Teoría Inteligencia Artificial

Blanca Cano Camarero

19 de junio de 2020

Índice

1. Agentes

Concepto de Agente. Agentes racionales comparados con agentes inteligentes. Arquitecturas de agentes.

1.1. Concepto agente

Un agente es cualquier cosa capaz de percibir su medio ambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores (Stuar Russell,Inteligencia Artificial. Un enfoque moderno, 2011, p. 37). Entendiendo por actuador al elemento que reacciona a un estímulo. mediante una acción. La secuencia de percepciones de un agente refleja el historial completo de lo que el agente ha recibido. Un agente tomará una decisión en un momento dado dependiendo de la secuencia completa o de una parte de ella. En términos matemáticos se puede decir que el comportamiento del agente viene dado por la función del agente que a cada percepción le asocia una acción. Un agente que cumpla lo anterior será autónomo (no necesita al humano) y proactivo (toma sus propias decisiones). Además, es deseable que sea social para servirse de otros agentes para lograr su objetivo.

Las características indispensables de los agentes son: la capacidad estímulo respuesta, un corpontamiento proactivo y autonomía. Otra caracterísitica posible sería la de ser social, es decir interactuar con otros agentes o humanos.

1.2. Agentes inteligentes y agentes racionales

Se **consideran inteligentes** a aquellos agentes capaces de superar el test de Turing, es decir si consigue aparentar ser un humano para ciertas pruebas.

Los **agentes racionales** son aquellos que buscan el comportamiento mejor dentro de las opciones de las que disponen.

1.3. Arquitecturas de los agentes

Russel define un agente como la arquitectura que tiene más el programa que utiliza.

La arquitectura coordina el manejo de la información obtenida por los sensores, ejecución de los programas y actuadores.

Clasificación de arquitecturas por su topología

- **Arquitectura horizontal.** Todas las capas que constituyen al agente con capaces de percibir el estímulo del entorno y realizar acciones.
- **Arquitectura vertical.** Una capa percibe los estímulos del entorno, transmite la información a otras capas *superiores* y es la última la que realiza las acciones.
- **Arquitectura híbrida.** Se comporta como la arquitectura vertical ya que todas las capas forman una cadena recibiendo y transmitiendo la información a adistinto niveles, pero es la capa que percibe el entorno la que se encarga de realizar las acciones.

Clasificación de arquitecturas por su abstracción.

- **Arquitecturas deliberativas.** Comprenden un sistema simbólico de la realidad, hipótesis sobre estos y un agente deliberativo cuyas decisiones se realizan a través de un razonamiento lógico basado en emparejamiento de patrones y manipulaciones simbólicas. Suelen estar diseñados con una estructura vertical y deben tener en cuenta el tiempo de respuesta y manejo de símbolos apropiado.
- **Arquitecturas reactiva.** Aquellas que no incluyen ninguna clase de modelo centralizaod de representación simbólica del mundo, y no hace uso del razonamiento complejo; se construyen bajo esquemas de percepción y acción, con lo que se busca una respuesta rápida y eficaz. Aunque existen agentes reactivos con memoria con el fin de conseguir retroalimentación.
- **Arquitecturas híbridas.** Combinan arquitecturas deliberativas y reactivas, un paradigma recomendado para la respuesta a problemas no previstos durante el desarrollo del plan.

2. Características de los Agentes reactivos y deliberativos.

Características de los Agentes reactivos y deliberativos. Similitudes y diferencias. Arquitecturas.

2.1. Agentes reactivos

Son aquellos agentes que siguen un ciclo acción-reacción y no poseen ningún modelo del mundo, aunque si posee memoria puede ir construyéndolo. Su comportamiento está basado en reaccionar a sensores, la cual puede modifificarse si existe retroalimentación, por información almacenada en memoria.

Es necesario para su buen funcionamiento que el diseño abarque todos los estados posibles. Sus características principales es que realizan pocos cálculos, almacenan todo en memoria y usan arquitecturas horizontales.

Un ejemplo de estrategias de diseño es:

- **Arquitectura de subsunción:** Consiste en agrupar comportamientos en módulos y dada una percepción, uno de los comportamientos se activará. (Pensar en un switch).
- **Agentes reactivos con memoria.** Mejorar la precisión de respuesta gracias a la memoria de estímulos recibidos. Un ejemplo de este es la implementación basada en pizarras, es decir una memoria común a todos los programas del agente (módulos de conocimiento).

2.2. Agentes deliberativos

Los agentes deliberativos son aquellos que tienen un modelo del mundo y elboran un plan según los efectos de su acciones sobre esos símbolos físicos. Sus decisiones se realizan sobre esos símbolos físicos, a partir de las hipótesis que poseen, no reaccionan a los eventos. Usan arquitecturas verticales.

A la hora de diseñar, la abstracción de la realidad para desarrollar el modelo o estados del agente, es donde reside la dificultad, ,es más, si se usa una arquitectura deliberativa pura y el sistema lógico no es completo podría no alcanzar nunca una solución (((((y si es completo no será consistente xD))))))).

Conclusión

Las diferencia esenciales son.

agentes reacitvos:

- Diseño explícito y completo de situaciones posibles.
- Pocos cálculos.
- Almacenan todo en memoria.
- Arquitecturas horzontales.

Los agentes deliberativos:

- Razonamiento lógico sobre modelos (conocen el mundo).
- Usan arquitecturas verticales.

3. Métodos de búsqueda no informada

Describir brevemente los métodos de búsqueda no informada.

3.1. Búsqueda primero en profundidad

En este método se expande el nodo más profundo de la frotera, es decir el último sucesor que se generó.

Se implementa como es natural con una pila. No asegura la optimalidad.

De él aparecen variantes como backtracking y descenso iterativo que comentaremos más adelante.

3.2. Búsqueda primero en anchura

Se exploran las posibles soluciones de generación en generación, es decir partir de un estado inicial, se analizan todos sus nodos sucesores, generando con ellos un nivel; de estos hijos sus sucesores conformarán un nuevo nivel.

No se pasa a explorar un nivel sin haber explorado antes todos los anteriores.

La solución es óptima si lor operadores de búsqueda tienen el mismo coste. La implementación más común es utilizar una cola de nodos abierto, es decir donde ir metiendo los nuevos nodos sucesores.

3.3. Búsqueda de coste uniforme

Es una variante del método anterior, el criterio de selección no es el de estar en un nivel más cercano a la raíz, si no que dada un función coste, el nodo que tenga menor coste.

Solo se tiene información de los nodos de la frontera, lo que hace que sea una búsqueda no informada.

3.4. Backtracking

Aparece un profundidad límite en la que seguir explorando; una vez alcanza ese límite se comienza a explorar no los sucesores a este si no los anteriores.

3.5. Descenso iterativo

Combina las ventajas de profundidad iterativa y anchura.

La idea es la siguiente, si cada nivel que generábamos con el método en anchura se correspondería a una búsqueda en profundidad con un límite de uno; es decir una línea de sucesores de un nodo, ahora el límite de profundidad a explorar aumenta progresivamente a más niveles generados.

3.6. Búsqueda bidereccional

Se ejecutan dos búsquedas simultáneas, una parte de la raíz o estado inicial y otra del estado al que queremos decir, es decir, haciendo una búsqueda hacia atrás del objetivo.

Se dará por finalizada cuando ambas alcancen el mismo nodo; se encuentren en el centro.

Esta búsqueda es completa y óptima.

4. El concepto de heurística. Como se construyen las heurísticas. Uso de las heurísticas en IA.

4.1. Concepto de heurística

La exploración exhaustiva es inabarcable por tanto surge la necesidad de métodos discriminantes de qué métodos explorar. Una heurística es una función que estima lo*prometedor* que un nodo puede llegar a ser.

4.2. Cómo construir una heurística

Debemos tener en cuenta la descripción de los nodos; el historial, los nodos visitados hasta ese punto de la búsqueda y cualquier conociento que se tenga sobre el problema puede ser útil.

Además de considerar un equilibrio en el coste computacional de esta función.

Las heurística provienen de modelos simplificados del dominio denominados **problemas relajados.**

4.2.1. Ejemplos

En la práctica 2 de la asignatura, para estimar la distancia restante desde un punto a otro utilizar la distancia manhattan o la línea recta.

Y para estimar el gasto de batería tener una función peso.

Por lo general no garantizan la solución óptima, pero sí que tienen resultado satisfactorios.

(Faltaba: Justificar la necesidad de las variantes estocásticas para evitar estancamientos y caer en óptimos locales)

5. Los métodos de escalada. Caracterización general. Variantes.

Su idea reside en seleccionar una mejor solución obtenida por una heurística en un entonorno local al nodo en curso, y de esta manera ir recorriendo la soluciones emergentes hasta alcanzar un alcanzando óptimo que será local o global en algunos casos siempre que se demueste.

Entenderemos por vecindad a ese entorno local.

Partiendo pues de la búsqueda en profundidad y modificando el criterio de selección del nodo sucesor a partir de cierta *vecindad* surge esta familia de nodos.

Podríamos utilizar estos métodos en los que se prima el objetivo más que el desarrollo hasta alcanzarlo. Otra característica es el poco uso de memoria que requieren, ya que solo se guarda los nodos para los que se va a hallar las vecindad.

En conclusión tienen las siguientes características:

- Poco uso de memoria cuya cantidad puede incluso ser constanste o acotarse.
- Encuentra soluciones aunque el espacio sea infinito (o lo sucientemente grande para que sea descabellado una búsqueda exhaustiva).
- Informado: necesita conocer el estado para seleccionar un nodo.
- No completo: No siempre es óptimo global el resultad, esto se debe a que no hace una exploración exhaustiva.

Veamos ahora distintos métodos de búsqueda.

5.1. Algoritmo de escalada simple

La vecindad de un nodo es el conjunto de hijos obtenidos secuencialmente hasta que aparece uno mejor que el padre (según la función de evaluación).

El problema que tiene este algoritmo es que depende del orden en que se generen los hijos. Además si la solución es imposible explorará todo el árbol de posibilidades.

5.2. Escalada por máxima pendiente

A diferencia del anterior ahora la vecindad son TODOS (no los explorados hasta el primero mejor) los hijos obtenidos secuencialemte. Se selecciona el mejor que pasa a ser el nodo en curso y el proceso se repite hasta obtener una solución o que no sea posible encontrar un en la vecindad mejor que el padre, que entonces se para el algoritmo.

No es un algoritmo completo, es rápido y útil si la función es monótoma.

Pero con este y el anterior surgen el problema de encontrarnos máximo o mínimos locales atractivos que nos impidan encontrar la solución.

Luego para intentar sortearlo surgen los algoritmo con variaciones estocásticas que cada iteración recorren el árbol de búsqueda de una manera diferente; un problema de esto es que se desconoce cuántas repeticiones hacer.

5.3. Escalada estocástica

Los método estocásticos se usan para intetar salir de los máximo mínimos locales. Todos tienen el problema que no saben cómo parar.

Como no hay razón para priorizar un sucesor que otro se escoge entre los sucesores con mejor valoración que el estado actual.

5.4. Escalada de primera opción

Ahora el azar se encuentra en el orden de generar los sucesores y la elección es el primero mejor.

5.5. Escala con reinicio aleatorio

Se repite varias veces la búsqueda, partiendo cada vez de un estado inicial distinto generado aleatoriamente. Si la probabilida de éxito es p, por tratarse de una binomial, la esperanza de reinicios es $\frac{1}{p}$. Destaca por lo sorprendentemente efectivo que resulta.

5.6. Algoritmo de enfriamineto simulado o temple simulado

Toma la idea de los principios de la termodinámica. Para evitar que se finalice en máximos locales se seleccionan de forma controlada movimientos peores. Los pasos son los siguientes: Cada nodo tiene una probabilidad de ser elegido menor que uno, se selecciona uno aleatoriamente; si es mejor se toma como nodo actual, si no dependerá inversamente de lo cerca que estemos de

la solución y de lo malo que sea; es decir cuanto peor sea y más cerca nos encontremos del objetivo menos probabilidad habrá de seleccionar un caso localmente desfavorable.

El temple simulado termina cuando se ha alcanzado el objetivo-

Existen otras variante como los algoritmos genéticosy los algoritmo de haz local.

5.7. Algoritmo de haz local

El algoritmo guarda la pista de k estados (no solo de uno). Los primeros estados son generados aleatoriamente, si alguno es la solución se para; si no se toman los k mejores y se vuelve a repetir el proceso. (no confundir con el algoritmo de reinicio aleatorio).

Una variante es la llamada búsqueda de haz estocástica, donde los k nodos seleccionados no son los mejores sino aleatorios.

Veamos que este concepto nos va a ser de utilidad para introducir los algoritmos genéticos.

6. Características esenciales de los métodos "primero el mejor".

Son variantes del algoritmo de Dijkstra, en vez de generar todos los mejores caminos nos quedamos con el primero y nos paramos cuando lo hayamos encontrado.

Es decir, expandiremos el mejor de los nodos bajo hipótesis de que nos acercará al objetivo de manera más rápida.

A la hora de implementarlo se usan dos listas, una de nodos abiertos y otra de cerrados.

En la de abiertos mantenemso aquellos generados pero no explorados (de los que todavía no se han derivado sucesores), es una lista que se ordena en función del valor asignado por una heurística, lo que en c++ podría ser una cola con prioridad.

La lista de cerrados contienen los nodos ya visitados con el fin de no repetirlos.

El procedimiento es el siguiente:

1. Se crea el grafo de búsqueda G con un nodo inicial I.

2. Se analiza la lista de abiertos con I y la de cerrados vacía. Se inicializa también la variable de éxito a un valor False.

3. Mientras la lista de Abiertos no esté vacía o su éxito no se True:

   (a) Quitar el primer nodo de abiertos (N) con mayor prioridad en función de nuestra heurística y meterlo en cerrados.

   (b) Si N es el estado final entonces éxito es True.

   (c) Si no expandir N generando su conjunto de Sucesores S y añadiéndolo a la lista de abiertos siempre que no estén en cerrado y no se repitan (trabajar con conjuntos sería lo ideal).

   (d) Se genera un nodo en G para cada s de S, (tener en cuenta que estos nodos deben de registrar su antecesor de alguna manera o representar de alguna manera los nodos de los que proceden).

4. Si ÉXITO entonces la solución de el camino desde I a N a través de nuestros nodos.

5. Si no no se ha encontrado solución.

Este algorimo se refleja perfectamente en el costo uniforme de la práctica 2.

(detallar más el algoritmo A* gestión abiertos y cerrados más compleja que en primero mejor por la forma de f(n))

7. Elementos esenciales del algoritmo A*

El algoritmo A^* es un método del tipo "Primero mejor"donde la función de evaluación es la suma de dos, una basada en datos conocidos del nodo, un historial de él y una heurística.

Descripción del algoritmo

- $g(n)$ la distancia del camino obtenido hasta el momento (desde el nodo inicio hata el nodo enésimo n).
- $h(n)$ es la heurística y expresa la distancia estimada desde el nodo n hasta el objetivo.
- Se impone la restricción de que la heurística no sobreestime el coste de llegar al destino, ya que esto causará peor exactitud.
- A la hora de implementarlo existen dos listas, una de cerrados y otra de abiertos.
- La lista de abierto contiene nodos que aún no se han expandido ordenados según f.
- La lista de cerrados contienen los nodos ya expandidos.
- En cada paso se selecciona el nodo más prometedor (calculado a partir de f), se inserta en cerrados es el objetivo se para la búsqueda sino se generan sus sucesores que se incluyen en abiertos.

Es decir si nos encontramos buscando el camino más corto, la primera función $g(n)$ nos dirá la distancia exacta que ha recorrido el nodo hasta entonces mientras que la función heurística $h(n)$ estimará la distancia (por ejemplo con la distancia euclídea o la Manhattan) y nuestra función de evaluación $f(n) = g(n) + h(n)$ expresará la distancia estimada de la ruta construída por el nodo n.

Respecto al diseño es algo más complejo que en los algoritmos de primero mejor, dos listas de nodos, abiertos y cerrados, donde en abiertos están los candidatos generados de nodos anteriores y de los que se va extrayendo el más prometedor, añadiéndolo a la lista de cerrados, es decir nodos ya visitados

y si esta no es la solución generando nuevos candidatos sin repetir los ya generados.

7.1. Propiedas

Llamemos $h^*(n)$ al coste de camino óptimo en cualquier nodo n y al coste del camino óptimo como $f^*(n) = g(n) + h^*(n)$ entonces lo ideal sería que si nuestra función heurística h fuera igual que h^*.

Sea pues a la hora de construir h tendremos las siguientes consideraciones:

- siempre que $h(x) > 0$.
- Una heurística se considera admisible si $h(n) \leq h^*(n)$ para todo nodo n.
- Si tenemos dos heurísticas admisible h_1, h_2 será mejor aquella h_2 que para todo nodo n cumpla que $h_1(n) \leq h_2(n) \leq h^*(n)$ ya que será mejor estimador.
- Será consistente la heurística si dados el nodo x y su sucesor y, $h(x) - h(y) \leq g(y) - g(x)$ es decir el gasto que produce el camino entre x, y es mayor o igual que lo que consumiría el trayecto estimado.

  Si se cumple la condición de consistencia entonces f es no decreciente y esto implicaría que que cada vez que se expande un nodo, se encuentra un camino que mejora al mismo.

Como consideraciones finales, nótese que si h es la función constantemente cero, entonces el algoritmo es idéntico al de Dijsktra. Una de las limitaciones que tenemos con este algoritmo es el costo en memoria, ya que requiere almacenar los nodos que se van generando y almacenar de algún modo esa jerarquía de generación.

Por otro lado encontrar una heurística adminsible y consistente no es un proceso sistematizado y puede dar problemas.

8. Elementos esenciales de un algoritmo genético

Los algoritmos genéticos son métodos adaptativos o sistemáticos que pueden usarse para resolver problemas de búsqueda y optimización, aplican estrategias biológicas basadas en la reproducción sexual y adaptativa.

Es además una variande de la búsqueda de haz estocástica, en la que los estados sucesores se generan combinando dos estados "padres". Introduciremos los siguientes tecnicismo:

- Población o cromosoma: Conjunto de k estados generados aleatoriamente y que representan soluciones.
- Cromosoma: Representa solución al problema, puede ser expresado como vectro binario. Conjunto que recoge las caractiristica, variables o atributos del vector que representa una solución; es decir, un elemento del cormosoma, Individuo o gen: Cada estado, representado por una cadena finita. lo más común es que sea binaria.
- Selección natural, reproducción sexual: Operador de cruce o selección, el encargado de ver cómo se cruzan los cromosomas.
- Mutación: Operador de modificación.
- Cambio generacional: Operador de reemplazamiento (cómo se reemplaza la función).

El algoritmo se compone de los siguientes paso:

1. Inicialización: Se genera aleatoriamente una población inicial; cromosomas que representan posibles soluciones.
2. Evaluación: A cada cromosoma de la población se le aplica una función de idoneidad o de evaluación, que a mejor solución devuelve resultados más grandes.
3. Cruce: se selecciona dos pares de manera aleatoria (u otro criterio) para la reproducción. Hay muchas variantes para esta regla de selección, se ha demostrado que un método selectivo hace que converja a la solución más rápido que la versión aleatoria.
4. Mutación: Cada posición está sugeta a una mutación aleatoria con una probabilidad independiente. Como la búsqueda por haz estocástica se combinan tendencia ascendente aleatoria que cambia la información entre hijos paralelos.

5. Esquema: Se puede trabar dejando posiciones inespecíficas.

El problema que tiene esta metodología es decidir con cuántas iteraciones parar.

9. Componentes de un juego

9.1. Definición de juego

Un juego es cualquier situación de decisión de varios agentes, (jugadores) gobernada por un conjunto de reglas y con un resultado bien definido, caracterizada porque ninguno de los jugadores con su sola acutación puede determinal el resutlado (independecia estratégica).

Es decir, nos encontramos en un entorno multiagente, donde la imprevisibilidad de estos introducen contingencias. En IA los juego son por lo general los de suma cero (la ganancio o pérdida de un jugador se equilibra con la del otro) , de dos jugadores, por turnos, deterministias, de información perfecta.

9.1.1. Definición formal de juego

Clase de problemas de búsqueda con los siguientes componentes.

- **Estado inicial**, posición tablero y el jugador que mueve.
- **función sucesor** lista de pares de (*movimiento,estado*), con todos los movimientos legales y su estado resultante.
- **test terminal**, determina si un nodo es terminal, es decir si se termina el juego en él.
- **Función utilidad, objetivo o rentabilidad**; da valor numérico a nodos terminales.

Se denomina **árbol de juego** a aquel cuya raíz es el estado inicial y sus sucesivos hijos los movimientos legales.

9.2. Las compentes de un juego

- **Número de jugadores.**
- Si es **de información perfecta**, toda la información del juego es conocida por todos los jugadores (vg ajedrez). o por el contrario **de información imperfecta** se oculta la información parcialmente (póker).
- **Determinista o no**, si el azar influye (la oca).
- Si es **Por turnos**
- Existencia o no de pagos colaterales. Que la acción de un jugador, beneficie a otro llegando así a un equilibrio de Nash.

- Juegos de **suma nula** el beneficio total para todo los jugadores suma 0, en contraposición de **suma no nula**, donde la ganancia de un jugador no necesariamente significa la pérdida del otro.

9.3. Equilibrio de Nash

(Wikipedia principal fuente, lo he ampliado porque me interesaba)

Es un concepto de solución para juegos multijugadores del cual se toma como hipótesis que todo jugador a adoptado su mejor estrategia y todos conocen la estrategia de los otros.

De aquí deducimos de que tras esto no beneficia cambiar de estrategia ya que el resto no la va a alterar y en tal cicunstancia ya teniamos la mejor.

Además el resultado maximiza el beneficio individual no el de conjunto, lo cual podría dar a que si todos coordinaran este fuera mejor para todos (en un juego de dos jugadores esto no tiene sentido, pero sí en .ªlianzas").

Un ejemplo sería el juego del dilema del prisionero, pero introduciendo más jugadores, donde la mejor estrategia individual sería declarar ya que minimiza la pérdida, sin embargo si todos cooperaran no confesando el beneficio global sería mayor.

(minimix juegos bipersonales de suma nula e información perfecta, 22-23 tema 4) casos representar todo árbol de juegos. y los que no 33-40 tema 4.

10. Ramificación. Algoritmo minimax. Poda alfa-beta

Qué es el factor de ramificación y cómo afecta a la complejidad de un juego? Describe en líneas generales el algoritmo minimax y el de la poda alfa-beta (Capítulo 6 del Russell Norvig)

10.1. Definición factor de ramificación

Dada una estructura de árbol el **factor de ramificación** es el número medio de hijos en cada nodo. Que esto en los juegos se manifiesta como el número de movimientos posibles por turno.

Cuanto mayor sea el factor de ramificación, mayor será la complejidad a la hora de buscar estrategias, ya que llegado cierto punto es imposible una búsqueda exhaustiva del árbol de juego para alcanzar la mejor solución.

En ajedrez el factor de remificación es 35, que por combinatoria en m movimientos se habrá habido una media de 35^m posibilidades. En el caso del Go el factor de ramificación es mayor, y este es el motivo por el que solo se ha conseguido una IA que juegue como un aficionado.

De aquí surge la necesitadad de estrategias de poda de nodos, pero antes veamos cómo sería una estrategia óptima.

10.2. Algoritmo minimax

Hipótesis o características en las que se basa:

- Juego por turnos de dos jugadores.
- Con información perfecta, es decir se conoce con totalidad la situación y posibilidades del oponente.
- De suma nula, el beneficio para un jugador es la pérdida del otro.

Dado un juego por turnos, de dos jugadores MIN y MAX; bajo hipótesis de que el oponente, MIN juegue perfecto; el algoritmo minimax nos provee de una estrategia óptima.

Esta estrategia se calcula en función al **valor minimax**, de cada nodo, que se obtiene de la siguiente manera:

- Si estamos en un nodo MAX, tomar el máximo de sus nodos hijos.
- Si estamos en nodo MIX, tomar el mínimo de los nodos hijos.

Esta idea proviene de que ambos juegan perfecto y elegiran la opción más beneficiosa.

Cada **capa** (nivel del árbol de juego) correspone a un turno de uno de los jugadores, que se va a alternan enter MIN, MAX. La idea es que MIN busca la jugada más beneficiosa para él; y como juega perfecto cogerá el valor mínimo (aquel que maximiza su beneficio y minimiza el de MAX) de cada nodo, mientras que MAX actuará de manera contraria, a esto se conoce como **decisión minimax.**

El **algoritmo minimax** por tanto es la implementación de tal exploración primero en profundidad completa de un árbol de juegos para sacar el valor minimax y después su selección según criterio de ir cogiendo máximo y mínimo alternadamente.

Código en haskell (En mi repositorio puede encontrarlo https://github.com/BlancaCC/haskell/
)

```
data Tree a = Void | Node a [ Tree a] deriving Show
-- vm es un la función valor minimax que es del tipo vm :: Ord(b) => a -> b

action :: Tree a -> [a]
action Void _ = []
action t = action t maximun
action n nsucc f = [ ns ] <> action ns maximun $ map (-) -- de esta manera si
  where  ns = head [ns | ns <- nsucc, (vm ns) == f $ map vm nsucc ]
```

La complejidad en tiempo del algoritmo minimax es $\mathcal{O}(b^m)$ donde m es la profundidad del árbol y b el número de movimientos legales. La complejidad en espacio será $\mathcal{O}(bm)$ si el algoritmo general todos los sucesores o $\mathcal{O}(b)$ si solo gnener los de un nodo.

Por tanto con un orden de tiempo exponencial, es poco práctico.

Árboles y juegos que se puedan resolver

No tods los juegos se pueden resolver. Los que tienen árboles compeljos resultan imposibles de explorar en su totalidad. Por tanto, debemos distinguir entre juegos en los que si podemos recorrer el árbol de manera completa y en los que no. En estos últimos son cruciales las heurísticas y la poda.

Juegos multijugadores óptimos

Esta misma idea se puede extrpolar a juegos de más de dos jugadores, donde ahora el valor de cada nodo sería un vector con un vector de valores. El problema a la hora de analizarlo son las alianzas, ¿cómo debemos de entenderlas? Por ello la complejidad de estas situaciones es mucho mayor.

10.3. Poda alfa-beta

La idea de este algoritmo probiene a que podemos devolver la misma solución que el minimax sin analizar todas las ramas, es decir **podando** las que tengamos la certeza de que la solución no es esa.

Ejemplifiquémoslo en con el siguiente caso concreto de un juego de dos capas:

```
MINIMAX-VALUE(raíz) = max(min(3,12,8), min(2,x,y), min(14,5,2))
                    = max (3, min(2,x,y),2)
                    = max (3,z,2) donde z menor o igual que 2
                    = 3
```

Se puede observar que independientemente del valor de x,y si este es menor o igual que 2 será ignorado en la función max aunque paseen e filtro de la min y si son mayores que 2 serám ignorados por la función min.

Por tanto no es necesario que los calculemos.

Descrición del algoritmo de la poda alpha-beta

Se definen las siguientes variables:

- α que recogerá el valor de la mejor opción encontrado para MAX, en cierto instante de la búsqueda. (Se inicia a menos infinito).
- β la mejor opción para MIN (el valor más bajo), se inicializa a más infinito.

Cuando un nodo es peor, es decir menor que el α para MAX y mayor que el β para MIN, podemos ignorar ese nodo, ya que no nos conducirá a un camino óptimo.

Eficiencia

La eficiencia sigue siendo exponencial. Además depemderá muchísimo de cómo se generen los sucesores, siendo el peorde los casos en el que aparecen primero los peores nodos; lo cual podría sugerirnos otro algoritmo y que guíado por cierta heurística escoja en orden los nodos más prometedores. Si se diera este caso la eficiencia sería $\mathcal{O}(b^{d/2)})$

11. Problemas que plantea el cálculo de predicados en la resolución de problemas de IA

3.- ¿Que problemas plantea el cálculo de predicados en la resolución de problemas de IA?

Problemas semánticos

- **Dificultad de expresión** de la heurísticas, metaconocimiento, jerarquía, herecia,igualdad, sentido común, etc mediante el el cálculo de predicados.
- Imposibilidad de una representación fiel del tiempo y por ende de un predicado que involucre variables temporales exactas; ya que físicamente se representaría con una variable continua (y siendos estrictos de cardinalidad infinito no numerable) lo cual obliga a su discretización.
- Difícil razonamiento a cerca de predicados, por la correspondencia biunivoca entre objetos del mundo... con que se interprestan los predicados esto dificulta
- Información incompleta y/o imprecisa (vaguedad, probabilidad)
- Excepciones difífil de abstraer una representación.
- La información puede ser incompleta o imprecisa o estar sujeta a la probabilidad. Luego aunque sepamos las leyes por las que se rige, la lógica tradicional nos impide o dificulta una representación en el cálculo de sus predicados. (Interesante pues la lógica difusa).
- Monotonia, es decir si una secuencia de fórmulas implica p, entontes sea tal secuencia unión otra fórmula que también lo implique.

Problemas computacionales

- Consistencia: Dos proposiciones opuestas son demostrables, dando lugar a que aquello que se muestra como verdadero no lo sea. Proviene en parte de que los predicados sean monótomos. Un ejemplo son las paradojas.
- Completitud: Existencia de proposiciones indemostrables. El cálculo de predicados es por ende semidecidible.

- Complejidad computacional: Lo algoritmos de demostración tienen un orden exponenciales, son computacionalmente complejos.

(Nota: Teoremas de Gödel: todo sistema axiomático no puede ser consistente ni completo al mismo tiempo).

12. Modelos de conocimiento heredable ¿Qué tipo de conocimiento organizan las redes semánticas? Describir en líneas generales el concepto de "frame".

Los modelos de conocimiento heredable asientan sus raíces en la psicología, en modelos de cómo las personas almacenamos la información.

Comencemos con las definiciones básicas:

Redes asociativas

Grafos donde cada nodo representa un concepto o proposición y los enlaces relaciones o categorías gramaticales.

Podemos encontrar las siguientes variedades según su objetivo:

Redes de clasificación aquellas empleadas para clasificar objeto por características; **redes causales** las que llevan asociada una relación de influencia, por los enlace; **y las redes semánticas** destinada y comprender el lenguaje natural y en las cuales vamos a profundizar un poco más.

Redes semánticas

La redes semánticas son aquellas destinadas a representar o a comprender el lenguaje natural. son un tipo de redes asociativas. Se representan con un grafo dirigido y etiquetado. Los nodos representas conceptos y los arcos relaciones entre ellos.

Bajo una visión de programador podemos entenderlo como clases y las respectivas relaciones entre objetosde la programación orientada a objetos, por lo tanto nos resulta natural que aparezca la herencia también en estos mecanimos de razonamiento.

Es decir un nodo o concepto hereda las propiedades de otro conceptos (nodos) de la jerarquía superior.

Uno de los problemas que tienen es que a la hora de tratar con sistemas complejos son difíciles de manejar.

Y este punto a mejora da lugar a la idea de sistemas basado en frames.

12.1. Sistemas basados en frames

El conocimiento concierne a individuos o clases de individuos y sus relaciones. y es almacenada en un **frame o marco** (objeto, unidad, concepto).

Un marco es una estrutura de datos que representan una situación estereotipada y posee atributos, **conjunto de campos o slot** selectivo que identifica los marcos.

Su principal tarea son las tareas de reconocimiento: dada una información esta activa marcos que a su vez activa a otros interconectados con los primeros dentro de una **red de activación**. Y esta dependencia es lo que antes habiamos denomidado herecencia o también conocido por reconocimiento descendiente.

Un inconveniente que tiene este mecanismo de propagación es que no se permite saber si cierto valor es heredado o explícito ni tampoco permite calcular valores de un atributo a partir de otro.

Para solucionar esto sugen las facetas.

12.1.1. Facetas

Una faceta es una propiedad asociada a un atributo, como por ejemplo su valor real, uno por defecto, cardinalidad ... o si es heredable o no.

13. Estructura y componentes de un sistema experto

Un sistema basado en conocimiento (SBC) es un programa que emplea masivamene el conocimiento para resolver un problema dentro de un dominio determinado.

Un sistem experto (SE) es un tipo de BSC se comporta como si se tratara de un profesional del ámbito. Es decir resuelve un problema y puede ser consultado de manera que justifique su razonamiento.

Necesita tres componentes básicas:

- Un conocimiento de base: que contenga el conocimiento necesario sobre el dominio a resolver.
- Un motor de inferencia, que permita razonar sobre el conocimiento de BC y los datos que proporcione el usuario.
- Interfaz de usuario: para la entrada y salida de datos de cada problema. A veces, los datos de un problema se conocen como Base de Hecho.

Tienen además implementados módulos para dar explicaciones y justificaciones sobre el porqué de las decisiones tomadas.

Para la representación de los formalismos se utiliza una tripleta (objetos, atributo, valor). Que se puede transformar en conocimiento útil mediante reglas de producción, redes asociativas o redes de marcos.

Pasos para construir un SBC

1. Extraer el conocimientos y extraer la forma abstracta y efectiva de representarlo.
2. Representarlo con el formalismo adecuado para que la máquina lo procese.
3. Realizar inferecias para generar información útil de él.

En el proceso de contrucción se extrae el conocimiento experto y se modela y procesa utilizando el lenguaje adecuado.

El esquema de diseño de un sistema experto basado en reglas contiene en memoria de trabajo la información relevante que el motor de inferencia utilizará para razonar las respuestas.

Algunos ejemplos de SBC son

- Anna, el agente conversacional del IKEA.
- Paciente Simulado Virtual para la formación de medicina.

14. Paradigmas de Aprendizaje Automático.

El aprendizaje automático es el campo de estudio o rama de la inteligencia artificial que tiene como objetivo que tiene como objetivo el desarrollo de técnicas que permitan modificar el mecanismo de decisión inicial de un agente en base a la experiencia.

Dicho de un modo más formal:

Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna tarea T existiendo alguna medida de rendimiento P, si su desempeño T; medidad por P, mejora la experiencia E.

Atendiendo a la definición, las distintas formas de mejorar la experiencia; de aprender son:

- **Aprendizaje memorístico** Introducción de un concepto o idea en memoria; es decir partir de una base de conocimiento y mejorarla. Es el aprendizaje clásico.
- **Aprendizaje deductivo** Se infiere el nuevo conociemiento a partir de reglas de deducción lógica; es un razonamiento artificial.
- **Aprendizaje analítico** Contruye una explicación para cada ejemplo en relación con un concepto dado y la generaliza para su futuro provecho. Vendría a ser *aprender a partir de ejemplos.*
- **Aprendizaje analógico** Consiste en entender y resolver una situación gracias a su parecido con otras situaciones anteriormente resueltas.
- **Aprendizaje inductivo** Consiste en aprender un concepto o clasificación a partir de ejemplos y contraejemplo. Es el más ampliamente estudiado y como buen ejemplos tenemos las redes neuronales.

El proceso de aprendizaje se basa pues en la realimentación y en función de qué tipo de conocimiento sea este podemos distinguir:

1. **Aprendizaje supervisado** Para cada entrada se dispone de un supervisor que porporciona la salida esperada; esta puede ser una clase, si estamos clasificando o un valor a aproximar, regresión. Es utilizado en los aprendizajes inductivos.
2. **Aprendizaje no supervisado** La agrupación se realiza en funcición de ciertas características, no se dispone de la salida deseada para cada entrada.

3. **Aprendzaje por refuerzo** Basado en el conductivismo; se aprende a partir de la información obtenida al realizar procesos de ensayo-error de los que se obtienen señales de benifico y coste. Es el más utilizado pero no por ello libre de problemas, ya que una retroalimentación negativa no implica que todo el conjutno de acciones pasada fuera erróneo. Es además supervisado en cierto sentido, pero no tiene porqué ser inductivo.

La herramientas destacables en el aprendizaje son:

- Técnicas estadísticas.
- Técnicas de árboles.
- Análisis clúster.
- Modelos bioinspirados: algoritmo genéticos, programación evolutiva, modelos conexionistas (redes neuronales).

15. Describir el problema del ruido y el del sobreajuste en aprendizaje automático.

En el aprendizaje inductivo el objetivo es encontrar una aplicación h tal que para cualquier par $(x, f(x))$ representando entrada y salida del conjunto de entrenamiento se tenga que $h = f$.

f estará **bien generalizada** si puede predecir ejemplos que no conozca.

Será **consistente** si satisface todos los datos.

Se utilizará el principio de la **Navaja de Ockham** a la hora de elegir funciones, es decir la más sencilla es más probable de que sea la cierta y por eso la tomaremos.

Finalmente un problema de aprendizaje es **realizable** si el espacio de hipótesis (el espacio de aplicaciones que intentan ajustarse al problema) existe aquella que lo hace de manera correcta.

Los problemas principales son los iguientes.

Ruido

Cuando se presentas dos o más ejemplos con la misma descripción y diferentes clasificaciones; aunque también puede ocurreir cuando los atributos no describen con suficiente detalle los datos o bajo un dominio no determinista.

Pensemos que en la matemática, para aplicaciones usuales, cada elemento del dominio tiene una sola imagen. Luego esto nos lleva a pensar que los datos no son correctos.

Como no podemos modificar el conjunto de datos, podemos pensar que una solución posible sería la *combinación* de esos valores conflictivos, en definitiva, realizar una interpolación. También se podría intentar clasificar los datos problemáticos en dos categorías distintas y ver cuáles producen mejores resultados quedándonos con estos.

Sobreajuste o overfitting

Es consecuencia de que h se aplique a partir de reglas poco significativas; es decir se aprenden hasta los errores. un ejemplo visual muy bueno resultado del reconocimiento sobre ajustado de imágnes es este https://www.nytimes.com/es/2017/08/17/es artificial-arte-musica-google-magenta-nsynth.html

Esto da lugar a que con el set de entrenamiento se consigan muy buenas soluciones, pero que falle en los test de prueba. Otra consecuencia es que h la función de ajuste necesite más información Las soluciones aumentar el tamaño del set de entrenamiento o bien añadiendo nuevos datos válidos

o por remuestro (redimensión, volteo de las imágenes previas); si estamos trabajando con árboles de decisión otra posibilidad sería podarlos impidiendo el desarrollo sobre atributos poco relevantes y que el ajuste sea perfecto a los datos de entrenamiento.

También se puede utilizar la validación cruzada. Que consiste en reservar datos para los test, y con esta reserva promediar los resultados o validar el aprendizaje.

16. Qué son y cómo se construyen los árboles de decisión

Los árboles de decisión, basados en el algoritmo de divide y vecerás; son un método de apredizaje inductivo. Toman como entrada un objeto o una situación descrita a través de un conjunto de atributos y devuelven una decisión. Esa decisión es el valor esperado para tal entrada.

Los tributos pueden ser discretos o continuos; y la salida discreta (clasificación), o continua (regresión).

Un árbol de decisión desarrolla una secuencia de test para poder alcanzar una decisión. Cada nodo interno se corresponde a un test sobre el valor de una de las propiedades y por ende las ramas en que se bifurca los posibles valores; es decir, cada hoja representa los posibles valores a devolver.

Forma de contrucción

Por aparatados anteriores podríamos pensar en inferir un árbol a partir de cualquier conjunto de entrenamiento, pero para generalizar no es una buena opción, luego se siguen las siguientes técnicas de inferencia:

- De manera trivial: Contruir un árbol de decisión cuyo camino esté basado en el ejemplo y la hoja sea la salida; es decir, el camino comprueba cada atributo y sigue el camino del ejemplo. El problema es que memoriza exactamente el problema pero no extrae ningún patrón aplicable a otras instancias.

- De manera óptima : En consonancia con idea de la navaja de Ockam se crea el árbol más pequeño posible consistente con todas las instancias(ejemplos). Esto no es computacionalemente viable por el coste.

- De manera pseudo-óptima: busca arreglar el problema anterior utilizando heurísticas más simples y así identificar árboles psudo-óptimos. La idea básica del algoritmo es hacer el test sobre el atributo más significativo, aquel que discrimina mayor número de ejemplos; y los ejemplos de entrenamiento que lleguen al nodo siguiente se utilizarán para elegir un nuevo atributo.