Teoria.pdf

Popa

Inteligencia Artificial I

3º Grado en Ingeniería Informática

Facultad de Informática
Universidad Complutense de Madrid

Inteligencia Artificial

Dos grandes enfoques:

- Simbólica: IA cognitiva y basada en conocimiento. Enfoque deductivo.
- Subsimbólica: Basada en datos. Conexionista. Enfoque inductivo.

IA Fuerte vs IA Débil:

- IA Débil: Sistemas aplicables a un tipo específico de problema, no tiene intención de mostrar inteligencia en general.
- IA Fuerte o Superinteligencia: Sistemas que igualan o exceden la inteligencia humana promedio. Sistemas que pueden realizar cualquier tarea intelectual (requiere creatividad, habilidades sociales, sentido común, ética, ...)

Representación de problemas

Se denomina problemas de planificación al proceso de búsqueda y articulación de una secuencia de acciones que permitan alcanzar un estado objetivo. Los problemas de optimización o configuración no importa el camino sino la solución.
El espacio de estados es un grafo dirigido de todos los estados alcanzables desde el estado inicial aplicando operadores válidos. Es importante recalcar que el espacio de estados no se construye entero. Según el paradigma del espacio de estados un problema se representa mediante:

- Estado inicial
- Operadores: Acciones que transforman un estado en otro.
- Comprobación de objetivo: determina si un estado es objetivo o no.
- Función de coste de camino: suma los costes de los operadores aplicados para llegar al estado objetivo desde el estado inicial.

El estado sólo incluye la información variable, la información fija del problema es una variable externa.

Búsqueda en el espacio de estados

El **espacio de búsqueda** sólo contiene los nodos generados en la búsqueda de la solución.
Aunque el espacio de estados sea finito, el espacio de búsqueda puede ser infinito por los posibles ciclos del grafo. Dos nodos distintos del árbol de búsqueda pueden referirse al mismo estado del espacio de estados.
Elementos del esquema general de búsqueda:

- **Nodos abiertos** (o frontera): Nodos pendientes de expandir.
- **Nodos cerrados**: Nodos ya expandidos.
- **Estados generados**: Aquellos que aparecen o han aparecido en nodos abiertos.
- **Estados generados pero no expandidos**: Aquellos que aparecen en la lista de nodos abiertos.
- **Estados expandidos**: Un estado se expande cuando un nodo que lo representa se quita de abiertos o se generan todos sus descendientes y estos se añaden a abiertos. Un estado puede ser abierto varias veces en diferentes nodos.

Esquema general de búsqueda Tree Search (sin control de repeticiones):

1. Coge el nodo inicial y lo mete en la lista frontera.
2. Saca el nodo de la frontera y genera hijos con la precondición.
3. Mete los hijos en la frontera y **descarta al padre**.
4. Elige el nodo frontera por el método de búsqueda asignada y genera hijos con la precondición.
5. Repite hasta encontrar el estado objetivo.

Esquema general de búsqueda Graph Search (con control de repeticiones):

1. Coge el nodo inicial y lo mete en la lista frontera.
2. Saca el nodo de la frontera y genera hijos con la precondición.
3. Mete los hijos en la frontera y **guarda al padre en explored-set**.
4. Elige el nodo frontera por el método de búsqueda asignada y genera hijos con la precondición, siendo estos diferentes a los que hay en explored-set.
5. Repite hasta encontrar el estado objetivo.

Búsqueda primero en anchura

Se explora el espacio de búsqueda haciendo un recorrido por niveles. Un nodo se visita sólo cuando todos sus predecesores y sus hermanos anteriores en orden de generación ya se han visitado. La estructura de frontera se implementa con una cola FIFO.

Es un algoritmo **completo**, su sistematicidad nos garantiza llegar al nivel donde se encuentra la solución. Si los operadores de búsqueda tienen coste uniforme la solución obtenida es **óptima**.

Complejidad en tiempo : $O(r^p)$ r = factor de ramificación p = profundidad mínima solución

Complejidad en espacio: $O(r^p)$

Búsqueda de coste uniforme

En cada paso, de los nodos en frontera se expande el nodo que tiene menor coste del camino hasta llegar a él. La frontera se implementa con una cola de prioridad. Si el coste del camino a un nodo coincide con su profundidad (o es proporcional) entonces es equivalente a primero en anchura.

Este algoritmo es **completo** si no existen caminos infinitos de coste finito. Es también **óptimo** si todos los operadores tienen coste ≥ 0.

Complejidad en tiempo : $O(r^p)$ r = factor de ramificación p = profundidad mínima solución

Complejidad en espacio: $O(r^p)$

Búsqueda primero en profundidad

Los nodos se expanden por orden inverso de generación (LIFO). Se extrae como nodo actual el último que entró en abiertos, después se expande el nodo actual y se quita de abiertos.

Este algoritmo **no es completo** ya que puede meterse en caminos infinitos. **Tampoco es óptimo** ya que puede haber soluciones mejores por otros caminos, no es recomendable si la máxima profundidad es grande.

Como ventaja necesita menos memoria y con "suerte" puede encontrar una solución sin tener que examinar gran parte del espacio de estados.

Complejidad en tiempo : $O(r^m)$ r = factor de ramificación m = profundidad máxima

Complejidad en espacio: $O(r * m)$

Búsqueda de profundidad limitada

Es la búsqueda en profundidad con límite L de profundidad para evitar descender indefinidamente por el mismo camino. El límite permite desechar caminos en los que se supone que no encontraremos un nodo objetivo óptimo.

El algoritmo es **completo sólo si L ≥ p** (profundidad mínima de la solución). **No es óptimo**, no puede garantizarse que la primera solución encontrada sea la mejor.

Complejidad en tiempo : $O(r^L)$ r = factor de ramificación L = límite de profundidad

Complejidad en espacio: $O(r * L)$

Búsqueda en profundidad iterativa

Es una aplicación iterativa del algoritmo de búsqueda de profundidad limitada. Combina las ventajas de primero en profundidad y primero en anchura. Suele ser el método no informado (ciego) preferido cuando el espacio de estados es grande y la profundidad de la solución se desconoce.

El algoritmo es **completo** y **óptimo**.

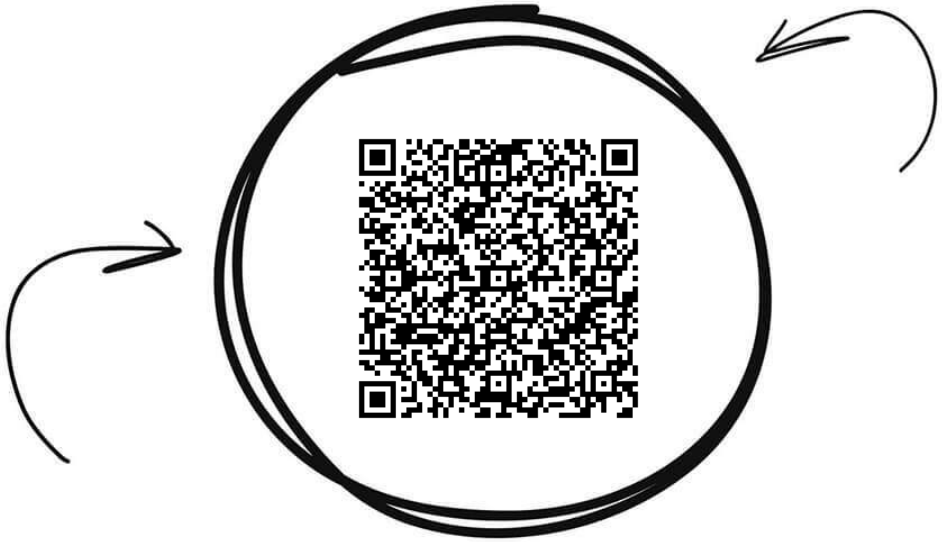
Complejidad en tiempo : $O(r^p)$ r = factor de ramificación p = profundidad mínima solución

Complejidad en espacio: $O(r * p)$

LA AUTOESCUELA EN TU UNIVERSIDAD

AUTOESCUELA EUROPEA

Inteligencia Artificial I



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

Búsqueda bidireccional
Ejecuta dos búsquedas simultáneas, una desde el estado inicial y la otra desde el objetivo. Es necesario conocer el estado objetivo, llegando a ser problemático si hay varios. Es necesario también poder obtener los predecesores de un estado.
El algoritmo es óptimo y completo si las búsquedas son en anchura y los costes iguales en todos los operadores.

Complejidad en tiempo : $O(r^{p/2})$ r = factor de ramificación ρ = profundidad mínima solución
Complejidad en espacio: $O(r * p/2)$

Búsqueda heurística

Los algoritmos de búsqueda heurística orientan la búsqueda aplicando conocimiento del dominio sobre la proximidad de cada estado a un estado objetivo, guiando así la búsqueda hacia el camino más prometedor.
La heurística mejora la eficiencia de la búsqueda aunque puede sacrificar la complejidad y la optimalidad. Las heurísticas proporcionan recomendaciones sobre la elección del sucesor más prometedor de un estado orientado acerca del orden de los sucesores de un estado.
La clave de la búsqueda heurística está en definir una función heurística adecuada. Una función heurística asocia a cada estado del espacio de estados un valor numérico que evalúa lo prometedor que es ese estado para alcanzar un estado objetivo. Se estima la distancia de un estado a un estado objetivo, los estados de menor valor heurístico son los preferidos (más cerca están de objetivo).
Una función heurística es **admisible** si el valor de la función heurística es menor o igual al coste real de ir desde n a un nodo objetivo por el camino de menor coste.
La búsqueda primero el mejor, selecciona primero el nodo más prometedor en base a una función de evaluación. Para gestionar frontera se usa una **cola de prioridad**, ordenada por el valor de la función de evaluación heurística. La función de evaluación puede ser de dos tipos:

1. Búsqueda voraz: Función que considera exclusivamente lo que falta, es decir, el coste mínimo estimado para llegar a una solución a partir del nodo n. **f'(n) = h'(n)**
   En cada paso intenta ponerse lo más cerca posible del objetivo. Depende del problema y de la calidad de la heurística es completa o no. En general **no es completa**, si hay ciclos puede entrar en rama infinita y no terminar. **No es óptima** ya que ignora el coste del camino completo.

   Complejidad en tiempo : $O(r^m)$ r = factor de ramificación m = profundidad máxima

   Complejidad en espacio: $O(r^m)$
2. Búsqueda A*: Función que considera el coste total estimado del camino desde el nodo inicial a un nodo objetivo pasando por el nodo n. Esta función está formada por el coste de camino desde un nodo inicial hasta n (g(n)) y el coste mínimo estimado para llegar a un nodo objetivo desde n (h'(n)). **f'(n) = g(n) + h'(n)**.
   Combina primero en anchura con primero en profundidad. Se cambia el camino cada vez que hay otros más prometedores. Si h' es admisible, la búsqueda f'(n) se denomina búsqueda óptima.
   Si **h' es consistente** (para cada nodo n y cada sucesor n' de n, el coste estimado de alcanzar el objetivo desde n no es mayor que el coste real de alcanzar n' más el coste estimado de alcanzar el objetivo desde n'), cada vez que expanda un nodo habrá encontrado un **camino óptimo** a dicho nodo desde el inicial.
   A* con Tree Search garantiza una solución óptima si la heurística es admisible.
   A* con Graph Search garantiza una solución óptima si la heurística es consistente
   A* es completa, óptima y óptimamente eficiente, pero en el caso peor la complejidad sigue siendo exponencial.

Búsqueda con adversario

Algoritmo α-β
Cada nodo tiene un intervalo de la mejor jugada que puede hacer y se inicializa como (α = -∞, β = ∞), el jugador MAX es α y MIN es β. Hacemos un recorrido en profundidad propagando hacia arriba los valores de las hojas en el correspondiente α si es un nodo MAX y β si es un nodo MIN quedandote con el mejor en cada caso. No confundir el valor del nodo con el intervalo α-β. En la bajada los intervalos (α, β) se heredan por los hijos del nodo.

Aprendizaje por refuerzo

El algoritmo Q-learning se basa en máquinas de estados. Cada estado tiene un conjunto de acciones que se pueden ejecutar. El personaje ejecuta una acción en el estado actual y la función de refuerzo le da un valor feedback. El valor de refuerzo no tiene que ser el mismo si se repite la acción en el mismo estado.
Cada episodio es una sesión de entrenamiento en la que el agente explora el entorno y recibe la recompensa hasta que alcanza el estado objetivo. Cuantos más episodios de entrenamiento mejor será la matriz Q. La matriz Q me dice que acción debo elegir. Para el estado actual elegir la acción con valor Q máximo
La matriz Q-values, de valores de calidad, guarda para cada estado y acción Q(s,a) lo aprendido hasta el hasta el momento. La actualización de los Q-values se hace con la siguiente regla
$Q(s,a) = (1 - \alpha)\, Q(s,a) + \alpha(r + \gamma \max(Q(s',a')))$

- α **velocidad de aprendizaje (learning rate)** es un valor entre 0 y 1 que indica cuánto aprender en cada experiencia. Con 0 no aprende nada de la nueva experiencia y con 1 olvida todo lo que sabía hasta el momento y se fia completamente de la nueva experiencia.
- γ **tasa de descuento (discount factor)** es un valor entre 0 y 1 que indica la importancia a largo plazo. 0 significa que sólo nos importan los refuerzos inmediatos, y 1 significa que los refuerzos inmediatos no importan, sólo los de largo plazo.

Búsqueda local

En la búsqueda local se empieza con una configuración inicial (aleatoria o no) y se hacen cambios (operadores) hasta alcanzar un estado sin sucesores mejores que él.
El inconveniente principal de este tipo de búsqueda es que la solución óptima global puede no ser alcanzada. Entonces podemos decir que no son **ni óptimos ni completos**, ya que pueden quedarse bloqueados en un óptimo local.
Hill Climbing
Se guarda un único estado. La función de evaluación alcanza su valor mínimo en la solución óptima. Se mueve siempre en la dirección que mejora la función de evaluación, si no puede falla. Se generan sucesores y en cada paso el nodo actual se sustituye por el primer sucesor mejor. Se usa solo si tiene una buena función heurística y ningún otro conocimiento útil.
En escalada por máxima pendiente se generan sucesores y en cada paso el nodo actual se sustituye por el sucesor mejor. Progresa muy rápido hacia una solución pero suele atascarse en óptimos locales. Entre los problemas destacan:

- Mínimo local: Hijos con una función heurística peor.
- Meseta: Hijos con una función heurística igual a la del nodo actual.
- Cresta: La función heurística no guía hacía ningún estado objetivo y termina en un mínimo local o en una meseta.

Para solucionar estos problemas de óptimo local podemos aplicar lo siguiente:

- Hacer backtracking a un nodo anterior y seguir el proceso en otra dirección.
- Reiniciar la búsqueda en otro punto (random restarting Hill Climbing)
- Aplicar dos o más operadores antes de decidir el camino a seguir.
- Dividir el espacio de búsqueda en regiones y explorar las más prometedoras.
- Usar otros algoritmos inspirados en analogías físicas y biológicas como el algoritmo de enfriamiento simulado o los algoritmos genéticos.

Enfriamiento simulado

Introduce un componente aleatorio al método de escalada, escogiendo aleatoriamente entre los hijos que mejoran la función de evaluación. También permite moverse en direcciones que empeoran la función de evaluación, saliendo de óptimos locales.

La probabilidad con que se permiten movimientos que empeoran disminuye con el número de iteraciones. En la fase de exploración se aceptan soluciones mucho peores mientras que en la fase de explotación se aceptan pocas soluciones peores.

Mecanismos de enfriamiento:

- Descenso exponencial: $Tk + 1 = \alpha * Tk$
- Criterio de Boltzmann: $Tk = T0 / (1 + log(k))$
- Criterio de Cauchy: $Tk = T0 / (1 + k)$

Algoritmos geneticos

Utilizan el principio de selección natural para resolver problemas de optimización más complicados. Se hace evolucionar a una población de individuos (cada uno de ellos representa una posible solución) sometiendolas a acciones aleatorias semejantes a las de la evolución biológica (mutaciones, recombinaciones genéticas...). Los individuos se seleccionan de acuerdo a una con una función de adaptación y se decide que individuos sobreviven y cuáles se descartan.

La representación de un estado (solución potencial a un problema) lo llamaremos cromosoma y la forma más típica de codificar cada gen es con valores binarios.

La **estructura** general de los algoritmos genéticos es la siguiente:

1. Generar aleatoriamente una **población inicial** de tamaño x.
2. Calcular la valoración de cada individuo mediante la **función fitness**, que indica la capacidad de cada individuo para resolver el problema.
3. Se **seleccionan** x individuos.
4. Aplicar operador genético de **cruce**, teniendo en cuenta la probabilidad de cruce.
5. Aplicar operador genético de **mutación**, teniendo en cuenta la probabilidad de mutación.

La inicialización de la población suele ser aleatoria pero conviene asegurar una mezcla adecuada e incluir en la población valores distintos para todos los genes. Todos los individuos deben ser válidos para el problema.

La función fitness debe evaluar la aptitud de los individuos de la población en relación al problema planteado. Debe penalizar las malas soluciones y premiar las buenas (tomando siempre valores positivos). Se calcula muchas veces así que debe ser un cálculo rápido.

La selección es el encargado de escoger qué individuos se van a reproducir, estará relacionado con el valor de aptitud (que nos da la función fitness). Tenemos tres tipos de selecciones básicas:

- **Selección elitista**: Se seleccionan de dos en dos para pasarlos a cruce. Garantiza la selección de los miembros más aptos de cada generación. En general no funciona muy bien.
- **Selección por ruleta**: Cada uno de los individuos de la población tiene una probabilidad de ser escogido. Los mejores individuos (según la función de evaluación) recibirán una probabilidad mayor que la recibida por los peores. Su mayor desventaja es que se vuelve ineficiente a medida que aumenta el tamaño de la población.
- **Selección por torneo**: Se eligen subgrupos de tamaño ρ de individuos de la población. Los miembros de cada subgrupo compiten entre ellos y se elige a un individuo de cada subgrupo para el cruce. Variando el número ρ de individuos que participan en cada torneo se modifica la presión de selección.

Una vez seleccionados los individuos, éstos son recombinados mediante algoritmos de cruce para producir la descendencia que se insertará en la siguiente generación. En estos algoritmos hay que tener en cuenta la probabilidad de cruce, la probabilidad de que dos cromosomas intercambian sus genes (entorno a 0,7)

- Sin permutación: **Cruce de 1 o 2 puntos**.
- **Permutación**: **Cruce de orden** (Seleccionar una parte arbitraria del primer padre, copiar esta parte en el hijo y copiar los números que no están en el primera padre empezando desde el punto de cruce y utilizando el orden en el que aparecen en el segundo padre) o **PMX**(flechitas).

Los algoritmos de mutación consisten en modificar unos genes del cromosoma con una probabilidad (> 10%) para garantizar que ningún punto del espacio de búsqueda tenga una probabilidad nula de ser examinado.

- Representación binaria : reemplazos aleatorios o intercambio de alelos.
- Representación decimal: incremento/decremento o multiplicación.
- Permutaciones: Intercambio, revuelto (seleccionar un subconjunto de genes y reordenar aleatoriamente sus alelos)

Sistemas basados en conocimiento

Para evaluar los **sistemas recomendadores** el conjunto de datos originales se divide en k subconjuntos. Se entrena con cada k subconjunto como conjunto de prueba del modelo, mientras que el resto de datos se tomará como conjunto de entrenamiento.
Precision es la fracción de ítems recomendados que son relevantes.
$Precision = |Recomendados \cap Relevantes| \ / |Recomendados|$
Recall es la fracción de ítems relevantes que son recomendados.
$Recall = |Recomendados \cap Relevantes| \ / |Relevantes|$
Se pueden comparar recomendadores usando la media armónica:
$F = (2 * Precision * Recall) / Precision + Recall$
Encadenamiento progresivo: Dirigido por datos (antecedentes-LHS). Busca la conclusión desde los datos (hechos) conocidos. Si los datos verifican las condiciones(LHS) de una regla, la regla se puede aplicar.
Encadenamiento regresivo: Dirigido por objetos(consecuentes-RHS). Puede seleccionar conclusiones(RHS) de reglas cuyos RHS equipares la consulta, o también puede intentar probar su validez buscando evidencias(LHS) que la soporten. Un antecedente(LHS) es cierto si sus hechos están en la MT del sistema, si no, se busca si es consecuente(RHS) de alguna regla R y se prueban recursivamente los antecedentes de dicha regla R. Si esto tampoco, se asume la hipótesis del mundo cerrado o se pregunta al usuario.
Estrategias de control o resolución de conflictos:
Principio de refracción: una regla sólo se dispara una vez con los mismos hechos.
Orden: Se selecciona la primera regla aplicable, orden lineal explícito en la BR (peligroso).
Prioridades heurísticas: Se selecciona la regla de mayor prioridad establecida por el control local.
Especificidad: Se les da prioridad a las reglas con más condiciones.
Arbitrariedad: Se selecciona aleatoriamente, se usa cuando no existen criterios adicionales disponibles.