Diseño de Algoritmos - Algoritmos II

Nazareno Aguirre, Sonia Permigiani, Gastón Scilingo,
Simón Gutiérrez Brida
Departamento de Computación
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Universidad Nacional de Río Cuarto

Clase 8: Búsqueda

Estrategias de Diseño de Algoritmos

Ya hemos visto una variedad de estrategias de diseño de algoritmos, que nos brindan herramientas para atacar el problema de construir soluciones algorítmicas para problemas.

Veremos ahora una familia de técnicas de diseño de algoritmos, que denominaremos Técnicas de Búsqueda. Dado que muchos problemas pueden formularse en términos de búsqueda (como veremos más adelante), estas técnicas tienen un alcance importante.

Problemas de Búsqueda

Un gran número de problemas puede ser resuelto aplicando técnicas de búsqueda.

Para poder aplicar esta técnica, es esencial poder describir el problema como una búsqueda de configuraciones exitosas a partir de ciertas configuraciones iniciales, mediante la aplicación de reglas predefinidas de reconfiguración o avance.

En general, un problema P corresponde a un problema de búsqueda cuando se cuenta con:

- una descripción precisa de qué constituye el estado o configuración
- una descripción del estado de partida (estado inicial)
- una descripción de aquellas configuraciones que se consideran exitosas (estados finales)
- una descripción de cómo se puede avanzar, en un movimiento, desde un estado hacia otros estados sucesores

El problema P es entonces encontrar, a partir del estado de partida, alguna configuración exitosa, usando las reglas de avance predefinidas.

Ejemplo: El Problema de las Jarras de Agua

Consideremos el siguiente problema:

> Tenemos dos jarras, con capacidades de 4 y 3 litros respectivamente. Las jarras no tienen líneas de medición, y queremos que la primera quede con exactamente 2 litros de agua.
>
> Contamos con una fuente de agua ilimitada, y podemos llenar alguna de las jarras, o volcar el contenido de una de las jarras en la otra.

estados posibles: la combinación de los posibles contenidos de las jarras A (de 0 a 4 litros) y B (de 0 a 3 litros)

estado inicial: las dos jarras vacías

estados finales exitosos: la jarra A con 2 litros, y la jarra B con cualquier contenido

reglas de avance: (i) vaciar jarra A, (ii) vaciar jarra B, (iii) llenar jarra A, (iv) llenar jarra B, (v) volcar el contenido de A en B (hasta que A se vacíe o B esté llena), (vi) volcar el contenido de B en A (hasta que B se vacíe o A esté llena)

Árboles de Búsqueda

Usando las descripciones de los elementos anteriores podemos construir un árbol de búsqueda, de la siguiente forma:

- los estados posibles son los nodos del árbol
- el estado inicial es la raíz del árbol
- Un estado e_h es hijo de otro estado e_p en el árbol ssi existe un movimiento (atómico) que lleva de la configuración e_p a la configuración e_h

El problema se reduce entonces a encontrar un camino que nos lleve del estado inicial a alguno de los estados exitosos.

Exploración del Espacio de Estados

Podemos intentar construir un camino desde la raíz del árbol de búsqueda hasta alguno de los nodos exitosos (es decir, una solución al problema) mediante la visita a los nodos del árbol.

Ya conocemos, de otras asignaturas, algunas estrategias simples de visita de nodos en árboles:

- Primero a lo ancho (breadth-first search): visita de los nodos por niveles hasta encontrar un estado exitoso
- Primero en profundidad (depth-first search): visita de los nodos en profundidad, considerando los hijos de un nodo antes que sus hermanos, hasta encontrar un estado exitoso

Ventajas y Desventajas de las Estrategias de Búsqueda Simples

Primero a lo ancho:

- Ventajas: Si existe un estado exitoso de profundidad finita, entonces será encontrado. Encuentra soluciones de profundidad mínima.
- Desventajas: Es necesario almacenar un conjunto de nodos exponencial con respecto al nivel que se está explorando.

Primero en profundidad:

- Ventajas: Sólo es necesario almacenar un conjunto de nodos linealmente proporcional a la profundidad que se está explorando.
- Desventajas: Las soluciones encontradas no son necesariamente de profundidad mínima. Aún habiendo estados exitosos de profundidad finita, puede no encontrarlos (puede perderse en ramas infinitas sin soluciones).

Una Combinación de Estrategias Simples de Búsqueda

R. Korf publicó en 1985 una estrategia que puede verse como una combinación inteligente de las estrategias simples primero en profundidad y primero a lo ancho. Esta estrategia se llama profundización iterativa (iterative deepening) y consiste en realizar búsquedas depth-first sucesivas, cada una con un límite de profundidad incrementado en uno con respecto al anterior. Es decir:

1. realizar depth-first search hasta profundidad 1
2. si no se encontró una solución, realizar depth-first search hasta profundidad 2
3. si no se encontró una solución, realizar depth-first search hasta profundidad 3

...

Ventajas y Desventajas de Iterative Deepening

Desventajas

- Se vuelve a visitar múltiples veces un gran número de nodos

Ventajas

- Si existe un estado exitoso de profundidad finita, entonces será encontrado
- Encuentra soluciones de profundidad mínima
- Sólo es necesario almacenar un conjunto de nodos linealmente proporcional a la profundidad que se está explorando
- Las múltiples visitas a nodos no afecta la tasa de crecimiento asociada al tiempo de ejecución (similar a depth-first search)

Otros Ejemplos

Otros ejemplos de problemas que pueden formularse como problemas de búsqueda son los siguientes:

- búsqueda de salidas en laberintos
- el problema de las 8 baldosas (8-tile problem, o 8-puzzle)
- el problema de las 8 reinas
- Acertijos del estilo del de Indiana Jones, o cruce del río con la gallina, el zorro, el maíz y el granjero
- juegos (e.g., Sudoku)

Mejorando las Estrategias de Búsqueda Básicas

Las estrategias generales de búsqueda pueden ser mejoradas utilizando algunas técnicas muy simples.

Las mejoras a las estrategias básicas de visita de árboles de búsqueda están basadas, fundamentalmente, en información adicional acerca del dominio de cada problema. Es por esto que estas técnicas se denominan usualmente técnicas de búsqueda informada.

La información adicional requerida por algunas técnicas de mejora de algoritmos de visita es en general una estimación de la "bondad" de un estado no final, que permite decidir qué caminos seguir durante el recorrido del árbol de búsqueda.

Es un tipo de información extra muy simple, que puede codificarse fácilmente como una función de valoración de estados.

Ejemplos de Funciones de Valoración

- Para el problema de búsqueda de salidas en laberintos, distancia a la salida más cercana
- Para el problema de las jarras de agua, $|A - 2|$
- Para el 8-puzzle, cantidad de baldosas en la posición correcta
- Para el problema de las 8 reinas, número de reinas que se atacan entre sí

La Técnica de la Escalada ([Steepest] Hill Climbing)

Utilizando una función de valoración de estados, podemos definir un nuevo algoritmo:

1. Comenzar con el estado inicial como estado actual

2(a). Si el estado actual es un estado exitoso, terminar exitosamente

2(b). Sino, obtener los hijos del estado actual, uno a la vez, hasta encontrar uno con mayor valoración que el estado actual o agotar todos los hijos

3(a). Si no existen hijos con mayor valoración que el estado actual, terminar

3(b). Sino, volver al paso 2, con el primer hijo con valoración mayor como estado actual

La Técnica "Primero el Mejor" (best-first search)

Esta técnica es una variante de la técnica general de visita breadth-first search, que aprovecha la información provista por la función de valoración de estados

1. Comenzar con una cola ABIERTOS sólo conteniendo al estado inicial

2(a). Si ABIERTOS está vacía, terminar

2(b). Sino, tomar al mejor estado de ABIERTOS como el estado actual

3(a). Si el estado actual es un estado exitoso, terminar exitosamente

3(b). Sino, obtener los hijos del estado actual y agregarlos a ABIERTOS

4. Quitar el estado actual de ABIERTOS

5. Volver al paso 2

Características de la Técnica "Best-First"

- Su performance depende fundamentalmente de la calidad de la función heurística
- Puede no encontrar estados exitosos de profundidad finita
- Si la función de valoración no es la adecuada, puede degenerar en una búsqueda depth-first, con un empeoramiento considerable en el espacio de almacenamiento utilizado

Búsqueda en Problemas con Adversarios

- Las estrategias generales de búsqueda que hemos visto permiten resolver problemas en los que en la búsqueda es realizada por sólo un "jugador".
- En problemas en los cuales se compite contra un adversario en la búsqueda de estados exitosos, las técnicas de búsqueda vistas no son las apropiadas.

Características de los Problemas de Búsqueda con Adversarios

Podemos reconocer algunas características propias de los problemas de búsqueda con adversarios:

- Existe un estado inicial bien definido

- A partir del estado inicial, se avanza mediante reglas de avance bien definidas, alternando movimientos con un adversario

- El objetivo del adversario, al igual que el nuestro, es conseguir llegar a un estado exitoso

- En general si el adversario tiene éxito, nosotros fracasamos

A diferencia de los problemas de búsqueda convencionales, en problemas con adversarios el avance no depende de un solo "agente".

Ejemplos de Problemas de Búsqueda con Adversarios

Los ejemplos clásicos de búsqueda con adversarios son juegos:

- ajedrez

- Ta-Te-Ti (tic-tac-toe)

- Damas

- Otello

- Go

La Técnica Min-Max

La técnica básica para solución a problemas de búsqueda con adversarios es Min-Max. Esta técnica se basa en considerar que el adversario puede realizar, cuando le toque, el mejor movimiento posible, y por lo tanto se independiza de las decisiones del adversario.

Al igual que para problemas de búsqueda convencionales, esta técnica se basa en la construcción de un árbol de juego.

Arboles de Juego

Los árboles de juego para búsqueda en problemas con adversarios se construyen de la siguiente forma:

- Las configuraciones del juego son los nodos del árbol

- La raíz es el estado inicial del juego

- Si e' resulta de la aplicación de una de las reglas de avance del juego a partir de e, entonces e' es hijo de e en el árbol

- Los niveles del árbol se clasifican como Min o Max, según corresponda el movimiento al adversario o al jugador principal

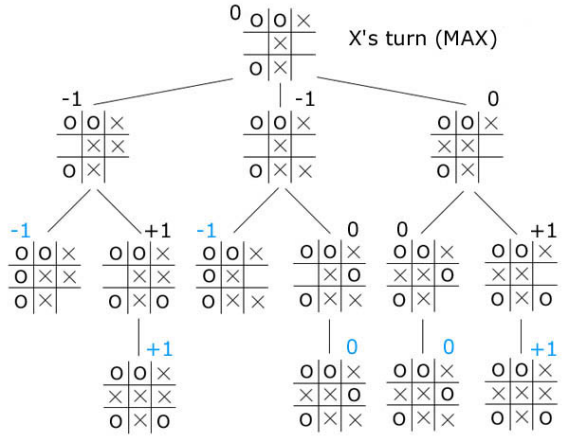
- las hojas se etiquetan con +1, 0 o -1, según corresponda a una configuración en la que ganó el jugador principal, fue empate, o ganó el adversario

- Cada nodo interior del árbol en un nivel Max (resp. Min) se etiqueta con el máximo (resp. mínimo) de los valores de los hijos

Algoritmo Min Max

```
Funcion minMax(n) --> Valor
    Si n es hoja
        retornar valor(n)
    Sino
        x:= MIN_VAL
        y:= MAX_VAL
        Para cada hijo n_k de n hacer
            Si n es Max
                x:= max(x, minMax(n_k))
            Sino
                y:= min(y, minMax(n_k))
            Fsi
        Fpara
        Si n es Max
            retornar x
        Sino
            retornar y
        Fsi
    Fsi
Ffuncion
```


Un Ejemplo



Cómo usar el Árbol Min Max para Buscar una Estrategia Ganadora

Suponiendo que contamos con el árbol de juego, la mejor estrategia que podemos seguir es elegir la rama con mayor valor en los turnos en los que nos toque mover.

Por supuesto, esto lo podemos hacer si contamos con el árbol, el cual además debe ser finito (pues necesitamos los valores de los estados finales, la hojas, para etiquetar los nodos internos).

Adaptando Min-Max para uso en la Práctica

En general, los árboles de juego son de una gran dimensión, por lo cual, aún siendo finitos, no se los puede construir por completo.

Una adaptación útil para poder aplicar la técnica en la práctica se basa en la utilización de una función de valoración de configuraciones, y un límite k en el número de futuros movimientos a considerarse:

- En el momento en el que nos toque ``jugar'', generamos el sub-árbol de juego que tenga como raíz la configuración actual, y que tenga hasta k niveles

- valoramos las ``hojas'' de este sub-árbol usando la función de valoración (en el caso en que las hojas no sean ya estados finales)

- valoramos los nodos internos usando el procedimiento Min-Max

Poda Alfa-Beta: Una Mejora a Min-Max

Incluso utilizando las técnica Min-Max ``adaptada'', para límites k de movimientos futuros razonables el tamaño del sub-árbol de juego a explorar es en general muy grande.

Debido a esto, es necesario buscar mejoras al algoritmo Min-Max. Una optimización interesante es la provista por la técnica de poda Alfa-Beta.

Algoritmo Min-Max con Poda Alfa-Beta

```
Funcion minMaxAB(n, alfa, beta) --> Valor
    Si n es hoja o esta a nivel maximo
        retornar valor(n)
    Sino
        Para cada hijo n_k de n y mientras alfa<beta hacer
            Si n es Max
                alfa:= max(alfa, minMaxAB(n_k, alfa, beta))
            Sino
                beta:= min(beta, minMaxAB(n_k, alfa, beta))
            Fsi
        Fpara
        Si n es Max
            retornar alfa
        Sino
            retornar beta
        Fsi
    Fsi
Ffuncion
```


Un Ejemplo