Proyecto 1 - Búsquedas

[Yeray Martínez Martínez](mailto:yeray.martinez@opendeusto.es)

[Adrián Martínez Tejerina](mailto:adrian.martinez.t@opendeusto.es)

## Formulación del problema

La manera en la que hemos formulado el problema es la siguiente: el mapa será una lista de listas, una matriz de N x M, en la que cada posición contiene una lista con una lista por cada estación en esa posición.

Para inicializar los datos del fichero, tenemos una función auxiliar que lee el fichero y nos devuelve las variables del problema en un formato de lista.

data <- load.from.csv(file)



En el siguiente *snippet*, estamos cargando en las variables de *problem*, las posiciones origen y fin que cargamos en data desde el mapa proporcionado. Las posiciones inicial y final son parte del problema porque permanecen constantes a lo largo de la resolución.

problem$initial\_pos <- c(data$initial\_pos[[1]], data$initial\_pos[[2]]) # (x, y)

problem$final\_pos <- c(data$final\_pos[[1]], data$final\_pos[[2]]) # (x, y)



En el siguiente *snippet*, cargamos en *problem* la lista de los distintos métodos de transporte que se encuentran en el fichero proporcionado. Siendo W (walking) el método por defecto, el cual se puede utilizar como transporte para cualquier casilla y que en el mapa no se contempla como parada. Es decir, las posiciones en las que haya únicamente *walking* como método de transporte, el valor de la lista será NULL.

problem$map <- data$map

problem$transport <- data$df # Default value is "W" (Walking), "M" (Metro), "B" (Bus), "T" (Tram)



A continuación, hemos definido las acciones posibles a través del ‘dataframe’ que hemos generado en la lectura del fichero. Como las acciones posibles son un elemento estático en el contexto del problema, pertenecen a *problem*. En cambio, la variable auxiliar que usamos para llevar el conteo de qué transportes se han utilizado pertenecerá al *state* ya que es algo que se va a modificar a lo largo de la resolución del problema.

vec\_temp <- c("north", "northeast", "east", "southeast", "south", "southwest", "west", "northwest")

transports\_used <- c()

for (i in 1:nrow(problem$transport)) {

if (problem$transport$mode[[i]] != "E") {

new\_action <- paste0("X", problem$transport$mode[[i]])

vec\_temp <- append(vec\_temp, new\_action)

transports\_used <- append(transports\_used, list(name = problem$transport$mode[[i]], value = 0))

}

}

problem$actions\_possible <- data.frame(actions = vec\_temp, stringsAsFactors = FALSE)

# This attributes are compulsory

problem$name <- paste0("Multimodal Planner ( Initial position:", problem$initial\_pos[[1]], ",", problem$initial\_pos[[2]], " | Final position:", problem$final\_pos[[1]], ",", problem$final\_pos[[2]], " )")



Por último, cargamos el nombre del problema y los estados inicial y final. Nuestro estado necesitará tener en cuenta la posición actual, el tiempo transcurrido (la suma del tiempo necesario para cada acción ejecutada), el dinero gastado (la suma del dinero necesario para cada acción ejecutada), el modo actual (“walking”, “metro”, “bus”...) y por último la lista de transportes usados (ya que necesitamos diferenciar si anteriormente se usó dicho transporte antes de sumar el dinero del billete).

problem$name <- paste0("Multimodal Planner ( Initial position:", problem$initial\_pos[[1]], ",", problem$initial\_pos[[2]], " | Final position:", problem$final\_pos[[1]], ",", problem$final\_pos[[2]], " )")

problem$state\_initial <- list(actual\_pos = problem$initial\_pos,

time = 0,

money = 0,

mode = "W",

transports\_used = transports\_used)

problem$state\_final <- list(actual\_pos = problem$final\_pos,

time = 0,

money = 0,

mode = "W",

transports\_used = transports\_used)



La función *is.applicable* sirve para comprobar en la posición actual, las condiciones que se deben complir para poder llevar a cabo el *effect*. En caso de que sea una acción de movimiento comprueba que la siguiente posición no estuviese fuera de los límites del mapa y en caso de que esté usando un transporte, que la próxima posición contenga una estación contigua. En caso de que la acción fuese de cambio de transporte, comprueba que la posición actual contenga una estación de ese transporte y que el transporte al que cambia no sea el actual, obviamente.

Hace uso de la función auxiliar llamada *is.consecutive*, que se encarga de que en métodos de transporte que no sean *walking* para comprobar que la estación a la que está intentando moverse sea consecutiva, ya que dichos métodos de transporte solamente pueden moverse a las paradas consecutivas.

is.applicable <- function (state, action, problem) {

(...)

}



La función *effect* sirve para aplicar una acción a un estado determinado. En caso de ser una acción de movimiento, el efecto consiste en cambiar la posición del vector actual\_pos y además añadir el coste temporal de dicha acción a la variable *time* que almacena la suma de todos los costes temporales de las acciones realizadas. En cambio, si es una acción de cambio de transporte, añade el coste temporal de la acción a *time*, añade el coste del billete si nunca se utilizó dicho transporte y además cambia la variable *mode* que controla el modo actual.

effect <- function (state, action, problem) {

(...)

}



La función *is.final.state* comprueba si el estado del nodo actual, es un estado final. Concretamente en este problema, es necesario que la posición (tanto en x como en y) sea la misma que la posición final cargada en el *problem*.

is.final.state <- function (state, final\_state, problem) {

result <- FALSE # Default value is FALSE.

result <- ((state$actual\_pos[1] == problem$final\_pos[1]) & (state$actual\_pos[2] == problem$final\_pos[2]))

return(result)

}



## Función de evaluación

Para la heurística hemos decidido usar la distancia euclídea entre los dos vectores de posición actual y posición final. Ya que las acciones permiten que el algoritmo se mueva en diagonal, la distancia de Manhattan no nos sirve, por lo tanto hemos decidido usar la antes mencionada. Lo siguiente es escalar el resultado a las proporciones de los costes del problema, en este caso vamos a usar el coste mínimo de tiempo, ya que en la función del coste hemos decidido usar solamente el tiempo como referencia y no el dinero.

get.evaluation <- function(state, problem) {

return (sqrt(sum((state$actual\_pos - problem$final\_pos)^2)) \* problem$min\_cost)

}

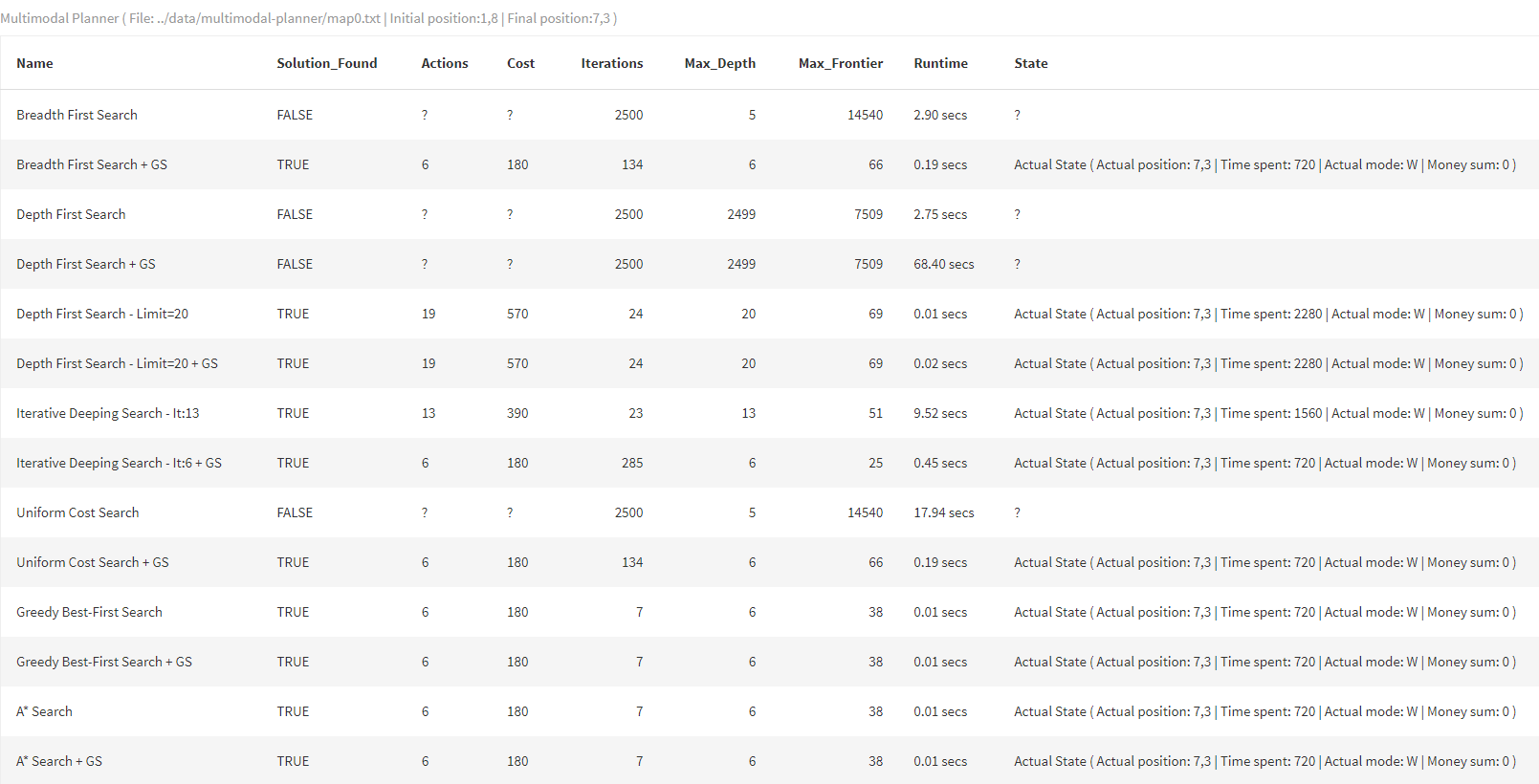


## 

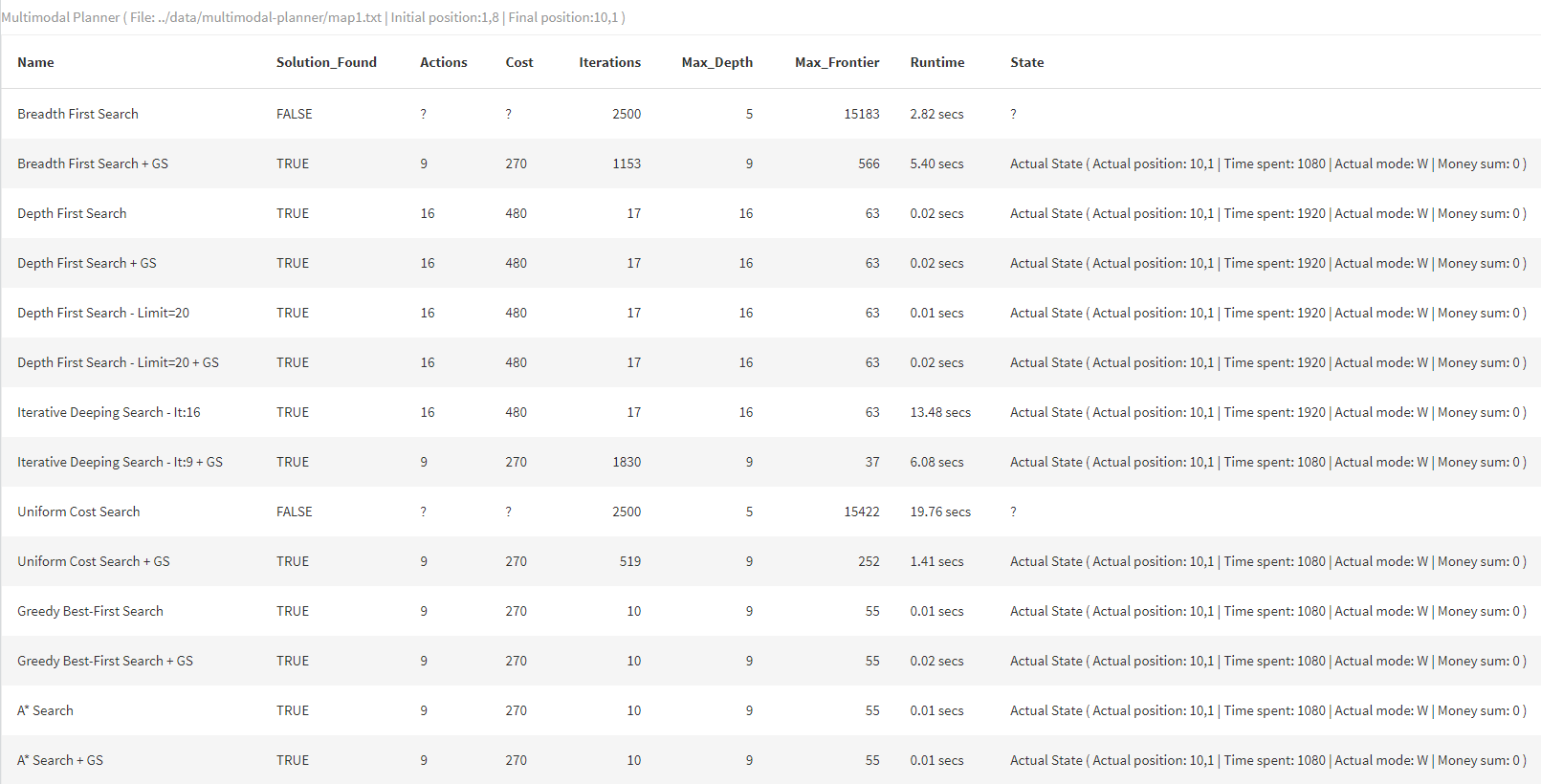
## Análisis de resultados

[Carpeta contenedora de las imágenes de los resultados en mayor resolución](https://drive.google.com/drive/folders/1x9b6_uoMImKFRhLht_f1QSkAPYBIVYXU?usp=sharing)

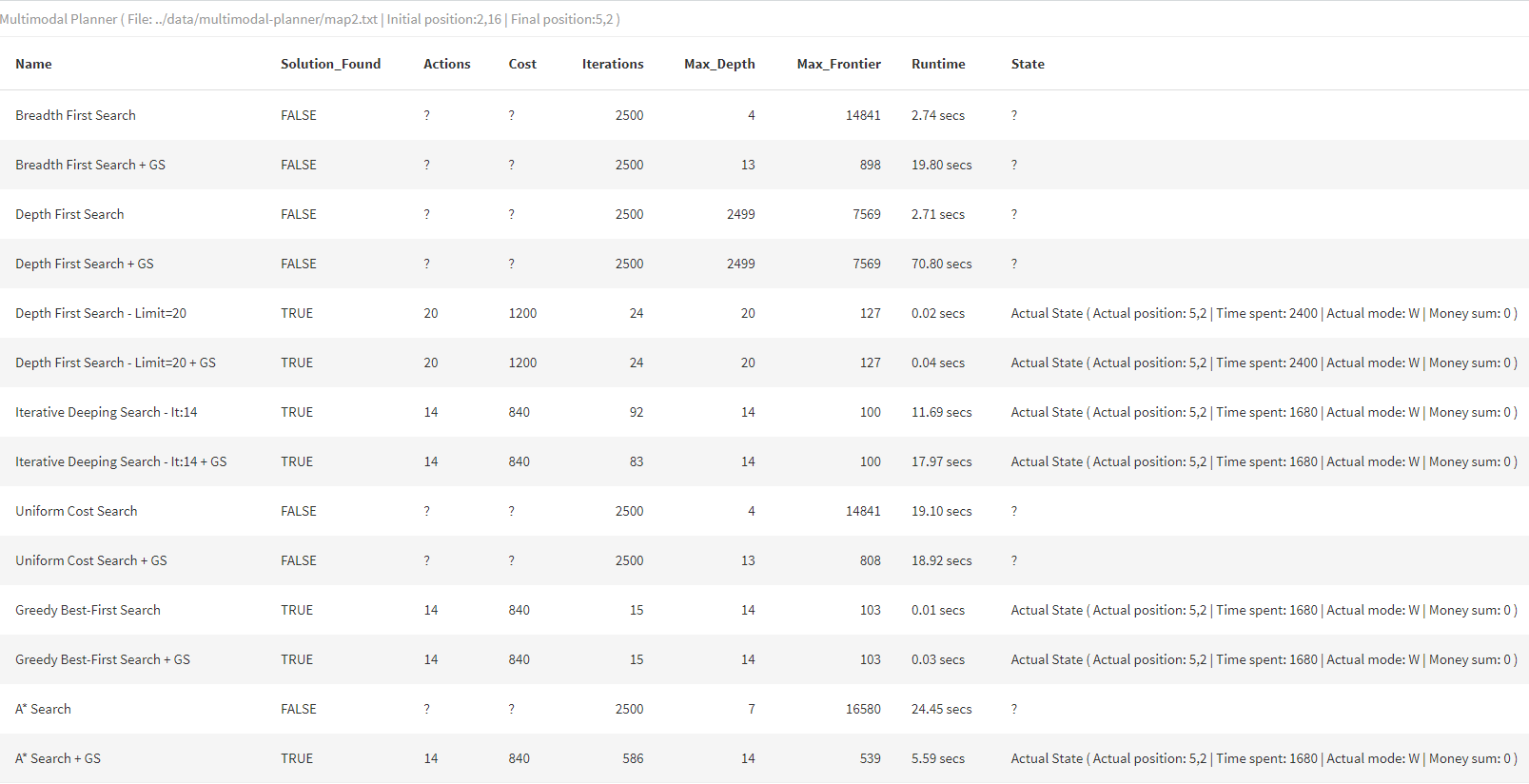
### Mapa 0



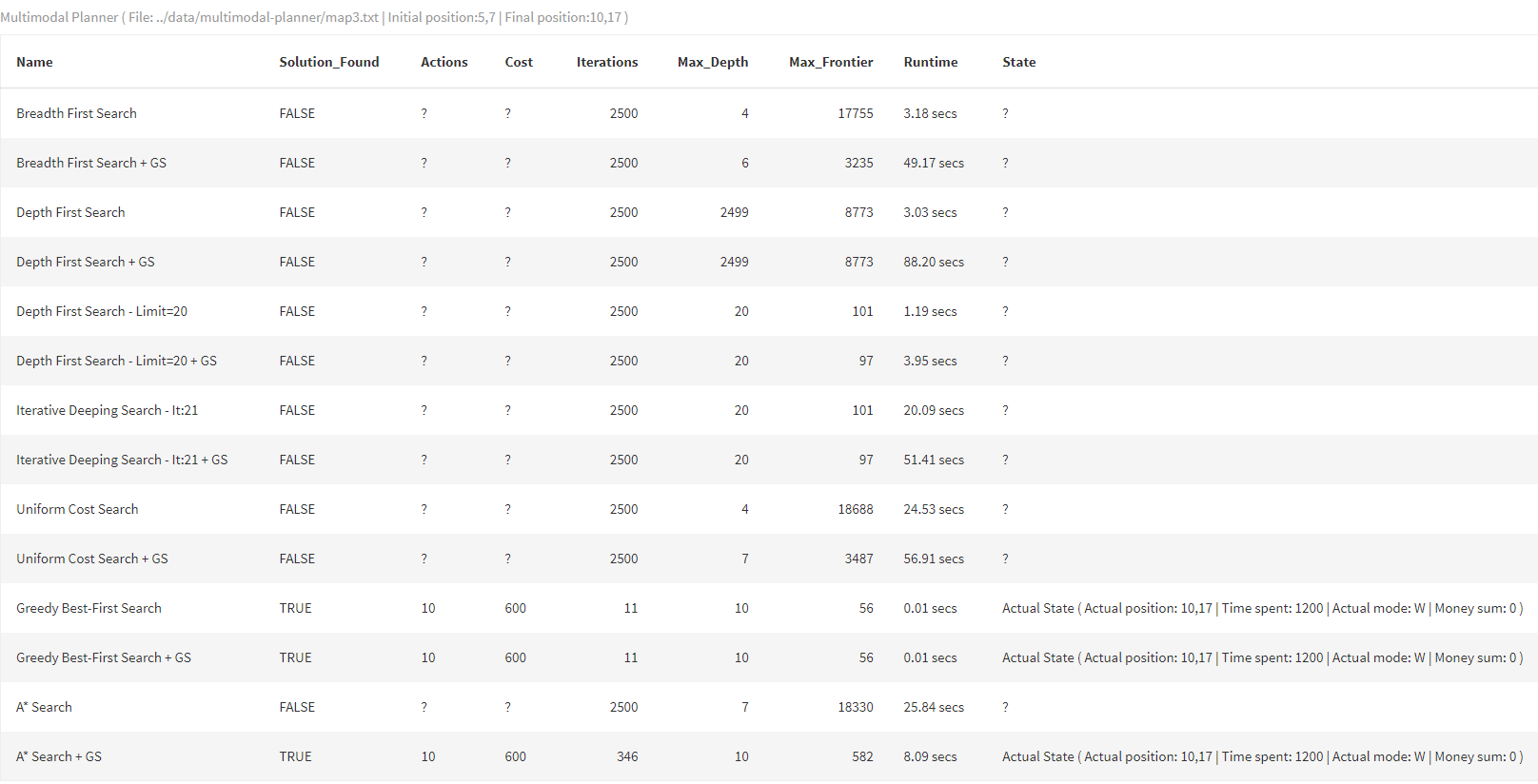
### Mapa 1



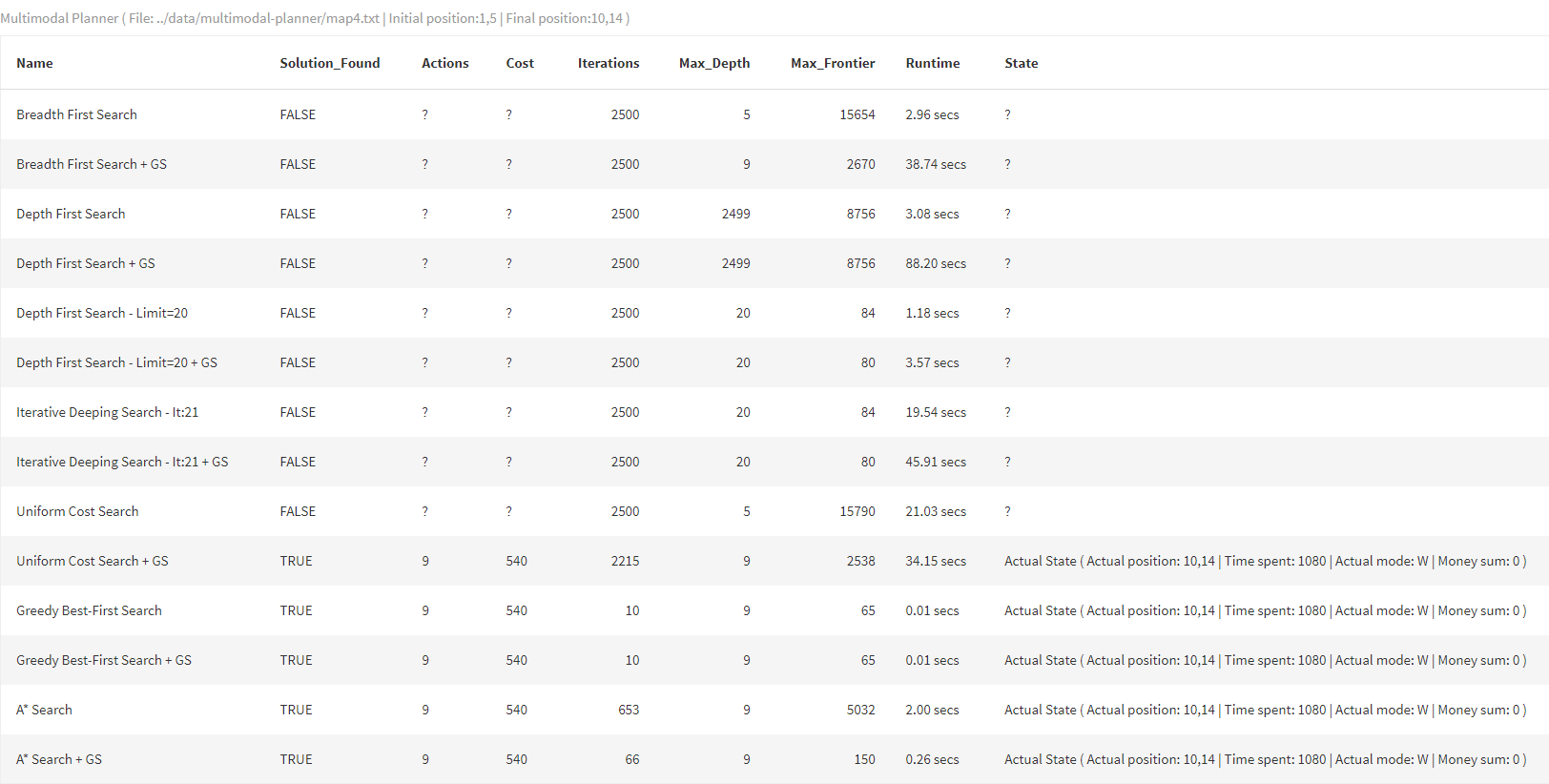
### Mapa 2



### Mapa 3



### Mapa 4



## 

## Conclusiones

A partir de los resultados obtenidos podemos observar que los algoritmos tienden a encontrar menos soluciones dependiendo de la complejidad de la entrada que se les proporciona, el cual en este caso son los mapas.

Otro punto a mencionar es que la influencia del orden en el que están definidas las acciones son un factor importante ya que las primeras acciones tienden a ser las que marca como solución. Esto es porque el algoritmo realiza primero dichas acciones, en nuestro caso moverse, por lo que tiene poca o nula tendencia a cambiar de método de transporte, las cuales están definidas al final. Dicho comportamiento se refleja en las capturas de pantalla adjuntas, en todas las soluciones se puede observar que la suma de dinero gastado es 0 ya que no han habido cambios de transporte.

Finalmente, con los datos obtenidos, se aprecia que los algoritmos como Greedy-Best-First y A\* tienden a encontrar solución en más casos y, además, estas suelen ser las mejores soluciones. Podría decirse entonces que dichos algoritmos, para este problema concreto, son los más óptimos.

# Bibliografía

[1] Documentación con una gran variedad de recursos y ejemplos en forma de tutorial para el aprendizaje del lenguaje de programación de R. Disponible en: [https://www.geeksforg  
eeks.org/r-programming-language-introduction/](https://www.geeksforgeeks.org/r-programming-language-introduction/).

[2] Hemos utilizado Stack Overflow como comunidad y portal de preguntas para ayudarnos a solucionar distintos problemas y dudas que nos han surgido a lo largo del aprendizaje y el desarrollo. Portal de Stack Overflow específico de R disponible en: [https://stackover](https://stackoverflow.com/questions/tagged/r)[flow.com/questions/tagged/r](http://flow.com/questions/tagged/r).

Hemos utilizado la extensión de GitHub Copilot integrada en R como IA generativa para ayudarnos y darnos sugerencias a la hora de hacer el código. Nos ha sido de gran ayuda frente a otros recursos ya que utiliza la información que posee del resto del código para brindarnos información más precisa y personalizada. Además, nos ha ayudado con la sintaxis del lenguaje de programación R, reduciendo las veces que hemos tenido que consultar la documentación.

Adicionalmente, hemos utilizado herramientas de IA generativa online, específicamente la herramienta de Google [Gemini](https://gemini.google.com/app), para intentar resolver algunas problemáticas que nos surgieron durante el desarrollo. A pesar de intentarlo, la respuestas que nos proporcionaba no eran correctas y, finalmente, no llegamos a utilizar las soluciones de dicha herramienta.