Tolerance is All You Need

Joaquín Badillo Granillo A01026364@TEC.MX
Escuela de Ingeniería y Ciencias
Tecnológico de Monterrey
Campus Santa Fe

Pablo Bolio Pradilla A01782428@TEC.MX
Escuela de Ingeniería y Ciencias
Tecnológico de Monterrey
Campus Santa Fe

Bajo la instrucción de:

Octavio Navarro Hinojosa
octavio.navarro@tec.mx

Gilberto Echeverría Furió
g.echeverria@tec.mx

1. Introducción

El rápido crecimiento urbano ha dado como resultado altas densidades poblacionales. En zonas centralizadas como la Ciudad de México, aunque la cercanía de los servicios es muy alta, la cantidad de vehículos que se trasladan por las calles ha hecho que el tiempo para llegar a dichos servicios sea muy largo en comparación con su distancia.

1.1 Propuesta de Solución

En este proyecto se consideró el problema de la movilidad urbana a partir de una simulación multiagente utilizando la librería de mesa (Kazil, Masad, y Crooks, 2020). En esta simulación, los agentes toman decisiones con un cierto grado de autonomía a partir de un conjunto de reglas finitas y sencillas, como el respetar las señales de tránsito (en particular los semáforos) y la navegación a partir de las rutas más cortas para alcanzar un destino.

Es entonces de nuestro interés observar si a partir de estas sencillas reglas y limitando el entendimiento que tienen los agentes del ambiente a pequeñas vecindades, pueden emerger comportamientos egoístas que resulten en congestionamientos. Asímismo, se planteó una simulación en la que es posible modificar la afluencia de agentes, de tal forma que se puedan observar los límites que tiene el diseño de una ciudad tras un incremento en la densidad poblacional.

2. Diseño de Agentes

Para que los agentes puedan interactuar con su entorno, se representaron calles, edificios, destinos, semáforos y vehículos como agentes.

Tanto los semáforos como los vehículos son capaces de cambiar su estado, pero únicamente los segundos tienen un objetivo: que es llegar su destino. Por esta razón, el diseño que es de nuestro interés es el de los vehículos, el cual se describe a detalle a continuación.

2.1 Capacidad Efectora (Actuadores)

Las acciones que puede realizar un vehículo están limitadas a:

1. Esperar.
2. Moverse.
3. Calcular una ruta.
4. Llegar a destino.

Esperar, moverse y llegar a su destino, son acciones sencillas y fáciles de implementar; mientras que el calculo de rutas permite una mayor libertad creativa. En esta simulación, los agentes calculan rutas a partir del algoritmo de A*, usando generalmente el cuadrado de la norma ℓ^2 como función de costos y la norma ℓ^1 como heurística. En caso de recalcular una ruta, un agente puede además usar la información de sus vecinos más próximos para incrementar el costo entre celdas. Cabe destacar que también se consideró el uso de la norma ℓ^∞ como función de costos, ya que los movimientos en el software de simulación asemejan en gran medida un tablero de ajedrez y no se observó ningún cambio en los resultados obtenidos por A*.

2.2 Percepción

Los vehículos son únicamente capaces de observar una vecindad en un radio de $\sqrt{2}$ unidades (usando la norma ℓ^2 como métrica). No obstante cuentan con un conocimiento previo de las calles que les permite calcular rutas eficientes (asumiendo que no hay otros vehículos).

2.3 Proactividad

Contemplando el objetivo principal de la simulación, que es alcanzar el destino en el menor tiempo posible, teniendo en cuenta que no son omniscientes los agentes, es decir, conocen las rutas, pero les es imposible saber si hay congestión en la ruta que seguirá. Por lo que los conductores vuelven a calcular la ruta cuando no es capaz de moverse y dependiendo de la nueva ruta adquirida es el comportamiento que sigue. Previendo que en momentos es mejor hacer una ruta absurdamente larga que esperar a que se descongestione una calle, los agentes cuentan con paciencia, la cual se reduce cada paso en el que esperan. En el momento en el que la paciencia esté por debajo de cierto rango, los agentes aceptan cualquier ruta como óptima. Lo que permite que todos lleguen a su destino evitando causar cuellos de botella.

2.4 Métricas de Desempeño

Si fuera de nuestro interés determinar el desempeño individual de un agente, una métrica relevante sería la cantidad de episodios que le tomó llegar a su destino, aunque debido a que la distancia que tienen que recorrer varía es aún mejor evaluarlos individualmente como

$$\frac{\text{Episodios Transcurridos}}{\langle \text{Origen}, \text{Destino} \rangle_{\ell^\infty}}, \tag{1}$$

donde $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\ell^\infty}$ representa la distancia de Chebyshev entre 2 puntos $\boldsymbol{x}$, $\boldsymbol{y}$.

No obstante, como es de nuestro interés determinar el rendimiento de los agentes a un nivel global, es más razonable evaluar el rendimiento que tienen de forma colaborativa y como responden a cambios en el flujo de entrada de agentes al sistema. Por esta razón se decidió que para un episodio en particular t, la métrica de desempeño R sea el número de agentes totales que han llegado a su destino

$$R(t) = \text{Número de llegadas después de } t \text{ episodios.} \tag{2}$$

Evidentemente si se tiene un mayor flujo de agentes y estos logran colaborar de forma efectiva, se espera que más agentes lleguen a sus destinos en un menor número de episodios. También cabe destacar que la métrica de desempeño es más interesante cuando la simulación llega a su condición de cierre, que es cuando se deben agregar agentes pero ninguna esquina del mapa (que es donde aparecen) está disponible.

Para poder variar el flujo de entrada en la simulación, esta cantidad se representó como un parámetro de línea de comandos (o una variable de entorno) para el servidor, que determina el número de episodios que debe transcurrir antes de volver a agregar agentes. Asimismo es posible especificar dicho valor desde la ruta `/init` del API, pasando el parámetro `cycles` desde un formulario web.

3. Arquitectura de Subsunción

	Evento	**Condiciones**	**Acción**
7	Nada	Nada	Calcular ruta
6	Seguir ruta	Celda disponible	Moverse siguiendo ruta
5	Seguir ruta	Celda ocupada por otro coche, es una diagonal y la paciencia es menor o igual a 0	Recalcular ruta (actualizando costos)
4	Seguir ruta	Celda ocupada por otro coche, es una diagonal y la paciencia es positiva	Esperar (restando paciencia)
3	Seguir ruta	La paciencia es negativa y llegó a un límite	Intentar moverse a una celda disponible aleatoria y recalcular ruta
2	Seguir ruta	Celda disponible (es su destino)	Llegar a destino
1	Seguir ruta	Celda disponible	Moverse
0	Seguir ruta	En un cruce con semáforo en estado rojo	Esperar

Tabla 1: Arquitectura de subsunción para los coches

4. Características del Ambiente

El espacio en el que viven los agentes es discreto y finito, el cual se representa con un tablero de $M \times N$ casillas,y este se crea a partir de un documento de texto, en el que se determinan los sentidos de las calles, las posiciones de los semáforos y los destinos a partir de una simbología previamente determinada.

El ambiente es relativamente inaccesible, ya que aunque los agentes son capaces de obtener rutas eficientes (en términos de distancia) y por ende tienen un cierto conocimiento del mapa, no son capaces de predecir el comportamiento ni la ubicación de otros agentes (a excepción de sus) vecinos más próximos.

El ambiente es un tanto estático; ya que aunque se introducen, eliminan y se mueven agentes vehiculares, los semáforos y los destinos no cambian de posición. Asimismo, el ambiente es no determinista, ya que el orden en el que los agentes toman las decisiones cambia entre simulaciones (debido al uso del `RandomScheduler` de mesa) al igual que por ciertas variables aleatorias[1] que se utilizan para definir la tolernacia de los agentes.

5. Controlador de Agentes

Para la parte gráfica de la simulación se utilizó Unity como herramienta, en donde se hizo la visualización de todos los agentes como vehículos puntualmente utilizando wavefronts (obj) como los modelos utilizados. El control de los agentes se hizo en su totalidad con la librería Mesa de Python, conectando la funcionalidad de la librería con Unity a través de un servidor de Flask. La forma en la que esto funciona es que se le brinda a Unity la capacidad de hacer peticiones al servidor que con el script AgentController, que es el responsable de recibir la posición de todos los agentes como existan en Mesa, para que con los scripts de movimiento, desplace los agentes con transformaciones de matrices, otra función del script es hacer que se instancíen los objetos en Unity para poder controlarlos, así como destruirlos cuando lleguen a su destino.

6. Resultados y conclusiones

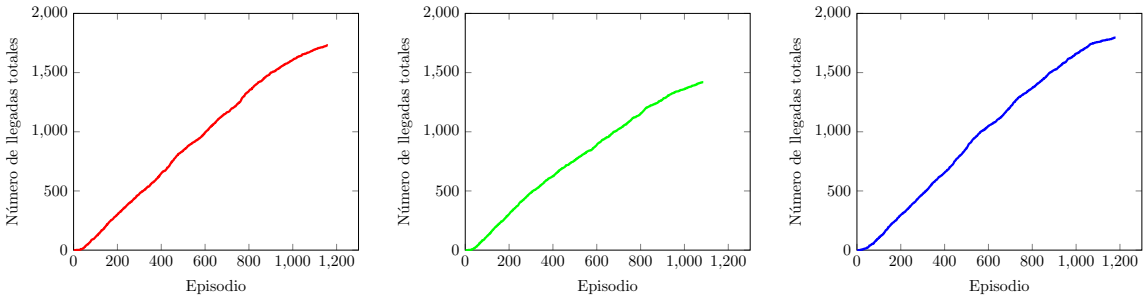
Originalmente se consideró una arquitecutra de subsunción en la que los agentes recalculaban una ruta cada vez que un vehículo obstruía la celda a la que se querían mover. Sin embargo, debido a que en el mapa había una glorieta, algunos agentes se notaban impacientes por desplazarse y por lo tanto se metían al carril interno. Como los agentes propuestos no cuentan con direccionales, a veces les resultaba imposible llegar a su salida ya que nadie cedía el paso en favor de su proactividad y con una afluencia alta de agentes terminaban en un *gridlock*. Después de un bloqueo de la rotonda, la simulación sobrevivía unas cuantas decenas de episodios más antes de que todas las calles quedaran congestionadas y que consecuentemente se llegara a la condición de cierre.

El bloqueo entre los vehículos dio pauta al principal descubrimiento de esta investigación, la tolerancia es todo lo que se necesita. Pues resultó evidente con la rotonda que cambiarse de carril no siempre es la mejor decisión y si no se introducen algoritmos de aprendizaje

1. En realidad son pseudo-aleatorias dado que son números generados computacionalmente y por ende son replicables con cierto conocimiento de la semilla y la secuencia.

que le permitan a los agentes determinar cuándo es conveniente cambiar de carril, resulta eficiente que esperen unos episodios antes de tomar la decisión. Además, para evitar que muchos coches se cambien de carril de forma simultánea, se generaron sus límites de tolerancia de forma pseudo-aleatoria y en caso de que este límite de paciencia se rebase, se tiene un segundo límite más agresivo con el que un vehículo preferirá moverse a cualquier espacio disponible: no solo cambiando su ruta por completo; sino también sacrificando su proactividad en favor del rendimiento de la simulación en su totalidad. Esta decisión, aunque parece ser irracional podría en realidad confundirse por un comportamiento inteligente, como si los agentes pudieran entender la existencia de un equilibrio de Nash.

Una vez realizados estos cambios y trás la observación de su rendimiento desde el visualizador de `mesa`, se decidió correr la simulación múltiples veces como proceso en segundo plano (agregando agentes cada 2 episodios) y escribir los resultados a un archivo para encontrar patrones en su rendimiento (ver Figura 1). La principal observación que se obtuvo de estos resultados fue que inicialmente la llegada de los agentes a sus destinos es rápida y casi lineal, pero eventualmente se estanca mostrando un comportamiento asintótico y dando fin a la simulación.



(a) Primera Simulación (b) Segunda Simulación (c) Tercera Simulación

Figura 1: Resultados agregando agentes cada 2 episodios y usando semáforos con tiempos de 10s entre estados

Referencias

Kazil, J., Masad, D., y Crooks, A. (2020). Utilizing python for agent-based modeling: The mesa framework. En R. Thomson, H. Bisgin, C. Dancy, A. Hyder, y M. Hussain (Eds.), *Social, cultural, and behavioral modeling* (pp. 308–317). Cham: Springer International Publishing.