

BÚSQUEDA DE RUTAS DE METRO

1. DESCRIPCIÓN DEL PROBLEMA

En este proyecto se quiere implementar un algoritmo evolutivo para buscar el mejor camino entre dos puntos de una red de metro. La calidad del camino no sólo depende del número de estaciones y trasbordos que contiene, sino también de cumplir determinadas restricciones sobre estaciones por las que se desea pasar o que se quieren evitar. A veces se encuentran soluciones válidas, aunque no óptimas, y a veces ni siquiera existe una solución que respete todas las restricciones.

Los algoritmos tradicionales no aseguran una solución óptima y rápida para la búsqueda de caminos en grafos. Por esta razón la programación evolutiva es una alternativa a considerar.

El algoritmo debe tener en cuenta las siguientes restricciones:

- El trayecto debe empezar y terminar en las estaciones especificadas, de forma que recorra el menor número posible de estaciones, y realizando el menor número posible de trasbordos.
- El trayecto debe evitar pasar por las estaciones que se indique, y debe pasar por otras estaciones indicadas. Estas restricciones pueden considerarse más o menos prioritarias. El algoritmo debe permitir especificar su importancia.

Entradas al programa

- Plano de metro:



Se lee de un archivo con el siguiente formato: la descripción de cada línea de metro comienza con una línea del archivo con el nombre de la línea. A continuación hay una línea de archivo para cada estación de la línea, que comienza con el nombre de la estación, seguida de la lista de nombres de líneas con las que tiene correspondencia y terminada en un *. Los datos de una línea terminan con una línea de archivo con la marca #. Por ejemplo:

Línea1 N

```
Congosto *
Villa_de_Vallecas *
Sierra_de_Guadalupe *
Miguel_Hernandez *
Alto_del_Arenal *
Plaza_de_Castilla Linea9 Linea10 *
...
```

```
#
Linea2 N
Cuatro_Caminos Linea1 Linea6 *
Canal Linea7 *
Quevedo *
San_Bernardo Linea4 *
...
```

- Datos de la consulta:

Los datos del trayecto que se quiere realizar pueden introducirse interactivamente o leerse de un archivo. Hay que especificar las estaciones inicial y final del trayecto, así como las estaciones por las que se desea pasar y las estaciones por las que no debe pasar.

La salida es una secuencia de estaciones, especificando en cada una a qué línea de metro pertenece. Los resultados, además de salvarse en un archivo, deben presentarse de forma que el usuario, de forma cómoda, pueda saber la longitud del trayecto, el número de trasbordos a realizar y si el trayecto respeta o no las restricciones solicitadas.

Observaciones

- Un trayecto no debe pasar dos veces por la misma estación.
- Hay que tener en cuenta que pueden existir líneas circulares.
- Hay que tener en cuenta que pueden existir correspondencias entre estaciones con distinto nombre. Por ejemplo, en la red de metro de Madrid, hay una correspondencia entre "Plaza de España" y "Noviciado".

2. DISEÑO DEL ALGORITMO

En primer lugar necesitamos definir la forma de representar la red de metro en la que se van a buscar los caminos. Una posibilidad es representar el plano de metro como un conjunto o lista de líneas de metro:

```
// Representación del PLANO de metro completo
tipo TPlano : vector de TLinea;
```

A su vez, cada línea podemos representarla como una secuencia o lista de estaciones. Además, asignamos un nombre a cada línea y un indicativo de si es o no circular.

```
tipo TEstaciones: vector de TEstacion;

tipo TLinea = registro{
  cadena_caracteres nombre; // nombre asignado a la línea
  TEstaciones estaciones; // secuencia de estaciones
                          // la línea
  booleano circular; // indicativo de línea circular o no
}
```

Para representar las estaciones utilizamos una estructura de datos que incluya información sobre el nombre de la estación, sobre el posible nombre que puede tener esa estación en otra línea y sobre las correspondencias con otras líneas a las que se puede cambiar en esa estación.

```
tipo TEstacion = registro{
  //nombre de la estación en esa línea
  cadena_caracteres nombre;
  //nombre de la estación en otra línea
  cadena_caracteres alias;
  //líneas con las que se cruza
  TCorrespondencias corresp;
}
```

Las correspondencias de una estación las podemos representar simplemente como un vector de nombres o cadenas de caracteres.

```
tipo TCorrespondencias : vector de cadena_caracteres;
```

También necesitamos representar los datos de una consulta que se realiza al algoritmo. Una consulta incluye la estación inicial del trayecto, la final, un conjunto de nombres de estaciones por las que no se quiere pasar y un conjunto de nombres de estaciones por las que se desea pasar.

```
tipo TLNombres : vector de cadena_caracteres;
tipo TConsulta = registro{
  // nombre de la estación inicial
  cadena_caracteres est_inicial;
```

```

// nombre de la estación final
cadena_caracteres est_final;
TLNombres est_prohibidas; // estaciones a evitar
TLNombres est_obligadas; // estaciones por las que pasar
}

```

2.1 Representación de los individuos

Sabemos que la solución a nuestro problema será una secuencia de estaciones del plano de metro considerado, que comience en la estación inicial del trayecto buscado y termine en la estación final de dicho trayecto. Por eso, cada individuo representa un trayecto entre la estación inicial y la final. Como hay estaciones que pertenecen a distintas líneas, cada estación especificada en un individuo debe incluir información sobre la línea a la que pertenece.

Por lo tanto, podemos representar a los individuos como listas de genes en las que cada gen representa a una estación del recorrido y a la línea a la que corresponde.

```

tipo Tgen = registro{
// posición en la lista de líneas del plano
entero línea;
// posición de la estación en la línea
entero estación;
}

```

Luego, cada individuo se representa como una cadena de genes como los descritos, junto con la información usual que necesita un algoritmo evolutivo.

```

tipo TGenes : vector de TGen;
tipo TIndividuo = registro{
TGenes genes; // cadena de genes(genotipo)
real adaptación; // función de evaluación
real puntuación; // puntuación relativa
real punt_acu; // puntuación acumulada para sorteos
}

```

2.2 Generación de la población inicial

Para reducir el espacio de búsqueda y hacer más eficiente el algoritmo, es conveniente que las secuencias de estaciones de los individuos de la población inicial, además de empezar y terminar en las estaciones inicial y final del trayecto

buscado, cumplan otras condiciones. Una de estas condiciones es que los trayectos no pasen dos veces por la misma estación, porque esto dará lugar a tramos de recorrido inútiles.

También para reducir el espacio de búsqueda podemos establecer un límite a la longitud de un individuo, pasado el cual si no se ha llegado a la estación final se empieza a generar de nuevo.

La idea de la generación de un nuevo individuo es partir de la estación inicial del recorrido, e ir generando estaciones por las que puede continuar, hasta llegar a la estación final, y teniendo en cuenta las condiciones mencionadas. De acuerdo con estas consideraciones, podemos seguir el siguiente esquema para generar los individuos de la población inicial:

- Se buscan las líneas en las que se encuentra la estación inicial y se selecciona una de ellas aleatoriamente.
- Para evitar que los trayectos avancen y retrocedan por una misma línea, es necesario establecer una dirección de movimiento (por ejemplo con una variable booleana *avanza* que toma el valor cierto para indicar avance o falso para indicar retroceso).
- Si la estación inicial está al comienzo o al final de la línea sólo hay una posible dirección, que queda así automáticamente determinada.
- Si la estación inicial es una estación intermedia, se elige una dirección aleatoriamente.
- De acuerdo con la dirección de avance del movimiento, se genera una secuencia de estaciones contiguas, hasta llegar a la estación final o al límite establecido para el trayecto. El procedimiento en cada paso o estación es el siguiente:
 - Se pasa a la siguiente estación de la línea de acuerdo con la dirección establecida del movimiento. Hay que tener en cuenta que si se trata de una línea circular, al llegar a la última estación de la lista que la representa se pasa a la primera, y al llegar a la primera en un movimiento de retroceso, se pasa a la última.
 - Se comprueba que la nueva estación no estuviera ya incluida en el trayecto del individuo. Si lo está, y la otra dirección de movimiento es posible, se intenta. Si no se deshecha el individuo.

- Para la nueva estación:
 - Si hay correspondencias se decide aleatoriamente si se continúa en la misma línea o si se cambia.
 - Si se cambia se elige aleatoriamente una de las correspondencias, y en la nueva línea se elige una dirección.

2.3 Función de adaptación

El objetivo es minimizar el número de estaciones del recorrido, pero teniendo en cuenta otras restricciones. Por ejemplo, podemos incluir una penalización por cada trasbordo realizado. Así mismo se penalizará la violación de las restricciones de inclusión o exclusión de las estaciones especificadas. Las penalizaciones deben ser parámetros de entrada al algoritmo.

En función de los valores que se les asigne, primaremos un tipo u otro de trayecto: con pocos transbordos, que nunca pasen por una de las estaciones que se desea evitar, o que pueda pasar si así se reduce considerablemente la longitud del trayecto, etc. La siguiente tabla muestra un ejemplo de posibles valores:

```

constante entero PENAL_TRANSBORDO 1
constante entero PENAL_PROHIB 10
constante entero PENAL_OBLIG 3

```

Un posible esquema de la función de adaptación será el siguiente:

```

funcion adaptacion(TI ndi vi duo i ndi vi duo, TPI ano pl ano,
                    TConsul ta consul ta, ...
{
  real adapt;
  entero num_transb; // número de transbordos
  entero num_prohib; // número de estaciones a evitar
  entero num_obliga; // número de estaciones a visitar

  // se inicializa la adaptación a la longitud
  // del trayecto
  adapt = tamaño(i ndi vi duo.genes);
  // se cuenta el número de transbordos del trayecto
  num_transb = { ... }
  adapt = adapt + num_transb * PENAL_TRANSBORDO;

  // se cuenta el número de estaciones prohibidas
  // por las que pasa el trayecto

```

```

num_prohib = { ... }
adapt = adapt + num_prohib * PENAL_PROHIB;

// se cuenta el número de estaciones por las
// que debe pasar y no pasa

num_oblig = { ... }
adapt = adapt + num_oblig * PENAL_OBLIG;

devolver adapt;
}

```

2.4 Operador de cruce

Los operadores genéticos deben tener en cuenta que deben generar individuos válidos. Por ejemplo, el operador de cruce puede buscar una estación en común entre los individuos a cruzar e intercambiar los segmentos a ambos lados de dicha estación. Un posible procedimiento para buscar un punto de cruce válido en los dos individuos es el siguiente:

- Se toman dos individuos a cruzar, con el procedimiento descrito en el capítulo 2.
- Se elige al azar una posición, el punto de cruce1, de la secuencia de estaciones que define el trayecto del primero de los individuos.
- Se busca en el otro individuo la estación correspondiente. Si se encuentra su posición es el punto de cruce2.
- Si no se encuentra, el punto de cruce1 avanza a la siguiente posición, y se repite el proceso. El avance del punto de cruce1 se realiza de forma circular, hasta volver al punto de partida. Es decir, si se llega al final de la secuencia de estaciones se pasa a la primera.
- Si después de comprobar todas las estaciones del primero de los individuos no se ha encontrado ninguna estación común que permita realizar el cruce, no se aplica el operador.

Hay que tener en cuenta que la posibilidad de que el cruce no se llegue a aplicar, si los dos individuos seleccionados no comparten ninguna estación, obliga a utilizar una tasa de cruce mayor que las que se utilizan en otros algoritmos evolutivos.

2.5 Operador de mutación

El operador de mutación selecciona aleatoriamente una estación del trayecto y genera un nuevo trayecto desde ella hasta la estación final o bien de la estación inicial a la seleccionada. Es decir, la aplicación de este operador sigue los siguientes pasos:

- Se elige al azar una posición, el punto de mutación, de la secuencia de estaciones que define el trayecto del individuo.
- Se decide aleatoriamente si se renueva el trayecto entre la estación inicial y el punto de mutación o entre el punto de mutación y la estación final.
- Se genera un nuevo trayecto para el tramo elegido, de la misma forma que se han generado los trayectos de la población inicial, pero entre las estaciones que corresponda en este caso.

2.6 Consideraciones adicionales

Como es habitual en los algoritmos evolutivos, es necesario hacer un estudio de los parámetros del algoritmo:

- Tamaño de la población
- Número límite de iteraciones del algoritmo
- Porcentaje de cruces
- Porcentaje de mutaciones

para alcanzar un rendimiento óptimo del sistema. En este caso, además de los parámetros también es interesante analizar y obtener resultados sistemáticos, de los valores de las penalizaciones que nos permiten obtener determinados tipos de trayectos.

Por ejemplo, podemos buscar los valores que nos permitan:

- Garantizar que se minimiza el número de transbordos realizados, quizás a costa de violar alguna de las restricciones de pasar o dejar de pasar por las estaciones prohibidas u obligadas.

- Garantizar que no se pasa por ninguna de las estaciones prohibidas, quizás a costa de obtener trayectos más largos o con más transbordos.
- Garantizar que no se pasa por ninguna de las estaciones prohibidas, ni se deja de pasar por ninguna de las obligadas.

3. TRATAMIENTO ALTERNATIVO DE LAS RESTRICCIONES

El proyecto abordado en este capítulo involucra diversas restricciones. Algunas de ellas, como el requisito de que los trayectos empiecen y terminen en las estaciones especificadas en la consulta del usuario, las hemos introducido en la codificación del individuo. No permitimos que existan individuos que no cumplan esta condición.

El resto de las restricciones, que hemos considerado más flexibles, como minimizar el número de transbordos, procurar que el trayecto pase por determinadas estaciones o deje de pasar por otras, se han tratado con técnicas de penalización. Sin embargo, existen otras alternativas.

Puede resultar interesante estudiar cómo afecta a los resultados y a la eficiencia del sistema las siguientes alternativas:

- Permitir que algunos (si se hace con todos el algoritmo sería lentísimo) individuos empiecen y terminen en estaciones aleatorias, penalizando a los que no correspondan a las estaciones del trayecto, y de manera que la penalización se vaya incrementando a medida que avanza la evolución.
- Introducir en la codificación las restricciones de estaciones prohibidas. Al generar los trayectos evitar las prohibidas hasta cierto número de intentos.

4. CON OTRAS RESTRICCIONES

También podemos refinar el proyecto introduciendo especificaciones más detalladas de la red de metro. Algunas posibilidades son:

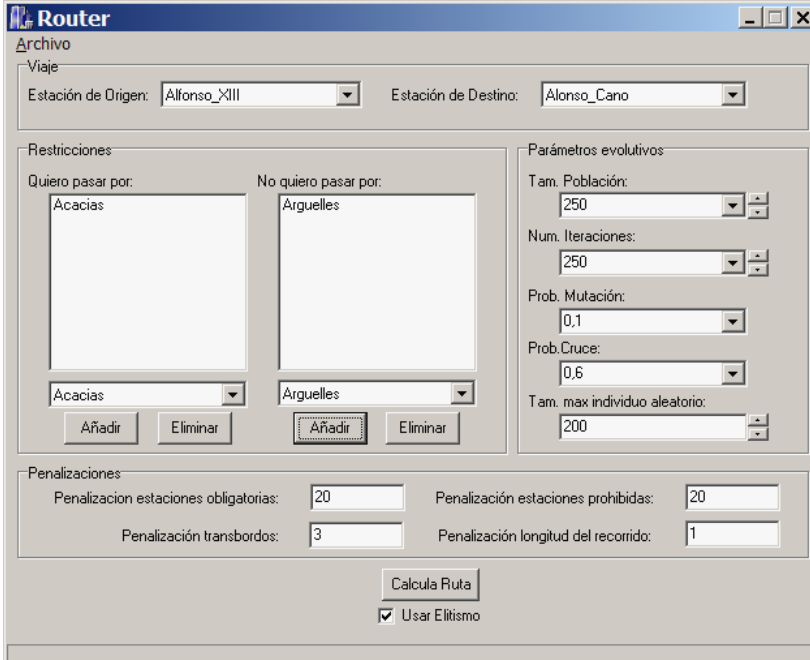
- Asignar un coste específico a cada transbordo: la distancia a recorrer por el usuario puede ser muy variable de unos transbordos a otros.

- Especificar la distancia entre estaciones.
- Especificar la velocidad media de los trenes en cada una de las líneas, que puede variar de unas a otras.

5. UN EJEMPLO DE INTERFAZ GRÁFICA

A continuación se muestra un ejemplo de aplicación que incluye una interfaz gráfica que permite seleccionar diferentes parámetros para el algoritmo, así como visualizar la solución obtenida.

La aplicación permite seleccionar los parámetros del algoritmo: estaciones de origen y destino, restricciones, penalizaciones, tamaño de la población, número máximo de generaciones, probabilidad de cruce y mutación, porcentaje de elitismo, método de selección, método de cruce y método de mutación.

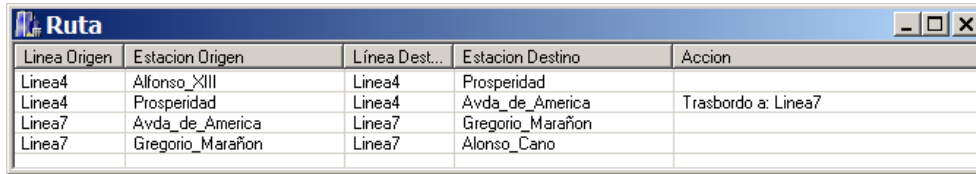


The screenshot shows the 'Router' application window. It has a menu bar with 'Archivo' and a title bar with standard window controls. The main interface is divided into several sections:

- Viaje:** 'Estación de Origen' is set to 'Alfonso_XIII' and 'Estación de Destino' is set to 'Alonso_Cano'.
- Restricciones:** Two lists are shown. 'Quiero pasar por:' contains 'Acacias'. 'No quiero pasar por:' contains 'Arguelles'. Below each list are 'Añadir' and 'Eliminar' buttons.
- Parámetros evolutivos:** 'Tam. Población:' is 250, 'Num. Iteraciones:' is 250, 'Prob. Mutación:' is 0,1, 'Prob. Cruce:' is 0,6, and 'Tam. max individuo aleatorio:' is 200.
- Penalizaciones:** 'Penalización estaciones obligatorias:' is 20, 'Penalización estaciones prohibidas:' is 20, 'Penalización transbordos:' is 3, and 'Penalización longitud del recorrido:' is 1.

At the bottom, there is a 'Calcula Ruta' button and a checked checkbox for 'Usar Elitismo'.

Al terminar la ejecución la aplicación muestra la ruta obtenida:



Línea Origen	Estacion Origen	Línea Dest...	Estacion Destino	Accion
Linea4	Alfonso_XIII	Linea4	Prosperidad	
Linea4	Prosperidad	Linea4	Avda_de_America	Trasbordo a: Linea7
Linea7	Avda_de_America	Linea7	Gregorio_Marañon	
Linea7	Gregorio_Marañon	Linea7	Alonso_Cano	