Planificación de rutas cooperativas

David Quirós Pérez

Resumen—Actualmente se utilizan vehículos de guiado automatizado (AGV) en almacenes industriales para la realización de tareas de transporte. El software que corren estos AGVs está pensado para trabajar en entornos no cambiantes o para seguimiento de rutas prefijadas. En este proyecto buscaremos soluciones para que un grupo de AGVs pueda trabajar en entornos cambiantes de forma cooperativa evitando conflictos (que coincidan en espacio y tiempo). De entre las soluciones que encontraremos, implementaremos una en el entorno de simulación NETLOGO. La solución elegida ha sido el Cooperative A*, un algoritmo derivado del clásico A*. Dicho algoritmo nos devolverá la ruta que seguirá cada AGV para realizar su tarea. Presentaremos algunos ejemplos de ejecución dónde podremos ver dichos resultados, aunque el algoritmo no es completo ya que hay un tipo de conflicto que no resuelve y no ha habido tiempo suficiente para implementar la solución.

Palabras clave—AGV, Planificación de rutas, Inteligencia artificial, NETLOGO, Cooperativo, Simulación.

Abstract— Nowadays, automated guided vehicles (AGVs) are used in industrial warehouses for performing transport tasks. The software running these AGVs is designed to work in static environments or to follow pre-set routes. This project will seek solutions for a group of AGV to work on changing environments cooperatively avoiding conflicts (coincidence on space and time). Among the solutions we find, we will implement one in the simulation environment NETLOGO. The chosen solution was the Cooperative A *, an algorithm derived from the classic A *. This algorithm will return the route to be followed for each AGV to perform its task. We present some embodiments where we see these results, although the algorithm is not complete as there is a kind of conflict not resolved and there has not been enough time to implement the solution.

Index Terms—AGV, Pathfinding, Artificial intelligence, NETLOGO, Cooperative, Simulation.

1 Introducción

En el ámbito industrial se utilizan AGVs (vehiculos de guiado automático) para realizar tareas de transporte, como por ejemplo llevar cajas de un sitio a otro. Actualmente el software que corren estos AGVs está diseñado para entornos no cambiantes o rutas prefijadas, pero en investigación encontramos software pensado para AGVs que se muevan en entornos cambiantes. De esta forma los AGVs pasan a llamarse agentes inteligentes. El objetivo de estos agentes es cumplir una orden de transporte, dada la localización de recogida y destinación del material los agentes deberán responder con el coste que tendrán para resolverla. Después de forma centralizada se asignan dichas órdenes a los agentes correspondientes.

En el caso que planteamos en este proyecto, nos encontramos en un sistema multiagente donde cada agente deberá planificar la ruta a un objetivo teniendo en cuenta las posibles rutas que tomaran los otros. Habrá casos en los que algun agente deberá esperar en su posición para dejar pasar a otro o cambiará de camino para evitar el conflicto. El sistema multiagente, dado una lista de tareas para los agentes, deberá encontrar rutas individuales que minimicen el coste global de ejecución de dichas tareas y también reducir el número de conflictos (coincidecia temporal y espacial entre agentes).

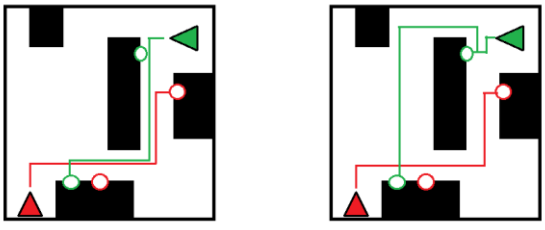

Figura 1: Representación de una posible solución.

- *E-mail de contacto: david.quirosp@e-campus.uab.cat*
- *Mención realizada: Enginyeria de Computadors.*
- *Treball tutoritzat per: Màrius Montón Macián (Microelectrònica i sistemes electronics)*
- *Curs 2014/15*

2 OBJETIVOS

A continuación se listan los objetivos definidos para este proyecto:

2.1 Objetivo principal

- Implementar un algoritmo de planificación de rutas cooperativas en el entorno de simulación NETLOGO.

Como fase inicial para realizar este objetivo tenemos que decidir algun algoritmo a implementar, para ello se hará una búsqueda de información para ver las soluciones existentes actualmente sobre este tema. Más adelante lo comentaremos en el estado del arte.

La siguiente fase será implementar la solución elegida. El entorno donde se hará dicha implementación será NETLOGO [4], es un entorno de modelado programable para la simulación de fenómenos naturales y sociales. Esta plataforma permite a los estudiantes hacer simulaciones abiertas y "jugar" con ellas, explorando su comportamiento bajo diversas condiciones. También es un entorno de creación que permite a los estudiantes, profesores y desarrolladores crear sus propios modelos. NETLOGO es bastante simple, pero lo suficientemente avanzado como para servir de poderosa herramienta para los investigadores en muchos campos.

Nosotros partiremos del modelo ya creado por Lluís Ribas, dicho modelo representa un almacén y los AGVs que lo habitan, con los elementos necesarios para la simulación.

2.2 Objetivo secundario

- Llevar la simulación del algoritmo implementado a los robots reales y realizar pruebas en vivo.

En el caso ideal de que el primer objetivo se lograse con resultados satisfactorios se intentará llevar la simulación a robots reales.

3 ESTADO DEL ARTE

La planificación de rutas es el problema clásico de inteligencia artificial, dónde el famoso algoritmo A* (Hart, Nilsson, & Raphael 1968) es el que se encarga de dar solución. El problema es que el A* está pensado para un único agente en entornos no cambiantes.

En [1][2], podemos ver como sobrepasar dichos límites del A*. David Silver expone varios algoritmos derivados del clásico A*. Una primera modificación sería el Local Repair A* dónde cada agente traza su ruta teniendo en cuenta los movimientos de sus vecinos inmediatos. Más adelante expone otras versiones que son el Cooperative A*, Hierarchical Cooperative A* y Windowed Hierarchical Cooperative A*. La gracia de estos algoritmos es que se añade la llamada “tercera dimensión”, el tiempo. En los casos que presenta David Silver los agentes trabajan sobre un espacio bidimensional. En éste tablero los agentes se desplazan con cuatro acciones que son moverse al norte, sud, este u oeste. Al añadir la tercena dimensión del tiempo también se les añade una quinta acción, la de esperar. Por ejemplo, si el tablero típico esta definido por las dimensiones espaciales Cell [x y], ahora tendremos Cell [x y t]. Si nos movemos al norte pasaremos a Cell [x, y+1 t+1], si decidimos esperar Cell [x y t+1]. David Silver también trata el tema de cuál es la mejor heurística a utilizar, de las cuales la básica y más utilizada es la distancia de Manhattan. Éste tipo de heurística ofrece un bajo rendimiento para escenarios complejos, pero para los casos con los que vamos a trabajar es admisible.

¿Pero como hace para que los agentes cooperen? Una tabla de reservas compartida. En dicha tabla se guardaran las posiciones que ocuparan los agentes y el tiempo en el que las ocupan, de esta forma cada vez que se quiera expandir un camino, primero se comprobará si a donde queremos ir está ocupado en ese momento. Además, a estos agentes se les asigna un orden de preferencia para poder resolver los posibles conflictos, por ejemplo, si el agente T1 y T2 se van a cruzar en el punto X en el siguiente tiempo, uno de los dos deberá replanificar su ruta (el que tenga menos preferencia). El problema de esto es que si el número de agentes es grande en comparación con las posibles rutas que se puedan tomar, los agentes con prioridades bajas probablemente no puedan alcanzar sus objetivos.

Los autores en [5] explican otra forma de modificar el A* para resolver el problema. En este caso trabajan con un grupo de agentes mayor al caso expuesto por David Silver. Separan los agentes en grupos, cada grupo de agentes tiene una ruta planificada que evita conflictos con los otros agentes del mismo grupo, para este paso utiliza un algoritmo llamado Operator Decomposition (OD). Cuando tiene dos grupos de agentes busca una replanificación de las rutas para uno de los grupos que evite las rutas del otro con el algoritmo Independence Detection (ID). Si no consigue encontrar alguna alternativa para los dos grupos los une y replanifica las rutas de los agentes internos con OD. La idea de hacerlo así es que el algoritmo OD es muy costoso en cómputo y para aligerar hace esta separación en grupos de agentes que puedan ejecutarlo de forma paralela.

En [3], se expone The Differential Evolution Algorithm (DE), un algoritmo genético que utiliza tanto una solución centralizada como distribuida del problema. El resultado de su experimento fue que la solución distribuida tiene un mayor rendimiento ya que el tiempo de computo de la versión centralizada es mayor a casua de que un único DE se encarga de manejar a todos los robots (posiciones de robots, posiciones destino, planificación de rutas para cada uno evitando colisiones con obstáculos u otros robots, etc). En cambio, de forma distribuida se puede separar en sub-tareas, cada cual ejecutada por un DE. La gracia esta en que cada DE se puede ejecutar de forma paralela.

4 DESARROLLO REALIZADO

Estas son las tareas que se han realizado durante el proyecto:

4.1 Elección del algoritmo a implementar

Como se ha descrito anteriormente en la descripción de objetivos, la idea de este proyecto es la de implementar un algoritmo que solucione el problema de la planificación de rutas cooperativas basándonos en la infomación que encontremos. En el estado arte se han mencionado varias ideas para posibles algoritmos, entre las cuales adoptaremos las de David Silver [1][2]. El motivo principal de esta elección es básicamente por viabilidad temporal, los algoritmos Cooperative A* se encuentran dentro del alcance al que puede llegar este proyecto.

El siguiente paso, antes de comenzar la implementar el algoritmo en el entorno de NETLOGO es el de montar el esqueleto en pseudocódigo:

```
function cooperative_a* (map,car,command,reserve)
  result.append( car.position, car, time=0, cost=0 )
  while result is not NULL
    current = result.pop()
    if current.position = command.goal
      break while
    endif
    for neighbor in current.neighborhood
      if check_reserve(reserve,neighbor.position, current.time+1)
        cost=current.cost+distance_cost(neighbor,command.start)
        result.sort-append( map[neighbor], car, time + 1 , cost)
        reserve.input( reserve, car, neighbor.position, car.time +1)
      endif
    endfor
return result
```

Esta fue la primera idea para el algoritmo, tenemos un A* clásico con algunos añadidos. Primero tenemos que cada nodo del resultado no solo contiene las posiciones de la ruta resultante y el coste, sino que también guardamos el tiempo que lleva. Y segundo se añade el concepto de la tabla de reservas donde a cada movimiento se comprueba si dicha posición en el siguiente tiempo está ocupada.

Esta versión sufrió bastantes modiciaciones durante la implementación por diversos temas que no se tenían en cuenta en ese momento, mas adelante se explican estos casos.

4.1 Familiarización con NETLOGO

El siguiente paso era comprender el funcionamiento de este entorno de simulación. El tablero de NETLOGO se divide en varias capas: la capa base son los patches, agrupaciones de pixeles para identificar distintos puntos del tablero. Por encima de los patches se colocan las tortugas, que son lo que se conocería como clases. Por ultimo está el observador, que puede dar órdenes a las tortugas. El porqué de este nombre se puede ver en la referencia [4]. En la siguiente imagen se muestra la interfaz principal:

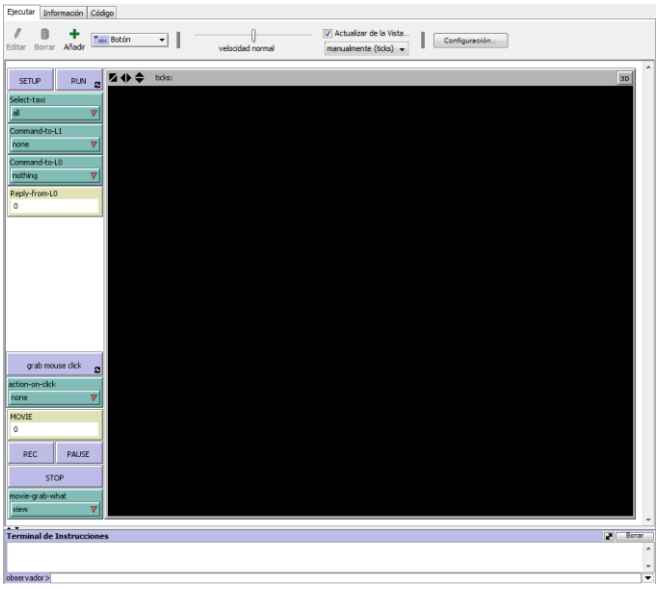

Figura 2: Interfaz principal de NETLOGO

Trabajamos sobre un entorno ya creado por Lluís Ribas. Este entorno posee los elementos necesarios para la simulación del comportamiento de un almacén manejado por AGVs. En el botón de SETUP se llamaran a todas las funciones destinadas a crear el entorno, en este caso serán los modelos de las estanterías, cajas y el mapa topológico de dicho almacén. El botón RUN para iniciar la simulación.

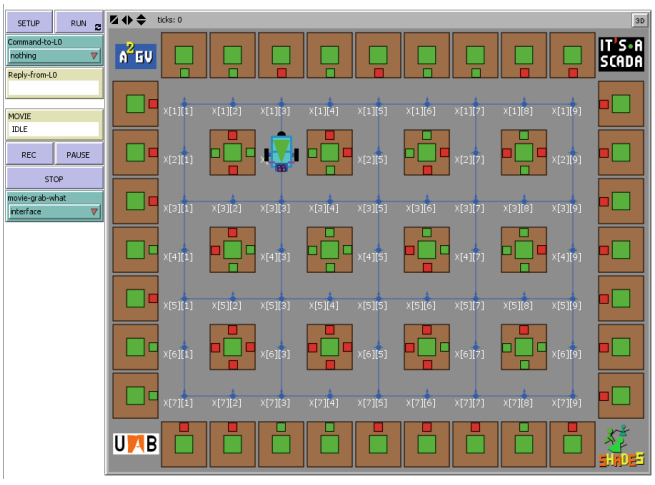

Figura 3: Representación del almacen sobre el que vamos a trabajar.

En esta imagen podemos ver los elementos nombrados anteriormente, las líneas y puntos azules representan el mapa topológico (grafo) sobre el que los AGVs planificarán sus rutas. Tanto el grafo como el taxi (agente) son tortugas, las cuales tienen definidas una serie de atributos. En NETLOGO las tortugas (clases) y los métodos no están relacionados como en Java u otros lenguajes orientados a objeto, es decir, cualquier tortuga podrá llamar a cualquier método. Por esto hay que ser cuidadoso en como se hacen los métodos, para quien están pensados. Por ejemplo, las tortugas no pueden dar órdenes a otras torugas, solo puede el observador, si dicho método da

ordenes a otras tortugas y se lanza desde una tortuga petará. Se ha decidido trabajar sobre este escenario porque ofrece mas oportunidades para conflictos ya que las calles entre estanterías son estrechas y sólo puede pasar un AGV al mismo tiempo. A continuación veremos como se ha llevado a cabo la implementación del Cooperative A*.

4.1 Implementación

El primer paso fue implementar el A* clásico para un único vehiculo en el entorno, la heurística implementada para reducir el numéro de nodos explorados es la típica de la distancia real entre dos puntos. Esta heurística se mantiene también para cuando trabajemos con más taxis a la vez.

Una vez verificado el correcto funcionamiento de este A* comenzaremos con los cambios para transformarlo en un A* cooperativo. Antes de comenzar hay que recalcar que trabajamos a nivel observador, es decir, las rutas se planificaran de forma centralizada. Tal y como teníamos el entorno hasta ahora el único elemento que faltaba para poder realizar la planificación cooperativa era la tabla de reservas que nombraba David Silver en sus artículos. Él trabajaba sobre grids, por eso utilizaba una tabla. En nuestro caso estamos trabajando sobre un grafo, en lugar de utilizar una tabla decidimos utilizar los arcos del grafo para guardar los tiempos en los que eran cruzados por los agentes. Cada vez que expandimos un nodo del A* reservamos todos los arcos de esa posición para que no puedan entrar otros taxis en ese tiempo.

En la primera versión que se implementó las rutas se planificaban de forma secuencial, es decir, primero hacíamos la ruta para el Taxi1, luego el Taxi2, etc. Esta fue una primera versión para comprobar que sistema de gestión de reservas funcionase. En este caso el primer taxi, al ser el primero en planificar su ruta tenia todo el mapa libre, por lo cual siempre resultaba en el camino óptimo. El segundo taxi ya se podía encontrar con conflictos con las reservas del primer taxi, el tercer taxi con las reservas del primero y segundo, etc. El orden en el que se planificaba la ruta para cada taxi determinaba su prioridad sobre los demás (la jerarquía). A pesar de que estamos trabajando en una versión centralizada, la idea es que fuese fácilmente exportable a una versión distribuida donde cada taxi pueda planificar su propia ruta junto a los demas. Por esto modificamos el flujo del código para que a cada tiempo se expandiera un nodo para cada taxi. Aun así, al hacerlo de forma secuencial se sigue manteniendo una prioridad entre taxis en función de cual reserva primero a cada tiempo.

Al hacer este cambio se vió que nos hacía falta saber el movimiento siguiente que harían los demás para poder ver sus intenciones y evitar los conflictos. A continuación mostraremos los conflictos con los que nos encontramos:

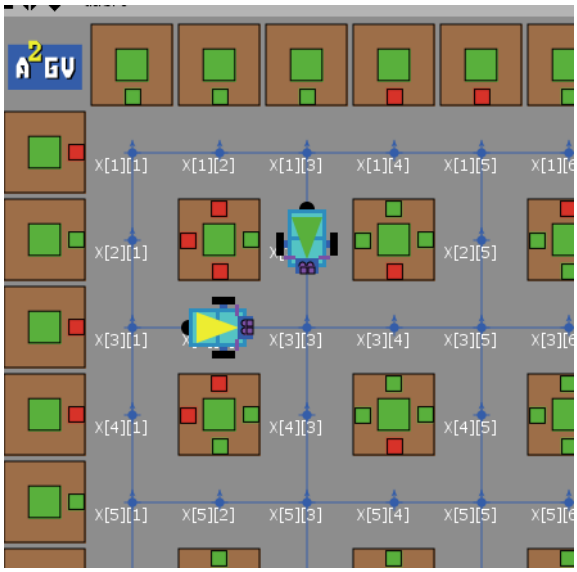

Figura 4: Primer conflicto tratado.

El taxi verde quiere ir hacia abajo y el amarillo hacia la derecha, pero en su siguiente movimiento coincidirán en el nodo X[3][3]. Suponiendo que el taxi verde reserva primero, quien que tendrá problemas para seguir planificando su ruta será el amarillo. Cuando el taxi amarillo intente expandir el nodo verá que el enlace entre su nodo actual y el siguiente esta reservado para el siguiente tiempo por otro taxi, por lo cual lo descartará "temporalmente". En estos casos lo ideal es que el taxi amarillo permanezca inmóvil durante un tiempo para poder continuar su ruta. Es decir, si los enlaces del nodo al que quiero ir están reservados para el siguiente tiempo, en lugar de expandir otra ruta posible, repetimos el mismo nodo en el que nos encontramos durante una iteración. A la siguiente iteración, al haber aumentado el tiempo ese enlace estará libre y podrá continuar su ruta. Pero aquí falta por contemplar otro caso que se ve en el siguiente conflicto tratado.

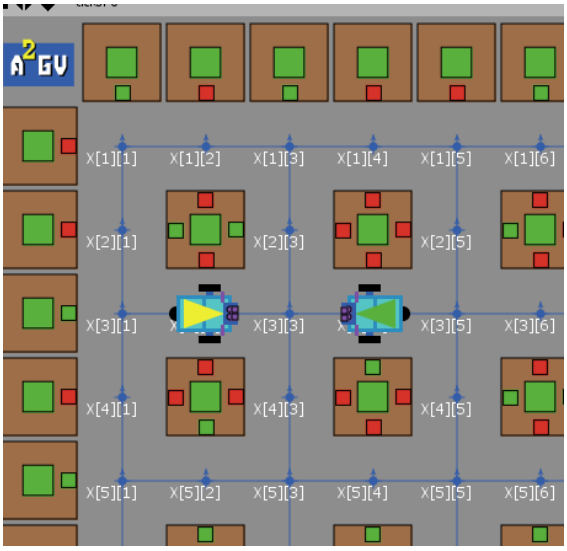

Figura 5: Segundo conflicto tratado.

En este segundo conflicto nos damos cuenta de la necesidad de preveer las intenciones de los otros taxis. En este caso el taxi verde quiere ir en la dirección opuesta al amarillo, mantenemos la priorioridad del taxi verde. Tal y como habíamos resuelto para el caso anterior, cuando el amarillo quiera avanzar verá el enlace reservado y repetirá nodo para simular una espera. El problema está en que ahora se van a cruzar ya que para los siguientes tiempos esos links están libres y resultará en un fallo de la planificación. La primera idea para resolver este conflicto fue reservar los nodos vecinos del que se evalua para un tiempo mas adelante, de esta forma ahora podemos ver las intenciones de los otros en el siguiente tiempo. Por desgracia sigue sin ser suficiente ya que para otros casos nos damos cuenta de que por ejemplo nos habría hecho falta dos tiempos más de previsión, o tres o más. Para que esta solución funcionase haría falta tener una previsión tan larga como el camino más largo posible, lo cual es una pérdida de tiempo de cómputo.

A parte del problema anterior también nos dimos cuenta de que guardar las reservas en los enlaces es innecesario, para ello también podemos usar los propios nodos, menos cálculos. Por lo tanto, para solucionar este conflicto decidimos cambiar las reservas a los nodos y la forma de comprobar su estado. Ahora, en lugar de comprobar el estado de la reserva los posibles nodos siguientes comprobamos el nodo actual para el tiempo actual. Por ejemplo, para los casos como el primer conflicto que hemos tratado antes (figura 4): el coche verde reserva primero el nodo en disputa, para ese mismo tiempo llega el taxi amarillo. Una vez el amarillo llegue comprobará si en ese mismo nodo y en ese mismo instante hay algun otro taxi, si la respuesta es positiva borramos el nodo actual de la ruta resultante y repetimos el anterior de la misma forma que hacíamos antes, la diferencia es que hay que corregir el coste acumulado también.

Para los casos del segundo conflicto añadiremos la tabla "tabú". En la función de comprobar la reserva ahora también comprobará los nodos vecinos del nodo actual. Lo que comprobará en los nodos vecinos será si hay algun otro taxi para ese tiempo, si encuentra alguno comprobará en las reservas del nodo actual si ese taxi se encuentra para el siguiente tiempo en su posición. De ser así, para evitar el conflicto lo que se ha de hacer es cortar ese enlace. Ahora cada taxi tendrá su propia lista de tabus dónde se almacenarán los enlaces cortados por tal de evitar esta clase de conflictos. Esta lista se comprobará a la hora de buscar nodos vecinos, si el enlace con este vecino esta cortado no puedo visitarlo, por la esencia del propio A* buscará una ruta alternativa.

5 Resultados

El objetivo del algoritmo es, dada un serie de tareas para los agentes del sitema, devolver las rutas óptimas evitando conflictos para resolverlas. Como se ha explicado anteriormente, para planificar dichas rutas, trabajamos sobre el mapa topológico (grafo) que define los cominos que pueden tomar los agentes. Los costes de las rutas que nos siven para definir si una ruta es mejor que otra ha sido la distancia recorrida, cada vez que expandimos un nodo de la ruta aumentamos el coste acumulado por la distancia del enlace que ha cruzado más el coste prefivisto para llegar al objetivo (heurística), tal y como hemos visto anteriormente, esta previsión es el cálculo de la distancia en línea recta entre la posición acutal y posición objetivo. Respecto a evitar conflictos, hemos utilizado el tiempo. Cada vez que avanzaba un tiempo la planificación los agentes guardan en los nodos el tiempo en el que han pasado para advertir a los demás agentes que ese nodo en ese momento estará ocupado. A su vez, el primer paso para evaluar si los agentes pueden ir a un nodo será comprobando que no haya otro agente en ese momento.

Al final el algoritmo devolverá una lista de nodos para cada agente. Esta lista de nodos representa el camino que seguirán. A parte, ya que estamos utilizando un entorno de simulación, para mostrar la solución de una forma más visual se han pintado los nodos de las rutas de un color para cada agente.

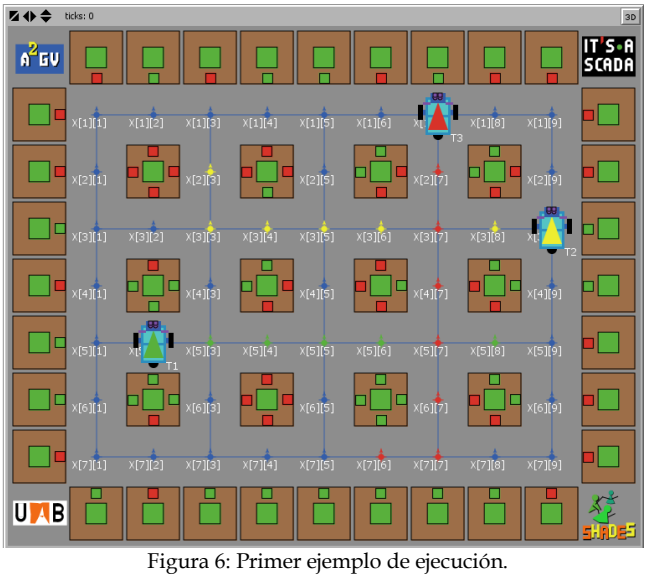

Figura 6: Primer ejemplo de ejecución.

En la imagen mostramos una ejecución con tres agentes, como dijimos antes los agentes tienen prioridades, en este caso son Verde > Amarillo > Rojo. Al taxi verde le hemos pedido que haga una ruta hasta el nodo X[5][8], el amarillo hasta X[2][3] y el rojo hasta X[7][6]. Los nodos pintados representan la ruta para los respectivos agentes. Las listas de nodos que ha devuelto para cada taxi són:
-Taxi Verde: [(node 157) (node 158) (node 159) (node 160) (node 161) (node 162) (node 163)]
-Taxi Amarillo: [(node 150) (node 149) (node 148) (node 147) (node 146) (node 145) (node 144) (node 138)]
-Taxi Rojo: [(node 134) (node 140) (node 140) (node 148) (node 154) (node 154) (node 162) (node 168) (node 176) (node 175)]

En la imagen, únicamente con los nodos pintados no se puede apreciar si en algun momento los agentes realizan alguna espera, con la lista de nodos podemos verlo. En

este caso tanto el taxi verde como amarillo no realizan ninguna espera pero el rojo, al ser el que tiene la prioridad más baja, si que espera y en este caso dos veces. Podemos ver que en la ruta del taxi rojo se repiten los nodos X[2][7] (node 140) y el X[4][7] (node 154). En el tiempo 2 el taxi amarillo reserva antes el nodo X[3][7] por lo cual hace esperar al taxi rojo, y en el tiempo 5 le pasa lo mismo con el taxi verde en el nodo [5][7].

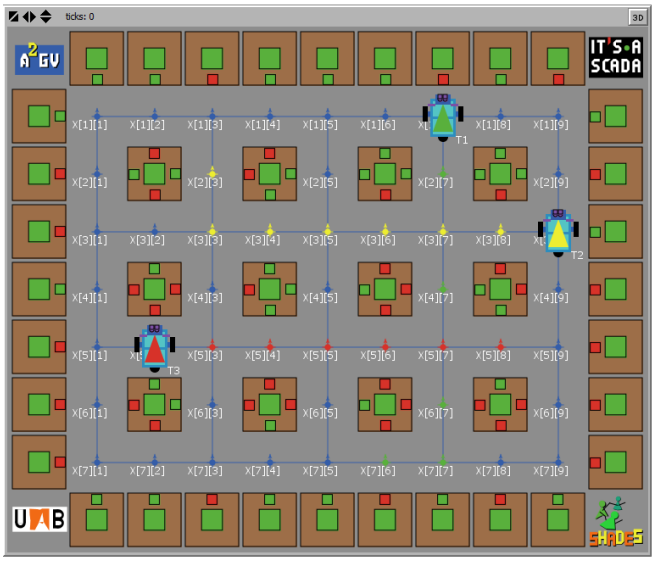

Figura 7: Segundo ejemplo de ejecución.

En este caso hemos cambiado el taxi rojo por el verde, en el resultado podremos ver como cambian las rutas a causa de las distintas prioridades.
-Taxi Verde: [(node 134) (node 140) (node 148) (node 154) (node 162) (node 168) (node 176) (node 175)]
-Taxi Amarillo: [(node 150) (node 149) (node 149) (node 148) (node 147) (node 146) (node 145) (node 144) (node 138)]
-Taxi Rojo: [(node 157) (node 158) (node 159) (node 160) (node 161) (node 162) (node 163)]

Como podemos ver ahora solo se realiza una espera entre todas las rutas, el taxi amarillo en el tiemo 2 en el nodo X[3][8] (node 149). Viendo esto nos damos cuenta de que la forma de repartir las prioridades afecta al resultado global del sistema, en el caso que al sitema le interese mantener unas prioridades en los agentes a causa de que las tareas también las tienen este algoritmo sería suficiente. En el caso que todas las tareas fueran igual de prioritarias, entonces se deberían calcular las rutas con todas las combinaciones de prioridades posibles entre los agentes para ver cual da el mejor resultado goblal. Lo normal en los almacenes actuales es que trabajen con pocos AGVs a causa de su elevado precio, por lo cual el aumento de cómputo que requiere comparar los resultados con todas las combinaciones de prioridades es admisible.

Con estos ejemplos hemos visto los primeros conflictos que hemos tratado en el desarrollo realizado [Figura 4], ahora veremos los casos del segundo conflicto [Figura 5].

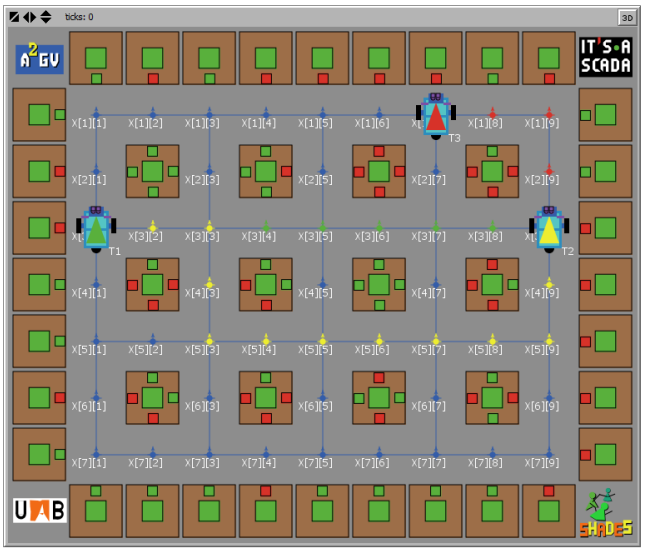

Figura 8: Tercer ejemplo de ejecución.

En este caso le pedimos al taxi verde que vaya a la posición del taxi amarillo y viceversa. Al taxi rojo lo mandamos al nodo X[2][9]. En el tiempo cuatro el taxi verde y amarillo se disputarán el nodo X[3][5], al ser el verde mas prioritario el taxi amarillo cortara el enlace entre X[3][5] y X[3][6] para cuando replanifique la ruta no tenga esa opción, de esta forma el A* lo llevará por una ruta alternativa. Como podemos ver la desviación de la ruta óptima para el taxi amarillo es considerable, pero la única solución que hay para resolver estos conflictos está en la fase de asignación de tareas.

Durante la fase de prueba de este tipo de conflictos nos dimos cuenta de que había otro tipo de conflicto que no habíamos tenido en cuenta.

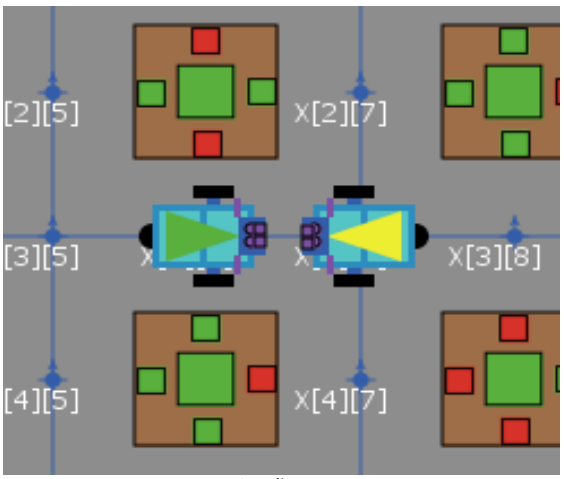

Figura 9: Conflicto sin tratar.

Este tipo de conflicto no puede resolverse de la forma en la que se planteó el escenario. Tal y como esta montado, cada vez que avanzamos en la planificación se comprueba que el nodo al que nos movemos no este ocupado por otro. En este caso, cuando los agentes avancen no verán conflicto ya que dejan libre sus posiciones para el siguiente tiempo.

Después de darle vueltas al caso para ver como solucionarlo, la solución que se nos ocurre es la de hacer backtracking de las rutas planificadas. Es decir, como no podemos solventar este conflicto sin el conocimiento de las intenciones de los otros agentes, necesitamos las rutas completas para poder hacer la comprobación de si se dan casos como estos. La comprobación se basaría en recorrer las listas de las rutas resultantes en busca de: si en mi estado actual con tiempo + 1 veo reserva de un agente que está en el nodo siguiente para este tiempo. Cada vez que se encuentre un caso del estilo habría que hacer algo similar como en la solución del tipo de conflicto 2, cortar el enlace entre esos nodos y meterlo en la tabla tabu para el agente con menos prioridad, y después relanzar la planificación para todos los agentes desde el tiempo anterior a ese conflicto. Esta solución no ha sido implementada a causa de falta de tiempo ya que la prioridad en el punto donde se encontraba el proyecto era otra, ahora la idea era hacer una simulación en condiciones haciendo que los agentes se movieran siguiendo las rutas.

Para hacer que los agentes se muevan siguiendo las rutas se ha utilizado el código que define el comportamiento de los AGVs. En el entorno que se nos facilitó para trabajar ya contenía la máquina de estados que define cada acción que puede realizar un AGV, entre los estados de esta máquina encontramos "robot-follow-path" el cual dada una lista de coordenadas polares los AGV la seguirán. Para ello se hizo una función que tradujera la lista de nodos que habíamos generado con nuestro algoritmo a dicha lista de coordenadas polares, el problema está a la hora de simular las esperas. Para ello se ha creado un nuevo estado para los AGVs que simula las esperas.

6 Conclusión

Tal y como definimos al principio del documento, el objetivo de este proyecto era el de crear un algoritmo que diese solución al problema de la planificación de rutas cooperativas. Hemos visto en la fase de desarrollo en qué consiste el algoritmo: un A* con unos cuantos elementos extra como la dimensión del tiempo, tablas de reserva y como resolver los conflictos que se crean entre los AGVs para dar con una solución realista.

El algoritmo es capaz de devolver rutas correctas en este tipo de entorno pero no es completo, como hemos visto en la parte de resultados hay un tipo de conflicto que no se planteó correctame, cuya solución requiere un tiempo que el proyecto no tiene. Así pues, el objetivo secundario del proyecto tampoco ha sido posible realizarlo.

A pesar de que hemos trabajado en una de las soluciones "sencillas" para resolver la planificación cooperativa, nos hemos dado cuenta tanto en la fase de diseño como de implementación la complejidad que tratan estos problemas. El tema de como tratar el tiempo para medir las intenciones de ocupación del espacio ha resultado complicado.

La complejidad de cómputo de este algoritmo es la misma que la del A*, depende de la heurística elegida. En el peor de los casos, con una pésima heurística, la complejidad del algoritmo es exponencial. Pero si utilizamos una heurística admisible (que no sobreestime el coste real) la complejidad puede llegar a ser polinómica. Por lo que para aplicaciones en tiempo real resultaría satisfactorio, sin olvidar que nos encontramos en un ámbito industrial dónde la cantidad de agentes es baja.

También podemos decir que el uso de la herramienta NETLOGO ha sido bastante satisfactorio. El tiempo que requiere aprender a usarlo y el tiempo requerido para montar el entorno gráfico merece la pena ya que te ayuda a identificar claramente el comportamiento de lo que se está desarrollando.

Las líneas de mejora están claramente marcadas. La primera sería dar solución al tipo de conflicto que no hemos podido tratar correctamente. Una segunda mejora sería la de transformar el código de forma que no se ejecutase en modo observador, sino que fuese a nivel de tortuga. Es decir, distribuir la tarea de planificación a cada agente teniendo en cuenta que todos deben compartir el estado del mapa topológico para que puedan tratar las reservas.

Agradecimientos

Principalmente darle las gracias a mi tutor Lluís Ribas por darme la oportunidad de utilizar su entorno para poder realizar el proyecto, a parte de los múltiples consejos para hacerlo correctamente. También agradecer a mis compañeros/amigos sus consejos y ánimos, parece que no pero ayuda. Y por último gracias a mi familia que son los que más tienen que aguantar mis tonterías.

Bibliografía

[1] David Silver, 2005. Cooperative Pathfinding. Department of Computing Science, University of Alberta.

[2] David Silver, 2006. Cooperative Pathfinding. Department of Computing Science, University of Alberta.

[3] Jayasree Chakraborty, Amit Konar, Uday K. Chakraborty and L.C. Jain, 2008. Distr ibuted Cooper ative Multi–Robot Path Planning Using Differential Evolution

[4] NetLogo itself: Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[5] Trevor Standley and Richard Korf. Complete Algorithms for Cooperative Pathfinding Problems. Computer Science Department University of California, Los Angeles.

APÉNDICE

A1. CASOS DE PRUEBA EXTRA

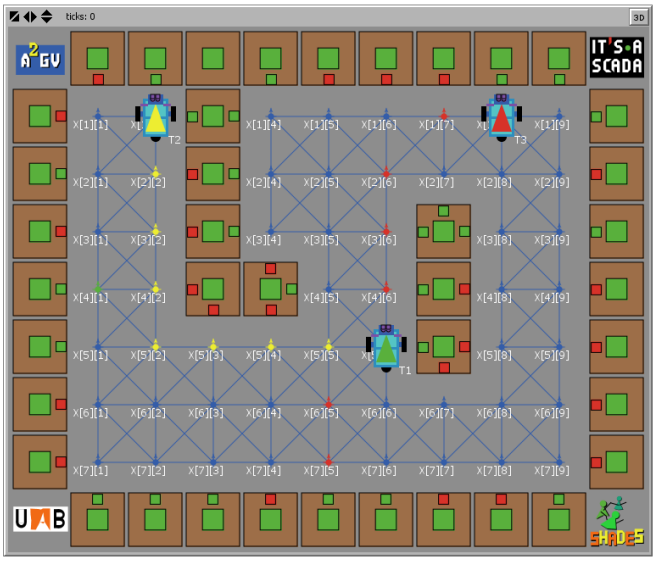

Figura 10: Taxi verde a nodo X[4][1], taxi amarillo al nodo donde se encuentra taxi verde y taxi rojo al nodo X[7][5]. Taxi amarillo realiza una espera en X[4][2] en el tiempo 4 para evitar el conflicto con el taxi verde que pasa por el nodo X[5][2].

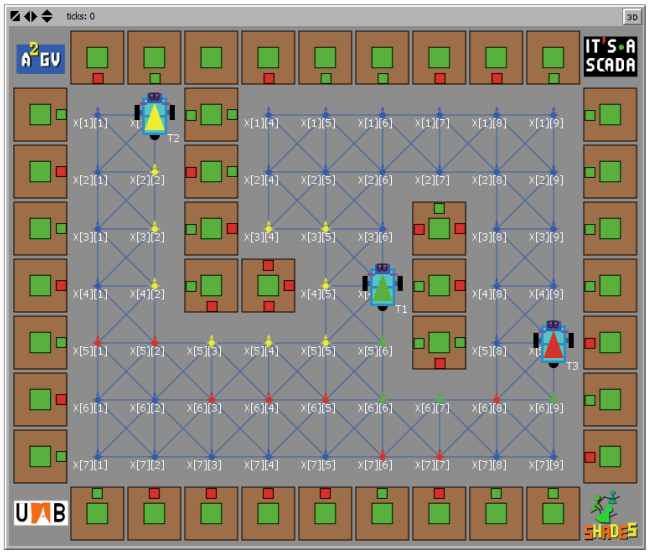

Figura 11: Taxi verde a nodo X[6][9], taxi amarillo a X[3][4] y taxi rojo a nodo X[5][1]. En este caso no se realizan esperas, el conflicto se da entre el taxi verde y rojo en tiempo 3 donde sus rutas ideales coinciden en X[6][7], como se puede ver el taxi rojo se desvía de su ruta ideal para evitar el conflicto.

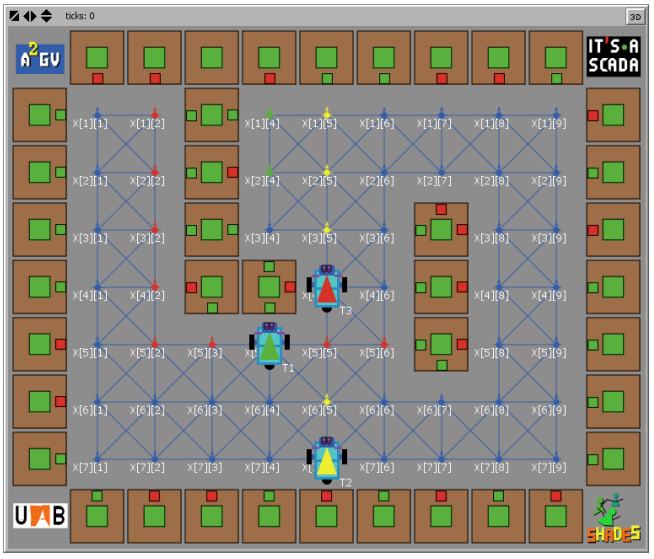

Figura 12: Taxi verde a nodo X[1][4], taxi amarillo a X[1][5] y taxi rojo a X[1][2]. En este caso el taxi rojo es el único que modifica su ruta ideal, primeramente se desvía hacia el nodo X[5][6] para dejar pasar al taxi verde y realiza una espera en esa posición para dejar pasar al taxi amarillo.

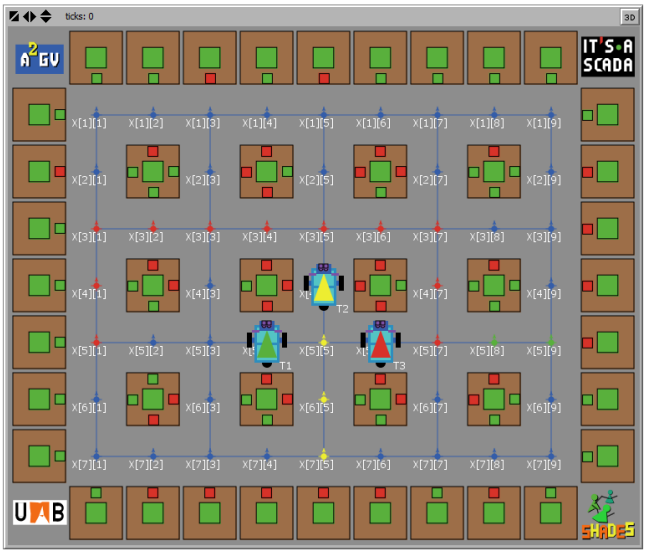

Figura 13: Taxi verde a X[5][9], taxi amarillo a X[7][5] y taxi rojo a X[5][1]. En tiempo uno los 3 taxis se disputan el nodo X[5][5], el taxi amarillo modifica su ruta esperando un tiempo en su posición y el taxi rojo debe modificar su ruta para dejar pasar al taxi verde.