INTELIGENCIA ARTIFICIAL
PROYECTO 2020/2021

Algoritmo A*

Metro de Atenas

Número de grupo: 12

Autores

Joel Medina Sánchez
180156 joel.medina.sanchez@alumnos.upm.es

Jia Yun Liu
180194 jiayun.liu@alumnos.upm.es

Álvaro Lozano Gil
180217 a.lgil@alumnos.upm.es

Carlos González Aguilar
180024 carlos.gonzalez.aguilar@alumnos.upm.es

Daniel Alameda Badurak
180173 daniel.alameda.badurak@alumnos.upm.es

Índice

1. Introducción

Esta práctica trata sobre el desarrollo de una aplicación externa, independiente que realice el cálculo de la ruta más corta entre dos estaciones de metro dadas por el usuario y que calcule el tiempo aproximado que se va a tardar en realizar dicho viaje. Para ello, es necesario utilizar el algoritmo A* para el cálculo óptimo de la ruta.

Como curiosidad personal del grupo, se debe decir que hemos quedado muy satisfechos con la realización de este práctica porque nos hemos visto obligados a aprender un nuevo idioma de programación dado un enunciado y se ha debido realizar una labor de investigación con poca información sobre los datos y ha sido como realizar un trabajo para alguien. Además, hemos podido aprender muchas funcionalidades muy útiles sobre las librerías, la creación de ejecutables, el desarrollo de conexiones entre clases de Python…

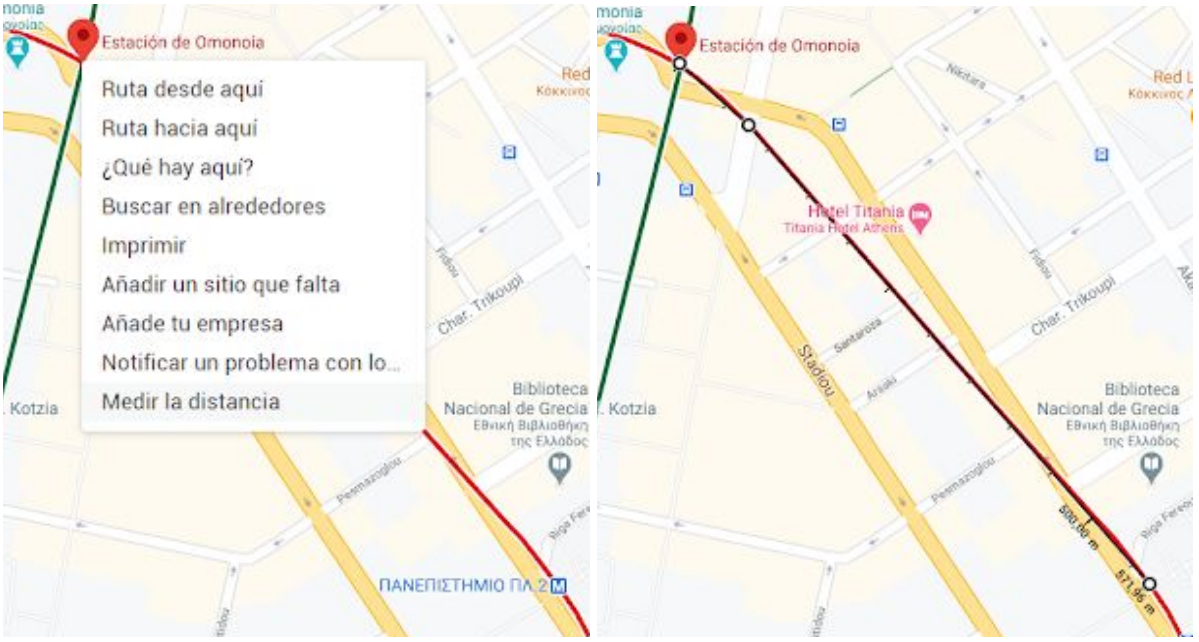
2. Desarrollo de la Práctica

En un comienzo, para el desarrollo de la práctica se comenzó por realizar una división del trabajo para que todos lo integrantes realizaran un trabajo equitativo. Por ello, una persona se encargó de la búsqueda de la información del Metro de Atenas. Otra persona debía implementar esa información en una estructura de datos de forma que se pudiera buscar y analizar dicha información de manera eficiente. Otra persona debía aprender y explicar al resto de compañeros el funcionamiento del algoritmo A*, para en conjunto poder buscar la manera más óptima para su implementación en código. Y otra persona debía encargarse de aprender a utilizar las librerías gráficas del lenguaje para encontrar una que fuera lo suficientemente completa para su implementación tal y como se quería. Finalmente, una persona debía encargarse de dirigir el proyecto, dividir las tareas y realizar el desarrollo de la memoria de la práctica supervisando el trabajo de los demás compañeros. Además, cada compañero debía explicar a los demás su trabajo para posibles mejoras, o simplemente para tener una idea general conjunta del proyecto.

En un primer momento, se decidió empezar a utilizar como lenguaje de programación, Python, ya que fue recomendado por el profesor encargado de la práctica y se creyó que sería un buen momento de aprender un nuevo lenguaje muy utilizado actualmente y con una versatilidad bastante amplia. Por ello, todo el desarrollo de la práctica supuso aprender un lenguaje nuevo, además de aprender el algoritmo y aprender la implementación de librerías gráficas en un lenguaje.

3. Obtención de datos

Tras buscar sin éxito un plano/mapa del metro de Atenas que indicase las distancias entre estaciones, usamos la herramienta Medir la distancia que proporciona Google Maps:



De esta forma se pueden obtener los datos del recorrido del metro con bastante precisión. Por ello, se obtuvieron los datos de todas y cada una de las estaciones en el plano real, pero cabe destacar que algunas estaciones que aparecían en el mapa del Metro, actualmente no existen, por lo que hubo que realizar una investigación sobre su posición antigua en el mapa. Una vez obtenidas todas las distancias entre estaciones, se vio que también iba a ser necesario un cálculo heurístico de las distancias, por ello, se decidió buscar de nuevo todas las estaciones, esta vez, apuntando las coordenadas de latitud y longitud de ellas, para así poder realizar un cálculo de distancia preciso y completo. Por ello, al final, tenemos 4 tablas de datos: Distancias entre estaciones de las 3 líneas y latitud y longitud de cada estación.

Para introducir todos estos datos en el código, en un primer momento se introdujeron en listas de tuplas. Pero a medida que aprendíamos el lenguaje vimos que para poder acceder en casos concretos a estaciones dado un nombre, iba a ser mucho más eficiente y sencillo el implementar las listas como diccionarios de Python. Por ello, la lista de coordenadas de cada estación viene dada por un diccionario donde el nombre de la estación es la clave y su definición es una tupla de latitud, longitud. Además, se le añadió una última tupla, que es a su vez una tupla, que contiene la/las líneas a las que pertenece dicha estación debido a que iba a ser información útil y de la que se requería un acceso rápido.

Toda la información está además almacenada en un archivo de Excel para facilitarnos la comprobación de cálculos a la hora de comprobar si los resultados del programa son correctos.

4. Implementación del Algoritmo A*

En un primer momento, se habló sobre la implementación manual del algoritmo, además, de que se empezó a estudiar la manera de implementar dicho algoritmo de una manera completa. Pero hubo un factor decisivo que cambió nuestra decisión. Para poder tratar las líneas y conexiones entre sí, iba a ser necesario tener implementado de alguna manera un grafo. Ya sea por tener un mayor control de las conexiones o por el simple desarrollo del algoritmo. Por ello, se comenzó a crear una implementación desde 0 tanto de los grafos como del algoritmo, pero al encontrarnos con algunos contratiempos decidimos buscar una alternativa.

Acabamos encontrando una librería (networkx) que no solo implementaba un código de grafo sencillo y práctico, sino que además contenía un método que realizaba el algoritmo A* dados unos parámetros. Estos parámetros debían ser el grafo en cuestión con las conexiones ya representadas, las estaciones origen y destino, la función heurística que debía utilizar para el cálculo heurístico de los tiempos de viaje entre estaciones, y la función encargada de calcular el valor de G.

Con este módulo, simplemente fue necesario pensar en la implementación de dichas funciones. Para asegurarnos de que podíamos utilizar una librería con esta función, se envió un correo al profesor esperando una confirmación. En un primer momento, el cálculo heurístico y el cálculo de la g iban a ser dados simplemente por distancias. Pero por consenso se decidió que estos cálculos iban a ser pasados a tiempos de viaje porque así podíamos añadir tiempos cuando se realizaban transbordos, cambios de línea… Por ello, para calcular la heurística entre dos estaciones, se realiza la siguiente operación:

$$H = \frac{\sqrt{(LatEst1 - LatEst2)^2 + (LongEst1 - LongEst2)^2} * 87.736 * 1000}{22.222}$$

El 87.736, es la constante por la que se debe multiplicar para transformar las coordenadas a kilómetros. El 1000 para transformar las distancias a metros. Y finalmente la división por 22.222 es la constante necesaria para transformar distancias en metros a segundos teniendo en cuenta que los metros de media viajan a una velocidad de 80 km/h (22.222 m/s). Este sería el cálculo de la heurística utilizada.

Para el cálculo de la G de la función se utilizaron los pesos de las aristas hasta el nodo de la estación objetivo dado el de la estación actual. Para el cálculo de los pesos de las aristas del grafo se tuvo en cuenta que cada parada, con o sin transbordo, iba a suponer un retraso de un minuto. De esta forma se añadía un grado de complejidad extra que podía favorecer un recorrido más largo y con menos estaciones a uno más corto y con más estaciones:

$$G = \frac{DistanciaEst1Est2 * 1000}{22.222} + 60$$

5. Interfaz gráfica

Para la parte de Interfaz Gráfica, la idea principal iba a ser desarrollar una ventana que se pudiera introducir una estación origen y una estación destino, apareciera el mapa del metro y además un listado de las estaciones, movimientos, transbordos y tiempos de viaje entre estaciones. Por ello, se encontró una librería que fuera capaz de realizar dichas implementaciones. Finalmente, se utilizó la librería tkinter.

Un compañero comenzó a desarrollar la estructura del programa, creando la ventana, los títulos… y comenzamos a ver que la interfaz cambiaba de un Sistema Operativo a otro. Cuando este compañero que utiliza MAC, ejecutaba el programa, los botones y las cajas desplegables eran mucho más suaves y bonitas que cuando se ejecutaba en Windows. Esto desanimó un poco, debido a que suponíamos que iba a ser algo unificado. Por ello, las capturas adjuntas en este documento van a ser desde el punto de vista de MAC.

Se propuso una estructura inicial, con un rectángulo superior donde se introducirían los datos, el mapa en un lado izquierdo y la lista de la ruta en un lado derecho.

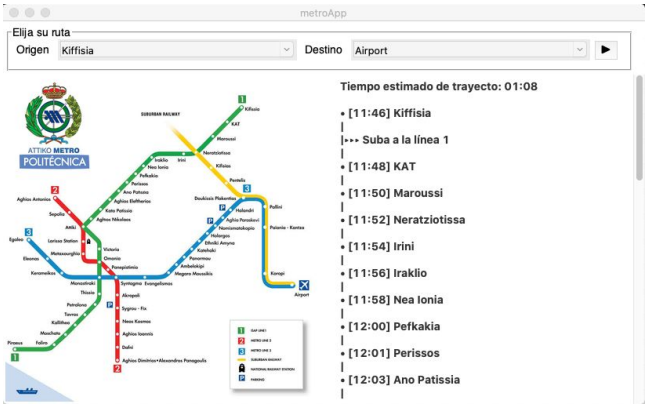
Se comenzó a tener problemas con las proporciones de la aplicación, debido a que la ventana se creaba en base a las dimensiones de su contenido, y no en base a un tamaño inicial. Por lo que para que todo se ajustara debía estar medido al completo. Por ello, se deshabilitó la opción de redimensionar la ventana. Se introdujo un botón que implementara la función de buscar la ruta utilizando el algoritmo y comenzó a tomar forma la aplicación. Además, de que ya se empezaba a tener una idea clara de cómo iba a quedar y de cómo se quería.

Finalmente, se introdujo el mapa, los diseños se hicieron más elegantes, se buscó un tipo de letra acorde, se reajustaron los tamaños y se le dio una forma final. Y cuando se creía tener finalizada la aplicación, se ejecutó en Windows y cambió completamente la estructura. Se veía mucho más grande, se había descuadrado todo y además no disponía de

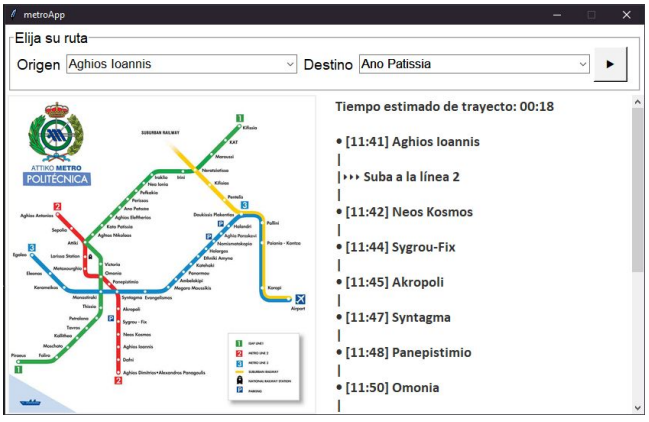
los mismos colores que tenía MAC. Por ello, un compañero que disponía de Windows, comenzó a cambiar la estructura de manera que quedara un poco más generalizada y que se pudiera utilizar en Windows. Quedó un poco más "fea" en MAC, pero al menos quedaba más parecido al diseño de Windows.

Con las limitaciones que esta librería contenía, se pudo obtener una aplicación un poco más elegante. Se comenzó a pensar en maneras de mejorar la aplicación gráfica. Se pensó en introducir un dibujo que se pintara a medida que se pulsaba una estación, o a medida que se ejecutaba el algoritmo, pero la librería no permitía pintar de manera sencilla dentro de la ventana tal y como se quería y algunos compañeros decidieron que iba a romper la aplicación.

Se planteó introducir colores a la ruta mostrada como resultado, pero no se podía realizar un cambio en el código tal y como estaba implementado porque el resultado del String no podía ser cambiado de color por ser un String. También se planteó la idea de introducir un zoom en el mapa, debido a que algunas estaciones no se llegaban a leer debido al tamaño de la imagen, pero la resolución de la imagen del metro era demasiado pequeña para poder realizar un zoom y que no se pixelara al completo. También se planteó introducir una pantalla de inicio a la aplicación antes de que iniciara la ventana con el mapa y la ruta directamente, pero de nuevo, se decidió por consenso, y algunos compañeros no estaban de acuerdo con la idea. Así que, se quedó un diseño de aplicación como el mostrado en la figura, donde se ve una comparación entre MAC y Windows.



Diseño en MAC



Diseño en Windows

6. Creación del Ejecutable

Con el proyecto terminado y ejecutándose correctamente, utilizamos el módulo de Python pyinstaller para generar un ejecutable de Windows .exe para facilitar la utilización del programa y su presentación. Para su generación se utilizó el siguiente comando:

```
pyinstaller --onefile --icon=logo.ico metroApp3.py
```

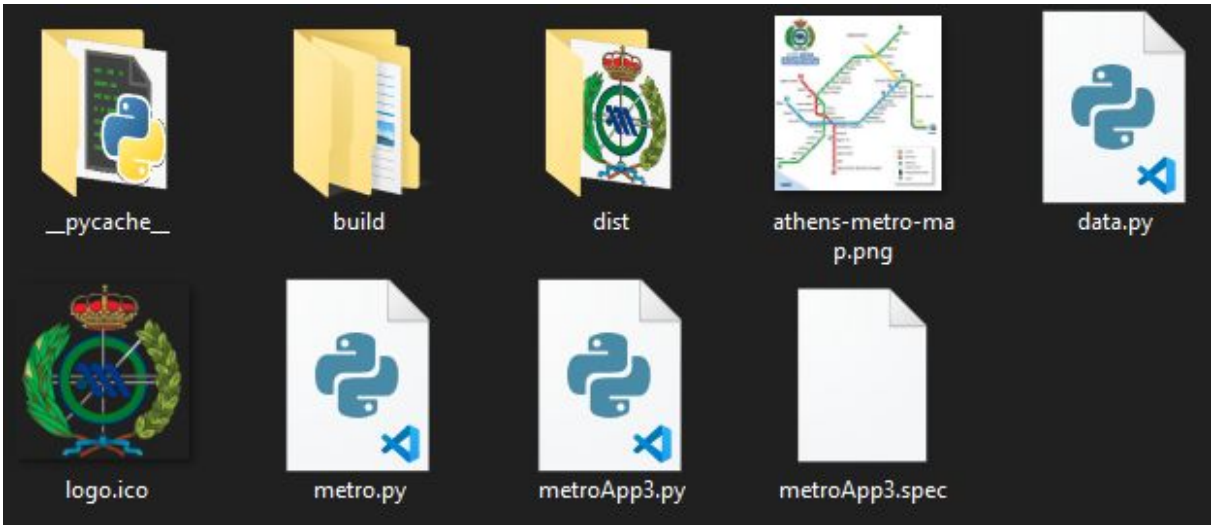
en la siguiente carpeta del proyecto, que contenía los 3 archivos .py, siendo metroApp3.py el que importaba a metro.py y a data.py. Además, la carpeta incluye la imagen del metro de Atenas y el archivo logo.ico para que el ejecutable generado tenga un icono personalizado.

Contenido de la carpeta del proyecto

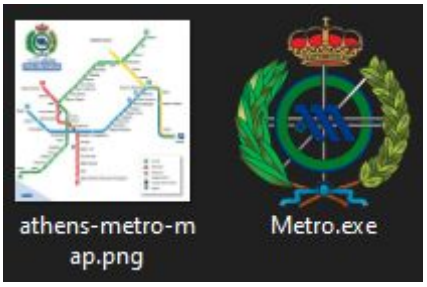
logo.ico

athens-metro-map.png

Tras la ejecución del comando, la carpeta queda así:



Ya que hemos usado el parámetro `--onefile`, el ejecutable de la carpeta dist contendrá todo el programa y sólo tendremos que meterlo en una carpeta con athens-metro-map.png para que funcione. Al haber usado `--icon=logo.ico`, el ejecutable usará el .ico especificado.
Las carpetas __pycache__, build y el archivo metroApp3.spec se pueden eliminar, ya que no nos servirán de nada.



Contenido de la carpeta final del ejecutable

Para comprobar que el ejecutable funcionaba, lo probamos en varias máquinas Windows con diferentes configuraciones, con y sin Python, y sólo en una, dio el siguiente error:

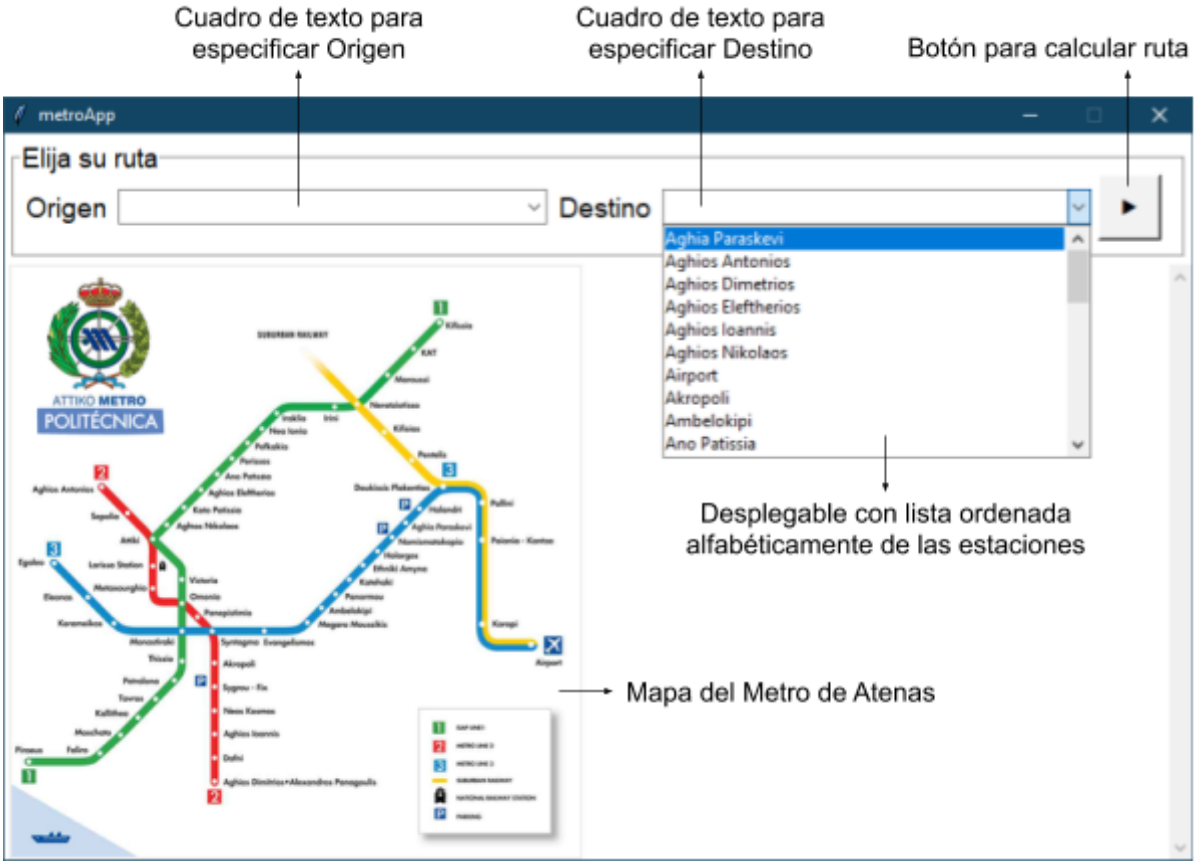
```
The current Numpy installation fails to pass a sanity check due to a
bug in the windows runtime
```

Lo consideramos un caso aislado pero a tener en cuenta, y en caso de experimentarlo, la solución sería ejecutar metroApp3.py, aunque para ello sería necesario descargar las librerías que importan los archivos .py.

Por esta razón, la parte ejecutable del proyecto consta de una carpeta que contiene el .exe del proyecto y de athens-metro-map.png.

7. Uso de la aplicación

La pantalla que el usuario se encontrará nada más abrir la aplicación es la siguiente:



Una vez introducidos los datos de las estaciones en los campos Origen y Destino y pulsado el botón de calcular ruta, el resultado será este:

La aplicación también tiene estados especiales que devuelven mensajes de error en el caso de faltar la estación de origen, de destino o ambas estaciones al pulsar el botón de calcular ruta, o en el caso de que tanto la estación de origen como la de destino sean la misma.

8. Anexos

8.1. Distancia entre estaciones

Línea 1		
Estación 1	Estación 2	Distancia (km)
Kiffisia	KAT	1,05
KAT	Maroussi	1,18
Maroussi	Neratziotissa	1,62
Neratziotissa	Irini	1,06
Irini	Iraklio	1,55
Iraklio	Nea Ionia	1,27
Nea Ionia	Pefkakia	0,7
Pefkakia	Perissos	0,68
Perissos	Ano Patissia	1,29
Ano Patissia	Aghios Eleftherios	0,55
Aghios Eleftherios	Kato Patissia	1
Kato Patissia	Aghios Nikolaos	0,53
Aghios Nikolaos	Attiki	0,95
Attiki	Victoria	1,1
Victoria	Omonia	1,01
Omonia	Monastiraki	0,95
Monastiraki	Thissio	0,47
Thissio	Petrolona	1,56
Petrolona	Tavros	0,94
Tavros	Kalithea	0,53
Kalithea	Moschato	1,64
Moschato	Faliro	1,79
Faliro	Piroeus	2,09

Línea 2		
Estación 1	Estación 2	Distancia (km)
Aghios Antonios	Sepolia	1,32
Sepolia	Attiki	0,87
Attiki	Larissa Station	0,82
Larissa Station	Metaxourghio	0,63
Metaxourghio	Omonia	0,74
Omonia	Panepistimio	0,57
Panepistimio	Syntagma	0,64
Syntagma	Acropoli	0,9
Acropoli	Sygrou-Fix	0,56
Sygrou-Fix	Neos Kosmos	0,78
Neos Kosmos	Aghios Ioannis	0,56
Aghios Ioannis	Dafni	0,87
Dafni	Aghios Dimetrios	1,02

Línea 3		
Estación 1	Estación 2	Distancia (km)
Egaleo	Eleonas	1,17
Eleonas	Kerameikos	1,83
Kerameikos	Monastiraki	1,29
Monastiraki	Syntagma	0,89
Syntagma	Evangelismos	0,99
Evangelismos	Megaro Moussikis	0,65
Megaro Moussikis	Ambelokipi	0,98
Ambelokipi	Panormou	0,91
Panormou	Katehaki	1,12
Katehaki	Ethniki Amyna	1,15
Ethniki Amyna	Holargos	0,91
Holargos	Nomismatokopio	1,14
Nomismatokopio	Aghia Paraskevi	1,06
Aghia Paraskevi	Halandri	0,91
Halandri	Doukissis Plakentias	1,22
Doukissis Plakentias	Pallini	4,18
Pallini	Peionia-Kantza	2,46
Peionia-Kantza	Koropi	8,52
Koropi	Airport	5,85

8.2. Posición geográfica de estaciones

Posición geográfica de todas las líneas		
Nombre	Latitud	Longitud
Kiffisia	38.073.664	23.808.134
KAT	38.065.867	23.804.002
Maroussi	38.056.209	23.804.889
Neratziotissa	38.045.094	23.792.907
Irini	38.043.732	23.782.714
Iraklio	38.046.258	23.766.038
Nea Ionia	38.041.591	23.755.116
Pefkakia	38.037.130	23.750.154
Perissos	38.032.793	23.744.444
Ano Patissia	38.023.771	23.735.913
Aghios Eleftherios	38.020.076	23.731.758
Kato Patissia	38.011.567	23.728.561
Aghios Nikolaos	38.006.917	23.727.650
Attiki	37.999.409	23.722.639
Victoria	37.993.068	23.730.237
Omonia	37.984.188	23.728.714
Monastiraki	37.976.127	23.725.546
Thissio	37.976.660	23.720.697
Petralona	37.968.615	23.709.208
Tavros	37.962.976	23.704.308
Kalithea	37.960.438	23.696.937
Moschato	37.960.438	23.696.937
Faliro	37.944.982	23.665.223
Piroeus	37.948.087	23.643.272
Aghios Antonios	38.006.669	23.699.490
Sepolia	38.002.625	23.713.493
Larissa Station	37.999.294	23.722.076
Metaxourghio	37.986.298	23.721.099
Panepistimio	37.980.379	23.733.008
Syntagma	37.975.110	23.735.637

Akropoli	37.968.753	23.729.558
Sygrou-Fix	37.964.271	23.726.496
Neos Kosmos	37.957.563	23.728.438
Aghios Ioannis	37.956.616	23.734.693
Dafni	37.949.205	23.737.214
Aghios Dimetrios	37.940.490	23.740.690
Egaleo	37.991.960	23.681.764
Eleonas	37.987.917	23.694.211
Kerameikos	37.978.560	23.711.461
Evangelismos	37.976.130	23.746.400
Megaro Moussikis	37.979.293	23.752.894
Ambelokipi	37.987.343	23.756.960
Panormou	37.993.184	23.763.396
Katehaki	37.993.197	23.775.913
Ethniki Amyna	38.000.057	23.785.722
Holargos	38.004.579	23.794.281
Nomismatokopio	38.009.271	23.805.643
Aghia Paraskevi	38.017.208	23.812.349
Halandri	38.021.502	23.821.168
Doukissis Plakentias	38.024.595	23.834.276
Pallini	38.005.815	23.869.552
Peionia-Kantza	37.984.222	23.869.965
Koropi	37.913.005	23.895.749
Airport	37.936.639	23.944.527