Proyecto de Simulación - Programación Declarativa - Agentes

Samuel David Suárez Rodríguez C-412

February 4, 2022

Abstract

En el siguiente proyecto se pretende realizar una simulación implementada en Haskell de un entorno de información completa en el cual interactúan robots y niños como entidades principales, los niños se mueven aleatoriamente por el entorno y generan suciedad. La tarea de los robots es mantener el entorno limpio a la vez de ubicar a los niños en corrales. Se presentan también dos modelos de agentes para resolver el problema, un agente reactivo y un agente inteligente, y analizaremos las diferencias principales entre estas variantes.

0.1 Enlace a Github

`https://github.com/samueldsr99/cleaning-robots`

1 Marco general

El ambiente en el cual intervienen los agentes es discreto y tiene la forma de un rectángulo de N × M . El ambiente es de información completa, por tanto todos los agentes conocen toda la información sobre el ambiente. El ambiente puede variar aleatoriamente cada t unidades de tiempo. El valor de t es conocido.

Las acciones que realizan los agentes ocurren por turnos. En un turno, los agentes realizan sus acciones, una sola por cada agente, y modifican el medio sin que este varíe a no ser que cambie por una acción de los agentes. En el siguiente, el ambiente puede variar. Si es el momen- to de cambio del ambiente, ocurre primero el cambio natural del ambiente y luego la variación aleatoria. En una unidad de tiempo ocurren el turno del agente y el turno de cambio del ambiente.

Los elementos que pueden existir en el ambiente son obstáculos, suciedad, niños, el corral y los agentes que son llamados Robots de Casa. A continuación se precisan las características de los elementos del ambiente:

- Obstáculos: estos ocupan una única casilla en el ambiente. Ellos pueden ser movidos, empujándolos, por los niños, una única casilla. El Robot de Casa sin embargo no pue- de moverlo. No pueden ser movidos ninguna de las casillas ocupadas por cualquier otro elemento del ambiente.

- Suciedad: la suciedad es por cada casilla del ambiente. Solo puede aparecer en casillas que previamente estuvieron vacías. Esta, o aparece en el estado inicial o es creada por los niños.

- Corral: el corral ocupa casillas adyacentes en número igual al del total de niños presentes en el ambiente. El corral no puede moverse. En una casilla del corral solo puede coexistir un niño. En una casilla del corral, que esté vacía, puede entrar un robot. En una misma casilla del corral pueden coexistir un niño y un robot solo si el robot lo carga, o si acaba de dejar al niño.

- Niño: los niños ocupan solo una casilla. Ellos en el turno del ambiente se mueven, si es posible (si la casilla no está ocupada: no tiene suciedad, no está el corral, no hay un Robot de Casa), y aleatoriamente (puede que no ocurra movimiento), a una de las casilla adyacentes. Si esa casilla está ocupada por un obstáculo este es empujado por el niño, si en la dirección hay más de un obstáculo, entonces se desplazan todos. Si el obstáculo está en una posición donde no puede ser empujado y el niño lo intenta, entonces el obstáculo no se mueve y el niño ocupa la misma posición. Los niños son los responsables de que aparezca la suciedad. Si en una cuadrícula de 3 por 3 hay un solo niño, entonces, luego de que él se mueva aleatoriamente, una de las casillas de la cuadrícula anterior que esté vacía puede haber sido ensuciada. Si hay dos niños se pueden ensuciar hasta 3. Si hay tres niños o más pueden resultar sucias hasta 6. Los niños cuando están en una casilla del corral, ni se mueven ni ensucian. Si un niño es capturado por un Robot de Casa tampoco se mueve ni ensucia.

- Robot de Casa: El Robot de Casa se encarga de limpiar y de controlar a los niños. El Robot se mueve a una de las casillas adyacentee, las que decida. Solo se mueve una casilla sino carga un niño. Si carga un niño pude moverse hasta dos casillas consecutivas. También puede realizar las acciones de limpiar y cargar niños. Si se mueve a una casilla con suciedad, en el próximo turno puede decidir limpiar o moverse. Si se mueve a una casilla donde está un niño, inmediatamente lo carga. En ese momento, coexisten en la casilla Robot y niño. Si se mueve a una casilla del corral que está vacía, y carga un niño, puede decidir si lo deja esta casilla o se sigue moviendo. El Robot puede dejar al niño que carga en cualquier casilla. En ese momento cesa el movimiento del Robot en el turno, y coexisten hasta el próximo turno, en la misma casilla, Robot y niño.

El objetivo del Robot de Casa es mantener la casa limpia. Se considera la casa limpia si el 60 % de las casillas vacias no están sucias.

2 Principales ideas

Como objetivo principal del problema está el mantener la casa limpia, sin embargo, para que un robot limpie una casilla con suciedad este debe emplear un turno, y los niños que son los encargados de generar la suciedad, la pueden generar el mismo turno en que se mueven, incluso, si se encuentran varios niños lo suficientemente cerca, pueden generar más de una instancia de suciedad el mismo turno, por lo que como primera observación tenemos que es mejor llevar los niños al corral, donde no son capaces de generar suciedad, y luego limpiar la suciedad en el entorno que va a ser fija.

Como segunda observación tenemos que, dada la suposición de que un niño al ser cargado por un robot no va a generar suciedad, y este robot mientras carga un niño puede limpiar, es más factible limpiar la suciedad que se encuentra en el camino hacia el corral más cercano del niño que llevar al niño y luego regresar a limpiar la suciedad, puesto que limpiar la suciedad estando en el mismo lugar tomaría solo un turno.

Siguiendo estas primeras observaciones diseñamos un conjunto de reglas de decisión que usarán los agentes posteriores.

- 1.1 - El robot está cargando un niño y la celda en la que se encuentra es un corral ⇒ Dejar niño en el corral.
- 1.2 - El robot está cargando un niño y la celda actual está sucia ⇒ Limpar la celda actual.
- 1.3 - El robot está cargando un niño ⇒ Llevar al niño al corral vacío más cercano.
- 1.4 - El robot no está cargando un niño y no hay niños fuera del corral y la celda actual está sucia ⇒ Limpiar la celda actual.
- 1.5 - El robot no está cargando un niño y no hay niños fuera del corral y existen celdas sucias ⇒ Moverse hacia la celda sucia más cercana.
- 1.6 - El robot no está cargando un niño ⇒ Moverse en dirección al niño que más rápido pueda llevar a un corral vacío.

Para esta última regla debemos analizar cómo evaluar cuál es el niño que más rápido puede llevar a un corral vacío, ya que este no tiene por qué ser el niño más cercano al robot, ni tampoco tiene por qué ser el niño que más cerca se encuentre de un corral, sino que es el niño que minimiza la suma de estas dos distancias, en la sección de implementación se especifica cómo implementar de manera eficiente este valor para cada niño.

3 Modelos de Agentes considerados

3.1 Agente Reactivo

El funcionamiento de estos agentes consiste en recibir determinada información del entorno, llamada percepción, y ejecutar una acción a partir de una función que reciba la percepción. La principal ventaja de estos agentes está en que pueden cambiar el comportamiento óptimo si el entorno cambiar de manera brusca, pues no se aferra en todo momento a un objetivo previamente definido, sino que va tomando acciones a medida que cambiar el entorno. En nuestro caso debido a que el agente no tiene información del pasado para tomar las decisiones actuales, este lo podemos considerar como un agente puramente reactivo.

Como desventaja de este método tenemos que las decisiones que toma cada agente son individuales, o sea, no dependen de qué decisiones hayan tomado los demás agentes. Esto pudiera llevar a serios problemas de optimización puesto que en los casos donde exista un único niño, todos los agentes tomarían la decisión de llevar al mismo niño al corral, y solo cuando uno de ellos lo cargue los demás no dejarán de considerar esta opción como "buena", cuando lo óptimo en estos casos sería que un solo agente lleve al niño al corral y el resto se dedique a limpiar la suciedad.

4 Agente Inteligente

Como vimos en el comportamiento del agente anterior, la principal dificultad de este es la falta de comunicación con los demás agentes, como cada uno actúa de manera individual eso puede llevar a tomar malas decisiones si los miramos a todos como un grupo. Para potenciar este comportamiento de trabajo en equipo decidimos implementar un modelo basado en un agente inteligente.

En nuestro agente inteligente, cuando un robot toma determinada decisión, este la guarda en un estado interno, y como el entorno es de información completa, cada robot puede saber qué quieren hacer los demás a la hora de tomar una decisión.

En este nuevo modelo el esquema de decisión no cambia, solo cambiar la manera en la que los niños y la suciedad están disponibles en base a la decisión que toma un robot. Por ejemplo, en nuestro agente inteligente al considerar el punto 1.6, no busca los niños que ya se encuentren en corrales puesto que no son niños disponibles, pero sí podrá buscar niños que ya están siendo buscados por otros agentes. Esto no pasa en este nuevo modelo, puesto que si un agente marca como objetivo que va a cargar a determinado niño, este deja de aparecer en la lista de niños disponibles para los demás robots. Esto permite que las acciones que tomen los robots no se intercepten y por tanto puedan hacer diferentes tareas al mismo tiempo, lo que mejora el aprovechamiento del tiempo a mediano y largo plazo.

5 Ideas seguidas para la implementación

Para poder tomar las siguientes decisiones:

- Llevar al niño al corral vacío más cercano.
- Moverse hacia la celda sucia más cercana.
- Moverse en dirección al niño que más rápido pueda llevar a un corral vacío.

Debemos tener apoyo de algún método por el cual podamos saber el camino mínimo a un conjunto de celdas en la matriz a partir de determinada celda inicial. Si tomamos la matriz como un grafo bidireccional en donde las celdas son los vértices, y existe una arista entre un par de vértices si y solo si estos vértices son adyacentes, o sea, comparten un lado en común. Entonces nos encontramos en presencia de un grafo bidireccional en el cual moverse por cualquier arista tiene el mismo coste. Para saber el camino mínimo de un vértice a un conjunto de vértices basta con hacer un recorrido BFS en el grafo e ir guardando la distancia con la que vamos visitando cada vértice. De esa manera respondemos la primera y segunda interrogante.

Para la tercera definamos $d(a, b)$ como la distancia entre el vértice a y el b. Luego $d(a, b) = d(b, a)$ puesto que el grafo es bidireccional. Lo que queremos saber es: dado un vértice a (la celda del robot), buscar un par de vértices (x, b) tal que x sea una celda de un niño libre para cargar, b sea una celda de un corral vacío, y se minimize $d(a, x) + d(x, b)$. Como $d(a, x) = d(x, a)$ y a es un valor prefijado, entonces con un simple BFS en x podemos saber $d(x, a) + d(x, b)$ iterando por los corrales vacíos en tiempo lineal. De esta manera computamos el niño que más rápido podemos llevar a un corral vacío con complejidad $O(N * C)$ siendo N la cantidad de niños y C la cantidad de corrales.

6 Ejecución

El proyecto fue desarrollado en el lenguaje de programación funcional Haskell utilizando la herramienta stack para el manejo de dependencias y el scaffold del proyecto, para ejecutar el código basta con escribir

```
stack run
```

Esto correrá la simulación con los valores predeterminados, los valores pueden ser cambiados desde el archivo `app/Main.hs`. Los valores admisibles para cambiar son

- n: Cantidad de filas de la matriz.
- m: Cantidad de columnas de la matriz.

- children: Cantidad de niños en el entorno.
- robots: Cantidad de robots en el entorno.
- obstacles: Cantidad de obstáculos en el entorno.
- dirt: Cantidad de suciedad inicial en el entorno.
- timeToShuffleEnv: Cantidad de iteraciones para el cambio de entorno aleatorio.
- seed: Semilla inicial (útil para la generación de números aleatorios).

En cada turno se imprime la matriz con las entidades que se encuentran en esa posición siguiendo el siguiente formato:

- Robot: R
- Niño: c
- Corral: C
- Suciedad: D
- Obstáculo: O

Así como los stats para el cumplimiento del objetivo del problema

- Epoch: Número de iteraciones desde el comienzo
- Free cells: Número de celdas libres
- Dirty cells: Número de celdas sucias
- Free cells percent: Porciento de celdas libres (no sucias)

6.1 Implementación de los agentes

Los agentes se encuentran en la carpeta `src/Agents/*.hs`, todos implementan una función `getAction` la cual recibe como entrada en entorno, el índice del robot que debe tomar la decisión, así como una función generadora en caso que se quiera realizar alguna operación aleatoria y debe retornar una tupla conteniendo el nuevo entorno, la acción realizada por el agente y la función generadora actualizada.

7 Consideraciones obtenidas a partir de la ejecución de simulaciones

Para poder comparar el performance de cada modelo por separado corrimos las mismas simulaciones con cada uno de estos dos agentes y comparando los resultados concluimos que los robot inteligentes son capaces de limpiar la casa en menos turnos que los reactivos, como era de esperarse. Para una matriz de tamaño 7x7, con 7 obstáculos, 5 niños, 3 celdas sucias iniciales y 4 robots, si estos son reactivos son capaces de mantener el mínimo de 60 porciento de celdas limpias a partir del turno 44, mientras que los inteligentes lo hacen en el 32. Los reactivos lograban eliminar toda la suciedad en el turno 57, mientras que los inteligentes lo hacían en el 45. Para matrices de tamaños mayores estas diferencias son aún mayores en promedio, por lo que concluimos que el trabajo en equipo es un factor clave para resolver el problema planteado en un tiempo menor y por tanto el modelo de agentes inteligentes resultó ser el mejor planteado.