Proyecto Corto #2-3 - Buscador de Zanahorias

Inteligencia Artificial

Grupo 1, I Semestre 2018

El actual proyecto (que se evaluará como proyecto corto #2 y #3) propone el desarrollo de algoritmos de búsqueda para resolver un problema de la familia de recorrido de laberintos. En particular, se pide a los estudiantes procesar tableros rectangulares representados como archivos de texto que contendrán un conejo y múltiples zanahorias. El objetivo primario de cada corrida de programa será que el conejo encuentre una cierta cantidad de zanahorias.

El proyecto se dividirá en dos partes, ambas contenidas en un programa por consola que se ejecutará en los diferentes modos especificando banderas diferenciadas. La primera consistirá en desarrollar un heurístico utilizado dentro de A* para genéricamente recorrer el tablero en busca de zanahorias. La segunda, consistirá en desarrollar un algoritmo genético que optimizará la colocación de señales direccionales para que el conejo recorra el tablero.

Representación del tablero en archivo

En aras de enfocar el desarrollo del proyecto en los algoritmos de búsqueda, se utilizará una interfaz rudimentaria para manejar las entradas y salidas de los programas. El tablero será un archivo de texto de N líneas y M columnas y cada caracter podrá ser únicamente:

- **C**: en mayúscula, identifica la posición del conejo. Sólo podrá haber uno por tablero.
- **Z**: en mayúscula, identifica la posición de una zanahoria. Puede haber múltiples zanahorias por tablero.
- **Espacio en blanco**: El caracter de espacio no tiene ningún efecto secundario pero sí debe estar presente para indicar una posición en el tablero por la que el conejo puede transitar.
- **<**: símbolo menor que, indica un cambio de dirección hacia la izquierda.
- **>**: símbolo mayor que, indica un cambio de dirección hacia la derecha.
- **A**: letra A mayúscula, indica un cambio de dirección hacia arriba.
- **V**: letra V mayúscula, indica un cambio de dirección hacia abajo.
- **Cambio de línea:** No tiene ninguna interpretación en el programa más que separar las distintas filas. Nótese que existen diferencias entre la manera en que WIndows y otras plataformas expresan cambios de línea (2 caracteres versus 1). La revisión se realizará en Linux o MacOS por lo que pide que los archivos **NO** utilicen el estilo de Windows. El profesor se reserva el derecho de calificar el proyecto corto con nota mínima de no

cumplirse, ya que la revisión primaria de los resultados depende de visualizar múltiples archivos de salida en este formato.

A manera de ejemplo, un tablero como el siguiente:

C			
		Z	
			Z

Sería representado como un archivo con las siguientes líneas:

- C, <blanco>, <blanco>, <blanco>, <cambio de línea>
- <blanco>, <blanco>, Z, <blanco>, <cambio de línea>
- <blanco>, <blanco>, <blanco>, Z, <cambio de línea>

Nótese que este será el formato de archivos de entrada y de salida, como se describe más adelante. Para indicar cuál es el archivo de entrada se deberá proveer una bandera mandatoria llamada --tablero-inicial <direccion/a/archivo.txt>

Algoritmo A*

Los estudiantes deberán crear un programa que mueva al conejo a través del tablero, una casilla a la vez, utilizando A*. Deben diseñar una función de costo para guiar la búsqueda A* que considere el costo acumulado y un heurístico para aproximar el costo futuro. El acumulado hasta un punto específico en la ejecución del algoritmo será simplemente la cantidad de pasos que ha dado el conejo. El heurístico de costo futuro se dejará a diseño de los estudiantes, sin embargo, deberá considerar los siguientes elementos:

- El ambiente no es completamente observable. Esto quiere decir que para predecir costo futuro no se debe asumir que se puede leer todo el tablero.
- El conejo tendrá un rango de visión. Si el estado actual tiene al conejo en la casilla (20, 18), por ejemplo, y el conejo tiene una visión de dos, podrá tomar en cuenta el contenido de las casillas (18, 16) hasta la casilla (22, 20). Podrá usar la información de todas esas casillas (i.e. si hay zanahorias o no) para estimar el costo futuro de un movimiento. Será controlada por una bandera llamada --vision <número>.
- El diseño del heurístico no debe incluir "memoria" que haga el ambiente implícitamente observable. Por ejemplo, los estudiantes podrían verse inclinados a mantener una estructura de datos con todas las casillas ya recorridas, de manera que no repitan el recorrido por regiones. En particular, se quiere que la huella de memoria del algoritmo sea constante y al mantener un histórico de visitas se hace la complejidad O(n), con n siendo el número de casillas del tablero. Se insta a los estudiantes a pensar en alta escalabilidad de sus algoritmos.

- Se indicará al inicio del programa cuántas zanahorias en total debe buscar el conejo para terminar su labor (con la bandera --zanahoria <cantidad>). Nótese que esto no quiere decir que sólo existan esa cantidad de zanahorias en el tablero, pero el conejo se dará por satisfecho después de esa cantidad.

Un ejemplo de comando utilizado para correr el programa, y su respectiva salida, se dan a continuación (detalle de nombres de módulos e instalación en sección de Entregables):

```
> python main.py --tablero-inicial entrada.txt --a-estrella --vision 2 --zanahorias 5
PASO: 00001 IZQUIERDA: 5 DERECHA: 3 ARRIBA: 8 ABAJO: 9 MOVIMIENTO: DERECHA
PASO: 00002 IZQUIERDA: 6 DERECHA: 5 ARRIBA: 5 ABAJO: 9 MOVIMIENTO: ARRIBA
...
PASO: 00105 FINAL
```

Nótese que los movimientos del conejo serán únicamente izquierda, derecha, arriba y abajo (no diagonal). Se espera que el programa muestre cuánto es el valor de la función de costo para cada una de las 4 acciones disponibles, en cada estado. Eventualmente, cuando el conejo encuentre la última zanahoria, simplemente mostrará un estado que diga FINAL. En la documentación deberá explicarse la función de costo en suficiente detalle como para poder calcular cada uno de los 4 valores "en papel", con sólo observar el estado del tablero.

Además de la salida en consola deberá generar un archivo de texto, con nombre dado por 5 dígitos que muestran el número de paso, que refleje el estado actual del tablero (favor generar ceros para números pequeños como 00004.txt). La revisión depende de la generación de todos estos archivos para poder corroborar la ruta tomada por el conejo para recolectar las zanahorias. De no estar presente el profesor se reserva el derecho de asignar nota mínima al proyecto.

Se recomienda escribir los archivos después de cada paso ya que ante programas que entren en ciclos infinitos deberá ser posible abortar ejecución y analizar el estado hasta ese momento.

Para efectos del informe los estudiantes deberán presentar el análisis de variación de costo al cambiar número de zanahorias y visión en un tablero de al menos 25 por 25 y 10 zanahorias.

Algoritmo Genético

Una modificación al problema planteado utilizará las señales de dirección mencionadas en la descripción del archivo (las señales no se utilizaban en A*). En esta segunda parte, el algoritmo no deberá tomar decisiones de cómo mover al conejo paso a paso, sino que asumirá que el conejo siempre se mueve hacia adelante y que se podrá cambiar su dirección con las señales.

El objetivo del programa será determinar la localización ideal de las señales para que el conejo pueda recolectar TODAS las zanahorias del tablero en la menor cantidad de pasos posibles y con la menor cantidad de señales.

El estado inicial estará dado por un archivo igual al del programa A*. Además, deberá indicarse a través de una bandera, cuál será la dirección inicial del conejo, lo cual es clave para colocar la primera señal. Lo anterior se controlará a través de cuatro banderas con el nombre de cada dirección: --derecha, --izquierda, --arriba y --abajo.

Cada individuo, para el algoritmo genético, será un tablero completo con sus correspondientes zanahorias, señales y localización del conejo. Con la bandera --individuos <número> se indicará al programa cuántos elementos tendrá cada generación. En un principio se arrancará de copias del estado inicial únicamente pero, conforme avance el algoritmo, comenzarán a divergir por el proceso aleatorio. Se repetirá el proceso por un número específico de generaciones (controlado por la bandera --generaciones <número>).

Se deja a diseño de los estudiantes los detalles específicos para realizar mutaciones, cruces y selección, sin embargo, deben seguir las consideraciones a continuación:

- Las mutaciones son operaciones puntuales. Por ejemplo, agregar 5 señales al mismo tiempo es un cambio muy abrupto. Agregar una señal en una casilla aleatoria, cambiar la dirección de una señal o remover una señal, es un ámbito de modificación apropiado para una mutación.
- Los cruces para este tipo de representación tienden a orientarse a escoger un punto para dividir un estado en 2, y después cambiar una división en un individuo por la análoga en el individuo con el que se cruza. En este caso, divisiones en una columna o fila específica serían un punto de partida apropiado.
- La función de aptitud debe combinar total de zanahorias recolectadas, cantidad de pasos del conejo y cantidad de señales. El diseño de la función final depende de los estudiantes, sin embargo, debe recalcarse que es válido tener una combinación lineal heurística donde se refleje que hay una penalización mayor por agregar una señal que por dar una cierta cantidad de pasos.
- La función de aptitud no influencia la selección de cruce y mutación. Las dos operaciones básicas de algoritmos genéticos son, en esencia, completamente aleatorias dada una política. La función de aptitud solamente sirve para ordenar la población de más a menos apto y seleccionar los que pasan a la siguiente generación.
- Los estudiantes deben documentar sus decisiones en detalle en el informe.

Un ejemplo de comando utilizado para correr el programa, y su respectiva salida, se dan a continuación:

```
> python main.py --tablero-inicial entrada.txt --genetico --derecha --individuos 3
--generaciones 1000
GENERACION: 00001
   INDIVIDUO 00001 APTITUD: 50
   INDIVIDUO 00002 APTITUD: 40
   INDIVIDUO 00002 APTITUD: 38
```

```
GENERACION: 00002
   INDIVIDUO 00001 APTITUD: 52
   INDIVIDUO 00002 APTITUD: 40
   INDIVIDUO 00002 APTITUD: 38
...
GENERACION: 1000
   INDIVIDUO 00001 APTITUD: 120
   INDIVIDUO 00002 APTITUD: 95
   INDIVIDUO 00002 APTITUD: 90
```

La salida en este caso será diferente a la de A*, por la gran cantidad de estados intermedios analizados en algoritmos genéticos. Se pide que los archivos de texto que muestran el conejo, zanahoria y señales estén organizados en una estructura de directorios de 3 niveles: <dirección>/<número generación>/<número individuo>.txt. En el caso anterior se tendría una carpeta llamada "derecha" primero, seguido del número de generación (5 dígitos) y luego el archivo del individuo (5 dígitos). Para la generación 1 y 1000, por ejemplo, los directorios se organizarían así:

- derecha
    - 00001
        - 00001.txt
        - 00002.txt
        - 00003.txt
    - 01000
        - 00001.txt
        - 00002.txt
        - 00003.txt

Nótese que esto tiene el efecto deseable que si se corre el programa con --izquierda en este punto, no sobre escribiría los resultados (utilizando derecha si lo haría).

Para efectos del informe, los estudiantes deberán analizar el efecto de al menos dos políticas de cruce y dos tasas de mutación diferente, a lo largo de diferente número de generaciones. Deberán exponer, a través de banderas, esta funcionalidad, de manera que se pueda reproducir en la revisión del proyecto. En el informe deberá documentarse en detalle cada una de las políticas.

Por último, en el informe deberán también abordar el tema de reproducibilidad de los resultados. Como algoritmos genéticos son esencialmente aleatorios, se espera que los estudiantes ejecuten el programa un número considerable de ocasiones para asegurarse que los resultados obtenidos pueden reproducirse en una fracción significativa de las corridas. El obtener un buen resultado en una sola corrida no es señal de un buen algoritmo, a menos que este resultado se pueda obtener con alguna consistencia.

Entregables

- Código fuente en un módulo llamado tec.ic.ia.pc2.g##, con pc1 representado Proyecto Corto 2, y g## que se reemplazará por el número de grupo asignado por el profesor (omitiremos el hecho que es también válido por la nota de proyecto corto #3). Se pide que el programa primario sea llamado main.py. Para efectos de revisión este podrá ser copiado afuera de la estructura de directorios entregada por los estudiantes. Se asumirá que el paquete se podrá instalar con pip en la máquina de revisión, y main.py importará lo necesario, asumiendo que los otros elementos ya están disponibles.

  ```
  from tec.ic.ia.pc1.g01 import generar_muestra_pais, generar_muestra_provincia
  ```

  El archivo main.py debe ser un esqueleto básico que procese banderas y llame a una jerarquía de clases encargada de las diferentes operaciones. No será aceptable que todo el código se incluya en el archivo principal. En la revisión se considerará el buen diseño de la solución como parte de la evaluación.

- Pruebas unitarias. En particular, operaciones específicas como moverse de una casilla a otra en A*, actualizando el estado correctamente, o realizar operaciones como cruce y mutación, se espera sean probadas unitariamente. Las mismas deben realizarse utilizando pytest.

- Un informe que aborde los requerimientos documentales mencionados a lo largo de esta especificación. Debe ponerse especial énfasis a documentar el heurístico de costo en A*, para poder reproducir los resultados "en papel", al igual que documentar en detalles los elementos de diseño de políticas para el algoritmo genético. Además, se espera que los estudiantes hagan un análisis detallado de los diferentes escenarios de evaluación descrito en cada una de las dos secciones.

Detalles de Entrega y Revisión

El proyecto corto deberá realizarse en los grupos de 3 personas pre-establecidos para otros proyectos. Se debe enviar todos los entregables a través de TEC Digital a más tardar el 23 de mayo del año en curso.

El profesor se reserva el derecho a asignar una nota de cero si los requerimientos no son cumplidos, de forma tal que impida ejecutar las funciones principales.

Consideraciones Python

- Se utilizará Python 3 (no 2.7)
- Se utilizará pip para instalar el módulo enviado. Se debe respetar la estructura de directorios apropiada para que la instalación sea exitosa.
- Se debe utilizar pytest para pruebas unitarias

- Seguir recomendaciones en http://docs.python-guide.org/en/latest/
- Seguir PEP 20. Una guía rápida con ejemplos está disponible en: http://artifex.org/~hblanks/talks/2011/pep20_by_example.pdf
- Utilizar PEP 8 como guía de estilo: http://pep8.org/