Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 — Inteligencia Artificial — 2' 2019

Tarea 2 – Sliders

1 Objetivo

En esta tarea usted deberá programar un algoritmo de búsqueda que lleve a encontrar soluciónes w-óptimas conforme al tiempo disponible para deliberar una solución, en un problema de búsqueda difícil como el puzzle Sliders. Particularmente, los objetivos de esta tarea son:

- Encontrar una heurística admisible para el problema del puzzle sliders.
- Explorar algoritmos de búsqueda Anytime Best First Search que entreguen resultados suboptimos, con una garantia de w-suboptimalidad.

2 Descripción del espacio de búsqueda

El problema a resolver es el puzzle sliders.

El puzzle Sliders es una mezcla entre el puzzle de 8, 15 o 24 y el cubo Rubik.

Este juego se juega sobre un tablero de $m * n$, y el objetivo es llegar a una configuración en la que los números aparezcan en forma consecutiva en el rectángulo.

Una diferencia de este problema con los juegos de puzzle es que este juego se puede jugar sobre grillas de cualquier tamaño, no necesariamente cuadradas, y no tiene un espacio vació.

En el juego sliders, las acciones son:

- desplazar en 1 cuadro, a la derecha o izquierda, una fila del puzzle.
- desplazar en 1 cuadro, hacia arriba o abajo una columna del puzzle.

Al desplazar una fila o columna el puzzle cambia de estado como si éste estuviera sobre un toro. La figura 1 muestra un ejemplo de cómo cambia el estado al aplicar la acción arriba sobre la columna 1 y al aplicar la acción izquierda sobre la fila 1. Además en la figura 2 se observa otro espacio de búsqueda donde la grilla es de tamaño 3x5.

3 Búsqueda Anytime

Un algoritmo de búsqueda anytime es aquel que entrega una solución con una calidad acorde al tiempo disponible para la deliberación. Los algoritmos de búsqueda solicitados en esta tarea están basados en weighted-A*, el cual consiste en "inflar" el valor de la heurística utilizando la función de evaluación $f(s) = g(s) + w * h(s)$, donde $g(s)$ es el costo acumulado, $h(s)$ es una estimación del costo de ir desde s hasta s_{goal} y w es un factor de suboptimalidad $w \geq 1$. Si la heurística es admisible y $w = 1$ el resultado entregado es óptimo; si $w > 1$ se dice que el resultado es w-suboptimo, es decir, el factor w determina una cota de suboptimalidad donde la solución es a lo mas w veces el óptimo.

Figure 1: Ejemplo de un problema Sliders de 3x3 y la generación de dos estados sucesores a partir de la ejecución de dos de sus acciones disponibles

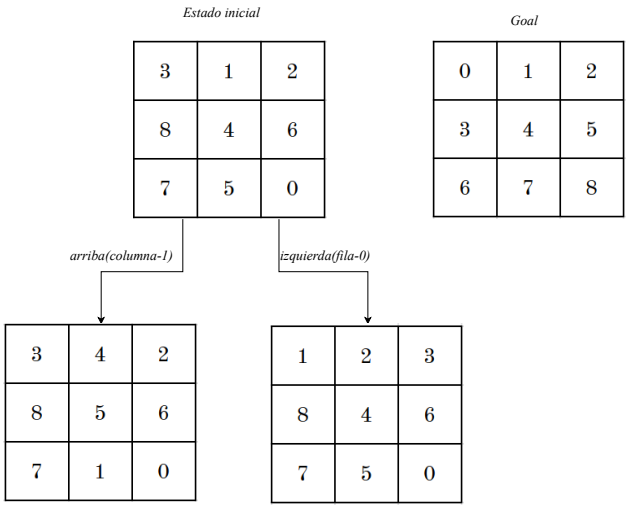


Figure 2: Ejemplo del estado inicial y el goal de un problema sliders en una grilla de 5x3

Estado inicial

1	2	3	11	14
0	12	8	4	6
7	10	13	5	9

Goal

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

3.1 Anytime Weighted A*

Anytime Weighted A* (AWA*) ejecuta wA* con un valor w inicial, y luego de encontrar una solución continúa la búsqueda con la intención de mejorar la solución encontrada, hasta que la lista $OPEN$ esté vacía, podando aquellos estados cuyo $g(s) + h(s)$ sea mayor al costo de la solución ya encontrada. En síntesis, una vez que el estado objetivo se extrae de $OPEN$, AWA* continua dentro del ciclo de búsqueda expandiendo estados hasta que la lista $OPEN$ este vacía. Puede encontrar mas información sobre esta estrategia de búsqueda en su paper [1]

3.2 Restarting wA*

RWA* reinicia la búsqueda cada vez que una solución es encontrada, disminuyendo el valor del parámetro w acorde a una política de planificación de w definida como $w = max(1, \phi * w)$, siendo ϕ un factor $0 < phi < 1$ que multiplica el valor anterior de w para obtener el nuevo w con el cual se realizará el siguiente episodio de búsqueda. Una diferencia fundamental entre AWA* y RWA* es que este último no conserva las listas de $OPEN$ y $CLOSE$ en los episodios de busqueda siguientes. En resumen, RWA* consiste en llamar repetidamente a wA*, podando aquellos estados cuyo valor de $g(s) + h(s)$ sea mayor al costo de la solución ya encontrada, y disminuyendo el factor de suboptimalidad w acorde a un factor ϕ.

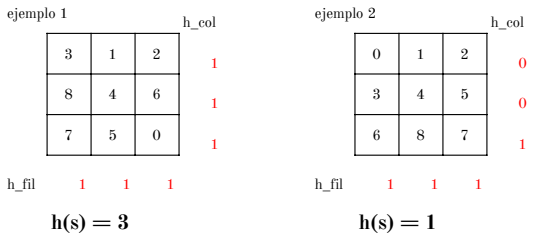
Este algoritmo, en su propuesta original, incorpora una tercera lista llamada $SEEN$, que almacena los estados vistos en iteraciones anteriores con la intención de rescatar la estimación de la heurística y no tener que calcularla. Para efectos de su tarea, no se requiere la implementación de esta tercera lista y puede calcular el valor de la aproximación heurística de los estados siguientes.

Puede encontrar mas información sobre esta estrategia de búsqueda en su paper [2]

3.3 Heurística entregada

En el código que a usted se le entrega, se incorpora una heurística admisible para el problema del puzzle sliders. Esta consiste en determinar si las filas o columnas del puzzle ya se encuentran ordenadas y se calcula de la siguiente forma: si la fila tiene los cuadros correctos y ordenados (ejemplo 0 1 2) suma 0 al valor de h_{fila}, si no (ejemplo 1 0 2 ó 0 1 5) suma 1. Lo mismo para las columnas, si esta la columna ordenada (ej 0 3 6) suma 0 al valor de $h_{columnas}$, si no lo están (ej 0 4 5) suma 1. La heurística del estado será definida como $h(s) = min(h_{filas}(s), h_{columnas}(s))$ Puede ver dos ejemplos de esta heurística en la figura 3. Esta función heurística no es muy informativa, dado que el mayor valor que estimará será siempre el mínimo entre el ancho y el alto de la grilla del puzzle.

Figure 3: Ejemplo del calculo de la heurística entrega para un problema Sliders de 3x3



4 Que se pide

En su tarea, usted deberá:

1. Encontrar una función heurística admisible para el problema del puzzle sliders mas informativa que la heurística trivial ya propuesta.
2. Realizar una implementación del algoritmo Anytime Weighted A* y
3. Realizar una implementación del algoritmo Restarting Weighted A*, de acuerdo a las descripciones dadas en el punto anterior.
4. Entregar un breve reporte donde describa la función heurística propuesta por usted y una comparación del comportamiento en los tiempos de búsqueda y los resultados entregados por cada algoritmo.

Este código, usted debe programarlo dentro del archivo *solution.py*, de acuerdo a la estructura dada dentro de ese archivo fuente.

Para este propósito, a usted se le provee código prefabricado el cual debe utilizar. Dentro de este código, usted encontrará los siguientes archivos:

- *search.py*: Archivo que incluye un motor de búsqueda genérico capaz de ejecutar diferentes estrategias de búsqueda. Este código se provee como base para el Anytime Sliders Solver que usted debe programar. Dentro de este archivo se incorporan principalmente dos clases: StateSpace que representa un espacio de búsqueda genérico (clase de la cual luego se heredará el espacio de búsqueda especifico del problema)
- *sliders.py*: Archivo que contiene las funciones necesarias para representar el problema del puzzle sliders. En este archivo usted encontrará una clase que contiene la generación de sucesores (successors), los estados sucesores luego de la ejecución de las acciones (slide), una función para determinar si el estado es un estado objetivo (sliders_goal_state) y un conjunto de problemas que le servirán a usted para ejecutar sus pruebas.

Puede encontrar una descripción mas detallada del código prefabricado en el archivo manual_search_engine.pdf desarrollado por Sonya Allin de la University of Toronto, a quien extendemos nuestros agradecimientos.

5 Reglas

En resumen, usted debe entregar

- La función heurística solicitada y los dos algoritmos anytime en el archivo solution.py
- En su entrega, no debe incluir los archivos search.py ni sliders.py, por lo tanto no puede modificarlos. Si considera que se debe hacer alguna corrección a estos archivos, abra una issue en el github del curso.
- Un breve reporte que incluya una descripción de la heurística propuesta junto a una comparación entre los algoritmos.
- Fecha de entrega: **Miercoles 16 de Octubre, 2019** a las **20:00 hrs.**
- Casi la totalidad de esta tarea será revisada de manera automatizada, por lo cual funciones incompletas o con errores sintácticos o semánticos no tendrán puntaje.
- Usted puede incorporar funciones o clases auxiliares en su entrega, pero el resultado final debe ser retornado al llamar a las funciones con la etiqueta IMPLEMENT.
- Solo puede utilizar bibliotecas estándar de python y numpy. Se recomienda no importar mas bibliotecas de las que ya están. Scipy, Pandas, networkx, etc. no están permitidos ya que al ejecutar el corrector, este arrojará la falta de un módulo y su calificación será deficiente.

6 Política de Integridad Académica

Los alumnos de la Escuela de Ingeniería deben mantener un comportamiento acorde al Código de Honor de la Universidad:

"Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad."

En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un procedimiento sumario. Ejemplos de actos deshonestos son la copia, el uso de material o equipos no permitidos en las evaluaciones, el plagio, o la falsificación de identidad, entre otros. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica en relación a copia y plagio: Todo trabajo presentado por un alumno (grupo) para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno (grupo), sin apoyo en material de terceros. Si un alumno (grupo) copia un trabajo, se le calificará con nota 1.0 en dicha evaluación y dependiendo de la gravedad de sus acciones podrá tener un 1.0 en todo ese ítem de evaluaciones o un 1.1 en el curso. Además, los antecedentes serán enviados a la Dirección de Docencia de la Escuela de Ingeniería para evaluar posteriores sanciones en conjunto con la Universidad, las que pueden incluir un procedimiento sumario. Por "copia" o "plagio" se entiende incluir en el trabajo presentado como propio, partes desarrolladas por otra persona. Está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la cita correspondiente.

References

[1] Eric A Hansen and Rong Zhou. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28:267–297, 2007.

[2] Silvia Richter, Jordan T Thayer, and Wheeler Ruml. The joy of forgetting: Faster anytime search via restarting. In *Twentieth International Conference on Automated Planning and Scheduling*, 2010.