no

Varias neuronas conctadas entre si = conexiones , cada conexión tiene un valor un peso

Si tenemos sólo neuronas con respuestas lineales, la red ofrecerá respuestas lineales.

Debemos ser capaces de salir de la linealidad mediante lo que se conoce como la **función de activación**: a para limitar el rango de valores de la respuesta de la neurona(normalmente a [0,1] o [-1,1]).

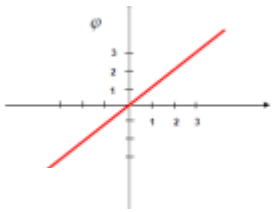
Funciones de activación

-**La función Umbral** respuesta 0 o 1

$$\varphi(x) = \begin{cases} 1 & si \quad x \geq 0 \\ 0 & si \quad x < 0 \end{cases}$$

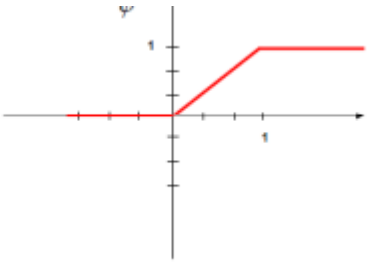
-**La función Lineal** : No limita la respuesta de la neurona

$$\varphi(x) = x \qquad x \in R$$



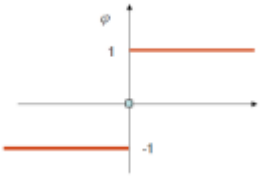
-**La función lineal acotada**

$$\varphi(x) = \begin{cases} 0 & si \ x \leq 0 \\ x & si \ 0 \leq x \leq 1 \\ 1 & si \ x \geq 1 \end{cases}$$



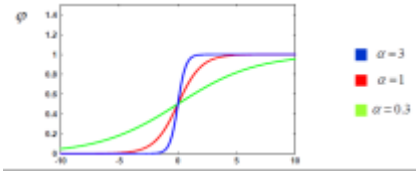
-**Función signo**

$$\varphi(x) = \begin{cases} -1 & si \quad x < 0 \\ 0 & si \quad x = 0 \\ 1 & si \quad x > 0 \end{cases}$$



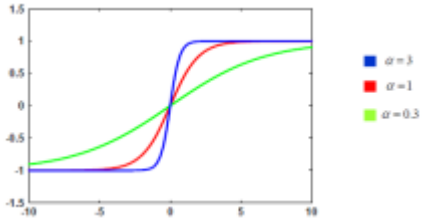
-**Función logística**: Es la versión continua de la función umbral y se utiliza en problemas de aproximación

$$\varphi(x) = \frac{1}{1 + \exp(-\alpha x)}$$



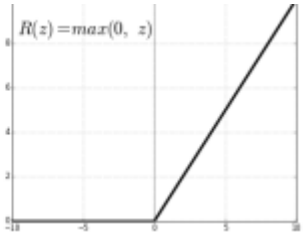
-**La función tangente hiperbólica**: version continua de signo , buenas propiedades analíticas

$$\varphi(x) = \tanh(x/2) = \frac{1 - \exp(-\alpha x)}{1 + \exp(-\alpha x)}$$



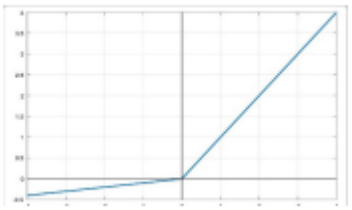
La función ReLU :transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran. Para redes convuncionales .De aprendizaje rápido pero genera neuronas muertas que son irrecuperables

$$f(x) = \max(0, x) = \begin{cases} 0 \ for \ x < 0 \\ x \ for \ x \geq 0 \end{cases}$$

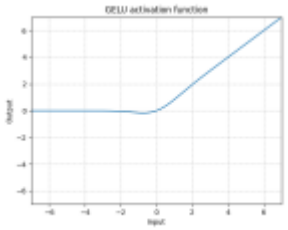


-La función Parametric ReLU para redes mas grandes mejora ReLU

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ a \cdot x\ for\ x \geq 0 \end{cases}$$



Función no nomotónica GELU: $GELU(x) = x * \varphi(x)$



Funcion Swish:aumentar la precisión de clasificación de ReLU ,mas coste

Funcion Mish :mejora lo anterior

Softmax para redes de clasificación.

Las funciones oscilatorias. Aún en experimentación.

Aprendizaje de una red

Aprendizaje: la red modifica los pesos de las conexiones para que las salidas de la red se vayan adaptando de manera paulatina al funcionamiento que se considera correcto

La modificación de los pesos depende del paradigma de aprendizaje que estemos usando:

- Aprendizaje supervisado: para cada ejemplo presentado a la red existe una respuesta deseada. La respuesta de la red se compara con su salida esperada, y en base a esa comparación se ajustan los pesos de la red
- Aprendizaje no supervisado: no se especifica a la red cuál es la respuesta correcta. La red descubre las relaciones presentes en los ejemplos

Separador lineal : da un conjunto de valores poder separarlos en 2 clases, si el conjunto no se puede separar en dos usar **redes multicapa**

APRENDIZAJE PERCEPTRÓN

Si dos neuronas están activas, su conexión se refuerza, actualizando los pesos con cada ejemplo.

Controlando el ritmo de aprendizaje ,decreciendo con el tiempo

DESCENSO GRADIENTE

Base de Backpropagation

Entrenamiento: minimizar el error cuadrático, moviéndonos en la dirección que lo reduzca

Buscar en espacios de hipótesis infinitos , puede ser lento no garantiza el mínimo global por los locales

Descenso de gradiente incremental : es mas rápido

REDES MULTICAPA MLP

Para representar funciones no lineales , entrenamiento lento

BACKPROPAGATION

grafos dirigidos,mínimo local,entrenamiento lento

TEMA 4.3

REDES NEURONALES CONVUNCIONALES

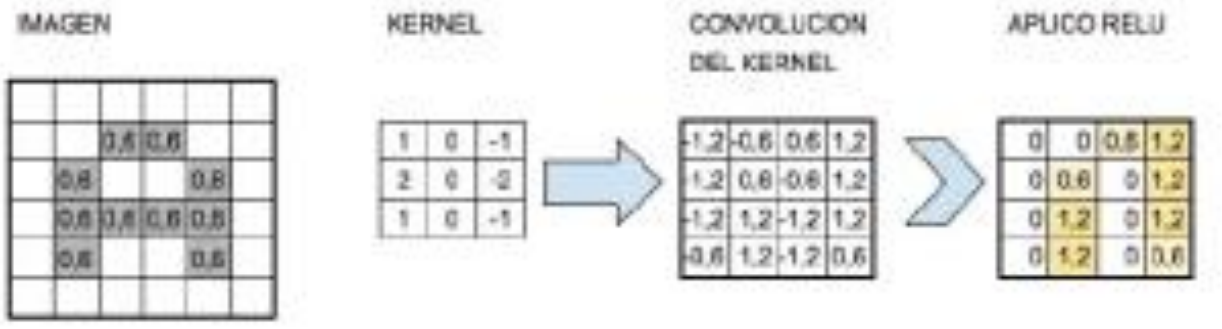
Variación de un perceptrón multicapa, usando matrices bidimensionales para tareas de visión artificial, como en la clasificación y segmentación de imágenes.

Las redes neuronales convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones

Primero deben ser entrenadas mas de 10.000 muestras y ya podrá reconocer imágenes y clasificaralas

La red tomo como entrada los pixeles de la imagen 20x20 por 1(si fuera a color=3)=1200 neuronas

Tomaremos una matriz llamada **kernel** (echo por backpropagation)y haremos el producto escalar , genereando capas ocultas (esto se hace varias veces). Al conjunto de kernel los llamaremos filtros

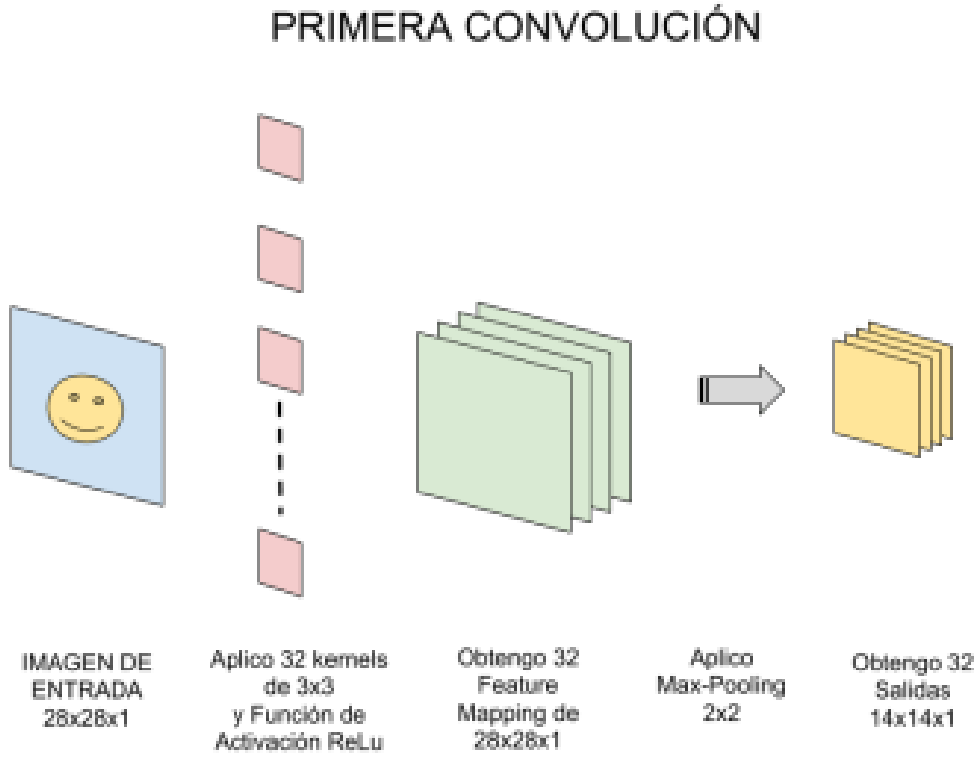


FINALMENTE OBTENGO UN MAPA DE DETECCIÓN DE CARACTERÍSTICAS

La función de activación es ReLu

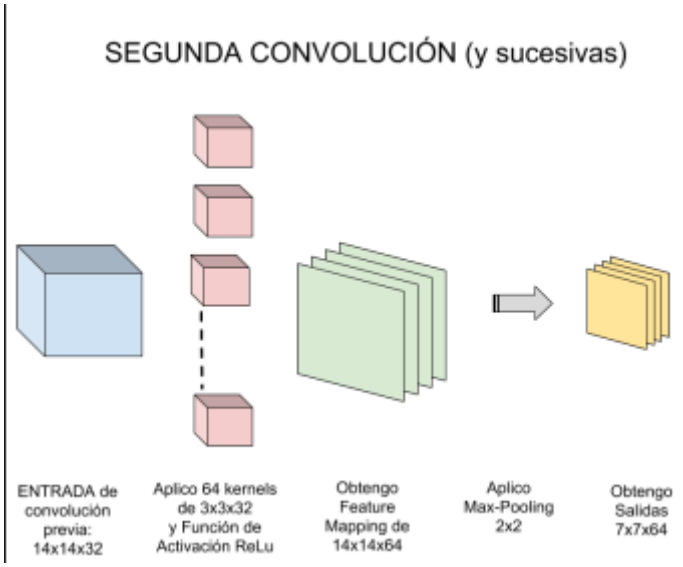
Muestreo (subsampling)

Luego cogemos una muestra de las neuronas mas representativas(mas importantes) para hacer una nueva convulsión y así reducimos el coste computacional (se suele hacer Max-Pooling y se reduce a la mitad)



La primer convolución es capaz de detectar características primitivas como lineas o curvas.

Cuantp mas convulsiones mas formas reconocera



A la 3º convolución cogemos la ultima capa oculta y la aplanamos (transformamos en red neuonal) y le aplicamos Softmax que conecta a la salida con el nº de neuronas como clasificaciones allá gatos y perros 2 neuronas , la salida será en forma de matriz [0,2 0,8] hay un 20% de prob de que sea gato 80% perro