FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE MADRID

UNIVERSIDAD POLITÉCNICA DE MADRID
FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

CÁLCULO DE TRAYECTOS MEDIANTE ALGORITMOS DE BÚSQUEDA INFORMADA SOBRE GRAFOS PONDERADOS NO DIRIGIDOS

AUTOR: Jesús Manuel Fernández Orchando
TUTOR: Vicente Martínez Orga
FECHA DE PRESENTACIÓN: Julio 2018

Resumen

El problema de búsqueda es un campo ampliamente desarrollado en el área de la Inteligencia Artificial. En este proyecto se propone la implementación de un sistema de búsqueda de trayectos en grafos ponderados no dirigidos minimizando el coste de dichos caminos. Como particularidad sobre los algoritmos ya existentes, el sistema emplea una clasificación de los nodos del grafo que permite obtener la solución del problema de una forma más eficiente. A su vez, esta clasificación permite reducir de manera considerable el espacio necesario para almacenar toda la información acerca de la definición del problema.

La herramienta es aplicable sobre distintos grafos siempre y cuando cumplan una serie de condiciones. El sistema presentado se ha aplicado sobre un ejemplo concreto de grafo: la Red de Metro de Barcelona. El usuario solicitará el cálculo de un trayecto entre dos estaciones cualesquiera de la red y el sistema responderá con un camino a través del grafo, que representa el trayecto entre las dos estaciones dadas minimizando la duración de éste e imprimiendo dicho tiempo/coste mínimo.

Gracias a la arquitectura software de sistema y la modularidad del mismo pueden realizarse modificaciones de manera sencilla tanto en la definición del problema, ya sea otra red del mismo transporte público u otro tipo de grafo, como en el algoritmo de búsqueda empleado para el cálculo de los caminos de coste mínimo.

Índice general

1. INTRODUCCIÓN

En Inteligencia Artificial el término *búsqueda* se refiere a un núcleo fundamental de técnicas que se utilizan en diversos dominios. El ser humano siempre ha requerido servirse de técnicas de búsqueda a la hora de resolver sus problemas más cotidianos. Este tipo de problemas pueden ser, por ejemplo, diagnóstico, resolución de problemas o planificación y, todos ellos, se pueden clasificar como problemas de búsqueda. Muchos de estos problemas a los que se enfrenta el ser humano tienen una apariencia simple pero, sin embargo, pueden volverse muy complejos según la dimensión del mismo, es decir, del número de alternativas que existan. Este número de alternativas se conoce como *espacio de búsqueda*. Es en este punto donde surge la necesidad de emplear la Inteligencia Artificial.

El procedimiento por el cual una máquina puede resolver un problema de búsqueda es, en esencia, el mismo que el del ser humano. La diferencia principal es que la capacidad de cómputo es sustancialmente mayor en la máquina. De este modo, problemas de grandes dimensiones que no podrían ser resueltos por el ser humano en un tiempo razonable, alcanzarían una solución aceptable mediante la aplicación de las técnicas de búsqueda de la Inteligencia Artificial. Una de estas maneras en las que se puede resolver un problema de búsqueda es mediante la aplicación de todas las posibles soluciones al mismo —es lo que comúnmente se conoce por *fuerza bruta.* Lógicamente, cuando la dimensión del espacio de búsqueda crece, cada vez es menos viable seguir empleando la fuerza bruta. De la necesidad de encontrar nuevas formas de buscar soluciones en un problema de búsqueda surgen nuevos algoritmos que optimizan la labor.

En cualquier problema de búsqueda es necesario distinguir dos estados básicos: el *estado inicial* y el *estado final* u *objetivo.* A partir de éstos, se ejecutará un algoritmo de búsqueda que permita encontrar soluciones al problema. Los algoritmos de búsqueda se podrán clasificar dependiendo del tipo de estrategia empleada: *informada* o *no informada.* La estrategia de búsqueda informada permite encontrar soluciones de manera más eficiente que en aquellas estrategias en las que no se cuenta con ninguna información adicional acerca del estado del problema *[Russell & Norvig, 2003]*. Esto es, en las búsquedas no informadas no se posee ninguna otra información que no venga dada por la propia definición del problema. De este modo, toda búsqueda se elaborará de tal forma que se compruebe en cada paso si se ha alcanzado el estado objetivo. Pero en el caso de emplear una estrategia no informada, cuando el estado objetivo no sea alcanzado, se continuará la búsqueda sin tener en cuenta cuál es la mejor opción para continuar.

El problema de búsqueda se representa normalmente como un *grafo.* El grafo es una estructura matemática formada por dos tipos de conjuntos denominados *nodos* (o ***vértices***) y *arcos* (o ***aristas***) *[Gross & Yellen, 2006]*. Los arcos permiten relacionar unos nodos con otros, dotando al grafo de diversos significados dependiendo del problema que estén representando. Por ejemplo, los grafos pueden emplearse para representar circuitos, estructuras moleculares, mapas o estados de un programa informático. De tal forma, puede observarse que la estructura del grafo permite representar situaciones más tangibles, como puede ser un mapa con conexiones por carretera entre distintas ciudades, o menos concretos, como el conjunto de estados en los que se puede encontrar un programa informático. En todos ellos, los nodos del grafo representan los elementos principales que forman el problema (ciudades en el mapa o estados en un programa) y los arcos se encargan de relacionar de alguna forma esos elementos (carreteras del mapa o entradas del programa informático).

En el caso de los problemas de búsqueda, los nodos del grafo representan los estados de dicho problema de búsqueda. Todos ellos tendrán ciertas características que permiten a un algoritmo de búsqueda distinguir entre unos y otros. Los arcos simplemente se encargarían de relacionar los estados de tal forma que se pueda conocer qué estados se alcanzan desde unos estados concretos. Así, si un nodo no tiene ningún arco que llegue hasta él, se podrá decir que no hay forma de alcanzar ese estado del problema. Por lo tanto, una de las clasificaciones existentes para un grafo se basa en la conectividad de sus nodos: grafos *conexos y* grafos *no conexos.* Un grafo conexo es aquel en el que todo nodo perteneciente al mismo es alcanzable desde algún otro nodo. Es decir, existe al menos un camino en el grafo que contiene al nodo que se quiere alcanzar.

El término *camino* es esencial en este proyecto y se define como una secuencia formada por vértices y aristas, de manera intercalada, que describe la conexión entre dos vértices concretos del grafo. En especial, muchos algoritmos se basan en la idea de encontrar en un grafo el camino mínimo —o de coste mínimo en el caso de grafos ponderados— como solución. Así, surge otra característica de los grafos: la *ponderación.* Un grafo es ponderado si sus aristas tienen un coste determinado. Para que ese coste aporte alguna información adicional en el problema no todas las aristas poseerán el mismo valor de coste. Dependiendo del tipo de problema, esos costes pueden representar resistencias en circuitos, kilometraje en mapas de carreteras, duración del recorrido, dificultad del tramo, etc. Es posible que el problema de búsqueda resida únicamente en encontrar un camino con el menor número de tramos recorridos. En este caso, simplemente se tratará de encontrar el camino mínimo y no se tendrá en cuenta si las aristas poseen coste alguno (equivale a que todas las aristas tengan coste con valor 1). Sin embargo, cuando el problema de búsqueda requiere

encontrar un camino con el coste mínimo posible, se debe considerar cada uno de los costes de los tramos entre nodos. El algoritmo escogido para la búsqueda se encargará de decidir en cada punto de la ejecución cuál es la siguiente opción a elegir para continuar con la elaboración del camino que dará como solución.

Otra de las características más importantes de los grafos, en cuanto a problemas de búsqueda se refiere, recae sobre la existencia de ciclos en el mismo. Un *ciclo* es un camino cerrado, es decir, el nodo inicial coincide con el nodo final. A la hora de resolver un problema de búsqueda, la existencia de ciclos en el mismo puede suponer un gran problema ya que el algoritmo podría estar buscando una solución recorriendo varias veces el mismo nodo, de tal forma que su ejecución se prolongue en tiempo infinito. Este inconveniente no se dará nunca en el caso de emplear árboles en el problema de búsqueda. Los *árboles* son un caso especial de los grafos, en el que para alcanzar cualquier nodo del árbol tan solo existe un camino posible. Esta característica provoca que se simplifique la ejecución del algoritmo aunque, como se verá en el capítulo siguiente, la manera en que se recorra dicho árbol condicionará considerablemente la búsqueda de la solución.

Por último, otra propiedad de los grafos que está muy relacionada con este proyecto es la existencia de aristas con un único sentido de recorrido. Estos grafos se conocen como *grafos dirigidos.* Es una limitación bastante importante cuando se requiere encontrar caminos dentro de un grafo ya que puede darse el caso de que el algoritmo se encuentre en un punto determinado de la ejecución en el que desde el nodo actual no existe ninguna arista que pueda tomarse en sentido de salida. De otro modo, aunque el grafo sea conexo puede ocurrir que haya nodos desde los que no se pueda mover *(pozos) y* nodos que no se puedan alcanzar porque todas sus aristas salen del mismo *(fuentes).* Si las aristas se pueden tomar desde cualquiera de los dos sentidos posibles el grafo se denomina *no dirigido.*

En este proyecto se emplea un algoritmo de búsqueda de caminos de coste mínimo en un grafo ponderado no dirigido. Se aplica sobre la red de Metro de Barcelona, que se puede representar como un grafo de tamaño considerable en el que los nodos serán las estaciones de Metro y las aristas, los tramos de vía que conectan las diversas estaciones. El problema de búsqueda, en este caso, se centrará en la obtención del camino de coste mínimo desde una estación de inicio a otra de destino entre las cuales desea moverse un viajero —se asume que el viajero desea realizar el trayecto evitando en la medida de lo posible la incomodidad de la realización de transbordos. Los tramos de vía poseen un coste, que caracterizarán la búsqueda dependiendo de qué tramo sea el menos costoso para el algoritmo. Este grafo tiene algunas peculiaridades, como por ejemplo, la existencia de distintas líneas de Metro que implican un cambio de

tren si se pasa de circular de una línea a otra. Los tramos que separan cada par de estaciones se pueden recorrer en ambos sentidos, por lo que el grafo de la red de Metro no es dirigido.

El algoritmo empleado está sujeto a cambios en la red de Metro, como podrían ser tramos cortados por obras o aparición de nuevas estaciones e, incluso, nuevas líneas de Metro. Y, si alguno de estos cambios repercute en el camino de coste mínimo, el algoritmo decidirá en el momento de la ejecución si la nueva alternativa es mejor o si, por el contrario, la mejor alternativa existente hasta el momento ya no puede ser escogida. Además, se pretende que esta aplicación calcule de la manera más rápida posible cuál es la ruta que mejor se adapta a las necesidades del viajero, por lo que es necesario que se empleen diversos métodos de poda de alternativas y gracias a los cuales no se requiera analizar a cada momento todos los posibles caminos existentes. Como se ha indicado anteriormente, la gran dimensión del espacio de alternativas en un problema de búsqueda influye directamente en el coste computacional de hallar una solución y, en este caso, el gran número de estaciones existentes en la red de Metro de Barcelona dificultaría la tarea de encontrar una solución aceptable en tiempo razonable si no se tomaran en consideración estos aspectos.

La estructura de este proyecto es la siguiente:

- **Estado del arte:** se describirán aspectos de las técnicas de búsqueda en la Inteligencia Artificial y de las formas de representación de un problema de búsqueda. Además, se analizarán posibles métodos de búsqueda de la solución en el grafo descrito anteriormente.
- **Desarrollo del sistema:** se detallará la aplicación propuesta en este proyecto de fin de carrera.
- **Resultados experimentales:** se mostrarán los resultados obtenidos tras la ejecución de la aplicación.
- **Conclusiones y líneas futuras:** se explicarán las conclusiones derivadas de los resultados obtenidos y posibles ampliaciones del sistema.

2. ESTADO DEL ARTE

Dentro del campo de la Inteligencia Artificial, el *problema de búsqueda* se define como la búsqueda de secuencias de acciones que permitan alcanzar uno o varios estados deseados *[S. J. Russell & Norvig, 2003]*. Existen problemas de distinta naturaleza que pueden ser clasificados como problemas de búsqueda, ya que es posible extraer una secuencia de pasos a seguir para alcanzar el objetivo del problema. Uno de los ejemplos de aplicación más típicos en los problemas de búsqueda es el de planificación. En este tipo de problemas suele ser un robot *[Koenig & Simmons, 1998]*, o máquina en general, el que debe desempeñar una serie de tareas hasta alcanzar una situación concreta, desplazamiento de objetos dentro de un espacio determinado, por ejemplo. Otro problema que puede servir como ejemplo de aplicación es aquel que se basa en juegos de dos oponentes, donde, en cada turno, el jugador correspondiente busca la mejor jugada posible. Además, las técnicas de búsqueda se emplean habitualmente para la búsqueda de caminos óptimos en grafos. Estos caminos, dependiendo del problema a resolver pueden ser los de menor número de aristas, los de coste mínimo, etc. Incluso, problemas de diagnóstico pueden verse desde el punto de vista de un problema de búsqueda, ya que se trata de emplear los síntomas observados en el tipo de problema para encontrar una causa común a todos ellos. En general, gran número de problemas, que no son nada ajenos a la vida de un ser humano, pueden clasificarse como problemas de búsqueda. Todos ellos, al modelarse como tal, tendrán en común una serie de conceptos básicos que se explican a continuación.

2.1. El problema de búsqueda

En cualquier problema de búsqueda se parte de un *estado inicial*. Ésta es la manera de describir la situación en la que se encuentra el problema al iniciar la búsqueda de la secuencia de acciones y que aportará la información necesaria para comenzar la toma de decisiones. Por ejemplo, una persona desea realizar un circuito turístico por Italia, visitando diversas ciudades del país. Quiere realizarlo de tal forma que recorra todas las ciudades de la manera más eficiente posible. Esto es, no debería visitar dos veces la misma ciudad ni debería estar perdiendo tiempo en los trayectos entre ciudades. La información básica que debería contener el estado inicial del problema sería la ciudad de la que parte el viajero. Además, si en el problema se tienen en cuenta otras características, también deben tomarse en cuenta en la descripción del estado inicial. Es decir, si el viajero no desea gastar demasiado dinero en los trayectos, una información útil que debería figurar en el estado inicial sería la cantidad de dinero con la que cuenta el viajero al inicio del viaje.

El otro elemento básico de un problema de búsqueda es el *estado final*. En él se describen las características del problema que deben cumplirse para dar por finalizada la

búsqueda de acciones. Es el objetivo o meta del problema. La secuencia de acciones se completará cuando, mediante su ejecución, pueda alcanzarse el estado final partiendo desde el estado inicial. En el ejemplo descrito anteriormente, el estado final podría representarse como un conjunto de ciudades visitadas que contiene todas las ciudades que deseaba visitar el viajero y una condición sobre el dinero restante al finalizar el circuito turístico.

Estado inicial del problema:
- *Ciudades visitadas = [Milán]*
- *Dinero = 800*

Estado final del problema:
- *Ciudades visitadas = [Milán, Venecia, Padua, Pisa, Florencia, Roma]*
- *Dinero > 50*

Figura 2.1: Posibles estados inicial y final del problema del circuito turístico por Italia.

Tanto el estado inicial del problema, como el estado final, pertenecen al espacio de estados definidos en dicho problema de búsqueda. En este conjunto se engloban todos aquellos estados en los que es posible que el problema se encuentre en un momento determinado.

La secuencia de acciones que constituirían la solución del problema sería aquella que contuviera los pasos a seguir para realizar el viaje. Algunas de estas acciones serían las de alquilar un vehículo para la realización de recorridos, comprar billetes de tren o autobús, etc. Tanto los estados inicial y final, como las acciones que se pueden realizar forman parte de la formulación del problema de búsqueda. La capacidad de encontrar una solución al problema está muy relacionada con el grado de detalle con el que se describen las acciones. Comparando las acciones *"efectuar un paso con el pie izquierdo" y "caminar"* se puede comprobar la diferencia de detalle entre ambas. Encontrar una solución a un problema en el que las acciones se detallan tanto como en el primer caso puede volverse tan complejo que no se logre hallar dicha solución. Sin embargo, si las acciones son demasiado generales, la solución del problema no será lo suficientemente adecuada. Por ejemplo, en el ejemplo del viajero no se puede tomar como única acción del problema la de *"visitar ciudades"*, porque no ayudaría a calcular el mejor recorrido a realizar. Será necesario desglosar esta acción en otras más simples, y así sucesivamente, hasta que tengan un nivel de detalle adecuado para el problema en cuestión.

Las acciones que se pueden realizar en un problema determinado se modelan mediante los *operadores [Palma Méndez & Marín Morales, 2008]*. Dichos operadores se ejecutan bajo unas precondiciones concretas, especificadas en la formulación del problema de búsqueda. Las precondiciones describen situaciones en las que se debe encontrar el problema para poder realizar la acción a la que están asociadas. Pueden estar formadas por una o varias expresiones, debiendo cumplirse cada una de ellas al

mismo tiempo para que la precondición se evalúe como cierta. Por otro lado, al ejecutarse un operador se producen unos efectos asociados. Los efectos se aplican sobre el estado del problema, modificando la situación en la que se encontraba.

Dependiendo del problema, los operadores pueden tener asociado un coste de ejecución. Es decir, el coste que acarreará a la solución del problema si el algoritmo decide ejecutar una acción concreta. En el ejemplo del circuito turístico por Italia, el coste de algunos de los operadores, como los que impliquen un desplazamiento entre dos ciudades, entre otras opciones podría representar el coste monetario de ese desplazamiento. Es decir, si existen operadores representando las acciones de *viajar en tren* o *viajar en autobús*, el algoritmo tendría información para decidir cuál de las dos opciones permite obtener un coste inferior para la solución del problema. Por el contrario, si la formulación del problema no otorga mayor prioridad de ejecución a ningún operador, el coste de todos ellos podría ser simplemente 1.

En consecuencia, se extrae que la correcta y adecuada formulación del problema es básica para aplicación exitosa del algoritmo de búsqueda. Ha de aparecer como información del problema únicamente aquella que realmente pueda repercutir en la solución. Esto es, en el ejemplo descrito, debe aparecer información que haga referencia a los movimientos del viajero a cada momento de la ejecución, aquella estrechamente relacionada con ese movimiento, como la compra de billetes, etc. Por otro lado, no sería necesaria en este problema otra información, considerada pues irrelevante, como la vestimenta del viajero, el número de comidas que realiza durante el viaje u otras acciones que se desempeñan en un viaje pero que no son determinantes a la hora de obtener una solución para este tipo de problema. De igual modo, como se ha mencionado anteriormente, es esencial saber escoger el grado de detalle, tanto de las acciones como del resto de información acerca del universo del problema, en la formulación.

Estado inicial del problema:
- *Ciudades visitadas = [Milán]*
- *Dinero = 800*

Estado final del problema:
- *Ciudades visitadas = [Milán, Venecia, Padua, Pisa, Florencia, Roma]*
- *Dinero > 50*

Acciones: viajar en tren, viajar en autobús, viajar en coche, comprar un billete de tren, comprar un billete de autobús, alquilar un coche, ...
Predicados: ciudadActual(Venecia), dineroActual(720), ...

Figura 2.2: Posible definición del problema del viajero del circuito turístico.

2.2. Buscando la solución óptima

La búsqueda es el proceso que se encarga de encontrar la secuencia de acciones más adecuada para el problema en cuestión *[S. J. Russell & Norvig, 2003]*. Para llevar a cabo esta tarea, existen diversos algoritmos que, aplicados a un problema en concreto, retornan una solución formada por una serie de pasos que deben ser ejecutados para alcanzar el estado objetivo. El aspecto que caracterizará esta búsqueda de la secuencia será la intención de minimizar cierto coste asociado a cada una de las acciones o maximizar el valor de una función que representa el beneficio de éstas. Aquella solución que, dentro del espacio de alternativas del problema, permita obtener el menor coste o el mayor beneficio será la *solución óptima* del problema. Pero en muchas ocasiones, la complejidad del problema dificulta en gran medida la capacidad de encontrar dicha solución óptima. Por este motivo, es posible que el algoritmo deba conformarse con una solución suficientemente buena, pero no la mejor. De este modo, algunos algoritmos están orientados a comparar las alternativas posibles en un problema de búsqueda y decidir cuál es la mejor de entre todas ellas, en lugar de buscar por sí mismo la solución óptima *[Clarke et al., 2003]*. Resulta más sencillo comparar soluciones e indicar cuál es mejor —tanto evaluando cuál tiene el menor coste como evaluando sus funciones de beneficio— que tener que partir de cero con la definición del problema y comenzar a buscar aquella solución que dé el mejor valor. Esto repercute directamente en el tiempo que se tarda en encontrar una solución al problema. Es decir, si el algoritmo aplicado se conforma con una solución aceptable, tardará menos tiempo en ejecutar que si debiera encontrar la solución óptima del problema de búsqueda. Es una de las razones fundamentales por las que se escogen este tipo de algoritmos, ya que existen situaciones en las que es más conveniente obtener una solución suficientemente buena en poco tiempo que obtener la mejor de todas las posibles en un tiempo elevado que, probablemente, provoque la inviabilidad de la aplicación del algoritmo.

2.3. Representando el problema de búsqueda

2.3.1. El grafo

La estructura de representación más comúnmente utilizada en un problema de búsqueda es el *grafo.* El grafo es una estructura matemática definida por dos conjuntos de elementos: el conjunto de nodos (o vértices), representado como *V,* y el conjunto de aristas, representado como *E [Gross & Yellen, 2006]*. Las aristas de un grafo pueden tener asociados uno o dos nodos, dependiendo de si los nodos que aparecen en los extremos de la arista son el mismo o no. De este modo surge el término de *adyacencia.* Dos nodos de un grafo son adyacentes si existe una arista que los conecta directamente,

es decir, si cada uno de esos dos nodos está en un extremo de la misma arista. Este término constituye uno de los más importantes en cuanto a problemas de búsqueda se refiere, como ya se detallará más adelante.

El término de conectividad de un grafo también supone un aspecto destacable en las tareas de búsqueda. Un grafo es *conexo* si cualquier nodo del mismo es alcanzable desde cualquier otro nodo del grafo. Esto significa que, dados dos nodos cualesquiera del grafo, debe existir un camino que los contenga. Un *camino* en un grafo está formado por una secuencia que alterna nodos y aristas y que representa la manera de desplazarse de unos nodos a otros a través de las aristas que los unen. Dependiendo del grafo, es posible que exista más de un camino uniendo dos nodos concretos. Todos los caminos se pueden caracterizar por el número de aristas que lo componen, el número de nodos por los que pasa e, incluso, un coste si las aristas del grafo tienen peso.

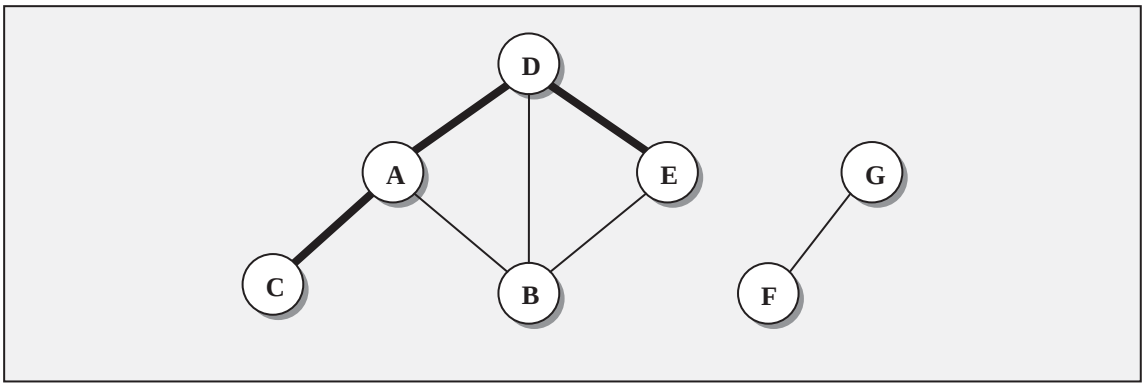


Figura 2.3: Ejemplo de grafo, donde los nodos adyacentes al nodo A son los nodos B, C y D. Se trata de un grafo no completamente conexo, puesto que existen nodos a los que no se puede llegar desde cualquier otro (F y G, por ejemplo). El camino marcado con trazo más grueso es uno de los posibles entre los nodos C y E.

Por lo general, un camino suele estar formado por aristas que pueden recorrerse en ambos sentidos. Cuando todas las aristas de un grafo pueden recorrerse de este modo, se dice que el grafo es *no dirigido.* Este caso presenta algunas ventajas a la hora de trazar caminos dentro del grafo al no existir tantas restricciones si se quiere alcanzar un nodo determinado desde otro cualquiera. Por el contrario, un grafo es dirigido (denominándose *digrafo*) cuando sus aristas solo pueden recorrerse en un sentido. En contraposición a los grafos no dirigidos, en los digrafos puede darse el caso en el que, siendo un grafo conexo, exista un nodo que no pueda alcanzarse desde ningún otro debido a que todas las aristas asociadas a dicho nodo se tienen que tomar en el sentido opuesto al mismo, esto es, todas las aristas salen del nodo y ninguna se dirige hacia él (como sucede con el nodo C en la Figura 2.4).

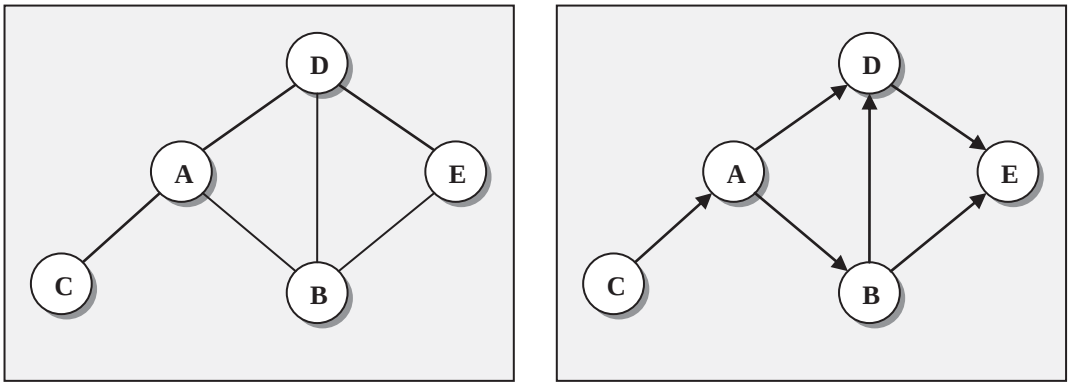


Figura 2.4: Representación de un grafo no dirigido (izquierda) frente a un digrafo (derecha).

En caso de existir en un mismo grafo tanto aristas dirigidas como no dirigidas, se dice que se trata de un *grafo mixto.* En la realidad quizá sea más fácil imaginarse este tipo de situación. Por ejemplo, si se desea representar con un grafo el mapa de calles de una ciudad, es muy común que existan tanto vías de doble sentido, que se expresarían como aristas no dirigidas, como vías de sentido único, que figurarían en el grafo como aristas dirigidas según el sentido de la marcha. Aún así, cualquier grafo no dirigido puede transformarse en un grafo dirigido *[Tarjan, 1971]*. Ante la idea de que las aristas de un grafo no dirigido pueden recorrerse en cualquiera de los dos sentidos, cada una de éstas puede sustituirse por otras dos aristas, cada una de ellas orientas en sentidos opuestos (Figura 2.5).

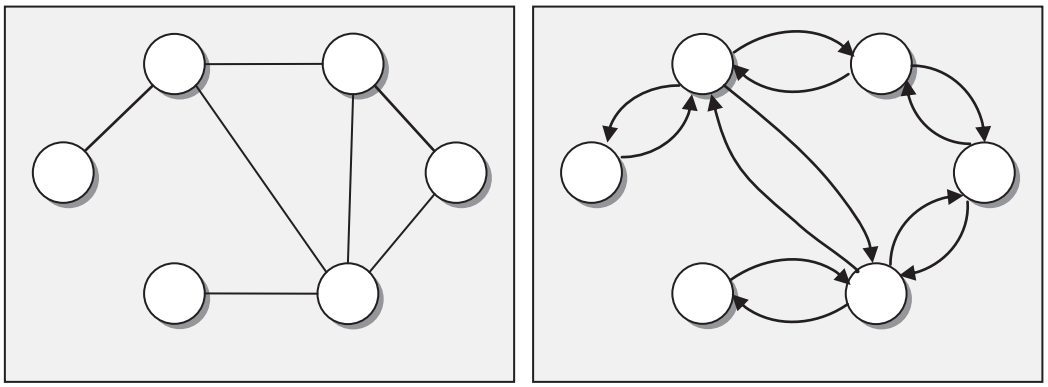


Figura 2.5: Grafo no dirigido (izquierda) transformado en un grafo dirigido equivalente (derecha).

Otra característica de los grafos que también debe tenerse en cuenta en la aplicación de un algoritmo de búsqueda es el grado de cada uno de sus vértices. El *grado* de un nodo de un grafo no dirigido es el número de aristas que inciden en el mismo, contando como dobles aquellas aristas cuyos dos extremos están sobre el mismo nodo (bucles). En el caso de los grafos dirigidos, este término se separa en dos conjuntos: el grado de entrada al nodo y el grado de salida. El primero hace referencia al

número de aristas que terminan en el nodo en cuestión; el segundo, al número de aristas que deben iniciar su recorrido en ese nodo. De este modo, cuando una arista contribuye al grado de entrada en un nodo, está contribuyendo a su vez en el grado de salida de un nodo adyacente a éste.

Dentro de un grafo y según la disposición de las aristas que lo forman, pueden existir ciclos. Un *ciclo* es aquel camino, con al menos una arista, que está cerrado, es decir, que el nodo inicial del camino y el nodo final son el mismo. Esta situación implica que para algún par de nodos en el grafo, exista más de un camino que los comunique. Para que esta circunstancia no fuera posible para ningún par de nodos, el grafo tendría que ser acíclico. De esta manera, en caso de haber un camino entre dos nodos del grafo —caso de los grafos conexos—, éste sería uno y solo uno. Esta característica se da siempre en un tipo de grafo denominado árbol. La estructura del *árbol* se define como un grafo acíclico y conexo. En él, dados cualquier par de nodos, siempre existirá un camino único que los conecte a través de una secuencia finita de aristas. Los problemas de búsqueda que puedan representarse con la estructura de un árbol se vuelven mucho más sencillos que aquellos que únicamente puedan representarse como un grafo común. La ventaja se debe precisamente a la característica de la existencia de un camino único entre dos nodos cualesquiera. Esto significa que será imposible que el algoritmo de búsqueda aplicado quede *atrapado* en un bucle infinito impidiendo que finalice su ejecución. Es decir, el algoritmo encargado de encontrar la solución al problema, al construir el camino óptimo, podría entrar en uno de los ciclos existentes en el grafo y, a través de él, pasar un número de veces infinito por el mismo conjunto de nodos sin apreciar que se sitúa en uno de los ciclos del grafo, de modo que no recorra a partir de ese momento ningún otro nodo para poder hallar la solución. Por tanto, en una situación ideal, el problema de búsqueda se representaría mediante un árbol para poder evitar con completa seguridad la situación anterior. Sin embargo, la mayoría de los problemas no puede expresarse de un modo en el que no exista ningún ciclo.

2.3.1.1. El grafo como herramienta de búsqueda

El espacio de estados del problema está representado en los nodos del grafo. Concretamente, por cada nodo del grafo, se tiene un posible estado del problema. Si dos estados del problema son adyacentes en el grafo que los representa significa que se puede transitar de un estado al otro en un solo paso en la ejecución del algoritmo, es decir, tras aplicar una de las acciones definidas en el problema. Es importante indicar que el grafo representa una situación estática del problema, esto es, su definición. Aporta información sobre todos los estados posibles del problema, desde qué estados se puede transitar a qué otros y la forma de hacerlo, etc. La forma de representar la ejecución del algoritmo se realiza mediante la estructura de árbol, de tal manera que el

estado inicial de la ejecución se correspondería con la raíz del árbol y los descendientes de dicho nodo serían los estados adyacentes al estado inicial. Esta diferencia se puede apreciar más claramente bajo el supuesto de que en la ejecución del algoritmo se pase dos veces por el mismo estado. En el árbol sí que aparecerán dos nodos representado a ese mismo estado, puesto que no pueden existir ciclos. Por el contrario, en el grafo, que tan solo representa la definición del problema, este estado aparecerá una única vez, porque se está haciendo referencia a la existencia de ese estado en el problema y no al número de veces que la ejecución se encuentra en dicho estado. La posibilidad de que un estado pueda ser recorrido más de una vez por el algoritmo de búsqueda puede estar relacionada con la presencia de acciones reversibles. Éste puede ser el caso de problemas como los de movimiento de bloques, movimiento de fichas sobre un tablero o búsqueda de rutas.

Por otro lado, es necesario indicar que no es lo mismo representar los distintos estados de un problema como un grafo a representar la situación, o el concepto, del propio problema con un grafo. Es decir, en todo problema de búsqueda es posible representar los estados como los nodos de un grafo y donde las aristas representan la forma de transitar de unos a otros durante la ejecución de un algoritmo. Pero no todos los problemas pueden estar representados con un grafo. Por ejemplo, un mapa de carreteras o un circuito electrónico pueden verse, en sí mismos, como un grafo. En el caso del primero, los nodos representan las ciudades y las aristas, los tramos de carretera y, en el segundo caso, los nodos pueden ser las puertas del circuito y, sus aristas, los tramos conductores. Sin embargo, un problema en el que una grúa deba situar cierto número de bloques de una determinada manera no puede imaginarse constituyendo un grafo en sí mismo. Para comprender mejor esta idea, se puede observar la Figura 2.6 y la Figura 2.7, ambas realizadas a partir del ejemplo del apartado *El problema de búsqueda.*

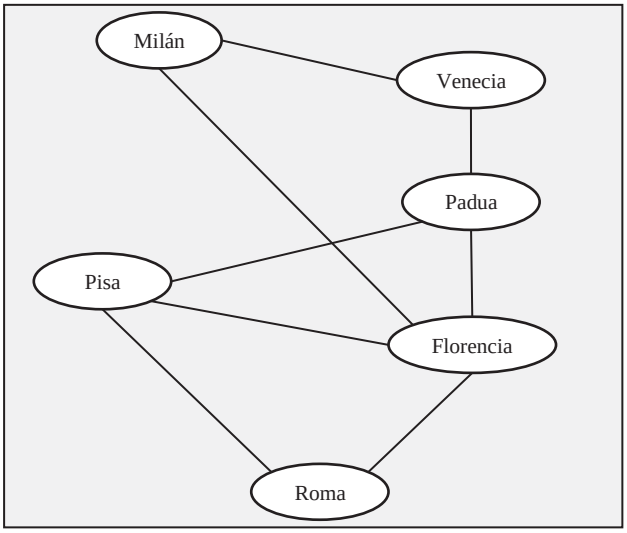


Figura 2.6: Representación del concepto del problema de un mapa de carreteras como un grafo.

Para aclarar aún más este aspecto, se puede mencionar que el grafo de la Figura 2.6 nunca cambiará para este mismo problema. Es decir, sean cuales sean los aspectos que se quieran tener en cuenta a la hora de resolver el mismo problema, por ejemplo, llevar la cuenta del dinero gastado por el viajero, contar el número de ciudades visitadas o conocer cuántas ciudades se pueden recorrer viajando únicamente en tren, la situación de las ciudades en el mapa siempre será la misma. Por el contrario, el grafo resultante o, concretamente, el árbol que representa la ejecución de un algoritmo específico (Figura 2.7) podrá variar dependiendo del propio algoritmo empleado para su resolución.

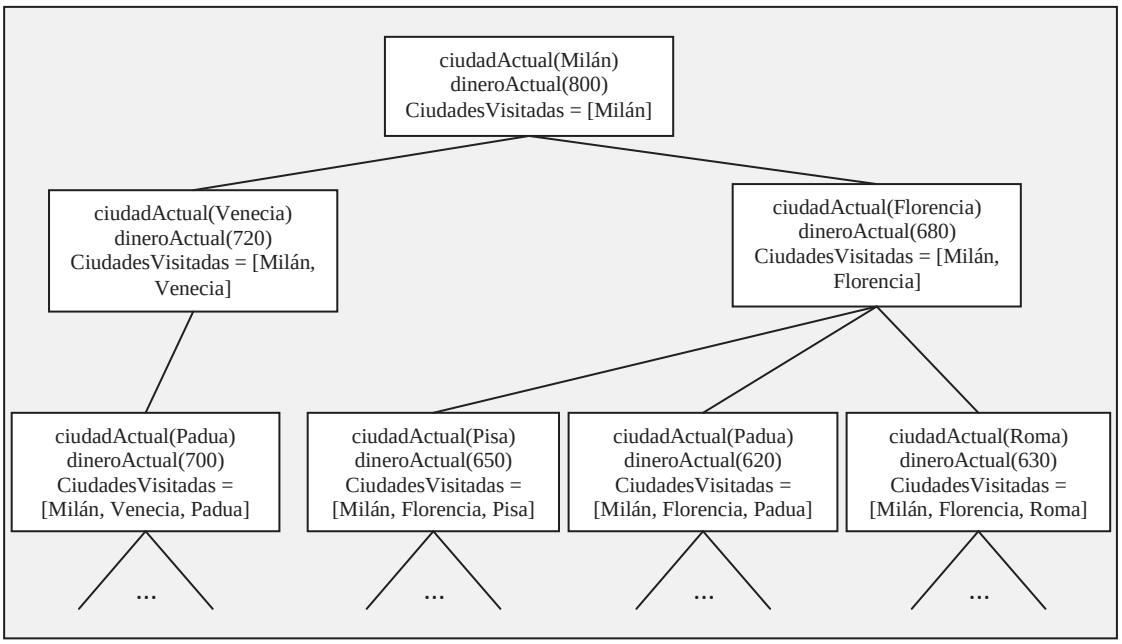


Figura 2.7: Árbol que representa la ejecución de un posible algoritmo de búsqueda.

En el caso de las figuras que representan el problema del circuito turístico no se ha incluido la posibilidad de deshacer acciones realizadas con anterioridad, como se comentaba anteriormente. En este problema de búsqueda el hecho de deshacer una acción se llevaría a cabo volviendo a una ciudad que se acaba de abandonar. Claramente, este puede suponer un gran problema para la ejecución de los algoritmos de búsqueda, puesto que dicha ejecución podría quedar por tiempo infinito indicando el trayecto únicamente entre dos ciudades del mapa, sin evolucionar hacia la solución del problema. La idea de que un algoritmo de búsqueda trate de deshacer acciones ya realizadas no es descabellada. Si se analiza ligeramente el problema del circuito turístico italiano, se puede extraer que la idea fundamental de un algoritmo de búsqueda sea viajar a la ciudad más próxima a la actual. En ese caso, se puede imaginar que el viajero acaba de llegar a Padua, procedente de Venecia. A la vista del gráfico de la Figura 2.6, desde Padua se puede viajar por carretera hasta Pisa, Florencia y Venecia —

nodos adyacentes al nodo representante de Padua. Sin embargo, si el algoritmo tiene únicamente en cuenta qué ciudad es la más cercana a la actual, decidirá que la mejor para esta labor será Venecia, volviendo nuevamente a la situación anterior.

En presencia de esta característica en los problemas de búsqueda, los árboles generados por la ejecución del algoritmo tienen infinitas posibilidades, puesto que el grafo que representa la situación en el mapa de las ciudades contiene infinitos caminos. Pero, obviamente, el aspecto más importante es que no todos esos caminos son válidos para obtener la solución, por lo que es por este motivo que un buen algoritmo de búsqueda debe poder distinguir en qué momento se está tomando una rama infinita del árbol, que no permite alcanzar el objetivo del problema.

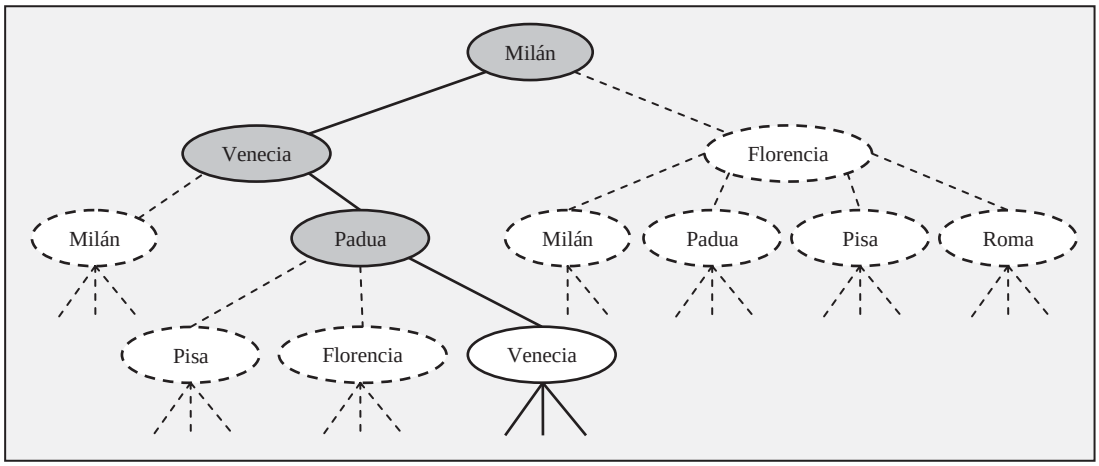


Figura 2.8: Consecuencia de acciones reversibles en un problema de búsqueda.

En el grafo o en el árbol de búsqueda del problema, las aristas representan la manera de cambiar de un estado a otro. Esto se realiza mediante la aplicación de las acciones definidas en el problema a los diversos estados donde sea posible realizarlas. Es decir, la aplicación de una acción concreta a un estado determinado conlleva que ese estado se transforme en otro —también representado en la formulación del problema, esto es, perteneciente al espacio de estados. Ambos estados serán adyacentes en el grafo de estados o uno descendiente del otro en el árbol de búsqueda. A partir de todos estos elementos, se pueden extraer algunos de los componentes básicos en un árbol de búsqueda *[S. J. Russell & Norvig, 2003]*:

- *Estado actual:* el estado, perteneciente al espacio de estados del problema, en el que se encuentra la ejecución del algoritmo en un momento determinado.
- *Nodo padre:* estado, perteneciente también al espacio de estados del problema de búsqueda, a partir del cual se ha derivado un estado concreto mediante la aplicación de una acción determinada.

- *Acción:* representada por una arista del árbol, es la acción aplicada a un nodo padre para generar un nuevo estado de la ejecución.
- *Coste del camino:* normalmente representado como *g(n),* representa el coste acumulado que acarrea realizar el camino a través del árbol de búsqueda desde el nodo raíz o inicial del problema hasta otro nodo concreto, *n.* Puesto que en un árbol, el camino entre dichos nodos es único, este coste también será único.
- *Profundidad:* es el número de pasos, o número de aristas recorridas, desde el estado inicial de la ejecución hasta un estado determinado de cual se quiere conocer su profundidad.
- *Nivel:* coincide con la definición de profundidad, pero se emplea en muchas ocasiones para hacer referencia a todos los nodos que están a una misma profundidad en el árbol.

En el árbol de búsqueda cabe distinguir dos tipos de nodos: aquellos que ya han sido desarrollados y aquellos que aún no han sido explorados. El primer grupo de ellos son todos los nodos del árbol que poseen al menos un descendiente. El hecho de que ya hayan sido explorados significa que se han analizado las posibles acciones aplicables a ese nodo y, por tanto, qué nodos se pueden alcanzar desde él. Una vez que un nodo se ha desarrollado y se conocen los nodos descendientes, no será necesario volver a desarrollarlo, puesto que no se obtendrá nueva información acerca de sus descendientes.

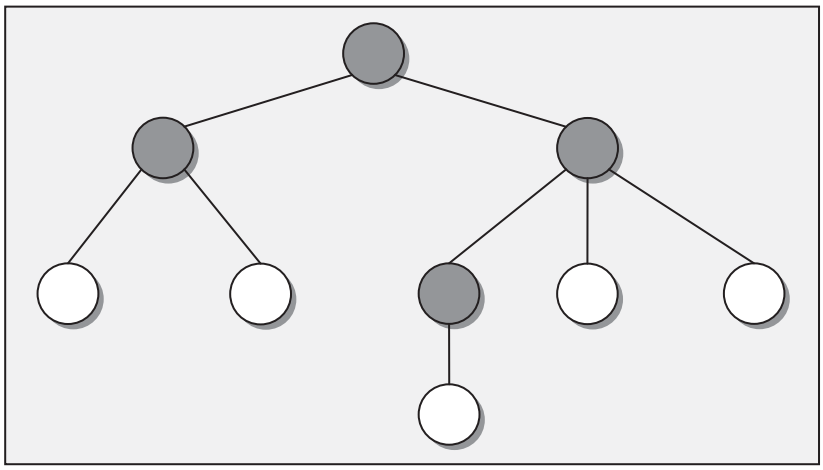


Figura 2.9: Los nodos sombreados representan nodos ya explorados del árbol de búsqueda; los nodos en blanco son nodos hoja.

Es importante recordar que el hecho de que un nodo explorado no se vuelva a desarrollar no implica que no se vuelva a pasar por el estado al que representa durante la ejecución. Un nodo en un árbol de búsqueda representa a un solo estado del espacio de búsqueda, pero un estado puede estar representado por distintos nodos a lo largo de la expansión del árbol.

El segundo grupo de la clasificación anterior está formado por los nodos hoja del árbol. Son todos aquellos nodos del árbol que figuran como posibles alternativas, pero por las que el algoritmo de búsqueda empleado aún no se ha decidido a continuar. Según el ejemplo del circuito turístico, se puede imaginar que el viajero ya ha visitado las ciudades de Milán, Venecia y Padua. Estas ciudades serían representadas el árbol de búsqueda como nodos desarrollados. El resto de ciudades a las que se puede viajar desde estas últimas serían posibles alternativas para dar el siguiente paso, que aún no habrían sido exploradas. De este modo, las ciudades que no ha visitado el viajero estarían representadas por nodos hoja del árbol.

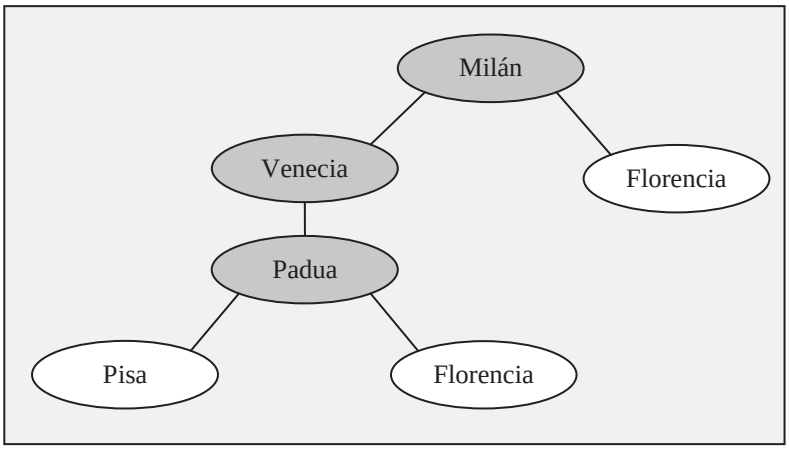


Figura 2.10: Las ciudades visitadas figuran como nodos explorados del árbol de búsqueda, mientras que las ciudades a las que se puede ir en el siguiente paso son hojas de árbol.

El algoritmo seleccionado para realizar las tareas de búsqueda escogerá un nodo hoja de árbol para desarrollarlo. Para ello, se regirá por una serie de criterios, dependiendo del funcionamiento del propio algoritmo, como podría ser el primero de una lista, aquel que permita obtener el mejor valor de una función, etc. Una vez seleccionado un nodo de todos los posibles, se encargará de expandirlo, obteniendo de esta manera nuevos nodos hoja. El algoritmo repetirá este proceso de expandir nodos hoja hasta que en uno de ellos esté representado el estado objetivo del problema.

2.3.2. Otras representaciones

Otra forma de representar un problema de búsqueda es mediante *computación evolutiva [Palma Méndez & Marín Morales, 2008]*. La computación evolutiva está inspirada en el comportamiento de la naturaleza *[Holland, 1975]*, atendiendo a la forma de evolución de los seres vivos (selección natural, aportación genética, etc, ...). Dentro de este campo se emplean los denominados *algoritmos evolutivos*, basados en técnicas probabilistas. La principal característica de este tipo de algoritmos es que las soluciones aportadas pueden variar para un mismo conjunto de datos de entrada, esto es, son

algoritmos no deterministas. Esto se debe a la forma en la que toman las decisiones, realizadas con aleatoriedad según los valores de ciertas probabilidades. En sus inicios en la Inteligencia Artificial, la computación evolutiva no era considerada una técnica eficiente, sin embargo, la aparición de máquinas capaces de realizar cómputos mayores en un tiempo mucho menor hizo que comenzara a resultar útil a la hora de resolver algunos de los problemas más típicos. Gracias a las altas prestaciones de los computadores actuales, los algoritmos evolutivos resultan ser más viables, puesto que son capaces de encontrar buenas soluciones en un tiempo razonable. Estos son algoritmos heurísticos, que se basan en la evolución que se obtiene a lo largo de su aplicación hasta obtener un resultado final. Debido a la potencialidad de los algoritmos evolutivos, es posible resolver algunos problemas de gran dimensión.

Los algoritmos evolutivos se basan en el concepto de población. La *población* de un problema es un conjunto de posibles soluciones que existe en un determinado momento en la ejecución de un algoritmo evolutivo. Cada uno de los elementos de ese conjunto se denomina *individuo,* o lo que es equivalente, cada solución del problema de búsqueda es un individuo. Para iniciar la búsqueda, se parte de una población inicial, generalmente creada de manera aleatoria, aunque siempre es posible agregar cierto conocimiento que permita facilitar la búsqueda de una solución lo suficientemente buena. Al igual que las soluciones en un grafo, representadas por un camino compuesto por nodos y aristas, poseen un coste, las soluciones dadas por un algoritmo evolutivo —es decir, cada uno de los individuos de la población— también poseen un valor que posibilita la comparación entre todas ellas con el objetivo de escoger la mejor. Este valor se obtiene a través de una *función de evaluación,* que se aplica a cada uno de los individuos de la población actual para conocer su adecuación, es decir, cómo de buena es para el problema al que atañe. Los individuos de la población se representan mediante una secuencia de genes —por ejemplo, secuencias de unos y ceros en un alfabeto binario— que los permita distinguir de cualquier otro. Así, dos individuos diferentes tendrán dos secuencias genéticas distintas, del mismo modo que dos individuos iguales deberán estar representados con la misma secuencia de genes.

Tal y como sucede en la naturaleza, las poblaciones del problema evolucionan a medida que suceden distintos eventos. En el mundo de los seres vivos, los individuos se reproducen y tienen descendencia, otros modifican alguna de sus características con el paso de tiempo a fin de adaptarse mejor a las condiciones de su entorno y, algunas especies, mueren. Todos estos aspectos son los que se quieren reflejar en el campo de la computación evolutiva mediante los *operadores* como son el de *selección*, *sustitución, cruce* o *mutación.* La aplicación de estos operadores modifica el estado en el que se encuentra el problema en un instante determinado, asemejándose de este modo a las acciones de un problema de búsqueda común. Cada uno de estos operadores realiza una labor especificada a continuación y que, aplicados reiteradamente en las poblaciones,

permite alcanzar una solución, si no óptima, lo suficientemente buena como para ser considerada la solución final del problema.

- *Selección:* obtiene de la población un número determinado de individuos hasta formar otra población del mismo tamaño que la anterior —extracción con reemplazamiento. Normalmente, esto implica que haya individuos que hayan sido seleccionados más de una vez y otros que no hayan sido escogidos nunca. La forma en la que se escogen los individuos se basa en la aplicación de una fórmula probabilista: si un individuo supera cierto umbral, será seleccionado.
- *Sustitución:* elimina algunos de los individuos de la población para ser sustituidos por algunos nuevos, resultantes de la aplicación de los otros operadores.
- *Cruce:* permite obtener individuos nuevos a partir de otros dos ya existentes. Para ello se selecciona parte de la representación de un padre —es decir, un número determinado de genes— y parte de la representación del otro padre y se juntan para formar al nuevo individuo, que contará con el mismo número de genes que sus padres.
- *Mutación:* modifica uno o varios de los genes del individuo, obteniendo un nuevo individuo.

Se considera que se alcanza una solución al problema de búsqueda cuando existe un individuo dentro de la población actual que supera un umbral especificado y que identifica una solución como suficientemente buena. También pueden establecerse otros criterios de parada, como puede ser el de aplicar los operadores un número de veces concreto. Cada vez que se aplica el conjunto de operadores descrito, se comprueba si este criterio de parada está siendo cumplido en alguno de los individuos existentes y, de ser así, se retorna la solución del problema. En ocasiones, existe más de un individuo que supera la condición para ser considerado una solución buena del problema. En este caso, el algoritmo puede devolver como solución cualquiera de ellas, ya que todas ellas se consideran suficientemente buenas como para satisfacer las necesidades del problema de búsqueda.

Al intentar representar el ejemplo del circuito turístico de este capítulo mediante computación evolutiva, se partiría de una población inicial formada por posibles caminos a seguir por el viajero. El algoritmo se encargaría de aplicar reiteradamente los operadores, con lo que la población irá evolucionando en otras nuevas compuesta por individuos diferentes. Puesto que cada uno de los individuos de las poblaciones es una solución al problema, todos deben tener la forma válida, es decir, cada individuo debe ser una secuencia que contenga todas las ciudades que se desean recorrer y ordenadas de manera en que se tenga en cuenta que no todo par de ciudades está conectado directamente por carretera (Figura 2.11).

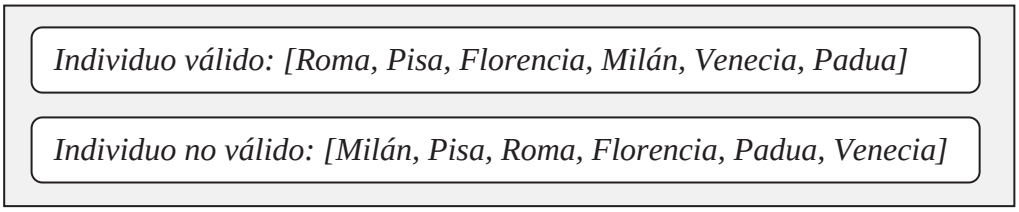


Figura 2.11: La secuencia de arriba representa un individuo válido puesto que cada ciudad está conectada directamente por carretera con la que le sigue en dicha secuencia. En el caso inferior, el individuo no es válido ya que no existe conexión directa entre Milán y Pisa.

El objetivo de la aplicación de un algoritmo evolutivo sería el de encontrar una secuencia de ciudades que permitiera al viajero recorrer todas ellas de la mejor manera posible, ya sea en el menor tiempo o con el menor coste monetario, por ejemplo. Para poder evaluar las soluciones que van surgiendo en las iteraciones del algoritmo se necesita una función de evaluación que indique la bondad de cada uno de los individuos de la población actual. En este caso, se podría poseer información adicional sobre el tiempo óptimo que se debería tardar en realizar el recorrido completo. La solución cuyo coste se acercara más al coste ideal sería mejor solución que cualquier otra.

Función de evaluación = Coste ideal - Coste real del recorrido en el individuo i

De esta manera, si la función de evaluación seleccionada fuera como la anterior, se trataría de minimizar su valor, pudiendo encontrar así la mejor solución al problema. Así, cuando se encuentre una solución cuyo valor en la función de evaluación sea menor que un umbral establecido, el algoritmo devolverá ésta como solución final.

2.4. Algoritmos de búsqueda

Cualquier algoritmo se define, de forma básica, como el conjunto ordenado y finito de operaciones que permite encontrar la solución de un problema *[Real Academia Española, 2018]*. Aplicando esta definición a los problemas tratados en este trabajo, un algoritmo de búsqueda se estructura como una secuencia de pasos, finita, que debe realizarse para alcanzar un objetivo establecido de antemano. Estos algoritmos se caracterizan por los siguientes aspectos *[Pearl, 1984]*:

- *Completitud:* característica que aquellos algoritmos que son capaces de encontrar, al menos, una solución al problema, si es que existe alguna.
- *Admisibilidad:* característica de los algoritmos que retornan como solución la óptima en caso de existir alguna. Este aspecto también es denominado ***optimalidad.***

Quizá éstas sean las características más importantes que debe poseer un algoritmo de búsqueda, sin embargo, existen algunas otras características que deben ser tenidas en cuenta al escoger un algoritmo puesto que influyen directamente en su viabilidad en el problema. Estas otras dos características son *[S. J. Russell & Norvig, 2003]*:

- *Complejidad en el tiempo:* evalúa cuánto tarda un algoritmo en encontrar una solución.
- *Complejidad en el espacio:* analiza cuánta memoria necesitaría el algoritmo para ser ejecutado.

Las dos últimas características pueden provocar que un algoritmo que siempre encuentra la solución óptima a un problema con completa seguridad no sea escogido para resolver un problema porque su coste en cuanto a tiempo o espacio que ocupe en la memoria de un computador sea excesivo. No resultaría viable aplicar un algoritmo de búsqueda a un problema que tarde un tiempo muy grande, aunque se sepa que encontrará la solución óptima. Tampoco se podría aplicar uno que ocupara tanto espacio en memoria que no existiera una máquina capaz de ejecutarlo. Por ello, es tan importante tener en cuenta los cuatro aspectos a la hora de elegir el algoritmo de búsqueda más apropiado para cada problema. Las complejidades en tiempo y en espacio están estrechamente relacionadas con el tamaño del problema, en tanto que es posible que un algoritmo, que resuelve sin dificultad un problema de búsqueda concreto, se vuelva completamente inviable si el mismo problema aumenta su espacio de búsqueda.

En este apartado se tratan algoritmos diseñados para trabajar con la estructura del grafo, que son los más comunes a la hora de resolver un problema de búsqueda. Estos algoritmos se pueden clasificar según el tipo de estrategia que emplean: estrategia de búsqueda no informada y estrategia de búsqueda informada o heurística. En los apartados *Estrategias de búsqueda no informada* y *Estrategias de búsqueda informada* se analiza cada uno de estos tipos.

Para completar el análisis de las distintas técnicas de búsqueda en árboles, se emplea el ejemplo del circuito turístico. Puesto que se trata de un problema en el que lo que se desea es recorrer todas las ciudades, los estados no estarán formados únicamente por la ciudad actual en la que se encuentra el viajero. Cada estado estará identificado por la lista ordenada de ciudades que se ha visitado hasta el momento. Así, sí el viajero se encuentra en la ciudad de Padua, el estado actual del algoritmo será distinto dependiendo de la secuencia de ciudades que haya visitado antes de llegar a esta ciudad. Se ha escogido que los estados del ejemplo estén representados por una secuencia de iniciales de las ciudades que se han visitado para alcanzar ese estado de manera ordenada. De este modo, el estado *M-F-Pd* hace referencia al estado alcanzado cuando el viajero se encuentra actualmente en Padua (Pd) después de haber visitado

previamente Milán (M) y Florencia (F), en este orden. El identificador dado a cada una de las seis ciudades en este ejemplo se representa de la siguiente forma:

Milán = M	*Pisa = Ps*
Venecia = V	*Florencia = F*
Padua = Pd	*Roma = R*

Figura 2.12: Representación de las ciudades del mapa.

Puesto que la ciudad inicial del recorrido es Milán y que el objetivo del problema es visitar todas las ciudades del circuito, el estado inicial y los posibles estados finales quedan de esta manera:

Estado inicial: M

Estados finales: M-V-Pd-Ps-FR, M-V-Pd-Ps-R-F, M-V-Pd-F-R-Ps, M-V-Pd-F-Ps-R, M-F-R-Ps-Pd-V

Figura 2.13: Estados inicial y finales del problema de búsqueda.

En este caso, existen cinco posibles estados finales, pues en todos ellos se recorren las seis ciudades del mapa, aunque en distinto orden. Tienen en común que la primera ciudad de la secuencia es Milán, ya que ésa es la ciudad origen del recorrido tal y como se indica en el estado inicial del problema. Esta condición provoca que muchas de las posibles combinaciones para recorrer las seis —permutaciones de seis elementos— ciudades desaparezcan. El resto de combinaciones existentes en las cuáles sí se comienza en la ciudad de Milán también se descartan al no existir camino directo entre todas y cada una de las ciudades del mapa (no es un grafo completo, ver Figura 2.14).

Como se puede observar en el grafo de la Figura 2.14, cada tramo, o arista del grafo, tiene un coste diferente que representa la distancia en kilómetros existente entre las ciudades de los extremos. En este apartado, a fin de realizar de manera más simplificada las explicaciones, se empleará una versión más sencilla del ejemplo en la que no se tendrá en cuenta el dinero gastado por el viajero ni el tipo de medio de transporte que utiliza para cada desplazamiento.

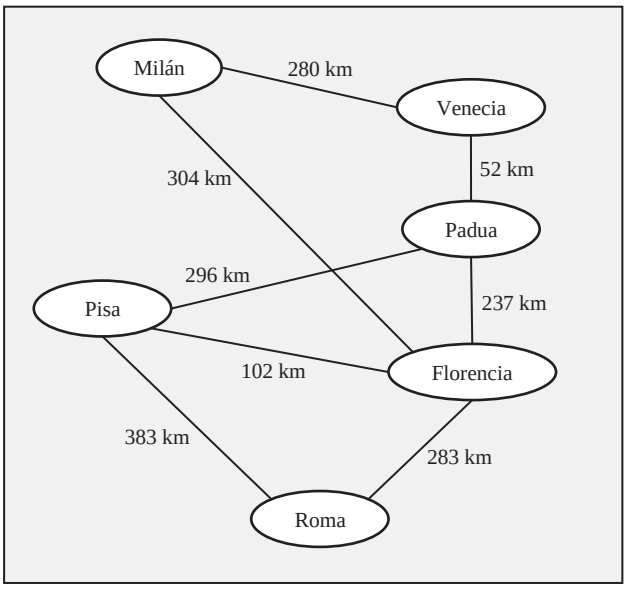


Figura 2.14: Grafo que representa las conexiones entre las ciudades del ejemplo del circuito turístico y sus distancias.

El árbol de búsqueda empleado por los algoritmos posee como nodo raíz el que representa al estado inicial del problema. A partir de él, se obtienen sus descendientes, que representan aquellos estados del problema que pueden alcanzarse desde el estado inicial. Puesto que se trata de un recorrido en el que se van visitando progresivamente las ciudades del mapa, los nodos de cada nivel estarán formados por una secuencia de un número determinado de ciudades. Esto es, el estado del nivel 1, que es la raíz del árbol, estará identificado por una sola ciudad; los estados del nivel 2 estarán identificados por una secuencia de dos ciudades, etc. Hay que recordar que solo se tendrán en cuenta aquellos estados que representen secuencias posibles entre ciudades, ya que al no tratarse de un grafo completo, no todas las secuencias serán válidas. El árbol de búsqueda correspondiente al problema se muestra en la Figura 2.15. En las aristas se indica el coste acumulado del camino que atraviesa cada una de esas aristas y los estados finales del árbol están representados con un trazo más grueso.

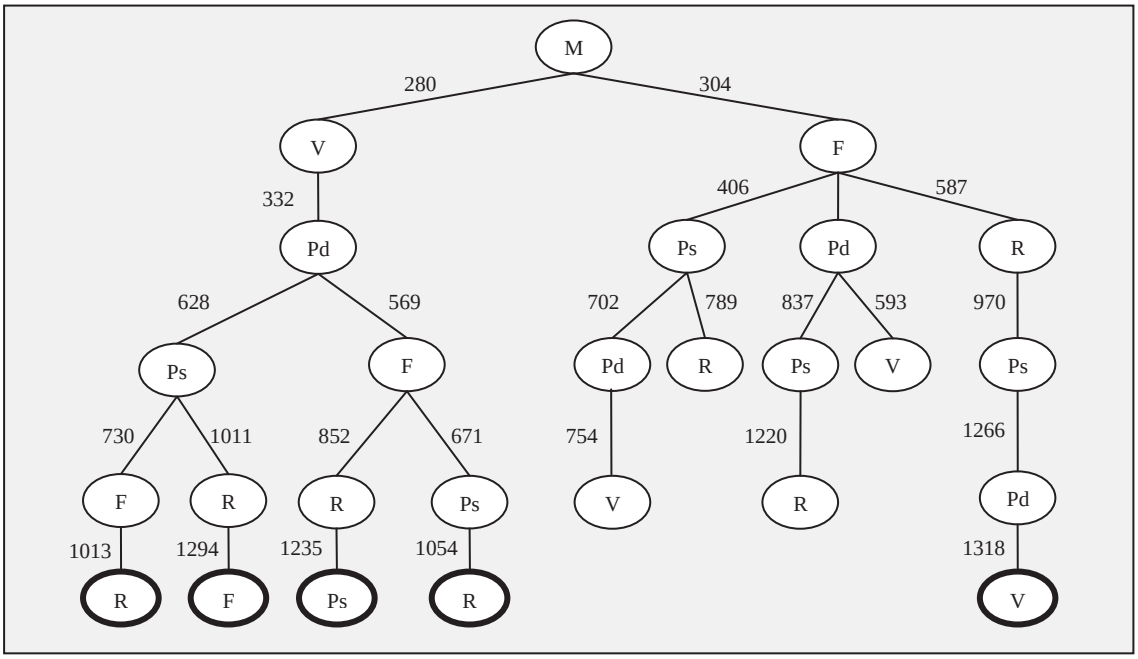


Figura 2.15: Árbol de búsqueda para el problema del circuito turístico.

2.4.1. Estrategias de búsqueda no informada

Los algoritmos de búsqueda que utilizan estrategias no informadas son aquellos que no cuentan con información adicional más allá de la especificada en la definición del problema. También es conocida como búsqueda a ciegas y funciona de manera sistemática, puesto que no se puede aplicar ningún tipo de conocimiento sobre el área a la que se aplica el problema. Al no poseer más información acerca del problema que permita guiar la búsqueda a través del grafo, lo único que puede hacer el algoritmo de búsqueda es desarrollar los nodos y comprobar si, en algún momento, uno de los nodos explorados coincide con el nodo que representa el objetivo del problema. Es decir, su única tarea será la de distinguir *estados objetivo* de *estados no objetivo* a lo largo de la exploración del grafo.

Por lo general, se empleará una estructura en forma de lista denominada *ABIERTA* en la que se van introduciendo los nodos que tienen la posibilidad de ser expandidos en los siguientes pasos del algoritmo. Los elementos de esta lista se van introduciendo de manera ordenada, siguiendo algún criterio propio del algoritmo aplicado, como podría ser el coste que aporta ese nodo a la solución del problema. Los recorridos de búsqueda de los algoritmos están diseñados, comúnmente, para realizarlos sobre una estructura concreta de grafo: el árbol. Por las características que poseen los árboles, la búsqueda del estado objetivo a través de ellos se hace mucho más sencilla que en un grafo cualquiera. De todas maneras, los métodos de búsqueda en árboles

pueden aplicarse a la búsqueda en grafos, ya que estos últimos también pueden verse como árboles (Figura 2.16).

La acción de expandir o desarrollar un nodo del árbol se basa en la obtención de los sucesores de dicho nodo que, en el caso de tratarse de un grafo común se correspondería con la obtención de los nodos adyacentes al actual.

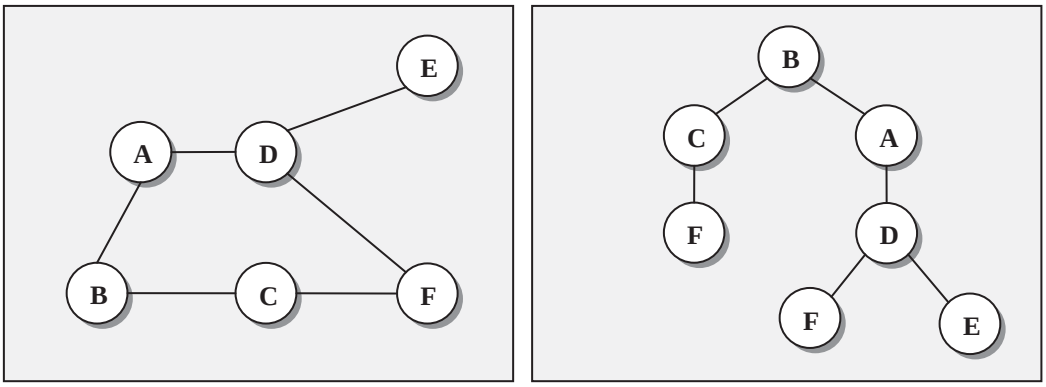


Figura 2.16: Representación del grafo de la imagen de la izquierda como árbol en la imagen de la derecha.

Para conocer cuáles son los caminos recorridos a través del árbol de búsqueda es necesario almacenar de alguna manera cuál es el nodo antecesor de cada uno de los nodos de la estructura. Esta información es útil, por ejemplo, si se necesita deshacer alguna acción, volviendo hacia atrás en el camino hasta un nodo concreto y buscando un nuevo camino a partir de él o, simplemente, para poder devolver la solución al problema como una sucesión de nodos del camino resultante. Por otro lado, también se requerirá guardar de alguna manera los costes que se van obteniendo en los caminos desde el nodo inicial o raíz del árbol hasta cada uno de los nodos explorados.

Un algoritmo aplicado a recorridos en árboles, sea cual sea la forma en la que recorre la estructura para expandir los nodos, comienza en el nodo raíz del árbol, que representa el estado inicial del problema, es decir, en este nodo está representada toda la información que se da como definición del problema de búsqueda. Para apoyar las definiciones de los tipos de búsqueda se va a emplear el ejemplo del viajero por el circuito turístico de Italia en su versión más sencilla, es decir, sin considerar el dinero que le queda al viajero ni qué medio de transporte utiliza para cada recorrido. Simplemente se tendrá en cuenta el recorrido que realiza para visitar todas las ciudades y el kilometraje existente entre cada desplazamiento. Así, los estados inicial y final quedarían de la siguiente manera:

El estado inicial de un problema de búsqueda es el primero que el algoritmo escogido inserta en la lista *ABIERTA.* El algoritmo de búsqueda expande los nodos del árbol basándose en su posición dentro de la lista de *ABIERTA*, de tal modo que expandirá aquel nodo que ocupe la primera posición en la lista. Por lo tanto, la labor más importante del algoritmo será la de escoger la mejor forma de ordenar los nodos

dentro de la lista *ABIERTA* cada vez que los introduce en ella. A partir de la forma en la que los algoritmos insertan los elementos en la lista *ABIERTA*, se obtiene una clasificación de los tipos de búsqueda: en *amplitud* y *en profundidad.*

2.4.1.1. Búsqueda primero en amplitud

En este tipo de búsqueda, tras introducir el nodo raíz en la lista *ABIERTA*, se insertan los sucesores de éste. Tras haber insertado cada uno de esos sucesores en la lista, se escogerá el primero de ellos y, al final de *ABIERTA*, se introducirán todos los sucesores de este último *[E. F. Moore, 1959]*. De manera iterativa se procede del mismo modo con todos los nodos hasta hallar el que represente al estado objetivo del problema. Así, los nodos del árbol se irán introduciendo por niveles, esto es, los nodos de un nivel concreto en el árbol no serán insertados en *ABIERTA* hasta que lo hayan sido todos y cada uno de los nodos del nivel inmediatamente anterior *[Korf, 1990]*. En este caso, la lista *ABIERTA* se puede ver más concretamente como una cola, en la que los últimos elementos insertados se colocarán en las últimas posiciones, es decir, sigue una estructura FIFO *(first-in-first-out).* Por lo tanto, los nodos que se visiten primero al recorrer el árbol serán los primeros en expandirse hacia sus sucesores.

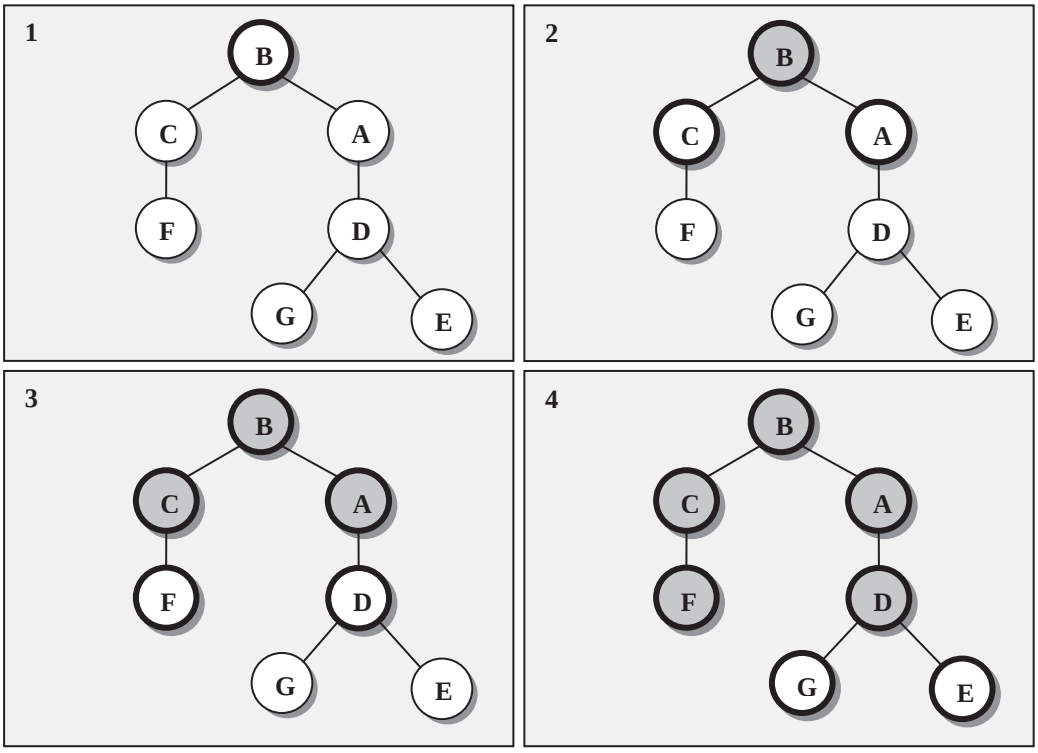


Figura 2.17: Recorrido en amplitud en un árbol.

En el gráfico de la Figura 2.17 se muestra el orden en el que se expanden los nodos aplicando un recorrido en amplitud. El contenido de la lista *ABIERTA* quedaría del siguiente modo al finalizar el recorrido del árbol:

ABIERTA = [B, C, A, F, D, G, E]

En el Algoritmo 2.1 se especifican los pasos a seguir para aplicar esta técnica de búsqueda sin información adicional del problema en árboles.

```
1:    ABIERTA = [nodoInicial]
2:    while noEsVacia(ABIERTA) do
3:        actual = primero(ABIERTA)
4:        if esObjetivo(actual) then
5:            return camino(nodoInicial, actual)
6:        succ = sucesores(actual)
7:        foreach s de succ do
8:            s.padre = actual
9:            s.coste = actual.coste + coste(actual,s)
10:           añadir(s, ABIERTA, final)
11:       end do
12:   end do
13:   return "No encontrado"
```

Algoritmo 2.1: Algoritmo para la búsqueda en amplitud en árboles.

Éste es un método de búsqueda muy simple que, sin embargo, tiene la desventaja de volverse muy costoso a medida que el árbol aumenta de tamaño. El coste, tanto en tiempo como en espacio, depende de algunos factores de forma del árbol como pueden ser:

- *Factor de ramificación:* número medio de sucesores por nodo.
- *Profundidad:* concretamente el número de niveles que tiene en total el árbol o nivel en el que se encuentra el estado objetivo del problema.
- *Coste de las acciones:* coste que conlleva aplicar cada una de las acciones que permita obtener uno de los sucesores a partir de otro nodo del árbol.

El hecho de que el coste de todas las acciones sea el mismo en el caso de la búsqueda en amplitud, por ejemplo, un coste igual a 1, asegura que el algoritmo que emplee este tipo de búsqueda sea admisible, en otras palabras, asegura la optimalidad del algoritmo. Si en el ejemplo del circuito turístico todas las ciudades estuvieran separadas por la misma distancia, equivaldría a un problema en el que el coste de todas

las acciones fuera igual a 1. En ese caso, la búsqueda de una solución al problema empleando este tipo de estrategia daría el resultado de la Figura 2.18.

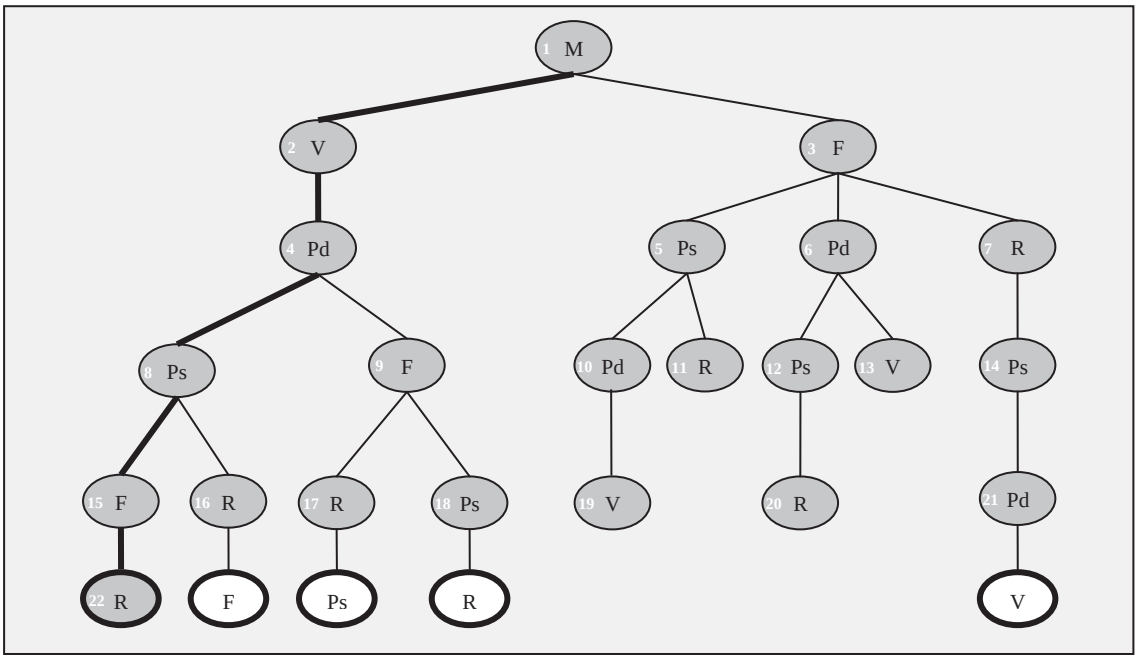


Figura 2.18: Resultado de la aplicación de la búsqueda en amplitud en el ejemplo del circuito turístico si todos los desplazamientos tuvieran el mismo coste.

Los números en blanco que figuran dentro de algunos nodos del árbol hacen referencia al orden en que han sido expandidos cada uno de ellos y, por otro lado, los nodos sombreados son aquellos que han sido introducidos en la lista *ABIERTA*, es decir, que han sido visitados. Puesto que, de todos los nodos objetivo existentes en el problema, el algoritmo encuentra primero el nodo 22R, se devolverá como resultado la ruta asociada a aquel camino que finalice en este estado:

Milán - Venecia - Padua - Pisa - Florencia - Roma

2.4.1.2. Búsqueda primero en profundidad

En la búsqueda en profundidad se desarrollan primeramente aquellos nodos que se encuentran en las ramas que se sitúan más a la izquierda del árbol *[Kumar, 1990]*. Esto implica que los sucesores de un nodo que se está explorando se colocan al principio de la lista *ABIERTA* y no al final, como sucedía en el caso de la búsqueda en amplitud. De este modo, la lista *ABIERTA* funciona como una pila, puesto que sigue la estrategia LIFO *(last-in-first-out)*. El algoritmo continuará explorando una misma rama

hasta encontrarse con un nodo que no tenga ningún sucesor —nodo hoja— momento a partir del cual comenzará a explorar la siguiente rama del árbol que parta del nodo antecesor a la hoja y que aún tenga sucesores sin desarrollar.

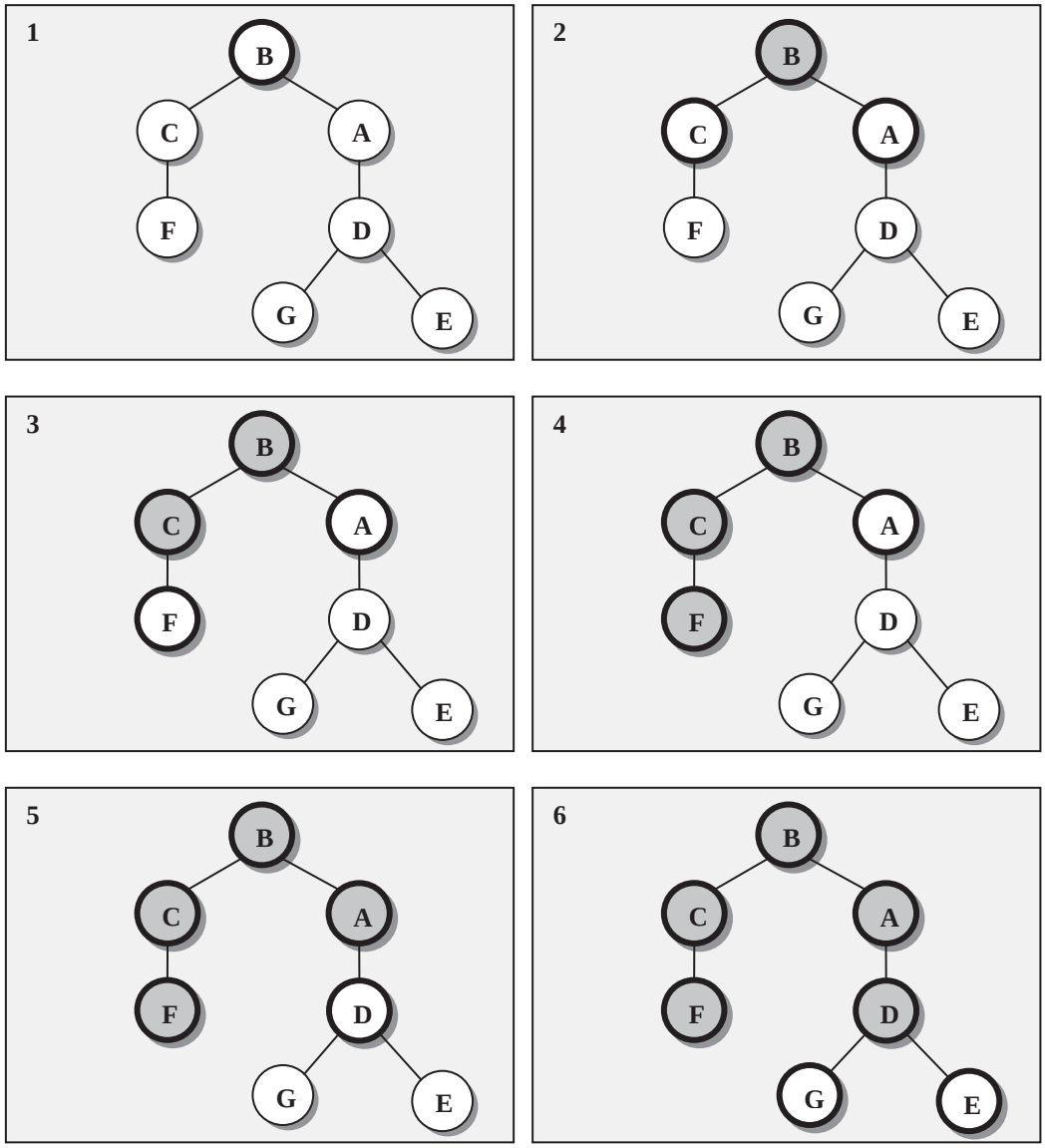


Figura 2.19: Recorrido en profundidad en un árbol.

El resultado de la aplicación de este tipo de búsqueda se muestra en el gráfico de la Figura 2.19. En la lista abierta, cada vez que se desarrolla uno de los nodos, se sustituye el nodo explorado por sus descendientes y, puesto que siempre se explora aquel nodo que se sitúa al principio de abierta, los sucesores se colocarán en las primeras posiciones, ordenados según la posición que ocupen en el árbol, de izquierda a derecha.

Por lo tanto, la evolución de la lista *ABIERTA* a medida que se recorre el árbol es la siguiente:

ABIERTA = [B]
ABIERTA = [C, A]
ABIERTA = [F, A]
ABIERTA = [A]
ABIERTA = [D]
ABIERTA = [G, E]
ABIERTA = [E]
ABIERTA = []

La opción más común al aplicar este método de búsqueda es borrar de la lista *ABIERTA* aquellos nodos que ya no son necesarios para la ejecución del algoritmo, siempre y cuando se vaya guardando la información necesaria de los caminos explorados. En el caso de la evolución de la lista *ABIERTA* que se ha realizado anteriormente, se puede observar que en el momento en que se llega a un nodo hoja éste es borrado de la lista al no poder ser sustituido por sus descendientes. Cuando la lista *ABIERTA* esté vacía significará que ya se ha explorado todo el árbol. Otra información que el algoritmo debe almacenar es la que hace referencia al nodo padre de cada uno de los nodos de un camino, es decir, cada vez que se desarrolle un nodo, éste debe figurar como padre que aquellos por los que va a ser sustituido.

```
1:    ABIERTA = [nodoInicial]
2:    while noEsVacia(ABIERTA) do
3:        actual = primero(ABIERTA)
4:        if esObjetivo(actual) then
5:            return camino(nodoInicial, actual)
6:        if actual.profundidad < limite then
7:            succ = sucesores(actual)
8:        else succ = <vacio>
9:        borrar(actual, ABIERTA)
10:       foreach s de succ do
11:           s.padre = actual
12:           s.coste = actual.coste + coste(actual,s)
13:           s.profundidad = actual.profundidad + 1
14:           añadir(s, ABIERTA, principio)
15:       end do
16:   end do
17:   return "No encontrado"
```

Algoritmo 2.2: Algoritmo para la búsqueda en profundidad en árboles.

La aplicación de este método de búsqueda puede conllevar un grave inconveniente si el árbol posee alguna rama infinita. En ese caso, el algoritmo podría entrar en dicha rama infinita, expandiendo continuamente cada uno de los nodos que pertenecen a ella, sin poder desarrollar el resto del árbol. Para que esto no ocurra, es posible modificar ligeramente el algoritmo para establecer una profundidad límite, a partir de la cual el algoritmo deja de desarrollar la rama en la que se encuentre, pasando a recorrer otra parte del árbol. Sin embargo, esta modificación puede provocar a su vez otra desventaja, ya que es posible que la solución al problema se encontrara en esa misma rama que estaba siendo recorrida por el algoritmo, pero a mayor profundidad del árbol. Por lo tanto, si esto ocurriera, la solución no podría encontrarse nunca. Es por ello que los algoritmos que emplean este método de búsqueda no son ni completos ni admisibles.

Otra forma de realizar este tipo de búsqueda es emplear métodos de *backtracking* en el algoritmo. Para ello se utilizan algoritmos recursivos donde la propia pila de la recursión es la que hace las funciones de lista *ABIERTA.* El algoritmo de *backtracking* se aplica a cada uno de los sucesores del nodo hasta que se encuentra algún estado objetivo alcanzable desde ese nodo. Una de las ventajas de la aplicación de algoritmos de *backtracking* es el ahorro de espacio durante la ejecución. La búsqueda de primero en profundidad ya supone un ahorro de espacio en comparación con la búsqueda en amplitud, pero si además se le añaden métodos de *backtracking* se puede conseguir un ahorro mucho mayor. A pesar de ello, los algoritmos de búsqueda en anchura se siguen empleando para evitar el problema de las ramas infinitas del árbol y la posibilidad de no encontrar la solución del problema cuando se define una profundidad límite de exploración del árbol.

Puesto que en este algoritmo de búsqueda también se trabaja con acciones que poseen el mismo coste, también se realizará una pequeña modificación en la formulación del problema de circuito turístico para emplearlo como ejemplo de aplicación de esta estrategia de búsqueda. Al igual que en el caso anterior, se asume que las distancias que separan las ciudades italianas son las mismas, por lo tanto en este caso, todas las acciones equivalentes a realizar los desplazamientos entre ellas tendrán coste 1. De este modo, el recorrido del árbol se efectúa de la manera mostrada en la Figura 2.20.

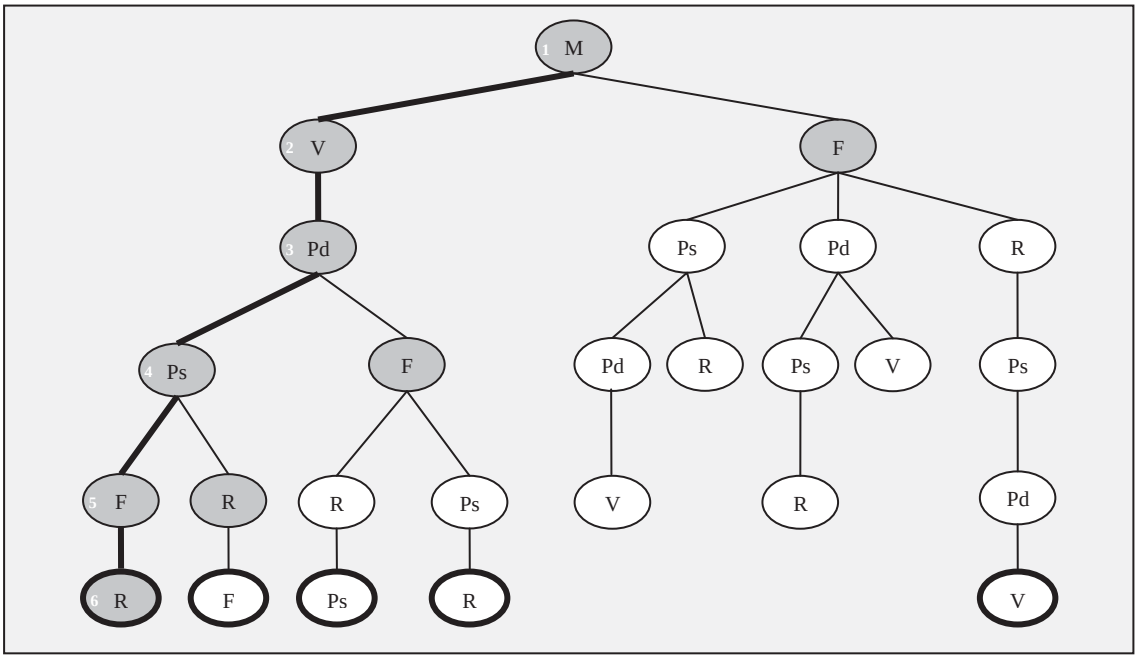


Figura 2.20: Resultado de la aplicación de la búsqueda en profundidad en el ejemplo del circuito turístico con las distancias iguales entre ciudades.

Siguiendo la misma representación que en el caso anterior, los nodos que aparecen sombreados son aquellos que en algún momento han sido insertados en la lista *ABIERTA* por haber sido desarrollado su antecesor. Los nodos que poseen una etiqueta numérica son los que han sido explorados por el algoritmo, indicando el orden en que se ha realizado dicha exploración. El algoritmo empleado encuentra en primer lugar el estado objetivo 6R, pero simplemente por motivos referentes a su posición en el árbol de búsqueda. Por ello, éste será el estado asociado al resultado del problema, de modo que la ruta que recomendará el algoritmo será:

Milán - Venecia - Padua - Pisa - Florencia - Roma

Tanto en el caso de la búsqueda en amplitud como en el de la búsqueda en profundidad se puede observar que los resultados dependen completamente de la situación de los nodos objetivo en el árbol de búsqueda, devolviéndose siempre como resultado aquel nodo objetivo que se sitúe más arriba del árbol, en el caso de la búsqueda en anchura, o más a la izquierda, en el caso de la búsqueda en profundidad.

2.4.1.3. Búsqueda de coste uniforme

La característica en la que no todas las acciones conllevan el mismo coste para su aplicación en el algoritmo de búsqueda es la que tiene en consideración un tipo concreto de búsqueda: la *búsqueda de coste uniforme.* Se trata de una estrategia muy

similar a la búsqueda en amplitud donde el aspecto de la existencia de costes distintos en cada rama está relacionado con la necesidad de que el coste asociado a la búsqueda debe verse incrementado con cada paso del algoritmo, asegurando de este modo que el primer nodo objetivo que se encuentre en el recorrido del árbol se alcanzará de la manera óptima. Podría suponer un problema el hecho de que siempre se pueda ejecutar una acción cuyo coste es nulo, porque esto provocaría que el algoritmo aplicara únicamente dicha acción en todas las iteraciones de manera que entre en un bucle infinito. Por ello, en caso de existir costes distintos en las acciones del problema, se debe asegurar que todas ellas tienen al menos un coste superior a una constante positiva definida de antemano.

El algoritmo de búsqueda en amplitud sufre una pequeña modificación para atender a esta característica. Este cambio afecta exclusivamente a la manera en la que se introducen los nodos visitados en la lista *ABIERTA.* Anteriormente, en la búsqueda en amplitud simple, los nodos visitados son siempre insertados al final de la cola de *ABIERTA.* En este caso, las posiciones iniciales de la cola quedarán reservadas a aquellos nodos que aporten un coste más bajo al camino, es decir, los nodos en *ABIERTA* estarán ordenados por coste. Dicho en otras palabras, el algoritmo explorará aquel nodo de la lista *ABIERTA* que tenga un menor coste y no simplemente el que ocupe la primera posición de la lista.

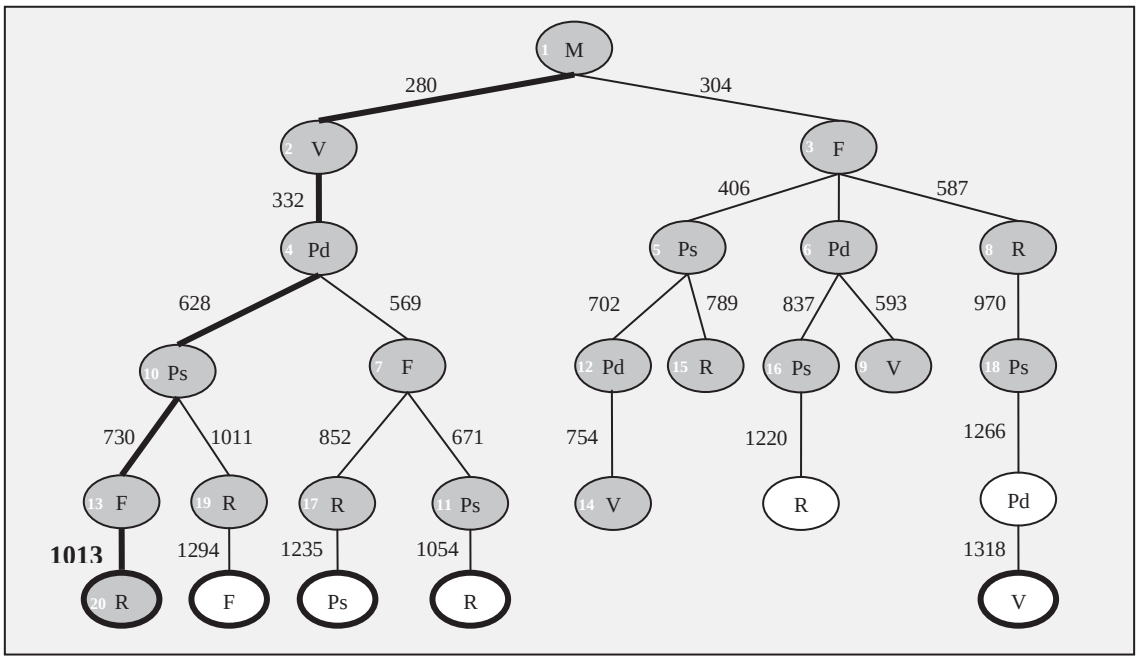


Figura 2.21: Resultado de la aplicación de la búsqueda de coste uniforme en el ejemplo del circuito turístico.

Atendiendo al ejemplo del circuito turístico, cuyo árbol de búsqueda tiene aristas de diferentes pesos, el orden en el que se recorren los nodos se puede observar en el gráfico de la Figura 2.21. Las etiquetas numéricas que figuran al lado de algunos nodos identifican el orden en que han sido explorados, mientras que el sombreado hace referencia a los nodos visitados, es decir, todos aquellos que han estado en la lista *ABIERTA.*

El algoritmo finaliza cuando alcanza uno de los nodos que representan un estado final, en este caso, el estado 20R. El camino recorrido por el algoritmo a través del árbol de búsqueda también se representa con un trazo de mayor grosor (el que se sitúa más a la izquierda del gráfico). Finalmente, del estado final alcanzado se deduce que el trayecto que conlleva menor coste para el viajero es el que se dibuja entre las ciudades de la siguiente manera:

Milán - Venecia - Padua - Pisa - Florencia - Roma

con un coste total de 1013 kilómetros realizados. Como se puede comprobar a la vista del gráfico de la Figura 2.21, no existe un camino hasta un estado final que posea un coste mejor (el resto poseen coste de 1294, 1235, 1054, 1318, respectivamente), por lo tanto el algoritmo ha encontrado la solución óptima cumpliendo de este modo con las condiciones de completitud y admisibilidad. El hecho de que esta solución coincida con las obtenidas por los dos métodos anteriores es casual, puesto que si el árbol hubiese sido construido en otro orden los algoritmos anteriores habrían retornado otra solución a partir de otro de los estados finales.

a) Algoritmo de Dijkstra

Se trata de un tipo de búsqueda de coste uniforme —se realiza sobre grafos cuyos pesos de las aristas son distintos entre sí— en el que a partir de un nodo inicial de un grafo se calculan los caminos de mínimo coste hasta cada uno de los nodos restantes del grafo *[Dijkstra, 1959]*. Es posible emplear este algoritmo en problemas de búsqueda como los analizados en este capítulo, pero posee el inconveniente de tener que trazar los caminos de coste mínimo a todos los demás nodos del grafo y, una vez hecho esto, comprobar cuál es el coste del camino que une el nodo inicial con el nodo meta (o nodos en caso de ser multiobjetivo). Esto implica que el coste de ejecución puede elevarse en gran medida en comparación con otros métodos que van escogiendo la mejor ruta.

2.4.1.4. Aspectos de la búsqueda en grafos

Aunque es posible transformar los grafos en árboles para después aplicar sobre estos últimos los algoritmos de búsqueda analizados, en ocasiones esta transformación no es recomendable, al menos tal y como se ha visto anteriormente. Existen casos en los que la repetición de uno de los estados del espacio de búsqueda dentro del árbol no simplemente indica la posibilidad de alcanzar dicho estado de diversas maneras. En algunos casos, el hecho de poder alcanzar un mismo estado a través de caminos distintos está haciendo referencia a la posibilidad de hacerlo con un menor coste. Esto es, si durante la ejecución de un algoritmo vuelve a aparecer un nodo que representa un estado por el que ya se pasó anteriormente, quizá se haya encontrado una mejor manera, menos costosa, de alcanzar este nodo desde el estado inicial del problema. Esta característica implica modificar la información almacenada acerca del camino a través del cual se alcanza un nodo del árbol para guardar en todo momento aquel camino que permite obtener el menor coste para alcanzar cada uno de los nodos. De este modo, sería más sencillo aplicar los algoritmos de búsqueda directamente a un grafo que a un árbol, empleando un algoritmo más general para la búsqueda en estas estructuras *[Nilsson, 1971]*.

a) Búsqueda bidireccional

Se trata de un tipo de estrategia realizada sobre grafos en el que se hacen al mismo tiempo dos búsquedas: una partiendo del estado inicial hacia el estado final, como en los algoritmos vistos hasta ahora, y otra partiendo desde el estado objetivo hacia el estado inicial *[Pohl, 1990]*. Cuando en las dos búsquedas se encuentre un nodo común, significará que se ha hallado el camino solución y el algoritmo parará en ese momento. El coste de la aplicación de un algoritmo de búsqueda bidireccional es menor que en los otros tipos de búsqueda no informada, tanto en tiempo como en espacio *[Kaindl & Khorsand, 1994]*.

2.4.2. Estrategias de búsqueda informada

En ocasiones, tan solo se cuenta con la definición del problema para realizar la búsqueda de soluciones dentro del espacio de estados. Ante esta situación, únicamente se pueden aplicar estrategias de búsqueda no informada, una búsqueda a ciegas que no permite discernir entre caminos que podrían ser mejores que otros. El hecho de tener que realizar una búsqueda sistemática sin ninguna ayuda adicional acerca del problema provoca, en algunos casos, que el coste de ejecución de un algoritmo sea tan grande que se vuelva inviable su aplicación. Los algoritmos que emplean estrategias de búsqueda

informada, o métodos heurísticos, poseen la capacidad de decidir qué camino parece dar el menor coste a la solución del problema desde el estado actual, gracias a esa información adicional del mismo.

En contraposición a la manera en que se expandían los nodos en las estrategias no informadas, donde se realizaba de forma completamente dependiente a la posición de cada nodo en el árbol, en los métodos heurísticos los nodos se expanden según el valor de una *función de evaluación, f(n),* donde *n* es el nodo del árbol al que se aplica la función. La función de evaluación otorga un valor a cada nodo permitiendo ordenarlos para conocer cuál es el mejor para la solución del problema. Por ejemplo, la función podría hacer referencia a las distancias de los nodos al nodo objetivo. Por lo general, se considera mejor aquel nodo cuyo valor en la función de evaluación es menor, aunque esto siempre depende de qué función se haya considerado en la formulación del problema. El nodo que posea el mejor valor de la función de evaluación en un momento determinado será el escogido por el algoritmo para desarrollarse en el siguiente paso.

La cantidad de conocimiento que se posee acerca del problema permite reducir el número de alternativas a explorar por el algoritmo de búsqueda, ya que muchas de ellas serán desechadas por no dar un valor suficientemente bueno en la función de evaluación *[Palma Méndez & Marín Morales, 2008]*. Sin embargo, el coste de ejecución del algoritmo puede verse peligrosamente incrementado si se posee demasiada información, puesto que en esta situación el algoritmo deberá analizar exhaustivamente cada nodo antes de tomar una decisión acerca de su bondad en la solución del problema. A causa de la necesidad de buscar un término medio en la cantidad de información sobre el problema, es posible que en ocasiones no se encuentre la solución óptima, pero sí una lo suficientemente buena en un tiempo muy razonable.

2.4.2.1. Búsqueda voraz primero el mejor

En la búsqueda voraz, para el cálculo del valor de la función de evaluación, no se tiene en cuenta el coste acumulado del camino hasta el nodo actual ni el coste de las aristas que comunican un nodo actual con sus descendientes, sino que simplemente se limita a continuar hacia el nodo adyacente que se sitúe más cerca del nodo objetivo *[Rosenbloom, 1990]*. De este modo, la función de evaluación de este método queda de la siguiente manera:

$$f(n) = h(n)$$

La función *h(n),* denominada función heurística y propia de los algoritmos heurísticos, representa una estimación del coste menor existente entre el nodo n y el

nodo meta. Al no emplear la información acerca del coste acumulado hasta el momento, puede darse la situación en la que no se alcance el camino de coste mínimo. Los algoritmos que emplean este método de búsqueda se basan en la idea de encontrar la solución óptima del problema escogiendo en cada paso la opción mejor, que en principio solo tiene una repercusión local. Por este motivo, no se pueden considerar algoritmos admisibles. Por otro lado, si el algoritmo que utilice este tipo de búsqueda no se diseña con cuidado, puede darse el problema de ejecutar una rama infinita del árbol, puesto que el algoritmo podría oscilar únicamente entre dos estados del problema si no se detecta que están repetidos. En caso de que esto ocurra, el algoritmo no podría encontrar la solución al problema, siendo por tanto un algoritmo no completo. La ventaja de este método de búsqueda es que al tener en cuenta únicamente la información heurística del problema, la ejecución del algoritmo se vuelve más rápida.

Para visualizar de manera más sencilla e intuitiva los algoritmos que emplean estrategias de búsqueda heurísticas, se define un nuevo problema similar al anterior. Al igual que en el ejemplo anterior, la temática del problema gira alrededor de un viajero que se mueve a través de unas ciudades italianas, pero en este caso no desea realizar un circuito turístico para conocer todas las ciudades del recorrido, sino que desea trasladarse entre dos de esas ciudades de la mejor manera posible.

En la Figura 2.22 se muestra el grafo que representa la situación del problema. Las etiquetas de las aristas del grafo representan la distancia en kilómetros entre los pares de ciudades de los extremos. El viajero se encuentra en Turín en el momento inicial de la búsqueda y desea trasladarse a Nápoles con el menor coste kilométrico que sea posible.

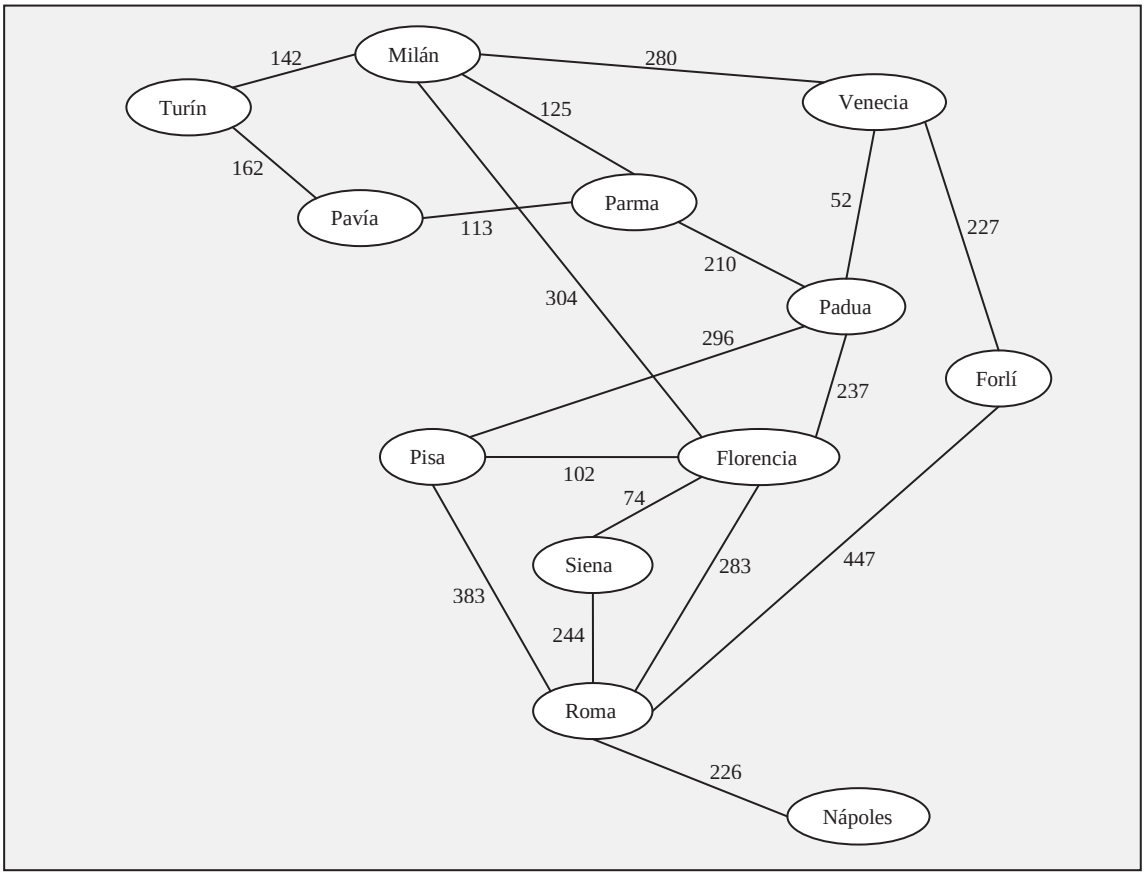


Figura 2.22: Representación del mapa de ciudades italianas como un grafo.

Para resolver este problema mediante el empleo de métodos heurísticos, se requiere la información que estime las distancias entre cada una de las ciudades del mapa y la ciudad objetivo, Nápoles (Figura 2.23).

Turín	*893*	*Pisa*	*571*
Milán	*774*	*Forlí*	*557*
Pavía	*764*	*Florencia*	*473*
Venecia	*734*	*Siena*	*437*
Parma	*658*	*Roma*	*226*
Padua	*695*	*Nápoles*	*0*

Figura 2.23: Distancias entre cada ciudad del mapa hasta la ciudad de destino, Nápoles.

El estado inicial del algoritmo se representa con la ciudad de Turín cuyo valor para la función de evaluación es igual a 893, puesto que éste es el valor de la función heurística. Para dar el siguiente paso, el algoritmo debe hallar cuáles son los nodos descendientes del nodo inicial, que en este caso son Milán y Pavía. Una vez se conoce cuáles son las posibles alternativas para dar el siguiente paso, se debe comprobar qué

camino parece el más prometedor para obtener un camino de coste mínimo entre las ciudades de origen y de destino. Para ello, se comparan los valores heurísticos de ambas ciudades:

$$h(Milán) = 774$$
$$h(Pavía) = 764$$

Puesto que la ciudad de Pavía es la que obtiene un valor menor en la función de evaluación, el algoritmo seleccionará este nodo para avanzar por el camino hasta el destino. En este momento, Pavía se convierte en el nodo actual del algoritmo y se repetirán los mismos pasos para conocer cuál de sus sucesores es el nodo más prometedor. Para este caso, en el que solo existe un descendiente, Parma, no es necesario comparar valores de la función de evaluación, y el algoritmo avanza hasta dicha ciudad. Todos estos pasos se repiten hasta que se alcance el nodo objetivo del problema, momento en el cual el algoritmo dará por finalizada su ejecución. El resultado de la ejecución del método de búsqueda de primero el mejor se muestra en la Figura 2.24.

Los nodos que en la Figura 2.24 aparecen sombreados y con un trazo grueso son aquellos que pertenecen al camino solución del problema, mientras que los que aparecen con un contorno discontinuo son aquellos que, a pesar de no ser parte de la solución, fueron visitados en algún momento por el algoritmo de búsqueda. El coste total del recorrido en esta solución se obtiene a partir de la suma de los costes de las aristas que pertenecen al camino, que en este caso es de 1231 kilómetros.

2.4.2.2. Algoritmo A*

Este algoritmo se basa en un tipo de búsqueda de primero el mejor *[P. E. Hart et al., 1968; Nilsson, 1982]*, con la particularidad de que en este caso la función de evaluación tiene la siguiente forma:

$$f(n) = g(n) + h(n)$$

La función de evaluación tiene dos componentes básicos: la función de coste acumulado, *g(n)*, y la función heurística, *h(n)*. La primera es equivalente a la empleada en los métodos de búsqueda no informada, y en ella se devuelve el coste acumulado del camino de coste mínimo existente, hasta el momento actual de la ejecución, entre el nodo inicial y el nodo n.

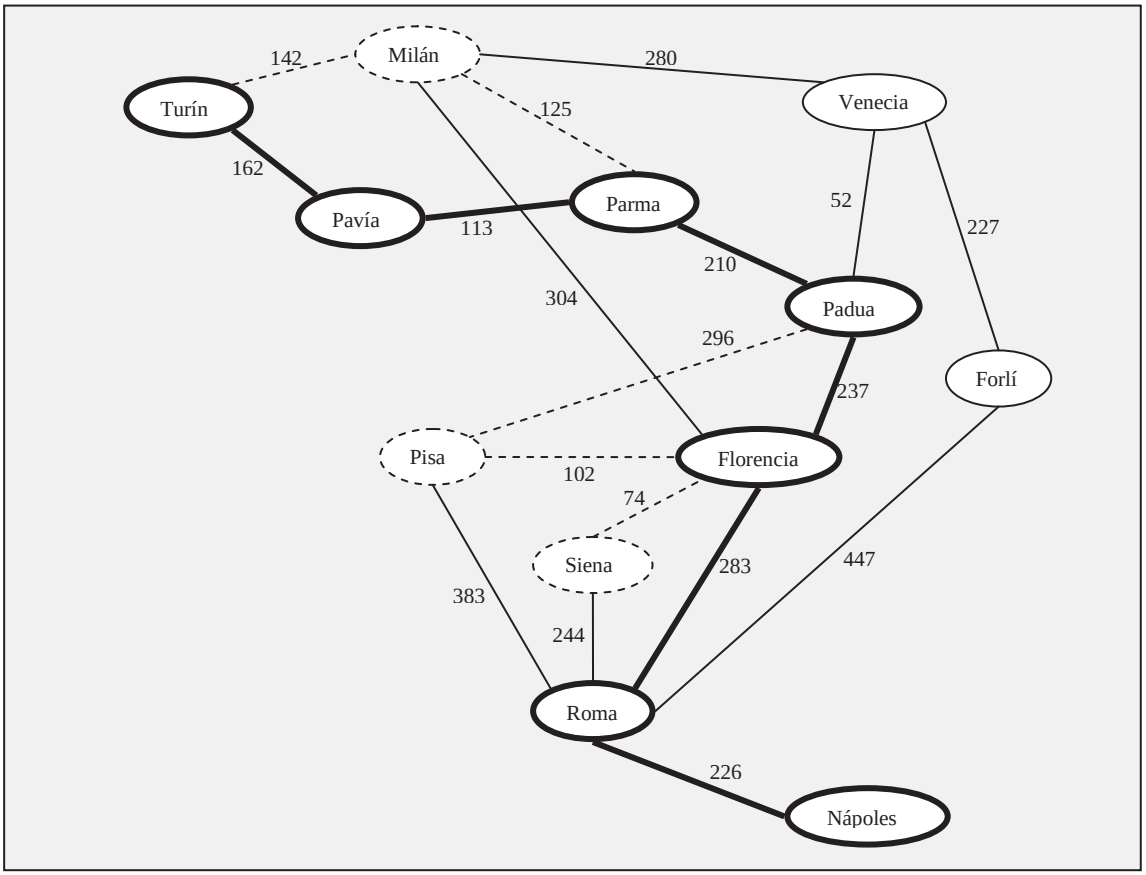


Figura 2.24: Solución del problema empleando la búsqueda voraz de primero el mejor.

La segunda es la misma función empleada en el caso anterior de búsqueda de primero el mejor, esto es, la estimación del coste del mejor camino existente entre el nodo n y el nodo meta. De la definición de estos dos componentes de la función de evaluación se puede extraer que el valor de la función *g(n)* cuando n es el nodo inicial es 0 y que el valor de la función heurística, *h(n),* cuando n es el nodo objetivo es 0. Ambos valores son aproximaciones de los costes *g*(n)* y *h*(n),* respectivamente, que se definen como:

- *g*(n):* coste del camino de menor coste existente entre el nodo inicial y el nodo n. El algoritmo debe encontrar durante su ejecución cuál es la mejor manera de alcanzar el nodo n desde el inicial, puesto que cualquier otro camino conllevará mayor coste. Por lo tanto, la función de coste acumulado que emplea la función de evaluación cumple la condición $g(n) \geq g^*(n)$.
- *h*(n):* coste del mejor camino entre el nodo n y el nodo objetivo, o el nodo objetivo más cercano a n en caso de ser un problema multiobjetivo. El algoritmo A* es un algoritmo optimista, puesto que estima este valor de forma positiva (nunca sobrestimará este coste). De este modo, se cumple que $h(n) \leq h^*(n)$ si la

heurística empleada es admisible *[P. E. Hart et al., 1968; Pearl, 1984]*. Por este motivo, en un problema de búsqueda donde el coste se mide en distancia, comúnmente se toma como valor de la función heurística la distancia aérea, que es la menor que puede existir entre dos puntos del grafo.

A diferencia de los algoritmos voraces de primero el mejor, que pueden no terminar en grafos con ramas infinitas, el algoritmo A* es capaz de reconocer cuándo está ejecutando a través de este tipo de ramas y modificar su recorrido para continuar por otro camino —siempre en el caso de que el grafo sea localmente finito (para todo nodo perteneciente al grafo existe un número finito de sucesores). Para ello, se basa en el valor que va adquiriendo la función *g(n),* ya que al recorrer la rama infinita incrementará su valor progresivamente hasta que sea suficientemente alto como para que el algoritmo A* prefiera continuar por otra rama del árbol de búsqueda con mejor valor de la función de evaluación *[Palma Méndez & Marín Morales, 2008]*. Este aspecto, junto con la característica de la función heurística, hacen del A* un algoritmo completo y admisible.

a) Especificación del algoritmo A*

El algoritmo A* introduce primeramente el nodo inicial del árbol de búsqueda en la lista *ABIERTA* y a continuación se obtienen los sucesores de este nodo. La forma de introducir los descendientes del nodo en la lista *ABIERTA* se basa en sus respectivos valores de la función de evaluación: se colocarán hacia las primeras posiciones de la lista aquellos descendientes cuyo valor de la función sea menor. Una vez ordenados todos los sucesores del nodo en la lista *ABIERTA,* se extrae el primero de ellos (el de menor valor en la función *f(n)*) *y* se procede del mismo modo con sus sucesores. Cuando el nodo objetivo sea seleccionado para expandirlo, el algoritmo finalizará devolviendo el camino solución, para lo cual será necesario haber almacenado la información acerca del nodo padre de cada nodo del grafo. Es posible que, a medida que se expanden los nodos del grafo, se halle un nuevo camino para un nodo, n, que ya se encontraba en la lista *ABIERTA.* En ese caso, se deben comparar los caminos que llevan al nodo en cuestión y averiguar cuál de ellos permite alcanzar el nodo *n* desde el inicial con un menor coste. Si el nuevo camino hallado posee un menor coste, se modifica la posición del nodo *n* en *ABIERTA* de acuerdo al nuevo valor de la función de evaluación. Al aplicar el algoritmo A* al problema planteado se puede observar con mayor claridad lo descrito anteriormente.

```
1:    ABIERTA = [nodoInicial]
2:    while noEsVacia(ABIERTA) do
3:        actual = primero(ABIERTA)
4:        if esObjetivo(actual) then
5:            return camino(nodoInicial, actual)
6:        succ = sucesores(actual)
7:        foreach s de succ do
8:            if esta(s, ABIERTA) then
9:                comprobarCamino(s, actual)
10:           else
11:               s.padre = actual
12:               s.g = actual.g + coste(actual,s)
13:               s.h = <obtener valor heurístico de s>
14:               introducir Ordenado(s, ABIERTA)
15:       end do
16:   end do
17:   return "No encontrado"
```

Algoritmo 2.3: Descripción del algoritmo de búsqueda A*.

El proceso de ejecución del algoritmo se muestra de forma progresiva en la Figura 2.25. La primera ciudad en añadirse a la lista *ABIERTA* es la ciudad de inicio del viaje, Turín. El algoritmo A* se aplica al grafo que representa el problema teniendo en cuenta los valores de coste real entre ciudades descritos en la propia figura del grafo (Figura 2.22) y los costes de la función heurística desde cada ciudad del mapa hasta el destino, reflejados en la tabla de la Figura 2.23.

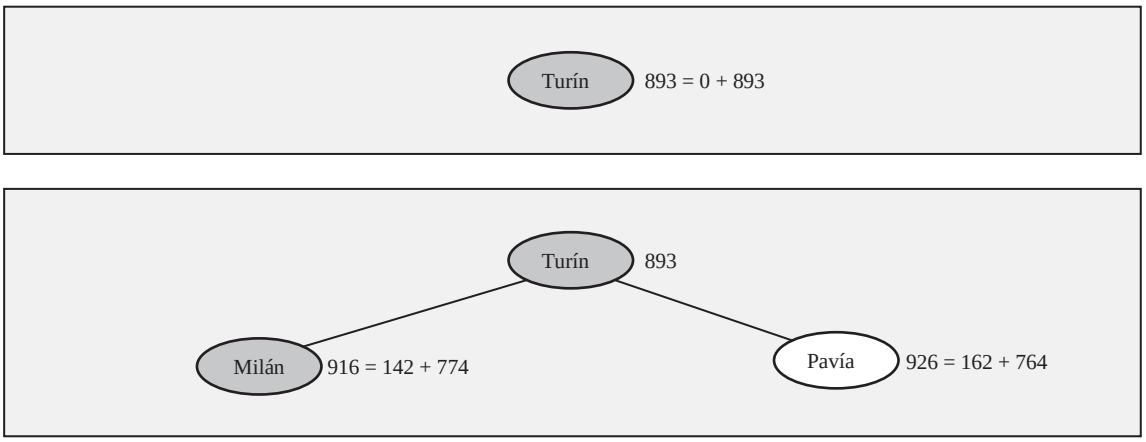
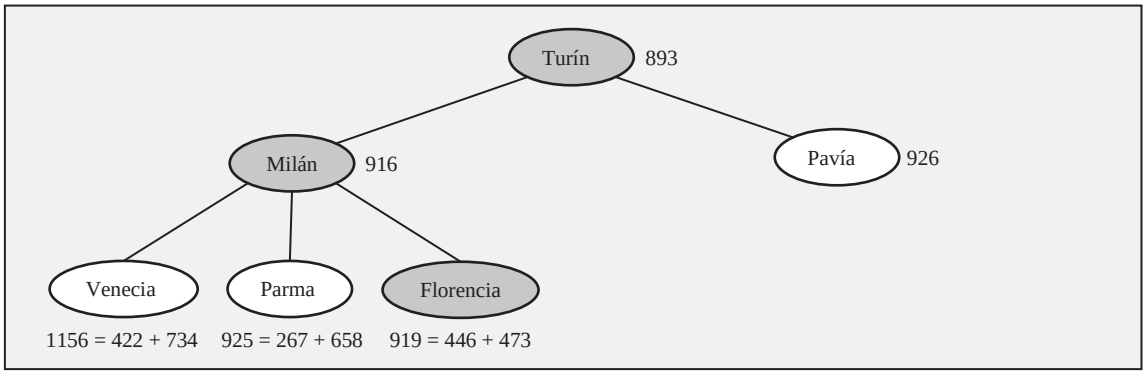
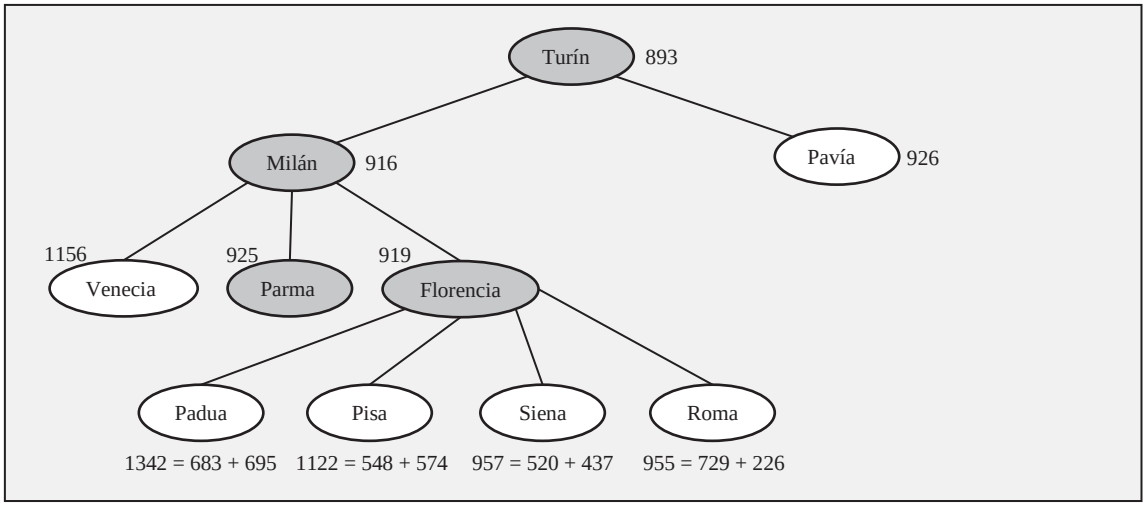


Figura 2.25a: Aplicación del algoritmo A* al problema del viaje de coste mínimo entre Turín y Nápoles.





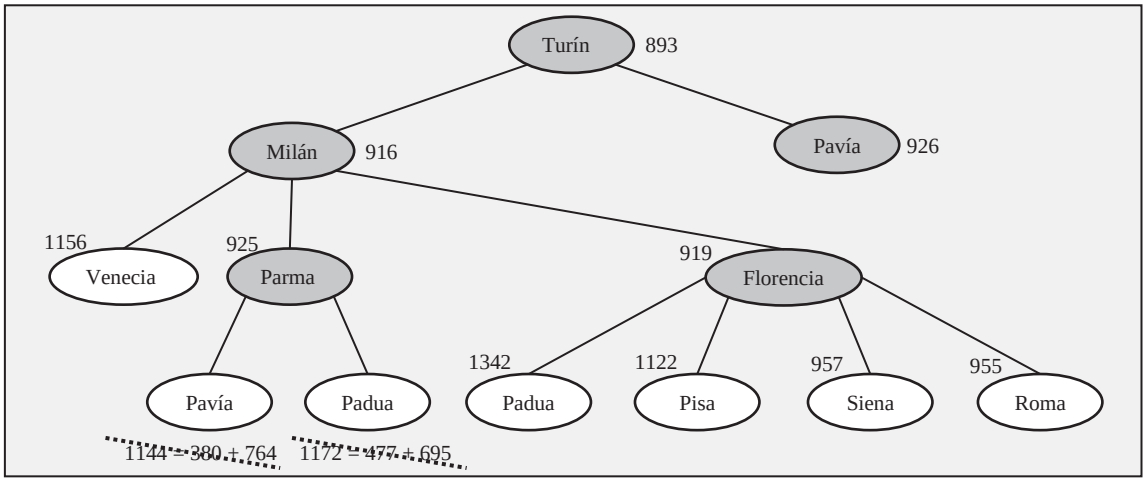


Figura 2.25b: Aplicación del algoritmo A* al problema del viaje de coste mínimo entre Turín y Nápoles.

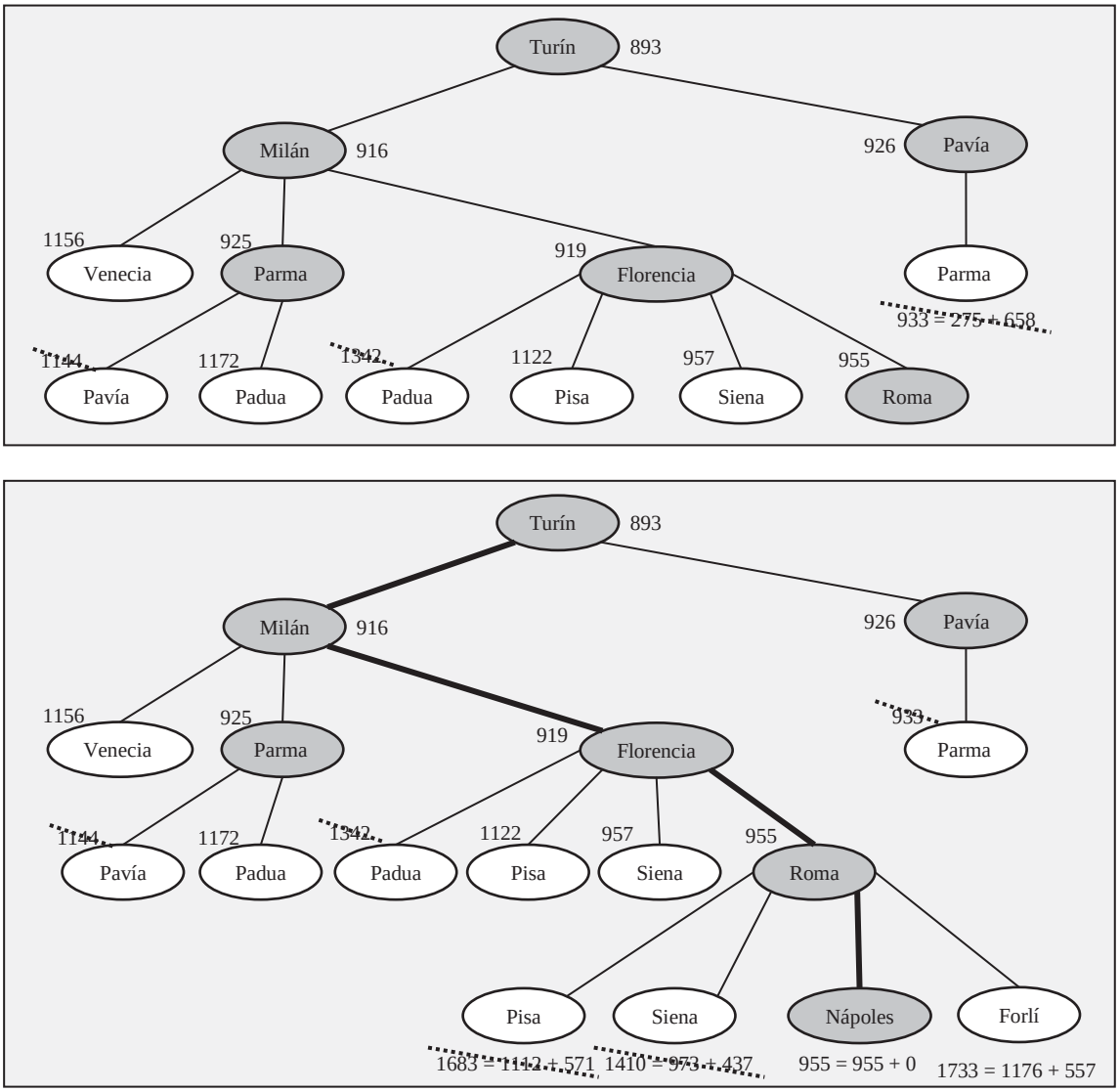


Figura 2.25c: Aplicación del algoritmo A* al problema del viaje de coste mínimo entre Turín y Nápoles.

En la Figura 2.25, donde se muestra la ejecución del algoritmo A*, empleado para resolver el problema de búsqueda del viajero que desea alcanzar la ciudad de Nápoles desde Turín, aparecen como nodos sombreados aquellos que en algún momento de la ejecución han sido expandidos por el algoritmo. Junto a cada uno de los nodos aparece una etiqueta numérica, que representa el valor de la función de evaluación, *f(n)*, cuando se alcanzó dicho nodo. También puede observarse que algunas de estas etiquetas se muestran tachadas. En este caso se están representando aquellos valores de la función de evaluación para un nodo *n* que superan el valor que poseía anteriormente dicho nodo. Es decir, una etiqueta numérica aparece tachada cuando en el árbol de alcanza un nodo que ya estaba en la lista *ABIERTA* y no mejora el coste del

camino existente hasta el momento. Finalmente, con trazo grueso, se muestra el camino de coste mínimo dado como solución por el algoritmo de búsqueda:

Turín - Milán - Florencia - Roma - Nápoles

con un coste total de 955 kilómetros recorridos. Al comparar los resultados obtenidos con este algoritmo y el algoritmo voraz de primero el mejor se puede apreciar que el camino devuelto como solución varía. En el caso del algoritmo voraz, el camino resultante era *Turín - Pavía - Parma - Padua - Florencia - Roma - Nápoles*, cuyo coste es de 1231 kilómetros de recorrido. Se trata de una diferencia sustancial en cuanto al coste del camino, puesto que al resolver el problema con ese tipo de algoritmo voraz se traza un trayecto de 276 kilómetros más que el trazado por el algoritmo A*. De este modo, se observa claramente la importancia de emplear una buena función de evaluación que tenga en cuenta las características del problema para resolverlo de la mejor manera posible, obteniendo los resultados óptimos[1].

2.4.2.3. Búsqueda con memoria acotada

El algoritmo A* requiere gran cantidad de memoria, sobre todo a medida que el espacio de estados crece y, por consiguiente, también aumenta la profundidad del árbol de búsqueda. Entre algunos de los algoritmos que permiten limitar la cantidad de memoria necesaria para la ejecución está el algoritmo IDA* (Iterative Deepening A*) *[Korf, 1985]* y el algoritmo SMA* (Simplified Memory-Bounded A*) *[S. Russell, 1992]*.

a) Algoritmo IDA*

El algoritmo IDA* combina la eficiencia del algoritmo de profundidad iterativa[2] y la optimalidad del algoritmo A*. En este caso, el algoritmo IDA* establece como límite de cada paso del algoritmo el valor de la función de evaluación, *f(n)*, que emplea el algoritmo A*, en lugar de la profundidad límite dada para el árbol. De este modo, inicialmente la profundidad límite del árbol hasta la que se realizará la búsqueda se establece como el valor de *f(nodo inicial)* que, dado que el heurístico es admisible —la estimación del coste hasta el destino posee un valor inferior al real— será inferior o

[1] El algoritmo A* cumple la propiedad de optimalidad.

[2] *Búsqueda iterativa en profundidad:* se trata de realizar una búsqueda en profundidad aumentando en cada paso del algoritmo la profundidad límite hasta que se alcance el nodo objetivo. Posee la desventaja de tener que recorrer nodos ya visitados para cada valor de la profundidad límite.

igual al valor de la solución óptima. Establecido dicho umbral se comienza una búsqueda en profundidad hasta que el valor de la función de evaluación de todos los nodos del árbol es superior al umbral especificado en cada paso del algoritmo. Así, cuando el algoritmo visita un nodo del árbol cuyo valor de la función de evaluación supera el límite, deja de explorar esa rama y continúa la búsqueda en profundidad por otra de las ramas del árbol, quedando de este modo como nodos hoja aquellos que han superado el umbral. Si al visitar nodos se alcanza el nodo objetivo con un coste que no supere el umbral establecido, se dará por finalizada la búsqueda. En el caso de que aún no se haya encontrado ningún estado final del problema, se comienza una nueva búsqueda en profundidad estableciendo como nuevo umbral el mínimo de aquellos valores de la función de evaluación que superaran el anterior umbral (Algoritmo 2.4 y Algoritmo 2.5).

```
1:    umbral = f(nodoInicial)
2:    camino = <obtener solución mediante búsqueda iterativa en profundidad>
3:    while (camino == "No encontrado" do
4:          umbral = nuevoUmbral
5:          camino = <obtener solución mediante búsqueda iterativa en
                                                              profundidad>
6:    end do
7:    return camino
```

Algoritmo 2.4: Descripción del algoritmo de búsqueda IDA* *(Iterative Deepening A*).*

Se ha discutido sobre los beneficios de la ordenación de los sucesores del nodo desarrollado con el fin de hacer más eficiente la búsqueda de la solución al problema *[Reinefeld & Marsland, 1993]*. Es posible aprovechar que el algoritmo empleado dispone de información acerca del problema (posee una función heurística) para ordenar los sucesores en la lista *ABIERTA* y, de este modo, desarrollar posteriormente aquellos nodos más prometedores en primer lugar. Por lo tanto, si los sucesores de un nodo expandido se ordenan en la lista *ABIERTA* se podrá encontrar la solución de manera más rápida en una iteración del algoritmo. En caso de que en esa iteración no se encontrara la solución, se visitarían el mismo número de nodos que en el caso de no ordenar los nodos de *ABIERTA.*

Lógicamente, puesto que emplea la misma heurística, el resultado de la aplicación de este algoritmo de búsqueda al ejemplo del viajero que viaja desde Turín a Milán es el mismo que el obtenido tras la ejecución del algoritmo A*.

```
1:      ABIERTA = [nodoInicial]
2:      while noEsVacia(ABIERTA) do
3:              actual = primero(ABIERTA)
4:              if esObjetivo(actual) then
5:                      return camino(nodoInicial, actual)
6:              succ = <vacio>
7:              if (f(actual) < umbral) then
8:                      succ = sucesores(actual)
9:              else nuevoUmbral = actualizarNuevoUmbral(f(actual))
10:             foreach s de succ do
11:                     s.padre = actual
12:                     s.g = actual.g + coste(actual, s)
13:                     s.h = <obtener valor heurístico de s>
14:                     introducirOrdenado(s, ABIERTA)
15:             end do
16:     end do
17:     return "No encontrado"
```

Algoritmo 2.5: Descripción del algoritmo de búsqueda iterativo en profundidad empleando la función de evaluación para establecer el límite.

Este algoritmo posee la ventaja de admisibilidad del A* y la reducción del espacio de memoria empleado propia de los algoritmos de primero en profundidad, aunque en cada nueva búsqueda en profundidad sea necesario volver a recorrer aquellos nodos que ya fueron visitados en la búsqueda anterior.

b) Algoritmo SMA*

Para intentar resolver la desventaja del algoritmo IDA*, en el que nodos ya visitados se tienen que volver a visitar en cada iteración, se propone el algoritmo SMA*. En él existe un espacio limitado en memoria y el problema debe ser resuelto empleando únicamente dicho espacio. Es un algoritmo que tiende a evitar la repetición de estados, puesto que esto puede conllevar el empleo de más cantidad de memoria. Se trata de un algoritmo completo y admisible en tanto en cuanto la memoria de la que se disponga permita almacenar todo el camino que represente la solución menos profunda del árbol de búsqueda. En caso de no haber suficiente memoria para almacenar la solución óptima, se devolverá la mejor posible.

2.4.2.4. Algoritmo BIDA*

Se trata de un algoritmo de búsqueda bidireccional propuesto por Manzini *[Manzini, 1995]* que, a diferencia de otros tipos de búsqueda bidireccional, donde ambas búsquedas se realizan al mismo tiempo, la búsqueda hacia delante y la búsqueda hacia detrás se ejecutan secuencialmente. Esto es, inicialmente se establece desde el nodo final un conjunto de nodos denominado *perímetro* y, posteriormente, se inicia la búsqueda desde el nodo inicial hasta encontrar alguno de los nodos pertenecientes al perímetro.

El perímetro que se constituye alrededor del nodo objetivo se construye de la siguiente forma:

- Se estipula una distancia, *d*. Todos aquellos nodos que se encuentren a una distancia *d* o inferior del nodo objetivo formarán parte del conjunto A_d
- A su vez, dentro de este conjunto se crea el conjunto P_d —perímetro—, que contendrá a aquellos nodos que tengan descendientes fuera del conjunto A_d *(*$P_d \subseteq A_d$*).* De otro modo, los nodos pertenecientes al perímetro del nodo objetivo son todos los nodos del conjunto A_d que puedan alcanzarse de algún modo desde nodos no contenidos en el conjunto A_d. Cuanto mayor sea la distancia *d* establecida, mayor será el número de nodos pertenecientes al perímetro.

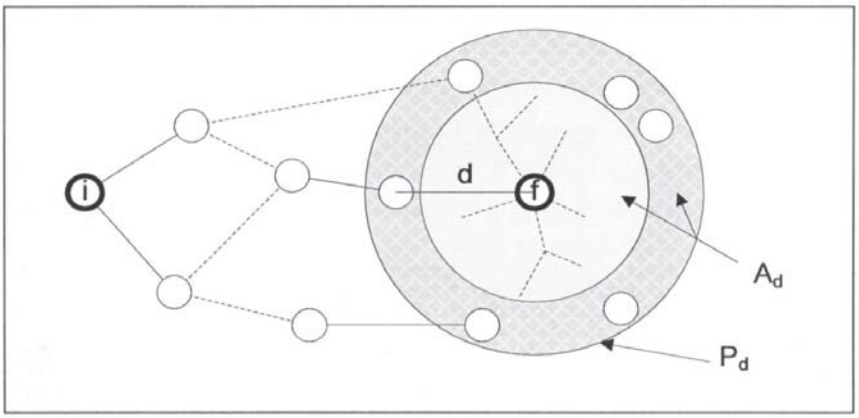


Figura 2.26: Representación de los elementos del algoritmo BIDA* en un grafo con nodo inicial, *i*, y nodo meta, *f*.

Una vez se conoce qué nodos pertenecen al perímetro del nodo objetivo, se debe almacenar la información acerca del valor heurístico que estima el coste entre cada uno de esos nodos y la meta, además del mejor camino para llegar al nodo final. Esta información permitirá calcular el valor de la función heurística para el resto de nodos del grafo durante la ejecución del algoritmo de búsqueda (en este caso, la estrategia de búsqueda que emplea el BIDA* se basa en el algoritmo IDA*). Por lo tanto, para un

nodo *n* que no forme parte del conjunto A_d, la función heurística se obtiene a partir de otros dos valores:

- *h(n,m):* valor heurístico que estima el coste entre el nodo *n* y un nodo, *m*, perteneciente al perímetro.
- *h*(m,f):* valor de la función heurística que representa el coste del mejor camino entre el nodo *m* perteneciente al perímetro y el nodo objetivo.

Para obtener el valor total de la función heurística en el nodo *n* es necesario sumar los dos valores anteriores, de tal modo que el nodo *m* escogido de entre todos los que pertenecen al perímetro sea aquel que permita obtener el mínimo valor de esta suma:

$$h(n) = \min_{m \in Pd} \{h(n,m) + h^*(m,f)\}$$

La forma de obtener el valor de la función heurística para cada nodo *n* posee el inconveniente de tener que realizar el cálculo para cada nodo, m, perteneciente al perímetro de nodo objetivo. Por lo tanto, será un cálculo más costoso que en los algoritmos analizados hasta ahora. De este modo, la función de evaluación con la que trabaja este algoritmo de búsqueda es:

$$f(n) = g(n) + h(n)$$

Sin embargo, el hecho de emplear el algoritmo IDA* como base de este tipo de búsqueda limita el número de nodos del perímetro que se comprueban para realizar el cálculo a la profundidad especificada para cada iteración del algoritmo, lo que supone una reducción del coste.

La solución al problema de búsqueda estará formada por el camino entre el nodo inicial hasta uno de los nodos del perímetro y el camino entre este mismo nodo del perímetro y el nodo meta. Este último es el que se hallaba almacenado como información relativa al propio nodo perimetral, por lo tanto, no será necesario realizar más cálculos una vez se alcanza el nodo del perímetro.

2.4.2.5. Algoritmos basados en árboles alternados

Los *árboles alternados* representan, por lo general, situaciones de un juego en el que participan dos contrincantes y donde además no interviene el azar, sino que son los propios jugadores los que permiten evolucionar al juego con sus decisiones a partir de una información completa del momento actual de juego. En el árbol se representan

como nodos cada una de las posibles situaciones del juego, y tendrá como descendientes aquellas otras situaciones del juego que puedan alcanzarse mediante movimientos legales en el juego. Puesto que en los juegos de este tipo los turnos de los jugadores se alternan, cada nivel en el árbol de búsqueda representará a dichos jugadores del mismo modo, siendo los niveles impares para un jugador y los niveles pares para su oponente.
Existen diversos métodos especializados en este tipo de problemas: Minimax *[Neumann, 1928]*, Alfa-Beta *[Knuth et al., 1974; Richards & T. P. Hart, 1961]* y SSS* *[Stockman, 1979]*.

a) Método Minimax

El objetivo de este tipo de métodos como el Minimax es escoger las mejores jugadas que puede efectuar el jugador principal del problema, que se enfrenta a un oponente.

Para ello, se define una función de evaluación para estos jugadores que tomará diversos valores según el movimiento que realicen en cada momento —mayor a medida que la jugada es mejor. El jugador principal siempre tenderá a escoger la jugada que le permita obtener un mayor valor de la función de evaluación (nivel *max* del árbol), mientras que el contrincante desea realizar jugadas que perjudiquen lo máximo posible al jugador principal, por lo que tenderá a efectuar aquellas jugadas que minimicen la función de evaluación (nivel *min* del árbol).

La ejecución del algoritmo parte de los nodos hoja del árbol de juego, que representan las situaciones finales de la partida, tras las cuales ya no es posible realizar ningún movimiento adicional. El valor de la función de evaluación tomado en los nodos terminales del árbol se va propagando hacia la raíz de tal modo que a los niveles donde se representa al jugador principal se trasmite el mayor valor de los nodos sucesores y a los niveles referidos al oponente se propaga el menor valor de todos los descendientes. Finalmente, se llega al nodo raíz del árbol de juego con un único valor de la función de evaluación que representa la puntuación máxima que puede esperar el jugador principal del juego, en caso de que realice las jugadas más convenientes durante toda la partida. Cualquier sucesión de jugadas de la partida se puede representar con el camino conexo existente entre el nodo raíz de árbol y un nodo terminal del mismo.

Es posible representar las posibles situaciones de una partida de juegos como el ajedrez, las tres en raya, el *backgammon*, etc. Sin embargo, el gran número de posiciones en las que se puede encontrar una partida de estos juegos provoca que en

ocasiones sea inviable la aplicación de este algoritmo para conocer cuál es la mejor secuencia de jugadas que puede realizar un jugador[3].

En la Figura 2.27 se muestra un ejemplo de aplicación del algoritmo Minimax sobre un árbol de juego donde el valor de la función de evaluación se muestra dentro del nodo correspondiente a cada posición.

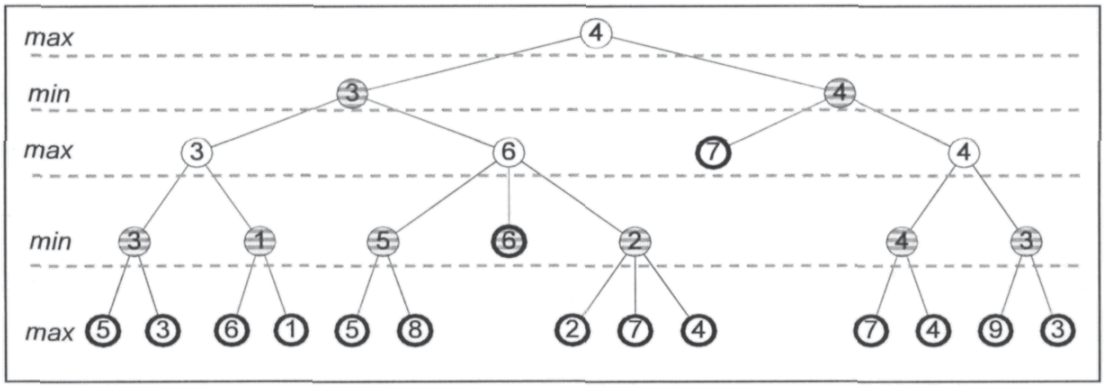


Figura 2.27: Aplicación del algoritmo Minimax a un árbol de búsqueda.

El jugador principal del juego representado en la Figura 2.27 podrá obtener una puntuación máxima con valor 4 en el caso de escoger siempre las jugadas más convenientes y que su oponente elija también las que más le perjudiquen. En el gráfico se muestran sin relleno aquellos nodos que representan movimientos del juego pertenecientes al jugador principal, mientras que con un relleno a rayas aparecen los nodos referentes a los movimientos del jugador contrario —cada grupo pertenece a un tipo de nivel distinto en el árbol, *max* y *min*, respectivamente. Todos los nodos hoja representan, como se ha dicho anteriormente, situaciones finales del juego y, en este gráfico, se resaltan con un trazo grueso.

b) Método Alfa-Beta

En ocasiones los árboles de búsqueda pueden poseer un gran número de nodos a través de los cuales se debe buscar la solución del problema, sobre todo en los casos de los árboles de juego, como se ha visto anteriormente. Para reducir este espacio de búsqueda en el que posteriormente se aplica el algoritmo Minimax se emplean métodos de poda como el Alfa-Beta. Las técnicas de poda deben ser tales que intenten eliminar la mayor cantidad posible de nodos y ramas del árbol sin afectar nunca a la solución del problema, esto es, la solución con o sin poda debe ser siempre la misma. Para llevar a

[3] Un tablero tan simple como el de las tres en raya necesita 9! nodos para representar todas las situaciones posibles de una partida.

cabo esta poda se emplean dos parámetros, α y β, que son los que dan nombre a este método:

- α: es el parámetro empleado por los nodos que pertenecen al nivel *max* del árbol de búsqueda. Representa el valor más alto que se puede obtener a partir de los sucesores del nodo al que se asocia el parámetro.
- β: es el parámetro que se utiliza en los nodos pertenecientes al nivel *min* del árbol de juego. Representa el valor más bajo posible de entre los descendientes del nodo al que está asociado este parámetro.

El método Alfa-Beta se basa en los valores de estos parámetros para tomar la decisión de podar las ramas del árbol de búsqueda. Si el método prevé que explorando una nueva rama del árbol encontrará valores mejores para los parámetros α y β, entonces no realizará poda alguna. Si por el contrario, deduce que no es capaz que encontrar un valor que pueda variar el resultado obtenido hasta el momento, podará todo el subárbol restante a partir de los siguientes sucesores sin explorar.

Si bien es cierto que se seguirán explorando ramas del árbol que no pertenecen a la solución del problema, la cantidad de nodos eliminados es tal que aumenta notablemente la eficiencia de la búsqueda realizada por el Minimax, aspecto que resulta de gran importancia en árboles de búsqueda de dimensiones tan grandes.

En la Figura 2.28 se muestra el resultado de la aplicación de este método de poda a un árbol de búsqueda donde los valores de la función de evaluación están representados en cada nodo. A partir de esta figura se explica la manera de proceder a la realización de las podas, tanto de simples ramas, como sucede en los niveles más profundos del árbol, como de subárboles enteros en los que se elimina una gran proporción de nodos.

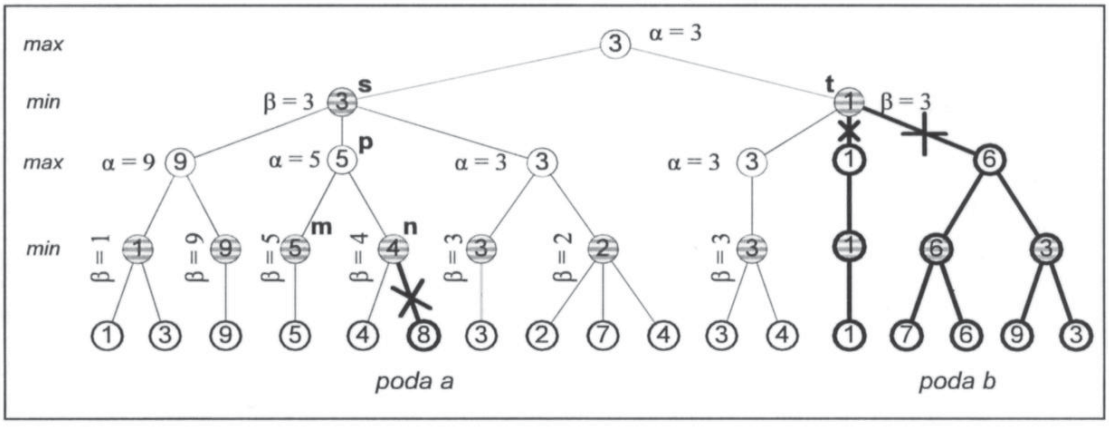


Figura 2.28: Aplicación del método de poda Alfa-Beta a un árbol de juego.

Siguiendo el funcionamiento de los parámetros α y β, cada nodo perteneciente al nivel *min* del árbol asigna al parámetro β el menor valor de la función que tienen sus

nodos hijo y, cada nodo perteneciente al nivel *max* del árbol asigna al parámetro α aquel valor que sea máximo de entre sus nodos hijo. Bajo esta regla, los valores se van propagando por el árbol hasta alcanzar la raíz. Teniendo en cuenta las exigencias de los parámetros de este método, el algoritmo es capaz de predecir si la exploración de una nueva rama resulta útil y, en caso contrario, podarla. Para ello, se pueden analizar las podas realizadas en el ejemplo de la Figura 2.28:

- *Poda α:* en el momento de la realización de la poda, el valor actual del parámetro β en el nodo *n* es $\beta = 4$. Por lo tanto, para que la nueva rama a explorar sea útil para la ejecución del algoritmo, se debería encontrar un valor inferior al actual, es decir, se desea encontrar un valor $\beta < 4$. Por otro lado, el nodo *m,* que es hermano del nodo *n,* posee un valor $\beta = 5$. Puesto que este valor del parámetro es mayor que cualquier otro que pudiera encontrar el nodo n, el padre de ambos, *p,* siempre escogerá el valor almacenado para el parámetro en el nodo *m* por ser un nodo perteneciente al nivel *max.* De este modo, no es necesario seguir explorando la rama nueva, ya que posea el valor que posea nunca será relevante para el resultado del algoritmo.
- *Poda β:* esta poda se realiza de manera similar a la anterior, pero en este caso se poda una mayor cantidad de nodos. Una vez se ha explorado el primer descendiente del nodo *t*, el parámetro β toma el valor 3. De este modo, se necesitaría recibir de alguno de los otros sucesores un valor tal que $\beta < 3$. Sin embargo, puesto que el nodo *s* ya posee un valor $\beta = 3$, el nodo raíz del árbol exigiría recibir de su otro descendiente un valor $\beta > 3$ para modificar el resultado obtenido hasta el momento. Como no existe ningún valor que cumpla al mismo tiempo las dos condiciones, $\beta < 3$ y $\beta > 3$, el algoritmo puede predecir con seguridad que el resto de descendientes del nodo *t* no aportarán ningún valor relevante a la solución del algoritmo y, por ello, se procede a realizar la poda de ambos subárboles.

c) Método SSS*

Se trata de otro método de búsqueda, similar a los anteriores, especializado en árboles de juego. En este caso, el algoritmo trabaja con una estructura de la forma *(n, s, h)* para almacenar la información relevante de cada uno de los nodos del grafo. Los elementos de esta tupia hacen referencia a:

- *n*: identificador del nodo en el árbol de búsqueda.
- *s:* estado en el que se encuentra el nodo.

- *Activo (a):* si el nodo ha sido alcanzado durante la expansión del árbol pero aún no se ha dado por finalizado su estudio.
- *Estudiado (e):* si ya se ha dado por concluido el análisis del nodo.

- *h:* valor de la función heurística que posee el nodo *n* hasta el momento.

Además, para la aplicación de este método se emplea una lista *ABIERTA* en la que se van introduciendo los nodos a medida que se alcanzan durante la ejecución y se ordenan de manera decreciente según el valor de la función heurística almacenado en la estructura anterior.

Inicialmente, se inserta en la lista abierta el nodo raíz del árbol con el estado *activo y* valor infinito en la función heurística, ya que en este momento aún se desconoce cualquier estimación del mismo. A partir de este primer nodo, se van introduciendo el resto de nodos del árbol dependiendo de los valores almacenados en la estructura según a Tabla 2.1.

Valores de la estructura (n, s, h)	Acciones
s = activo *n* de tipo *max* y *no terminal*	Añadir los sucesores de *n* al principio de *ABIERTA* con el estado *activo* y el valor de *h* igual al del padre. Borrar el nodo *n* de *ABIERTA.*
s = activo *n* de tipo *min* y *no terminal*	Añadir solo el primer sucesor de *n* al principio de *ABIERTA* con el estado *activo* y el valor de *h* igual al del padre. Borrar el nodo *n* de *ABIERTA.*
s = activo *n terminal*	Se recoloca el nodo *n* tras los nodos activos y de tal forma que quede delante de aquellos nodos con estado *estudiado* con un valor de *h* menor que el *n* (si hay empates, ordenar según su aparición en el árbol, de izquierda a derecha). Modificar su estado a *estudiado*. Si el valor *h* es mayor que el que existe en la hoja, sustituirlo por el nuevo valor.
s = estudiado *n* de tipo *max* y con hermanos sin estudiar	Añadir el siguiente hermano de *n* al principio de *ABIERTA* con estado *activo* y valor *h* igual al de *n*. Borrar el nodo *n* de *ABIERTA.*

s = *estudiado* *n* de tipo *max* y sin hermanos sin estudiar	Añadir el padre de *n* al principio de *ABIERTA* con estado *estudiado* y valor *h* igual al de *n*. Borrar el nodo *n* de ABIERTA.
s = *estudiado* *n* de tipo *min*	Añadir el padre de *n* al principio de *ABIERTA* con estado *estudiado* y valor *h* igual al de *n*. Borrar el nodo *n* de *ABIERTA*. Borrar de *ABIERTA* todos los sucesores del padre.

Tabla 2.1: Operadores para la expansión de nodos en el algoritmo SSS*.

El algoritmo extrae de la lista *ABIERTA* en cada paso siempre el primer elemento, representado en la Tabla 2.1 como nodo *n.* En el momento en el que el nodo extraído de la lista *ABIERTA* sea el nodo raíz del árbol y su estado sea *estudiado,* se dará por finalizada la aplicación del algoritmo de búsqueda y valor minimax del juego será el que se almacene en el campo *h* de la estructura.

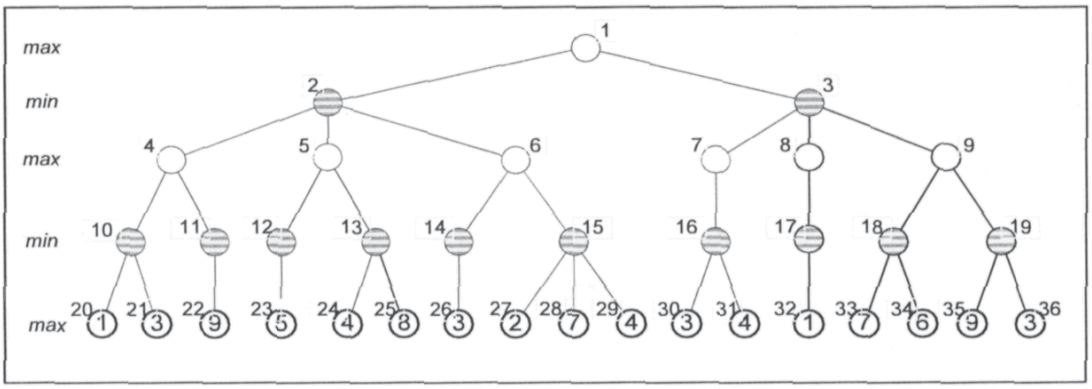


Figura 2.29: Árbol de juego para el ejemplo de aplicación del algoritmo SSS*

Como ejemplo de aplicación de este método de búsqueda del valor minimax de un árbol de juego, se muestra en la Tabla 2.2 (leyéndose de arriba abajo y de izquierda a derecha) una ejecución a partir del árbol de la Figura 2.29. En esta figura los nodos se representan con la etiqueta que aparece al lado de cada uno de ellos, que servirá como identificador en la estructura empleada por el algoritmo. El valor minimax obtenido es 3, el mismo que el resultante de la aplicación del método Alfa-Beta y puede observarse en el nodo raíz cuando su estado es estudiado, al final de la ejecución.

Lista ABIERTA (I)	Lista ABIERTA (II)
(1, a, ∞)	**(23, a, 9)**, (13, a, 9), (30, e, 3)
(2, a, ∞), **(3, a, ∞)**	(13, a, 9), **(23, e, 5)**, (30, e, 3)
(4, a, ∞), (3, a, ∞)	**(24, a, 9)**, (23, e, 5), (30, e, 3)
(10, a, ∞), **(11, a, ∞)**, (3, a, ∞)	(23, e, 5), **(24, e, 4)**, (30, e, 3)
(20, a, ∞), (11, a, ∞), (3, a, ∞)	**(12, e, 5)**, (24, e, 4), (30, e, 3)
(11, a, ∞), (3, a, ∞), **(20, e, 1)**	**(5, e, 5)**, ~~(24, e, 4)~~, (30, e, 3)
(22, a, ∞), (3, a, ∞), (20, e, 1)	**(6, a, 5)**, (30, e, 3)
(3, a, ∞), **(22, e, 9)**, (20, e, 1)	**(14, a, 5)**, **(15, a, 5)**, (30, e, 3)
(7, a, ∞), (22, e, 9), (20, e, 1)	**(26, a, 5)**, (15, a, 5), (30, e, 3)
(16, a, ∞), (22, e, 9), (20, e, 1)	(15, a, 5), **(26, e, 3)**, (30, e, 3)
(30, a, ∞), (22, e, 9), (20, e, 1)	**(27, a, 5)**, (26, e, 3), (30, e, 3)
(22, e, 9), **(30, e, 3)**, (20, e, 1)	(26, e, 3), (30, e, 3), **(27, e, 2)**
(11, e, 9), (30, e, 3), (20, e, 1)	**(14, e, 3)**, (30, e, 3), (27, e, 2)
(4, e, 9), (30, e, 3), ~~(20, e, 1)~~	**(6, e, 3)**, (30, e, 3), ~~(27, e, 2)~~
(5, a, 9), (30, e, 3)	**(2, e, 3)**, (30, e, 3)
(12, a, 9), **(13, a, 9)**, (30, e, 3)	**(1, e, 3)**, ~~(30, e, 3)~~ **FIN**

Tabla 2.2: Ejecución del método SSS* sobre un árbol de juego

La desventaja de este método de búsqueda en árboles de juego es el coste computacional que conlleva la reubicación de los nodos en la lista *ABIERTA.* Al reordenar los nodos se debe comprobar inicialmente el estado de aquellos que ya se encuentran en la lista y, después, el valor heurístico de los nodos con estado *estudiado,* acarreando de este modo un incremento del coste de ejecución.

2.5. Problemas típicos de búsqueda

2.5.1. Problema de las n-reinas

Este problema *[Rivin et al., 1994]*, propuesto inicialmente por Max Bezzel para $n = 8$, trata de colocar n reinas en un tablero de ajedrez de tal forma que no se amenacen entre ellas. Se dice que la figura de la dama amenaza a cualquier otra figura del juego cuando se sitúan sobre la misma fila, columna o diagonal (Figura 2.30).

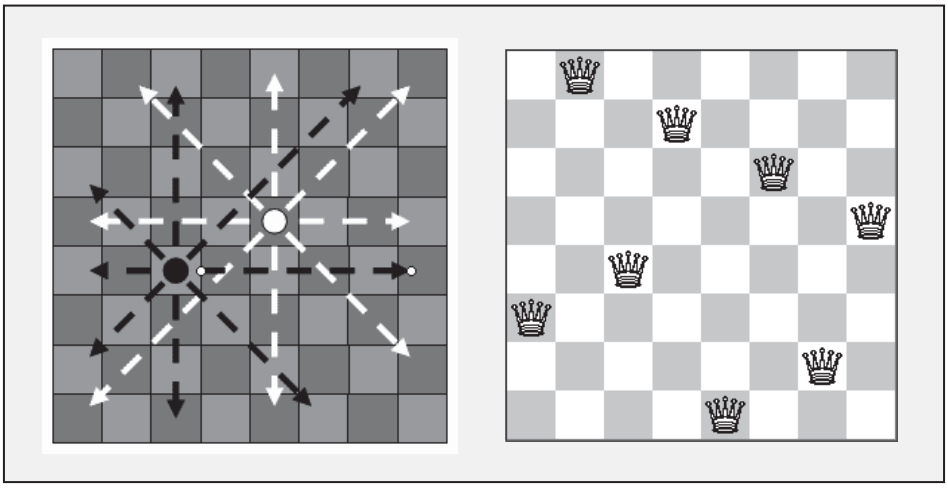


Figura 2.30: Situación en el tablero de las casillas amenazadas por la reina y posible solución.

El método de resolución del problema de búsqueda intentará hallar al menos una de las soluciones que cumplen la condición de que ninguna reina amenace a otra en el tablero —es posible que exista más de una solución para un número fijo, *n*, de reinas. Para ello, se deberá definir una función de evaluación que determine la bondad de cada una de las soluciones, de tal modo que aquellas soluciones en las que haya menos reinas amenazando a otras sean mejores.

2.5.2. Problema del viajero

Se trata de un problema en el que se representa a un viajero que desea trasladarse desde una ciudad de origen a otra de destino pasando una única vez por cada una de las otras ciudades del mapa, de tal forma que el coste del recorrido sea mínimo. Es un problema aparentemente sencillo que encierra una gran complejidad a medida que el número de nodos del grafo que representa el mapa aumenta. Con tan solo 12 nodos en el mapa, las posibles rutas a trazar entre ellos son 479.001.600, de las cuales el algoritmo de búsqueda debe encontrar la de menor coste. Si se tratara de un mapa con 20 ciudades, su resolución se volvería inviable por completo.

El problema del viajero, denominado comúnmente *TSP*, del inglés *Travelling Salesman Problem*, es un problema NP-completo, por lo tanto, está clasificado como uno de los tipos de problema más complejos, puesto que no se ha hallado forma de resolverlo en tiempo polinómico.

3. DESARROLLO DEL SISTEMA

Una de las aplicaciones de los problemas de búsqueda más utilizadas es la de obtención de caminos de coste mínimo entre nodos de un grafo, tal como se ha explicado en el capítulo anterior. El grafo puede representar distintos escenarios y sus características se adaptarán al tipo de problema en cuestión, por ejemplo, la existencia de aristas dirigidas y no dirigidas, la estructura en forma de árbol en caso de inexistencia de ciclos, etc. Dependiendo de estos aspectos referentes a la definición del problema y su forma de representación y, además, teniendo en cuenta otras características deseables como velocidad o calidad de la respuesta, se escoge el algoritmo de búsqueda gracias al cual se obtendrá una solución.

3.1. Descripción del problema

En este proyecto se propone el desarrollo de una aplicación que permita hallar caminos de coste mínimo entre dos nodos cualesquiera de un grafo de gran dimensión. En estos casos, en los que existe un número elevado de nodos, una de las características más deseables del método de resolución es su eficiencia. No basta tan solo con obtener una solución al problema con la total seguridad de ser la óptima, sino que acaba por ser imprescindible una política de ahorro de tiempo, puesto que una complejidad excesiva del algoritmo o una búsqueda demasiado exhaustiva podrían provocar la no terminación de la búsqueda.

Se trabaja con grafos cuyas aristas no están dirigidas, por lo que no es necesario tener en cuenta el sentido del arco a la hora de evaluar los trayectos que se pueden realizar a partir de un nodo dado del grafo o en la realización del cómputo del grado del mismo. Además, es posible la formación de ciclos dentro de la estructura, por lo que, generalmente, el algoritmo de búsqueda del camino de coste mínimo no se verá aplicado a árboles. Es por ello que se requiere especial cuidado en la construcción de caminos entre dos nodos pertenecientes al grafo, puesto que comúnmente existirá más de un camino que conecte dichos nodos. Si no se considerara esta característica, el método de resolución podría entrar en un bucle en el que se estén recorriendo permanentemente los mismos nodos del grafo sin poder avanzar a la solución del problema.

Los problemas representados poseen un único nodo raíz de la búsqueda y un único nodo objetivo. Por este motivo, la dificultad de la búsqueda radica en el amplio número de caminos posibles entre los nodos raíz y objetivo, ya que en caso de no existir ciclos en todo el grafo, la búsqueda se resumiría en encontrar el único camino existente

entre estos dos nodos, sin importar cuál pudiera ser el coste del camino. Lógicamente, resulta completamente inviable la exploración de todos los caminos existentes entre los dos nodos, pues la gran cantidad de nodos del grafo y la existencia de un elevado número de ciclos implicarían un tiempo de ejecución demasiado grande como para que el algoritmo sirva de utilidad. Así, existe la necesidad de buscar un método que resuelva la búsqueda de tal modo que se evite recorrer el grafo más de lo imprescindible para hallar la solución óptima del problema. A medida que el árbol de búsqueda se expande durante la ejecución del algoritmo, el tiempo que tarda en encontrar la solución puede elevarse de manera exponencial hasta impedir que se pueda hallar la solución buscada.

Por otro lado, el grafo con el que se representa el problema puede ser modificado en cualquier momento, de manera que puedan surgir nuevos nodos, conectados a su vez al resto del grafo mediante la aparición de nuevas aristas, o pudiendo dejar de existir nodos o aristas que anteriormente estuvieran representados en la estructura. Este aspecto tiene como consecuencia el completo descarte de la idea de almacenar de algún modo caminos de manera estática, de forma que para un par de nodos dado, se pudiera conocer de antemano cuál es la mejor ruta existente, obtenida en una primera ejecución del algoritmo de búsqueda y simplemente consultada en búsquedas posteriores. En otras palabras, todas estas posibles variaciones deben ser tenidas en cuenta por el método de búsqueda seleccionado, de tal forma que sea capaz de adaptarse a la estructura actual de la red y encontrar de igual modo la solución óptima del problema bajo las nuevas condiciones, puesto que el camino entre los dos nodos dados podría ser modificado en cualquier momento y una solución anterior dejaría de ser válida. Las soluciones que devuelva el sistema propuesto en este trabajo, por lo tanto, deberán ser construidas en el momento de realizar cada búsqueda.

Las aristas de los grafos con los que se trabaja poseen su propio coste, expresado con un valor mayor que 0. Esta información viene dada como definición del problema de búsqueda y está considerada como el valor real que conlleva recorrer la arista a la que está asociado este valor. Además, con la definición de los problemas de búsqueda tratados se tiene una información adicional que permite orientar la búsqueda y facilitar de este modo el alcance de la solución óptima. Se trata de una estimación del coste que acarrea el mejor camino entre el nodo al que está asociada y el nodo objetivo del problema: información heurística. A partir de estos dos tipos de coste, el real y la estimación, el algoritmo de búsqueda trabajará por encontrar la solución óptima existente entre los dos nodos dados. Con este fin, se han empleado técnicas basadas en algunos de los algoritmos descritos en el capítulo anterior, de la manera que se detalla en la sección *Técnicas de búsqueda* que se describe a continuación.

3.2. Técnicas de búsqueda

Aunque el objetivo del proyecto consisten en resolver el problema de búsqueda sobre la estructura de un grafo, es posible aplicar técnicas de búsqueda en árboles adaptando el grafo, como se ha explicado en el capítulo dos. Así, se parte de la idea de búsquedas simples como las realizadas por las técnicas de primero en amplitud y primero en profundidad. Con el uso de estas técnicas se podría construir un árbol a medida que se recorre el grafo que representa el problema de búsqueda, de modo que los nodos adyacentes a uno actual, que todavía no hayan sido visitados por el algoritmo, figurarían como descendientes del nodo en cuestión en el árbol de ejecución. Ambas representan formas de recorrer los nodos de un árbol de manera sencilla, sin embargo, la búsqueda que se desea realizar requiere, no solo el hecho de hallar un camino que conecte los nodos raíz y objetivo, sino que se haga mediante el camino de menor coste posible. Para ello, es necesario tener en cuenta los costes de cada una de las aristas que se recorren en el camino.

La búsqueda de coste uniforme sí tiene en consideración los distintos costes que pueden tomar las aristas del grafo y escoge aquel camino de menor coste que permita alcanzar un nodo objetivo. Este método recorre el árbol de búsqueda de tal manera que siempre expande aquel nodo que en el momento actual de la ejecución posea el menor coste acumulado en el camino que conecta éste con el nodo raíz de la búsqueda, lo que permite que cuando se alcanza un nodo objetivo en el árbol es seguro que se ha conseguido con el menor coste posible. El inconveniente de este método de búsqueda es que no emplea la información adicional que se pueda poseer acerca del problema, algo que simplificaría y aumentaría la probabilidad de encontrar la solución óptima. Es posible que un algoritmo de búsqueda no sea capaz de hallar la solución óptima del problema si tan solo emplea el coste real de cada arista al realizar la denominada *búsqueda a ciegas*, puesto que el hecho de que una rama de árbol de búsqueda posea un menor coste hasta el momento no garantiza que a medida que avanza la ejecución esa rama siga conteniendo el camino de menor coste que una los nodo inicial y objetivo. Además, cuanto mayor es el grafo en el que se está realizando la búsqueda, mayor es el error que se puede cometer, respecto a la solución óptima del problema, al realizar una búsqueda que tan solo tenga en cuenta el coste real acumulado hasta un instante determinado de la ejecución. Puesto que en este proyecto se ejecuta el algoritmo de búsqueda a partir de un grafo de grandes dimensiones, el empleo del método de coste uniforme y, en general, de una técnica de búsqueda no informada, no resulta lo más conveniente.

Al igual que en el caso anterior, el algoritmo de Dijkstra no realiza una búsqueda informada de la solución del problema. Por otro lado, la necesidad de recorrer todos los

nodos del grafo para calcular el coste de alcanzar cada uno de ellos desde el nodo inicial provoca una elevada complejidad en grafos de gran tamaño. El aspecto de visitar más nodos de los necesarios para hallar el camino de mínimo coste entre los nodos inicial y objetivo es uno de los que se desean evitar para poder emplear un algoritmo lo más eficiente posible. Es por este motivo que no resulta uno de los algoritmos más útiles para la búsqueda de soluciones en este tipo de grafos.

Dado que se posee información adicional acerca del problema de búsqueda, resultan muy convenientes aquellos algoritmos que ejecutan de manera informada. El algoritmo voraz de primero el mejor considera siempre en cada decisión la alternativa más prometedora, teniendo en cuenta únicamente aquel descendiente con menor estimación del coste hasta el nodo objetivo. La desventaja surge al ignorar el coste acumulado hasta el momento desde el nodo inicial de la búsqueda. Esta característica, como se pudo observar en el capítulo 2, puede conllevar la modificación del camino escogido como solución del problema, retornando un camino que no coincide con la solución óptima.

El algoritmo A* tiene la característica de aprovechar tanto la información real del problema como la estimada desde cada nodo hasta el final. Esto se considera como una gran ventaja, puesto que permite hallar la solución óptima del problema en caso de existir alguna. El algoritmo trabaja de forma que, cada vez que debe tomar alguna decisión acerca de la alternativa a escoger, valorará el coste acumulado de cada alternativa junto con el coste estimado para alcanzar el destino, de modo que asegura escoger siempre la mejor alternativa existente. No se trata de un algoritmo de elevada complejidad, por lo que su empleo en grafos de gran tamaño resulta muy conveniente. Entre las características que favorecen su eficiencia se encuentra el especial cuidado en evitar la creación de ciclos en el camino construido como solución. Además, puesto que en cada decisión escoge siempre la mejor alternativa, se asegura visitar únicamente el mínimo número de nodos necesarios para obtener la solución óptima, con lo que la proporción de estos nodos visitados con respecto al total no suele ser muy grande. Por lo tanto, el algoritmo A* estaría considerado de gran utilidad para resolver el problema concerniente a este proyecto.

Otros algoritmos basados en A*, esto es, los algoritmos IDA*, SMA* y BIDA* cumplen también la propiedad de optimalidad, de modo que cualquiera de ellos podrá obtener la solución óptima de un problema realizando una búsqueda informada. Cada uno de ellos posee sus propias características, pero no todas resultarían útiles para el problema planteado. El algoritmo IDA* efectúa una búsqueda de manera que limita la cantidad de memoria empleada, algo que en un grafo grande puede resultar necesario. Sin embargo, tiene la desventaja de tener que recorrer de nuevo en cada iteración los nodos visitados en la iteración anterior. Esto produce un aumento del tiempo necesario para la ejecución del algoritmo, que no resulta una característica deseable en esta aplicación. En caso de que se almacenaran estáticamente las rutas entre cada par de

nodos del grafo (aspecto que conllevaría una gran capacidad de almacenamiento en redes grandes) no sería tan importante la velocidad en encontrar una solución, puesto que esta labor solo se realizaría la primera vez que se ejecuta. Pero debido a la característica de variabilidad del grafo con el que se trabaja, que como se ha explicado anteriormente hace descartar la posibilidad de este tipo de almacenamiento, el algoritmo deberá encontrar la solución óptima con cada ejecución, por lo que es preferible que el tiempo requerido en dichas ejecuciones sea el menor posible.

El algoritmo SMA* intenta solventar el inconveniente de IDA*, realizando ejecuciones en las que no se tenga que visitar reiteradamente los mismos nodos del grafo hasta encontrar la solución. Sin embargo, en el empeño por reducir la cantidad de memoria empleada fijando dicha cantidad para que el algoritmo no pueda sobrepasarla, conlleva la posibilidad de que no sea capaz de devolver la solución óptima del problema de búsqueda, retornando como solución la mejor posible dentro de ese espacio de memoria disponible.

El algoritmo bidireccional BIDA*, además de las ventajosas características del algoritmo A*, posee otras que podrían resultar de interés a la hora de resolver el problema tratado en este proyecto, como se describirá más adelante en este capítulo. El establecimiento de un perímetro formado por los nodos más cercanos al nodo objetivo del problema permite simplificar la búsqueda en cierto modo, una vez que se conozca cuál es el mejor camino entre los nodos pertenecientes al perímetro y el nodo meta.

3.3. Estructura del problema

La técnica o algoritmo empleado para la resolución de este problema de búsqueda deberá tomar en cada nodo explorado del grafo la decisión acerca de cuál es la mejor alternativa existente a partir de él y, por lo tanto, hacia dónde continuar la búsqueda. Sin embargo, no siempre es necesaria la toma de decisiones por parte del algoritmo, ya que podría darse el caso de que el nodo actual, en el que se encuentra el algoritmo, tan solo posea un descendiente. En estos casos, no cabe cuestionar cuál es la mejor alternativa ni el coste de ésta, sino que tan solo se podrá continuar por el único camino existente hasta un nuevo nodo en el que se requiera tomar una decisión. De este modo, surge una clasificación de los nodos pertenecientes al grafo del problema: *nodos de decisión* y nodos *simples.*

Los nodos de decisión son aquellos en los que el algoritmo está obligado a escoger una alternativa para continuar la búsqueda. El grado de todos ellos posee siempre un valor de al menos 3 aristas. Esto es, para que el algoritmo se vea obligado a

tomar una decisión para continuar hacia el siguiente paso deben existir al menos dos alternativas. Puesto que una de las aristas que llegan al nodo actual ha sido empleada por el algoritmo para alcanzar dicho nodo, éste deberá poseer como mínimo dos aristas más, que se utilizarán como aristas de salida, para ser denominado nodo de decisión. Cuanto mayor sea el grado de un nodo más complejo será el análisis acerca de cuál es la mejor opción para continuar, ya que se deberán calcular los costes para cada uno de ellos.

Se consideran nodos simples de un grafo aquellos en los que no cabe la duda sobre el siguiente paso a dar por el algoritmo, puesto que tan solo existe una arista de salida del nodo. Es decir, si el nodo actual tiene grado 2 significará que una de las aristas que posee servirá como entrada al nodo y la restante será obligatoriamente la arista de salida. Lo mismo sucede con aquellos nodos de grado 1, en los que ni siquiera existiría una salida por la que el algoritmo pueda continuar la búsqueda. Por lo tanto, en caso de explorar estos nodos, sería como punto de partida, empleando de este modo su única arista como salida del propio nodo.

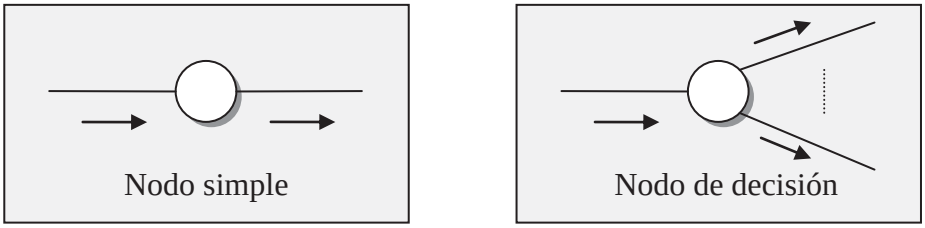


Figura 3.1 Clasificación de los nodos del grafo según su grado.

El sistema propuesto empleará las técnicas de búsqueda ya conocidas, pero limitando su uso tan solo a los nodos de decisión, rebajando de este modo la complejidad de las operaciones durante la ejecución. El hecho de no realizar ningún cálculo de costes sobre los nodos simples elimina cierto número de operaciones que, cuando la proporción de nodos simples sobre la totalidad de nodos del grafo es elevada, reduce notablemente el coste de tiempo de cada búsqueda. Obviamente, el algoritmo propuesto en este trabajo no elude totalmente los nodos simples del grafo, puesto que esto no permitiría construir el camino completo devuelto como solución al problema de búsqueda, sino que guarda la información necesaria y continúa explorando el siguiente nodo.

Los nodos inicial y final de la búsqueda podrán pertenecer a cualquiera de estos dos tipos, no variando en ningún caso el resultado obtenido, puesto que esta clasificación se realiza únicamente con motivos de eficiencia al ahorrar operaciones al algoritmo. La clasificación sirve de ayuda para que el algoritmo sepa reconocer aquellos casos en los que la solución de un paso ya es conocida de antemano y no se requiere ninguna evaluación para obtener más información.

Como se ha mencionado anteriormente, el tipo de grafo con el que se trabaja no posee aristas dirigidas. De este modo, no es necesario realizar ningún cómputo que permita comprobar por qué aristas es posible salir del nodo en el que se encuentra el algoritmo empleado. En caso contrario, la complejidad de las operaciones se elevaría porque, además de realizar las relacionadas con la búsqueda de caminos entre nodos, habría que añadirle la comprobación de las aristas que deben emplearse únicamente como entrada al nodo y las que servirán exclusivamente a la salida del mismo. De hecho, un nodo perteneciente a un grafo en el que la totalidad de las aristas que posee son de entrada nunca podría ser el nodo inicial de una búsqueda, pues en caso de serlo resultaría imposible alcanzar cualquier nodo objetivo desde él.

Tratándose de grafos conexos donde no existen aristas dirigidas se puede asegurar que existe un camino entre cualquier par de nodos dados como origen y destino de la búsqueda y, por lo tanto, el algoritmo escogido deberá devolver algún camino.

3.4. Arquitectura software

Basándose en la estructura establecida para el problema, se ha creído conveniente emplear el paradigma de la orientación a objetos. Este paradigma se centra en los datos que conforman el problema y, en el caso tratado en este trabajo, el grafo posee dos tipos de elementos principales: las aristas y los nodos. Cada uno de ellos es una entidad en sí mismo y tiene unas propiedades muy concretas que hacen pensar que el paradigma que se corresponde de manera más adecuada sea uno que se base en los datos y no en las funcionalidades.

La modularidad que se emplea mediante las clases del paradigma de la orientación a objetos permite al sistema ser fácilmente ampliable y variable. De esta manera, si se separa claramente la implementación del algoritmo de búsqueda del resto de clases del sistema, se reducirá en gran medida la complejidad a la hora de sustituir el algoritmo empleado por cualquier otro que resuelva el mismo problema.

3.4.1. Un grafo con clase

Puesto que se trabaja con grafos, el sistema desarrollado debe tener en cuenta los elementos básicos de esta estructura a la hora de definir las clases que se implementarán mediante el uso del paradigma de orientación a objetos:

- *Aristas:* clase del sistema que contiene la información relevante a cada arista del grafo, como puede ser su coste o el par de estaciones que conecta. Además,

poseerá los métodos necesarios para realizar operaciones con las aristas o conocer la información acerca de ellas que el algoritmo pueda necesitar.

- *Nodos:* es aquella clase del sistema que debe contener la información de cada vértice del grafo, como el identificador del nodo dentro del grafo o la relación con las aristas adyacentes. De igual modo será necesario almacenar la información relevante a los costes que se emplearán durante la ejecución del algoritmo, como son el coste que conlleva llegar desde el nodo inicial hasta él, el coste estimado para alcanzar el destino desde el propio nodo y el valor que posea en la función de evaluación en un momento determinado.

Para facilitar el control de las operaciones sobre ambos tipos de clases y puesto que existen ciertas acciones que se realizan sobre un conjunto de los mismos, se necesitan colecciones de datos, aristas y nodos, conformando nuevas clases en el sistema:

- *Colección de aristas:* se trata de la clase del sistema que controlará las operaciones realizadas sobre las aristas, ya sea sobre un conjunto de ellas o sobre una arista en concreto. Es la única clase capaz de buscar una arista determinada del grafo, pues almacena la lista que contiene todas las aristas pertenecientes a la estructura.
- *Colección de nodos:* es la clase del sistema que actúa de manera homologa a la colección de aristas. Permite la realización de operaciones sobre un conjunto de nodos del grafo o sobre un nodo determinado. Posee la lista completa de todos los nodos existentes en el grafo, por lo que es la única clase con capacidad para efectuar este tipo de operaciones.

El resto de clases existentes en el sistema deberán comunicarse con los elementos básicos del grafo a través de las colecciones anteriores, preservando de este modo la privacidad de los datos almacenados.

3.4.2. Almacenamiento de datos

Junto con la definición del problema, es decir, aquellos datos relevantes a las aristas pertenecientes al grafo y los nodos, se posee una información adicional que permitirá que el sistema propuesto emplee un método de búsqueda informada. Para hacer esto posible, es necesario almacenar la información sobre los costes que conlleva efectuar el mejor camino entre un par de nodos cualesquiera del grafo y que esta información sea accesible en todo momento de la ejecución del algoritmo.

Estos valores que representan una estimación acerca del coste de la mejor ruta para conectar cada par de nodos se almacenan en una base de datos relacional. Este tipo de bases de datos es uno de los más utilizados actualmente, ya que permite estructurar la información que contiene de manera sencilla e intuitiva. Los datos se almacenan en tablas tras hacer un análisis y clasificación de los mismos. Además, todos los datos guardados en diferentes tablas pueden vincularse unos con otros mediante las denominadas *relaciones* —característica del *modelo relacional*. De este modo se construye la relación entre los nodos del grafo para conocer cuál es el coste estimado entre ellos. Debe tratarse de una base de datos dinámica, puesto que los datos que se almacenan en ella pueden ser modificados con el tiempo debido a la posible variabilidad del grafo. Podría darse el caso en el que surjan nuevos nodos y aristas o se eliminen algunos de los elementos ya existentes y, por consiguiente, la estimación de la mejor ruta entre un par de nodos determinado podría cambiar.

Se emplea el lenguaje SQL *(Structured Query Language)* para realizar la comunicación con la base de datos donde se almacena la información heurística del problema. Este lenguaje declarativo permite obtener con el uso de un reducido número de sentencias la información necesaria sobre la estimación de los costes. Por otro lado, se utiliza este lenguaje por las ventajas que aporta el hecho de tratarse de un lenguaje muy estudiado y que, por tanto, las mejoras sobre su eficiencia son notables con respecto a otros lenguajes.

El sistema propuesto debe comunicarse, entonces, con la base de datos donde se almacenan los costes necesarios para la ejecución del algoritmo. Para ello, es necesaria la creación de una nueva clase que permita separar las consultas del resto de la implementación del sistema:

- *Base de datos:* es la clase que se encarga de la comunicación con la base de datos que almacena la información del grafo, en especial los costes de los caminos. Emplea el lenguaje SQL para llevar a cabo esta labor y obtener así los datos que después comunicará al resto del sistema para continuar la ejecución del método de búsqueda.

3.4.3. Algoritmo de búsqueda

Otra de las partes fundamentales del sistema es la clase encargada de implementar el algoritmo de búsqueda. La razón de la separación en una clase independiente es, a parte de la modularidad implicada en la orientación a objetos, la posibilidad de cambiar el algoritmo de búsqueda en cualquier momento reduciendo al máximo los cambios en el resto del sistema, como se ha explicado anteriormente.

- *Algoritmo de búsqueda:* esta clase implementa los pasos realizados por el algoritmo escogido y se comunica con el resto de clases del sistema para conocer los datos referentes al grafo del problema.

Debido a la falta de visibilidad sobre los objetos de las clases de las aristas y los nodos, el algoritmo de búsqueda obtiene la información necesaria para su ejecución a través de las colecciones de datos, como por ejemplo, la obtención de sucesores o aristas adyacentes del nodo actual o los distintos tipos de costes del mismo, que se habrán recuperado previamente de la base de datos donde se almacenan. Con todo ello, el algoritmo irá construyendo el camino solución, en caso de existir, y devolverá el resultado al finalizar su ejecución, cumpliendo de este modo con el cometido del sistema propuesto.

3.4.4. Interfaz del sistema

El usuario interactúa con el sistema desarrollado indicándole las opciones del trayecto deseado. Para facilitar dicha interacción se ha implementado una interfaz gráfica que permita seleccionar los nodos de origen y destino entre los cuales se desea encontrar el camino de coste mínimo. Del mismo modo, también es deseable que tanto el grafo del problema como el camino solución se muestren por pantalla para una fácil comprensión por el usuario. Las clases pertenecientes a la implementación de la interfaz gráfica se han elaborado dentro de una librería distinta de la que contiene la implementación del resto del sistema para obtener una mayor independencia. De esta manera, la interfaz gráfica podrá ser modificada en cualquier momento sin afectar al núcleo del sistema donde se tratan los elementos básicos de la estructura del grafo y el algoritmo de búsqueda.

Cada elemento perteneciente a la interfaz desarrollada, tales como el menú, la barra de herramientas, la región de dibujo —*canvas*— o la región para la selección de opciones de búsqueda (selección de nodos inicial y objetivo), es una clase independiente del paquete de presentación. Dentro de estas clases se implementan las características propias de cada uno de estos elementos, las operaciones que se realizan cuando se activan ciertos eventos y los métodos que permiten comunicarse unos elementos con otros y con el resto del sistema. Los eventos pueden ser activados por el usuario de la aplicación, por ejemplo mediante botones o movimientos de ratón. La región de dibujo se encarga de la representación del grafo por pantalla, para hacerlo visible al usuario, y a su vez, permite mostrar en el propio grafo el camino devuelto como solución del problema. En este mismo gráfico el usuario también puede seleccionar los nodos inicial y objetivo de la búsqueda, por lo que este elemento debe comunicarse con la clase que implementa el algoritmo de búsqueda.

Además de representarse el camino solución del problema a través de la pantalla, es necesario mostrar el coste de dicho camino, que se hará mediante una ventana informativa tras finalizar la búsqueda.

3.4.5. Lenguaje de programación del sistema

Según las características que requiere el sistema propuesto, como el uso del paradigma de orientación a objetos, se considera que el lenguaje que mejor se adapta a estas necesidades es *C++*. Se trata de un lenguaje que trabaja con elementos vistos como objetos, agrupándolos en clases cuando existen conjuntos de éstos que cumplen internamente las mismas propiedades, como sucede en esta aplicación. Es un lenguaje muy potente que permite realizar operaciones de bajo nivel en contraposición a otros lenguajes como sucedería con *Java*. Además, la gran ventaja de desarrollar una aplicación empleando el lenguaje *C++* es la independencia de la plataforma sobre la que se ejecuta, permitiendo que una misma aplicación pueda ser utilizada en diversos sistemas operativos sin realizar cambios sobre la misma. Existen también numerosos métodos ya implementados para este lenguaje que pueden ser reutilizados en beneficio de la claridad del código y una mejor eficiencia.

3.5. Algoritmo de búsqueda del sistema

El tipo de problema que se pretende resolver mediante el desarrollo de esta aplicación posee una información adicional que resulta de gran ayuda para encontrar el camino óptimo entre dos nodos cualesquiera de un grafo dado. Por lo tanto, de todos los algoritmos de búsqueda analizados en este trabajo, servirá de mayor utilidad aquel que realice una búsqueda informada para encontrar la solución. Como ya se ha visto anteriormente, existen varios algoritmos que cumplen con esta característica, pero de entre ellos se debe escoger aquel que aporte mayor eficiencia. No se trata de un sistema en el que no interese una rápida obtención de resultados, como podría suceder en sistemas en los que solo se realiza una búsqueda y los resultados no van a variar en ningún momento, por lo que posteriormente tan solo se realizarían consultas sobre los datos obtenidos. En este caso, no es suficiente con efectuar una búsqueda entre dos nodos del grafo y almacenar el resultado para consultas futuras, ya que el grafo podría haber variado y, por tanto, el resultado óptimo se vería afectado. Este hecho provoca la necesidad de realizar una búsqueda cada vez que se ejecuta la aplicación sobre un par de nodos y, al tratarse de grafos con un número elevado de nodos, la eficiencia a la hora de encontrar la solución cobra importancia.

Debido a estas razones se ha considerado que el algoritmo de búsqueda A* es el que mejor se adapta a las necesidades del problema. En el sistema propuesto, este método actúa sobre los nodos de decisión del grafo del problema, es decir, este tipo de nodos son los único que resultan *visibles* para el algoritmo de búsqueda, puesto que solamente en estos nodos del grafo se pueden tomar decisiones.

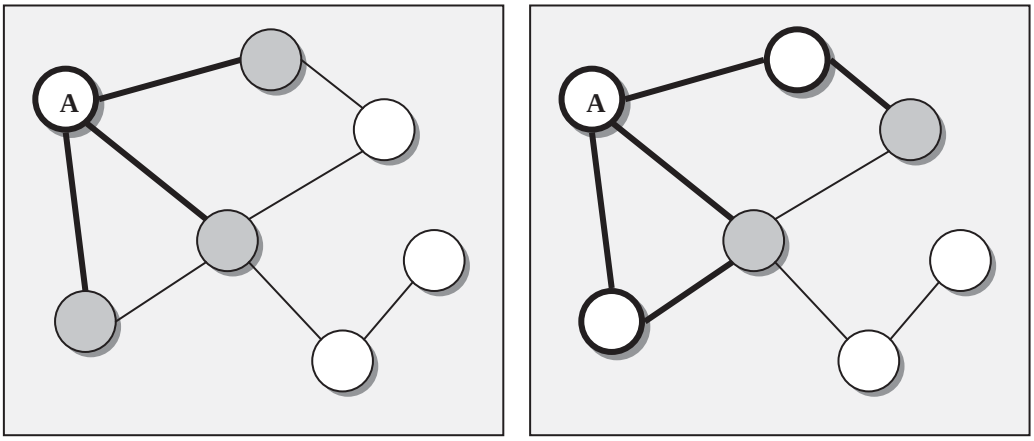


Figura 3.2 En la figura de la izquierda se representan sombreados los descendientes del nodo A en una ejecución normal del algoritmo A*. En la figura de la derecha se muestran los descendientes del nodo A si el algoritmo solo considera los nodos de decisión.

La eficiencia del algoritmo aumenta cuanto mayor es la proporción de nodos simples con respecto a la totalidad de nodos del grafo. Como se puede observar en la Figura 3.2, ya no solo se reduce el número de nodos en todo el grafo sobre el que trabaja el algoritmo de búsqueda, sino que además puede ocurrir que el número de descendientes de un nodo dado se reduzca, disminuyendo de este modo el número de operaciones que se deben realizar para decidir qué nueva alternativa es mejor para avanzar al siguiente paso de la ejecución.

En la implementación del algoritmo A* con esta pequeña modificación permite la ejecución común del algoritmo como si el grafo del problema solo estuviera compuesto por los nodos de decisión. Sin embargo, a la hora de construir el camino solución, siempre se debe almacenar cuál es realmente el padre de cada nodo, ya sea simple o de decisión. Si no se guardara esta información se podrían devolver como resultado de la ejecución caminos que realmente no existen dentro del grafo del problema (Figura 3.3).

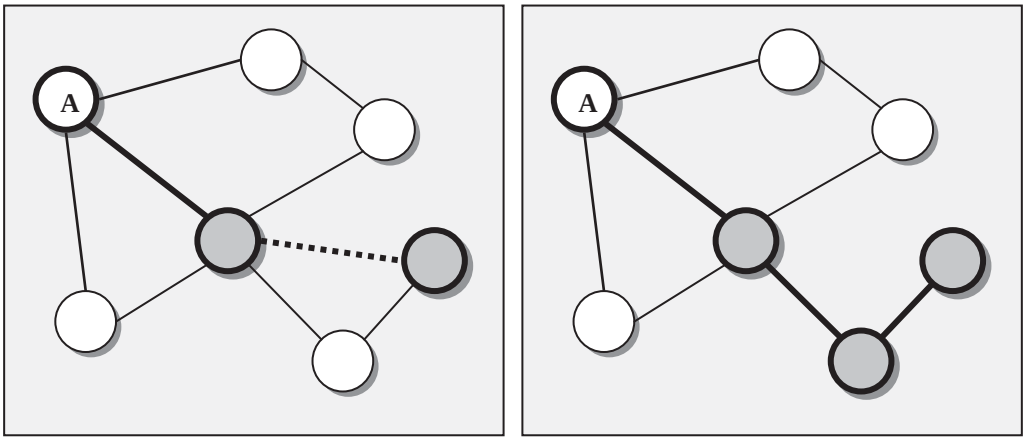


Figura 3.3 En la figura de la izquierda no se almacenan correctamente los antecesores de cada nodo, por lo que se devuelve un camino erróneo. En la figura de la derecha se consideran los nodos simples para construir la solución real.

3.5.1. Tratamiento de los nodos inicial y final

Cuando el nodo inicial o el objetivo son nodos simples del grafo se deben realizar algunas operaciones adicionales para adaptar el método explicado a esta situación. Al tratarse de un nodo simple, tendrá, a lo sumo, dos descendientes y éstos, a su vez, pueden ser nodos simples o de decisión. El objetivo en estos casos es avanzar el estado de la ejecución hasta hallarse en un nodo de decisión, a partir del cual comienza a aplicarse el algoritmo de búsqueda A* tal como se define. Por lo tanto, si el nodo de origen de la búsqueda es un nodo simple existen, como mucho, dos nodos de decisión hasta los que avanzar, aunque éstos no tendrían por qué ser adyacentes al inicio y podrían encontrarse a mayor profundidad, como se representa en la Figura 3.4.

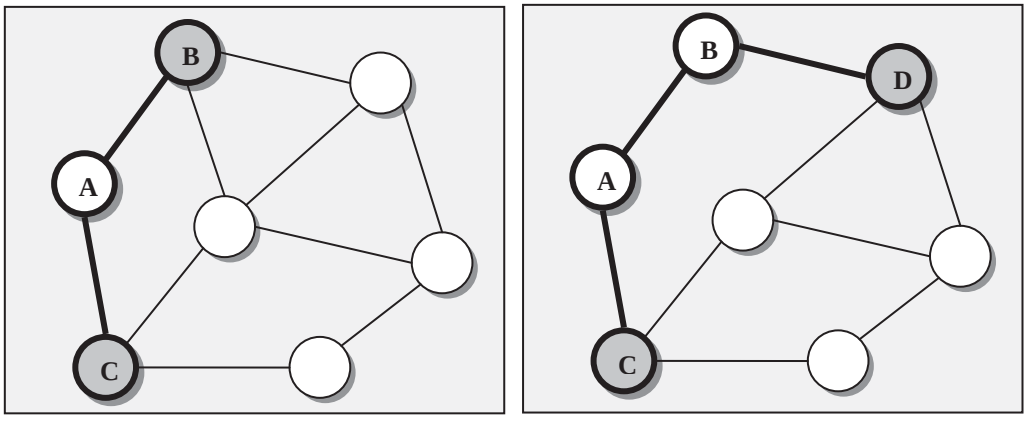


Figura 3.4 En la figura de la izquierda los dos nodos adyacentes al nodo A son nodos de decisión, por lo que se tratará de avanzar hasta B o C. En la figura de la derecha el nodo B es un nodo simple, así que se intentará avanzar hasta C o D, que son los nodos de decisión más cercanos.

En este caso, en el que el nodo inicial es un nodo simple y se debe alcanzar un nodo de tipo decisión antes de comenzar a aplicar el algoritmo A*, aparecen por tanto dos posibles nodos que podrían ejercer de nuevos nodos de inicio para el algoritmo, ndo_1 y ndo_2. Para decidir cuál de estos dos nodos heredará el papel de nodo inicial del algoritmo A*, se evalúan mediante la función de evaluación:

$$h(n) = g(n) + h(n)$$

Ésta es la función de evaluación empleada comúnmente por el algoritmo A*, en la que se tienen en cuenta tanto el coste real del mejor camino entre el nodo inicial y el nodo *n*, denominado *g(n)*, y el coste estimado para el camino óptimo entre el nodo *n y* el nodo meta. Así, se calculan los valores

$$f(ndo_1) = g(ndo_1) + h(ndo_1)$$
$$f(ndo_2) = g(ndo_2) + h(ndo_2)$$

Aquel nodo, ndo_1 o ndo_2 que permita obtener un valor menor para la función de evaluación será el que esté contenido en el camino solución entre el nodo inicial original y el nodo objetivo de la búsqueda y, por tanto, pasará a ser el nuevo nodo de origen para la búsqueda realizada por el algoritmo A*. En esta función de evaluación, el componente *g(n)* para los nodos ndo_1 y *ndo2* se construye a partir de la suma de los costes reales de las aristas del único camino entre el nodo inicial simple y los nodos ndo_1 y ndo_2, respectivamente.

A partir de los datos de la Figura 3.5, las funciones de evaluación correspondientes a los nodos de decisión equivaldrían a las siguientes expresiones:

$$f(ndo_1) = g(ndo_1) + h(ndo_1) = c_{11} + c_{12} + h(B)$$
$$f(ndo_2) = g(ndo_2) + h(ndo_2) = c_{21} + h(C)$$

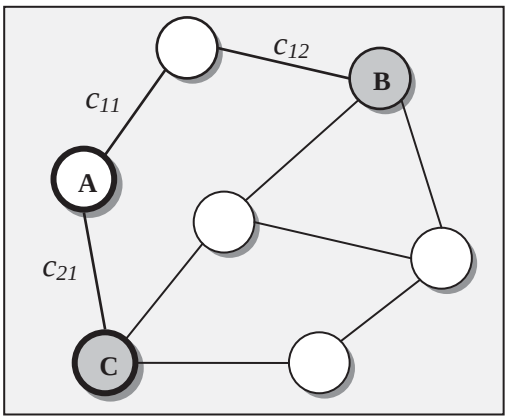


Figura 3.5 Cálculo de la función de evaluación de los nodos de decisión B y C.

Al igual que en la aplicación del algoritmo A* en el resto del grafo, los nodos simples han sido obviados de los cálculos para conocer cuál es el primer nodo de decisión sobre el que se aplica el algoritmo de búsqueda. Solo es necesario calcular la función de evaluación sobre el par de nodos de decisión antes de comenzar la aplicación del algoritmo A*. Los nodos simples que se encuentran en el camino entre el nodo inicial y cada uno de los nodos de decisión tan solo se tienen en cuenta a la hora de construir el camino que los conecta.

Del mismo modo en que se trata el caso en el que el nodo de origen de la búsqueda es un nodo simple, se debe tratar aquel en el que es el nodo objetivo el que es un nodo simple, aunque con una ligera variación puesto que en esta situación aún no se conoce cuál es el coste real del camino entre el nodo origen hasta cada uno de los posibles nodos de destino, ndd_1 y ndd_2. Si el nodo meta es un nodo simple, puede tener, a lo sumo, dos nodos de decisión cercanos, de entre los cuales uno de ellos será alcanzado por la ejecución del algoritmo A* en busca del nodo objetivo. Para calcular cuál de los dos nodos de decisión más cercanos al nodo meta es el que pertenece al camino de coste mínimo entre los dos nodos establecidos para la búsqueda se evalúa cada uno de estos dos nodos mediante la función, *f(n)*, descrita anteriormente. Sin embargo, puesto que no se conoce el valor real que conlleva realizar el recorrido desde el nodo inicial hasta los nodos ndd_1 *y* ndd_2, la función de evaluación con la que se comparan este par de nodos adquiere la siguiente modificación:

$$f(ndd_1) = g(n) + h'(n, ndd_1) + h(ndd_1)$$
$$f(ndd_2) = g(n) + h'(n, ndd_2) + h(ndd_2)$$

En el supuesto de que el nodo inicial de la búsqueda sea un nodo de decisión, es decir, en el que el nodo inicial que se tomará en la aplicación del algoritmo A* coincide con el dado por el usuario para calcular el camino, el nodo *n* de las expresiones anteriores será el mismo que el nodo inicial. Esto significa que, si el nodo *n* es el nodo inicial de la búsqueda, se cumpliría que $g(n) = g(nodo\ inicial) = 0$. Por lo tanto, el cálculo del valor de la función de evaluación para los nodos de decisión más cercanos a la meta se reduciría a:

$$f(ndd_1) = h'(nodo\ inicial, ndd_1) + h(ndd_1)$$
$$f(ndd_2) = h'(nodo\ inicial, ndd_2) + h(ndd_2)$$

La función $h'(n_1, n_2)$ permite obtener el coste estimado del mejor camino entre los nodos n_1 *y* n_2. De este modo, el cálculo de las expresiones anteriores permite conocer desde el nodo inicial qué nodo, ndd_1 o ndd_2 otorgará el camino de coste menor al atravesarlo para alcanzar el destino. Por ello, cuando el nodo meta de la búsqueda es un nodo simple del grafo, la aplicación del algoritmo A* tomará como nodo objetivo

visible el nodo ndd_1 *y* ndd_2, según sea aquel que posea un menor valor en la función de evaluación.

En el caso de la Figura 3.6 se debe decidir si es el nodo B o el C el que se emplee como nodo de destino del algoritmo, dado que el objetivo de la búsqueda es un nodo simple. Para ello se usan dos pares de estimaciones:

$$f(B) = h'_1 + h_1$$
$$f(C) = h'_2 + h_2$$

El nodo que devuelva un valor inferior en las expresiones anteriores será seleccionado como el nodo objetivo con el que trabajará el algoritmo A* durante su ejecución.

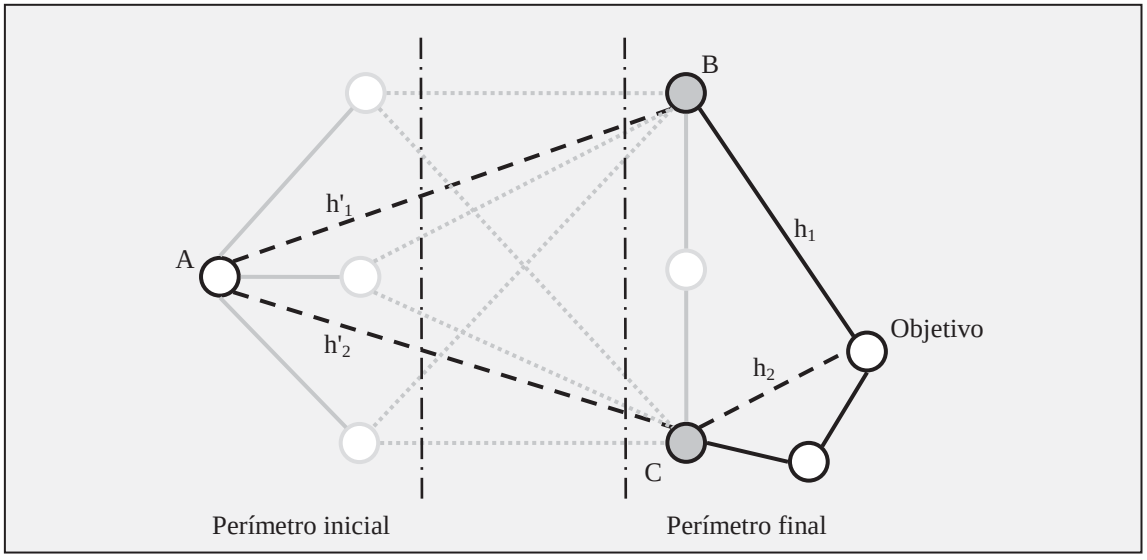


Figura 3.6 Estimaciones de los caminos que conectan los nodos inicial y objetivo atravesando los nodos B y C, respectivamente.

En el caso en el que tanto el nodo inicial como el nodo objetivo de la búsqueda sean nodos simples del grafo (Figura 3.7), se deben realizar algunos cálculos adicionales puesto que se poseen dos pares de nodos que comparar. Es decir, para obtener qué nodos se considerarán origen y destino de la búsqueda por el algoritmo A* se debe conocer qué par ndo_i - ndd_j devuelve el menor valor de la función de evaluación.

$$f(ndo_1\_ndd_1) = g(ndo_1) + h'(ndo_1, ndd_1) + h(ndd_1)$$
$$f(ndo_1\_ndd_2) = g(ndo_1) + h'(ndo_1, ndd_2) + h(ndd_2)$$
$$f(ndo_2\_ndd_1) = g(ndo_2) + h'(ndo_2, ndd_1) + h(ndd_1)$$
$$f(ndo_2\_ndd_2) = g(ndo_2) + h'(ndo_2, ndd_2) + h(ndd_2)$$

A través del cálculo de estas cuatro expresiones se podrá conocer qué nodos deberá atravesar el camino solución para obtener el menor coste entre los nodos dados por el usuario como extremos de la búsqueda.

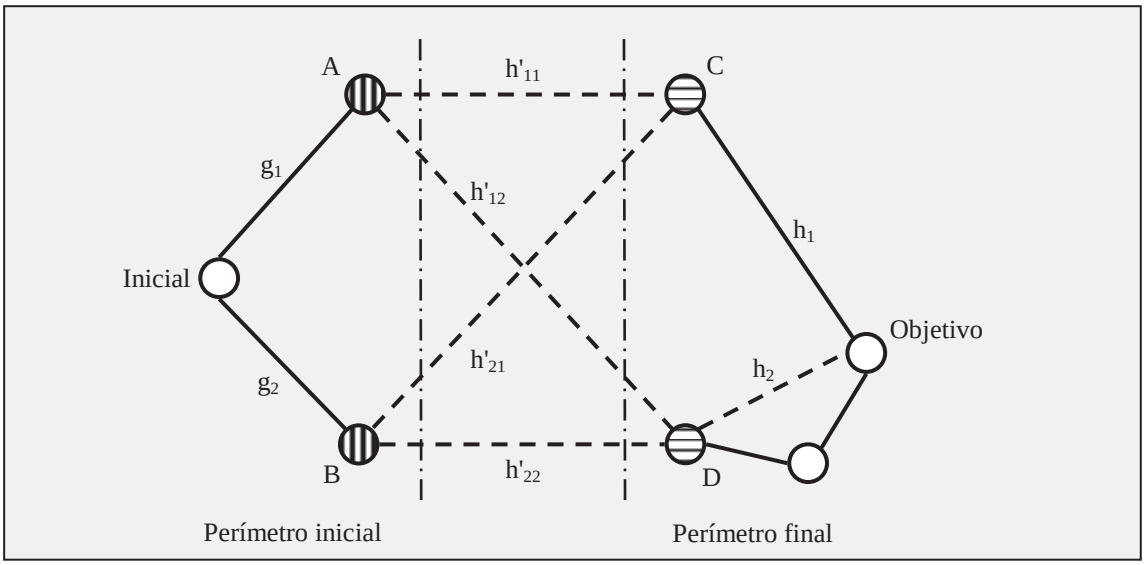


Figura 3.7 Búsqueda con nodos inicial y final de tipo simple.

Según los datos de la Figura 3.7, las expresiones a calcular equivaldrían a las siguientes expresiones:

$$f(AC) = g_1 + h'_{11} + h_1$$
$$f(AD) = g_1 + h'_{12} + h_2$$
$$f(BC) = g_2 + h'_{21} + h_1$$
$$f(BD) = g_2 + h'_{22} + h_2$$

Una vez se hayan comparado los valores devueltos por las expresiones anteriores, se escoge aquel de menor valor, obteniendo de este modo la pareja de nodos de origen y destino para la ejecución del algoritmo A* contenida en el camino de mínimo coste entre los nodos originales de la búsqueda. Estos dos nodos, ndo_i - ndd_j, serán los empleados en la aplicación del algoritmo A* para su ejecución habitual. Todo esta estrategia es similar a la idea del algoritmo BIDA*, en el que se construye un perímetro alrededor del nodo objetivo con nodos que se sitúan a una distancia menor a otra dada de dicho nodo meta. Cuando se hayan obtenido todos los nodos pertenecientes al perímetro, y para los cuales se conoce el camino óptimo que los conecta con el nodo objetivo, se aplica el algoritmo A* desde el nodo inicial hasta alcanzar alguno de los nodos del perímetro. En la estrategia definida en esta sección, se construye un perímetro similar alrededor de los nodos inicial y final cuando éstos son nodos simples del grafo. Si solo se contabilizan los nodos de decisión para construir el perímetro alrededor de los

nodos inicial o final, se puede decir que se incluirán en este perímetro aquellos nodos que se encuentre a distancia 1 del nodo central. Debido a las características de estos nodos centrales, en el perímetro siempre habrá, como máximo, dos nodos. Al igual que sucede con el algoritmo BIDA*, el algoritmo A* se aplica sobre aquellos nodos que no pertenecen al perímetro. Sin embargo, en lugar que finalizar la búsqueda cuando se alcance alguno de los nodos definidos en el perímetro, se puede estimar, antes de comenzar la búsqueda, cuál será de todos ellos el que aportará un coste menor al camino solución. Puesto que el perímetro contará, a lo sumo, con dos nodos, sigue resultando eficiente realizar los cálculos necesarios para efectuar dicha estimación.

3.5.2. Aplicación del algoritmo A*

Dados dos nodos de origen y de destino de la búsqueda que sean de tipo decisión en el grafo, se ejecuta el algoritmo A* como se ha descrito en el capítulo 2, pero teniendo en consideración únicamente aquellos nodos en los que exista más de una alternativa posible. Para calcular la función de evaluación en cada nodo visitado, *f(n),* durante la aplicación del algoritmo, se deben consultar los valores *g(n) y h(n).* El primero de ellos, que representa el coste real del camino óptimo entre el nodo inicial y el nodo *n,* estará almacenado como atributo del objeto de la clase *nodo,* puesto que se ha ido acumulando durante la ejecución del siguiente modo:

$$g(n) = g(\text{nodo padre}) + \text{coste}(\text{nodo padre}, n)$$

La expresión *coste(nodo padre, n)* hace referencia al coste de la arista que conecta al nodo *n* con el nodo almacenado como antecesor suyo.

El segundo de ellos, que es la estimación del coste del mejor camino entre el nodo *n* y el nodo final, se debe obtener mediante una consulta a la base de datos donde se almacena toda la información adicional acerca del problema, de lo que se encargará la clase *base de datos.* Esta consulta requiere como entrada el par de nodos *n-nodo final,* ya que el valor de la función heurística se define entre dos nodos dados del grafo. El valor devuelto en esta consulta SQL se almacena en el objeto de la clase *nodo* al que corresponda *h(n)* y, a partir de su obtención, se podrá calcular el valor actual de la función *f(n)* según la siguiente expresión:

$$f(n) = g(n) + h(n)$$

Sin embargo, existe una ligera variación en el momento de efectuar la consulta a la base de datos cuando el nodo objetivo es un nodo simple del grafo. Como se ha

explicado anteriormente, al forzar al algoritmo a establecer un camino solución que pase por un determinado nodo de decisión, *ndd,* la forma de obtención del valor *h(n)* requerirá dos consultas: una para obtener el valor heurístico entre el nodo *n* y el nodo de decisión *ndd* y otra para extraer el valor de la función heurística del mejor camino entre el último nodo de decisión, *ndd,* y el nodo objetivo de la búsqueda. Estos cálculos son internos a la clase *base de datos,* que devolverá, tanto en este caso como en el que el nodo objetivo original ya sea un nodo de decisión, un único valor como *h(n).* Por lo que antes de devolver un valor para que el algoritmo continúe ejecutando, la clase realizará la suma de los dos valores, *h' y h,* cuando se cálculo requiera su obtención por separado.

En la Figura 3.8 el algoritmo se encuentra en la situación en la que el nodo actual es el nodo A. Éste posee tres nodos descendientes, denominados en la figura como n_1, n_2 y n_3. El algoritmo debe decidir cuál de estas tres opciones es la mejor para continuar la búsqueda, esto es, cuál de estos tres nodos parece ser el más prometedor para construir el camino solución a través de él. Suponiendo que en el momento que se representa en la Figura 3.8 no existen más nodos hoja en el árbol de ejecución del algoritmo, se deberán calcular las siguientes expresiones para comparar después sus valores:

$$f(n_1) = g(n_1) + h'_1 + h$$
$$f(n_2) = g(n_2) + h'_2 + h$$
$$f(n_3) = g(n_3) + h'_3 + h$$

Aquella función $f(n_i)$ que posea el valor más bajo indicará qué nodo, de entre todos los nodos hoja del árbol desarrollado hasta el momento, permitirá obtener un camino de coste menor que el resto de alternativas.

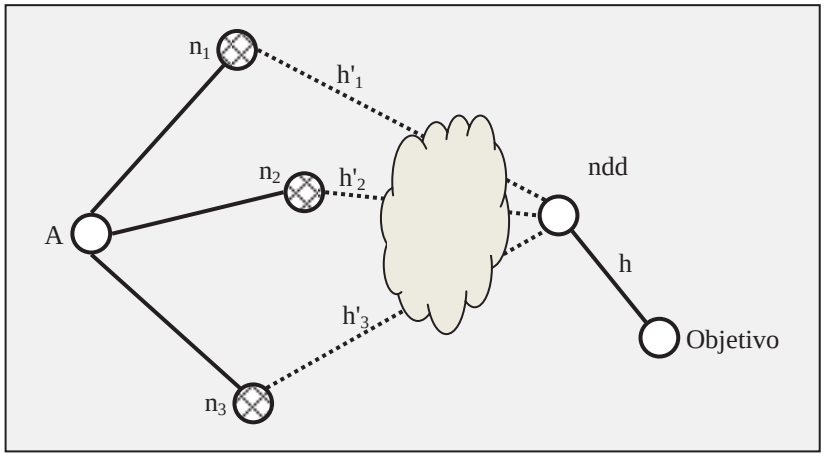


Figura 3.8 Aplicación del algoritmo A* sobre el grafo dado.

Todos los nuevos nodos encontrados se introducirán ordenados en la lista *ABIERTA*, de tal forma que los de menor coste queden en posiciones más bajas de la lista para dotarles de mayor prioridad a la hora de extraerlos para desarrollarlos. Cuando se explora un nodo que se encuentra en la lista *ABIERTA*, se traslada desde dicha lista a la lista *CERRADA* del algoritmo. El almacenamiento de los nodos antecesores de los nodos recorridos durante la ejecución no solo es necesario para calcular los costes, sino que también resulta imprescindible a la hora de obtener el camino solución. De entre todos los nodos que se encuentren en la lista *CERRADA* al finalizar la ejecución del algoritmo A* solo existirá un conjunto de ellos que permitan construir un camino desde el nodo inicial al nodo final -no es posible encontrar ciclos mediante la aplicación de este algoritmo, por lo que tampoco será posible hallar más de un camino entre los nodos inicial y final. De este modo, a partir del nodo objetivo *ndd y* a través de los nodos antecesores, se reconstruye el camino solución hasta hallar el nodo inicial.

En caso de que el nodo inicial original de la búsqueda fuera un nodo simple del grafo, se necesita añadir también a la solución devuelta el camino que conecta este nodo con el primer nodo de decisión de la búsqueda, aquel que el algoritmo A* ha tomado como su nodo inicial, *ndo.* Puesto que el nodo inicial original es un nodo simple, tan solo existirá un camino que conecte este par de nodos, por lo que no será necesario realizar ningún cálculo más. Lo mismo sucede si el nodo final original de la búsqueda es un nodo simple, es decir, dado que solo existe un camino que conecta el nodo final de la búsqueda realizada por el algoritmo, *ndd*, con el nodo objetivo original, no se requerirá la realización de ningún otro cálculo adicional. Simplemente se anexionará el camino construido con la aplicación del A* con el que conecta el nodo *ndd* con el nodo objetivo.

A partir de este momento, el sistema puede devolver el camino calculado junto con el coste asociado, que podrá tomarse como el camino de coste mínimo entre los dos nodos dados por el usuario como origen y destino del trayecto. En este camino, se mostrará la sucesión de nodos del grafo que pertenecen al camino solución, incluyendo ya en este caso los nodos simples. Es decir, si en un segmento del camino calculado por el algoritmo A* entre dos nodos de decisión consecutivos existen uno o varios nodos simples, éstos también deben estar reflejados en la solución final devuelta por el sistema (Figura 3.9 y Figura 3.10).

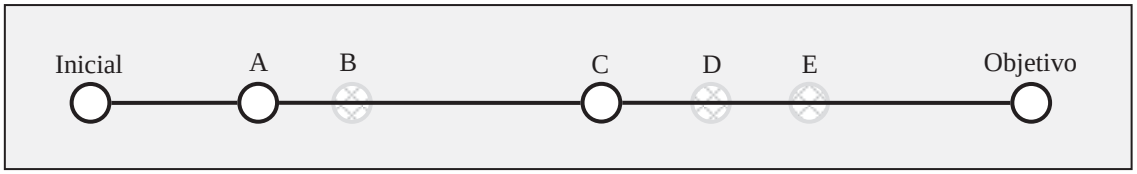


Figura 3.9 Camino devuelto tras la aplicación del algoritmo. Formado por los nodos de origen y destino y los nodos de decisión del camino óptimo: [inicial, A, C, objetivo].

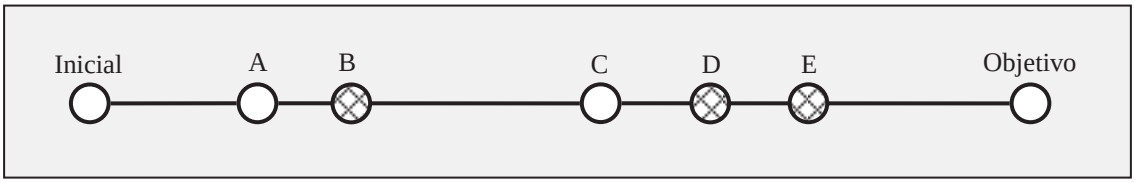


Figura 3.10 Camino devuelto por el sistema. Incluye los nodos simples del camino, además de los nodos pertenecientes al camino devuelto por el algoritmo: [inicial, A, B, C, D, E, objetivo].

La sucesión de nodos y aristas que conforman la solución del problema se muestra a través de la interfaz gráfica desarrollada. En ella se muestra el grafo completo y se resaltan aquellos nodos y aristas que pertenecen a la solución devuelta por la aplicación. De este modo, resulta fácil comprobar las características de la solución, como el hecho de que ninguna solución contendrá ningún ciclo y que conecta realmente los nodos dados por el usuario como extremos del camino. La interfaz gráfica muestra a su vez una ventana de información mediante la cual se indica al usuario el coste del mejor camino existente entre los nodos propuestos.

3.6. Usabilidad de la aplicación

Al ejecutar la aplicación, el usuario deberá seleccionar el grafo sobre el que quiere calcular algún camino. Para ello mediante el menú principal se escoge cualquiera de los grafos disponibles en la aplicación. A partir de este momento, en el que el grafo ya aparece dibujado en pantalla, el usuario podrá realizar las búsquedas deseadas seleccionando previamente un par de nodos bajo el papel de nodos inicial y objetivo. Estos nodos se podrán seleccionar a través de un listado con todos los identificadores de nodos existentes.

El usuario podrá en todo momento cancelar la búsqueda actual o esperar a su finalización para iniciar una nueva búsqueda para dos nodos diferentes sin necesidad de reiniciar la aplicación. Del mismo modo, el usuario podrá cerrar la aplicación en el momento que desee, aun estando activa la ejecución del algoritmo de búsqueda. El estado en el que se encuentra la ejecución en el momento del cierre de la aplicación no es almacenado, por lo que en caso de haber parado la ejecución del algoritmo no se continuará la siguiente vez que se inicie la aplicación.

4. RESULTADOS EXPERIMENTALES

En este proyecto se pretende aplicar la búsqueda de caminos de coste mínimo entre dos estaciones cualesquiera de la Red de Metro de Barcelona (año 2018). La Red de Metro se puede representar claramente mediante la estructura de un grafo, por lo que el sistema propuesto en el capítulo anterior permitirá resolver el cálculo de rutas de coste mínimo entre dos puntos dados de esta red de transporte.

4.1. Topología de la Red de Metro de Barcelona

Si nos ceñimos estrictamente a la Red de Metro de Barcelona ésta se compone de más de 150 estaciones, todas ellas pertenecientes, al menos, a una de las 11 líneas existentes. Esta red se representa como un grafo donde cada una de las estaciones sería un nodo y los tramos que conectan cada par de estaciones se mostrarían como aristas de dicho grafo. Los tramos que enlazan cada par de estaciones están formados por dos vías diferentes, que permiten recorrer dicho tramo en ambos sentidos. De este modo, estos tramos podrían ser vistos como dos aristas dirigidas, de sentidos opuestos, que conectan dos estaciones. Pero, dado que esta característica se cumple para todo tramo de la Red de Metro, resulta más cómodo abstraer la idea de la existencia de dos vías de sentidos opuestos y representar los tramos mediante una única arista no dirigida, transformando esa representación en otra más sencilla. La característica principal de los tramos entre estaciones, analizado como un problema de búsqueda, es su coste, que en todos ellos posee un valor positivo.

Las líneas de Metro están formadas por una sucesión de tramos consecutivos. Generalmente, no suelen tener ningún tramo en común, pero existen algunas excepciones dentro del grafo en las que dos estaciones están conectadas por dos tramos con idénticas características. En estos casos, se trata como si únicamente hubiera una arista que conectara los dos puntos del grafo, puesto que para la aplicación del algoritmo de búsqueda solo es necesario conocer los extremos de cada arista y su coste, ambas características comunes en esta excepción. La única característica que distingue una arista de otra en este caso es su pertenencia a una línea de Metro. Para ello, el algoritmo siempre continuará su recorrido a través de la arista —o sección de línea— que pertenezca a la misma línea empleada por el algoritmo para recorrer el tramo anterior.

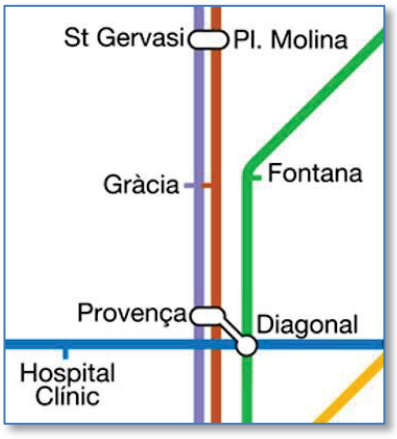


Figura 4.1 Tramo de Metro con arista doble.

Por ejemplo, en un momento determinado, la ejecución del algoritmo de búsqueda se encuentra en la estación de *Gràcia* y el antecesor de este nodo del grafo en la búsqueda es la estación de *St Gervasi*. Este par de estaciones está conectado únicamente por un tramo de la línea 6 de Metro (color violeta). Si el siguiente paso a dar por el algoritmo indica que se debe seguir hacia la estación de *Provença*, se deberá recorrer un tramo en el que existe una arista doble, tal como se puede apreciar en la Figura 4.1. Este tramo contiene una arista perteneciente a la línea 6 (color violeta) y otro perteneciente a la línea 7 (color marrón), por lo que el algoritmo decidirá, antes de avanzar hasta la estación de *Provença*, qué línea atravesará para lograr su cometido. Puesto que la línea que estaba recorriendo en el último tramo (aquel existente entre *St Gervasi* y *Gràcia*) era la línea 6, el algoritmo continuará a través de la misma línea en la siguiente sección del recorrido, evitando realizar de este modo un transbordo de trenes. En general, siempre que sea posible continuar el camino por la misma línea en la que se encuentra el algoritmo actualmente —siempre y cuando se mantenga el camino de coste mínimo y se construya entre las estaciones de origen y destino dadas—, no se realizará ningún transbordo.

La forma en la que se conectan las distintas líneas de Metro provoca la existencia de ciclos en el grafo que lo representa. Si bien no existen ciclos formados por un único nodo, como podría suceder en un grafo común, sí que aparecen numerosos ciclos construidos gracias a la comunicación entre líneas de Metro. Debido a esta característica de la red, este problema no podrá representarse como un árbol e implicará que entre dos estaciones cualesquiera del Metro pueda existir más de un camino que las conecte.

4.1.1. Transbordos

A su vez, las líneas de Metro se conectan entre sí mediante ciertas estaciones denominadas estaciones de transbordo. Únicamente en este tipo de estaciones es posible la realización de un cambio de línea, mientras que el resto de estaciones de la red pertenecen solamente a una de las líneas de Metro. Las estaciones de transbordo son prácticamente equivalentes a los nodos de decisión que se explicaron en el capítulo anterior. De este modo, el algoritmo de búsqueda solo tomará decisiones en este tipo de estaciones, ya que en aquellas estaciones que solo pertenecen a una línea de Metro (nodos simples) únicamente existe la posibilidad de continuar la búsqueda hacia la siguiente estación de la línea por la que se circula. En la Red de Metro empleada existen 28 estaciones de transbordo, por lo que las otras 123 estaciones son simples. Sin embargo, existe una ligera diferencia entre las estaciones de transbordo y los nodos de decisión explicados en el capítulo anterior. Como se muestra en la Figura 4.2 la estación de Metro de *Zona Universitària* es una estación de transbordo, puesto que conecta las líneas 3 y 9 de la red, pero por el contrario este nodo en el grafo tan solo posee grado 2, algo que no coincide con la definición dada para los nodos de decisión. Éste es el único caso en el que una estación de transbordo no se ajusta a la definición de nodo de decisión, por lo que se considerará una excepción pero se tratará del mismo modo que cualquier otra estación de transbordo de la red.

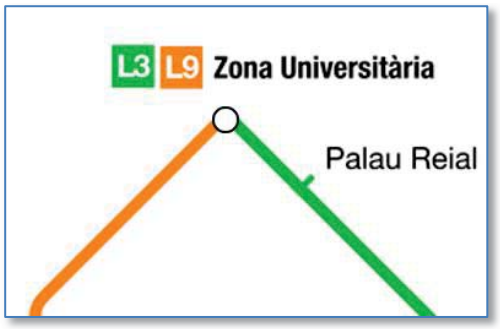


Figura 4.2 *Zona Universitária* es una estación de trasbordo que no se ajusta a la definición de nodo de decisión.

En este punto se destaca la importancia que posee el hecho de que el algoritmo solo tenga en consideración los nodos de decisión del grafo para calcular la ruta de menor coste hasta el destino. A pesar del gran tamaño de este grafo, el número de nodos que se emplean durante la ejecución es considerablemente menor al total, por lo que la eficiencia del algoritmo se ve mejorada con respecto a la aplicación común de éste. Como se ha visto en el capítulo de *Desarrollo del sistema*, aunque en los nodos simples no se tomen decisiones, el algoritmo no puede prever este aspecto, por lo que analizaría del mismo modo cada uno de los nodos del grafo, ya sean simples o de decisión, elevando tanto el tiempo de ejecución y la complejidad que se podría llegar incluso a la

no terminación del algoritmo. En este ejemplo de aplicación se puede comprender la importancia de la aportación de conocimiento adicional a la definición del problema, haciendo distinciones entre los nodos del grafo.

Además, existe un coste adicional que debe ser tenido en cuenta por el algoritmo empleado: el coste de la realización de transbordos. Los cambios de línea durante un trayecto a través de la Red de Metro implican un coste de espera que podría marcar la diferencia entre dos caminos solución del problema, hasta tal punto que sea preferible evitar la realización de un trasbordo que suponga una larga espera a favor de un camino con mayor número de estaciones. Como se ha indicado anteriormente, se ha decidido que el algoritmo escoja, como norma general, mantenerse en la línea actual por la que está construyendo el camino solución, a no ser que, a pesar de la realización de un transbordo, el coste del camino que atraviesa un cambio de línea sea inferior, esto es, que efectuar un trasbordo sea más prometedor para alcanzar la solución que continuar por la misma línea por la que se circula.

Puesto que no todos los transbordos cumplen las mismas características, ha sido necesario realizar una pequeña clasificación de éstos para poder tratarlos como corresponde:

- *Transbordos cortos:* son aquellos que conllevan el tiempo medio de realización. El cambio de línea siempre se efectúa dentro de la misma estación y se considera un transbordo corto porque los andenes correspondientes a las diferentes líneas están próximos entre sí, por lo que un usuario del transporte no gastará demasiado tiempo en cambiar de línea. El tiempo medio de realización de un transbordo se ha establecido en cinco minutos, en base a los tiempos más comunes que implican realizar esta acción. Se tratan como un nodo de decisión cualquiera, pero se debe añadir al coste total aquel correspondiente a la realización de un transbordo común. Es decir, al coste acumulado hasta la estación de *Universitat*, en el caso de la figura Figura 4.3, se sumará el coste estipulado para un transbordo corto, cinco minutos, en este proyecto. También deberá tenerse en cuenta este coste de realización de transbordo a la hora de escoger cuál es el nodo sucesor más prometedor para continuar la búsqueda, esto es, no es suficiente con sumar su coste una vez el algoritmo ha avanzado hasta la estación de transbordo, sino que tiene que decidir previamente si ésa es la acción más conveniente.

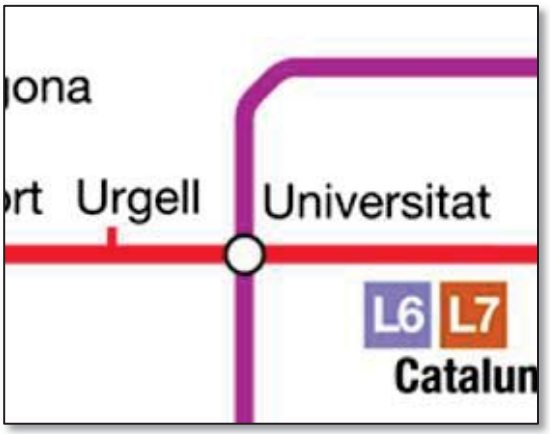


Figura 4.3 Estación con transbordo corto en *Universitat*.

- *Transbordos largos:* los andenes pertenecientes a las líneas entre las que se desea realizar un transbordo no están próximos entre sí, por lo que el usuario pasará más tiempo que en el caso anterior desplazándose de un andén a otro. Estos transbordos también se efectúan dentro de una misma estación de Metro. Al igual que en el caso anterior, este tipo de transbordos se tratarán teniendo en cuenta el coste adicional que supone efectuar un cambio de línea en estaciones de este tipo. La única diferencia con respecto a los transbordos cortos es que estos transbordos tienen un coste propio, nunca inferior al tiempo medio de un transbordo simple. Para conocer el coste concreto de los transbordos largos se debe consultar una tabla concreta en la base de datos denominada *transbordosEspeciales* y en la que figura cuál es el coste del transbordo dependiendo del par de líneas implicadas.

Figura 4.4 Estación con transbordo largo en *Catalunya* (representada como un nodo por cada línea conectada, aunque en realidad se comporte como un nodo único).

- *Transbordos con cambio de estación:* en este tipo de transbordos el usuario del transporte se desplaza por sí mismo entre un par de estaciones para realizar el cambio de línea. Un ejemplo de este supuesto es el transbordo existente entre las estaciones de *Provença* y *Diagonal*. A efectos de la búsqueda del camino de coste mínimo equivale a una arista más del grafo, por lo que se tratará como si el transbordo en sí conformara una línea de Metro por sí mismo, puesto que su comportamiento es el mismo desde el punto de vista del algoritmo. La diferencia radica en el coste, que será mayor que en el caso de poder recorrer este tramo en tren. Este tipo de transbordos se representan en la base de datos con una tabla para cada transbordo similar a las del resto de líneas de la red de Metro, ya que su funcionamiento es el mismo. Así, el sistema lo verá como una línea más de Metro, pero aplicando el coste particular de este tramo, que habrá sido almacenado ya en la base de datos atendiendo a la imposibilidad de hacerlo mediante un tren.

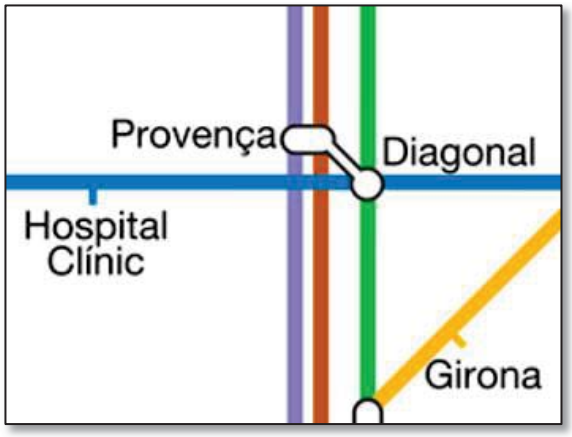


Figura 4.5 Transbordo con cambio de estación entre las estaciones de Provença y Diagonal.

Algunas estaciones de transbordo del Metro combinan varios tipos de los explicados anteriormente, es decir, podría ocurrir que dentro de una misma estación existan transbordos cortos y largos. Éste es el caso de la estación de *La Sagrera*, en la que, dependiendo del par de líneas entre las que se realice el cambio, el coste del transbordo podrá variar. Por este motivo, siempre que en una estación de Metro exista un transbordo largo se almacenan en la tabla todos los costes asociados a las distintas combinaciones de pares de líneas que confluyan en dicha estación, ya representen un transbordo largo o corto. Después, simplemente con consultar el coste según la estación y el par de líneas asociado, se obtendrá el coste concreto del cambio de línea que se está realizando.

Por otro lado, este coste de espera se ve influenciado por las condiciones horarias en las que se realice el trayecto, ya que no todos los horarios ni todos los días

poseen la misma frecuencia de trenes. Esto es, a menor frecuencia de trenes por las estaciones, como el que podría darse en días no laborables u horarios de poca afluencia de viajeros, mayor será el tiempo de espera en las mismas, incrementándose de este modo el tiempo medio de recorrido. Como consecuencia de ello, podría verse modificado el camino devuelto por la aplicación dependiendo de las condiciones en las que se efectúe el recorrido. Si dos caminos distintos, atravesando uno de ellos un transbordo y el otro un mayor número de estaciones, poseen una mínima diferencia de coste que hace preferir el transbordo ante la otra posibilidad, podría suceder que, cambiando las condiciones del viaje por otras en las que haya menor frecuencia de trenes, se acabe por descartar el camino que contiene el transbordo porque ha aumentado lo suficiente su coste como para superar el del otro camino. La forma en que afecta estas condiciones en la implementación de la aplicación se centra en el coste medio de realización del transbordo. Si existe una menor frecuencia de trenes en un momento determinado, el tiempo medio de espera al realizar un cambio de línea aumenta un valor estipulado previamente —y siempre el mismo incremento en todos los transbordos realizados durante la búsqueda. Por el contrario, si se concretan unas condiciones de viaje que implican una mayor frecuencia de trenes, como viajar en las franjas horarias de entrada y salida de los trabajos y en días laborables, el tiempo medio que conlleva un transbordo se reduce. Esta disminución también se realiza en base a un valor escogido previamente e igual para todos los transbordos que se efectúen en la misma búsqueda.

4.1.2. Costes de las aristas

Uno de los aspectos más importantes en cualquier problema de búsqueda es el coste que posee cada una de las aristas en el grafo sobre el que se ejecuta el algoritmo. Puesto que los resultados se esta aplicación se han obtenido a partir de un problema real, es necesario que los costes asignados a las aristas tengan un valor en el que se consideren las características concretas de este problema para alcanzar de este modo caminos que coincidan con la realidad.

En la Red de Metro de Barcelona los tramos existentes entre cada par de estaciones poseen longitudes distintas, por lo que la idea de dotar a todas las aristas con el mismo valor (por ejemplo, 1) queda descartada. Lógicamente, puesto que los tramos son recorridos por un objetivo físico, no sería posible de ningún modo que tramos más cortos fuesen recorridos en el mismo tiempo que tramos más largos si la velocidad de ese objeto es constante. Es decir, podría producirse un error en el cálculo del camino de coste mínimo aristas más largas son consideradas bajo el mismo coste que aquellas que

son más cortas y, por tanto, podrían ser, en principio, más convenientes para la solución del problema.

La opción utilizada para realizar la asignación de costes a las aristas del grafo será la longitud del tramo, atendiendo a los inconvenientes de la opción anterior. Para la elaboración de este proyecto fin de carrera se ha conseguido obtener la longitud real de cada tramo de la red, de forma que se puede calcular, a partir de una velocidad media por línea, el tiempo entre cada par de estaciones.

Debido a las características de la estructura de la red, algunos trenes pueden circular más rápido en unas líneas que en otras, aspecto que afecta directamente al tiempo de espera del tren en cada estación desde que realiza su parada hasta que inicia de nuevo la marcha. De esta forma, cada línea de Metro tiene asociado un coste adicional que representa el tiempo que permanece parado el tren en cada estación de esa línea. Cuando el algoritmo de búsqueda empleado traza una sección del camino entre dos nodos de decisión, *A y B,* es probable que dentro de dicha sección existan uno o varios nodos simples. Bajo el punto de vista de la búsqueda en la red de Metro, en esas estaciones simples se han realizado paradas, por lo que se debe tener en cuenta ese coste adicional en el coste total de recorrer esa sección del grafo. En los valores almacenados en la base de datos, los costes de parada en estaciones simples que se encuentran entre estaciones de transbordo ya han sido tomados en cuenta. Sin embargo, cuando uno de los extremos de la sección es una estación simple —cuando la estación de origen o la de destino son estaciones simples— no se ha almacenado este coste completo[4], por lo que se deberá añadir el coste de parada por cada estación simple que se encuentre dentro de la sección recorrida.

4.1.3. Particularidades de la búsqueda

En función de la división de la Red de Metro en un conjunto de líneas, es posible aplicar una optimización a la búsqueda para que la complejidad de la aplicación del algoritmo se vea reducida. En un grafo cualquiera no es posible conocer cuántos nodos de decisión quedan por analizar desde el nodo actual en el que se encuentra la ejecución y el nodo objetivo. Sin embargo, la existencia de diferentes líneas de Metro permite conocer cuántos nodos de decisión es necesario analizar para alcanzar el nodo meta. En el grafo que representa el plano de Metro, los nodos de decisión se corresponden con las estaciones de transbordo, como se ha concretado a lo largo de este capítulo, por lo tanto,

[4] En la base de datos solo se almacenan los costes entre estaciones de transbordo, que son las empleadas por el algoritmo de búsqueda. Por otro lado, por cada línea de Metro se almacenan los costes de las estaciones al origen de la línea. Se ha realizado de este modo para reducir el tamaño de la base de datos y almacenar únicamente la información imprescindible.

la cuestión que se trata en esta sección se reduce a prever si desde una estación de transbordo actual será necesario hacer un transbordo hasta la estación de destino. Para ello, el sistema se sirve de las diferentes líneas de Metro existentes, esto es, si la estación de transbordo en la que se encuentra la ejecución del algoritmo en un momento determinado se sitúa sobre la misma línea de Metro que la estación destino, se puede asegurar que no será necesario realizar ningún transbordo en iteraciones posteriores para alcanzar la meta. En este punto, se considera que el algoritmo ya no debe tomar ninguna otra decisión para obtener el mejor camino, por lo que dejará de ejecutar y se devolverá como solución la ruta resultante de concatenar el camino trazado por el algoritmo y el que conecte la estación de transbordo actual y la estación de destino. Por lo tanto, el algoritmo de búsqueda ejecuta mientras no se alcance la última estación de transbordo del recorrido o hasta situarse sobre la línea de Metro que contenga tanto a la estación actual como a la de destino.

4.1.4. Variabilidad de la red

La Red de Metro de Barcelona es una red de transporte público que se ve modificada continuamente como consecuencia de la adición de nuevas estaciones o líneas de Metro, supresión temporal de tramos por reformas, etc. En este caso cobra mayor importancia la necesidad de calcular la ruta de coste mínimo cada vez que se ejecuta una nueva búsqueda, puesto que si se almacenaran de manera fija las rutas entre cada par de estaciones, como se comprobó en el capítulo anterior, cualquier modificación en el grafo no sería reflejada y podría provocar errores en los caminos devueltos como solución. Únicamente ejecutando el algoritmo cada vez que se inicia una nueva búsqueda se puede obtener el camino de coste mínimo para el grafo del problema, sea cual sea la modificación que se haya aplicado sobre él.

Como consecuencia de la adaptabilidad de la aplicación, se podrán calcular las rutas de coste mínimo en cualquier plano de Metro u otro grafo, siempre y cuando se haya almacenado previamente la información necesaria y en el formato adecuado en la base de datos con la que se trabaja.

4.2. Cálculo de caminos en la Red de Metro

A continuación se muestran algunos de los resultados obtenidos más representativos de la aplicación de las búsquedas de caminos de coste mínimo realizadas con el sistema propuesto en este proyecto fin de carrera. En estos resultados se ven reflejados aquellos aspectos sobre los que poseen mayor influencia las particularidades de la Red de Metro sobre la ejecución del algoritmo de búsqueda implementado.

Todos los resultados se muestran mediante una captura de pantalla de la ventana de la aplicación sobre la que se observa la evolución de la ejecución de la búsqueda para los parámetros introducidos por el usuario del sistema. Estos parámetros se habrán seleccionado mediante un listado de elementos (estaciones de origen y destino, franja horaria o día de la semana en el que se realiza el viaje).

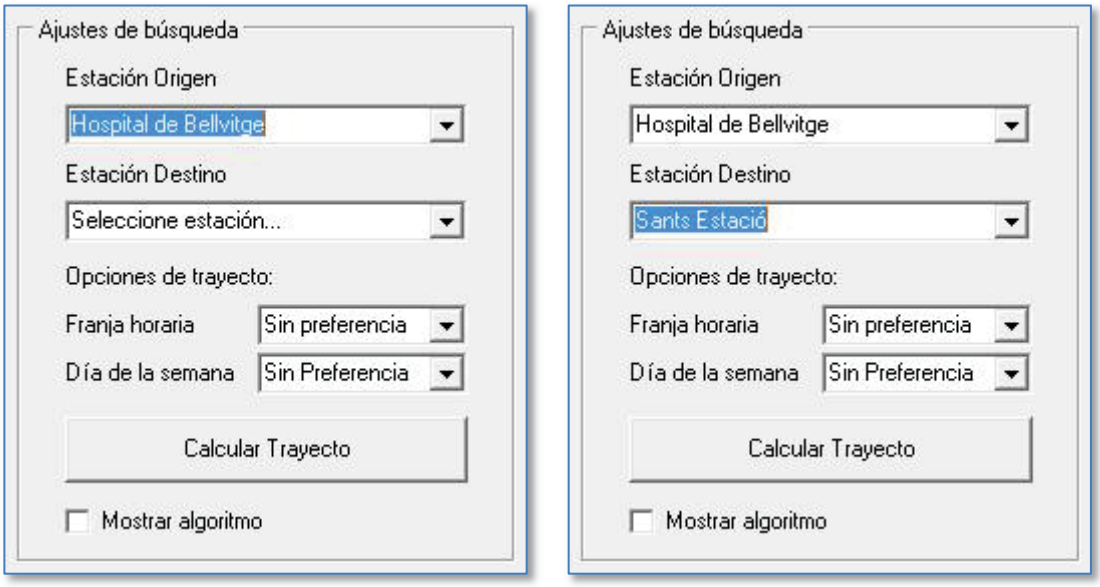


Figura 4.6 Selección de parámetros de la búsqueda.

Existen varios aspectos básicos que deben comprobarse para analizar el funcionamiento de la herramienta desarrollada. Se debe comprobar, entre ellos, que la aplicación realmente optimiza la búsqueda evitando el estudio de nodos del grafo que no aportan una información relevante a la búsqueda, como es el caso de los nodos simples, puesto que en éstos el algoritmo no tendrá que tomar ninguna decisión. Sin embargo, en caso de que alguno de los extremos del camino sea una estación simple, la aplicación deberá demostrar que se ha construido correctamente el camino hasta alcanzar la primera estación de transbordo. En general, los casos de prueba que se han considerado son los siguientes:

- **Camino de coste mínimo entre dos estaciones de transbordo:** es el caso de prueba básico. El algoritmo A* comenzará a ejecutarse desde el primer momento y hasta el nodo objetivo de la búsqueda, puesto que ambos extremos son *visibles* en la ejecución del algoritmo.

- **Camino de coste mínimo entre una estación de origen simple y una estación destino de transbordo:** la aplicación deberá construir un camino inicial hasta la primera estación de transbordo y después comenzar la ejecución del algoritmo A* hasta el nodo objetivo.
- **Camino de coste mínimo entre una estación de origen de transbordo y una estación destino simple:** es similar al caso anterior, sin embargo, éste modifica el nodo objetivo que considera el algoritmo de búsqueda y, además, deberá construir la función heurística a partir de dos valores.
- **Camino de coste mínimo entre dos estaciones simples:** conjuga en un mismo caso de prueba las dos particularidades anteriores, ejecutándose el algoritmo A* tan solo entre la primera y la última estaciones de transbordo del camino y concatenando los caminos de lo conectan a los extremos de la búsqueda.
- **Camino de coste mínimo entre estaciones de la misma línea de Metro:** según la política escogida en este trabajo, si el camino construido hasta el momento alcanza la línea de Metro sobre la que se sitúa la estación de destino, el algoritmo finaliza su ejecución y continúa de manera directa por la misma línea hasta alcanzar la estación meta. Este caso comprueba el correcto cumplimiento de esta característica, evitando la realización de transbordos adicionales.
- **Camino de coste mínimo entre dos estaciones en una franja horaria de mayor frecuencia de trenes:** a partir de dos estaciones de transbordo del mapa, se calcula el coste de la ruta realizada durante un intervalo horario en el cual existe una mayor frecuencia de trenes. Esto afecta directamente al coste acarreado en la realización de transbordos durante el recorrido.
- **Camino de coste mínimo entre dos estaciones en un día no laborable:** bajo las condiciones de este caso de prueba existe una menor frecuencia de trenes de la Red de Metro, lo que influye de manera directa a la realización de transbordos existentes en el camino construido.

4.2.1. Camino de coste mínimo entre dos estaciones de transbordo

El caso de prueba básico debe comprobar el buen funcionamiento de la aplicación devolviendo un camino válido entre las dos estaciones seleccionadas por el usuario indicando el coste asociado a dicho camino. Además, deberán haberse tenido en cuenta los costes relacionados con la realización de transbordos durante el recorrido y

aquellos relacionados con la espera en las paradas de los trenes en las estaciones simples intermedias.

Captura 4.1 Búsqueda del camino de coste mínimo entre dos estaciones de transbordo.

En la Captura 4.1 se muestra el camino óptimo calculado en la ventana emergente. La ruta calculada entre las estaciones de transbordo de *Passeig de Gràcia* y *La Sagrera* recorre las siguientes estaciones de la Red de Metro:

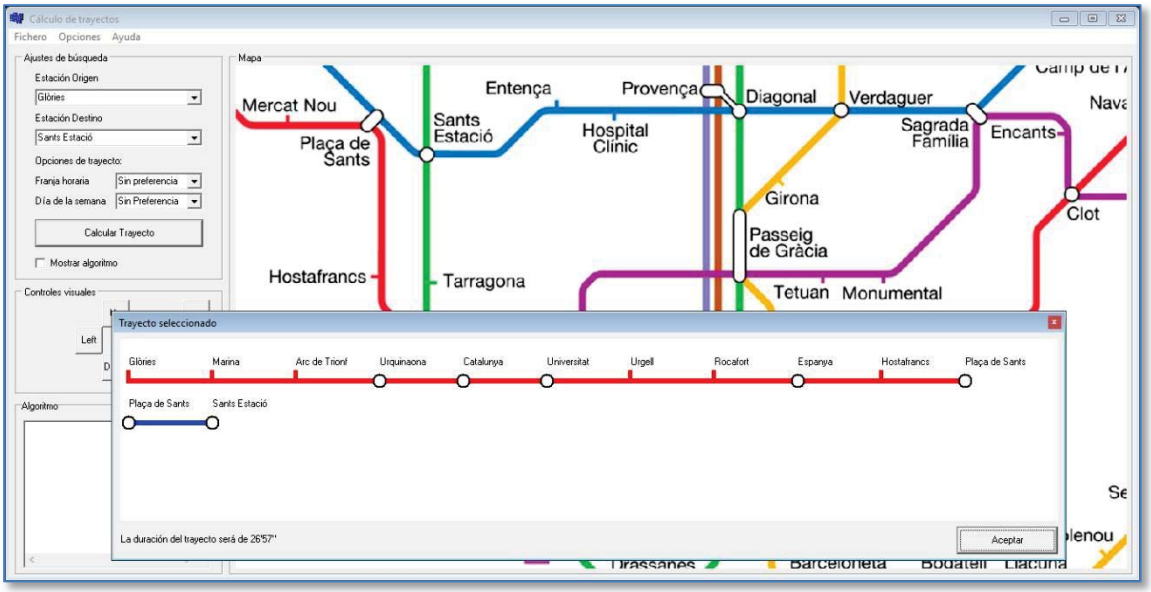
(Línea 2) Passeig de Gràcia - Tetuan - Monumental - Sagrada Familia
(Línea 5) Sagrada Familia - Sant Pau / Dos de Maig - Camp de l'Arpa - La Sagrera

El coste total de la ruta aparece al finalizar la búsqueda en un nueva ventana. En este caso, el coste total de la ruta es de 20 minutos y 13 segundos, evaluados desde que el usuario comienza su viaje hasta que abandona el último tren en la estación de destino. Como se ha indicado anteriormente, este coste incluye el correspondiente a los transbordos realizados durante el trayecto, como el que se ha efectuado en la estación de Sagrada Familia.

4.2.2. Camino de coste mínimo entre una estación de origen simple y una estación destino de transbordo

En este caso de prueba, el algoritmo de búsqueda A* no comienza a evaluar el grafo hasta encontrarse en una estación de transbordo, por lo que deberá comprobarse si

el cálculo de la ruta que conecta la estación de origen original de la búsqueda y la primera estación de transbordo del recorrido es correcta. En todo momento de la búsqueda la estación de destino es la misma, pero durante la ejecución de esta búsqueda se analiza cuál de las estaciones de transbordo adyacentes a la estación de origen es la que permite obtener un camino de menor coste hasta el destino.



Captura 4.2 Búsqueda del camino de coste mínimo a partir de una estación de origen simple.

Según se observa en la Captura 4.2, las dos estaciones de transbordo adyacentes a la estación de *Glòries* son *Clot* y *Urquinaona*. A partir de estas dos últimas estaciones se calcula cuál posee un valor inferior de la función de evaluación, con el valor del coste real acumulado desde la estación de origen, *g(n), y* el valor heurístico hasta el destino, *h(n)*. En este caso, la estación que aportaba un valor inferior de esta función es *Urquinaona*, por lo que el resto del camino de coste mínimo hasta el destino es calculado por el algoritmo A* desde *Urquinaona* hasta *Sants Estació* de la siguiente manera:

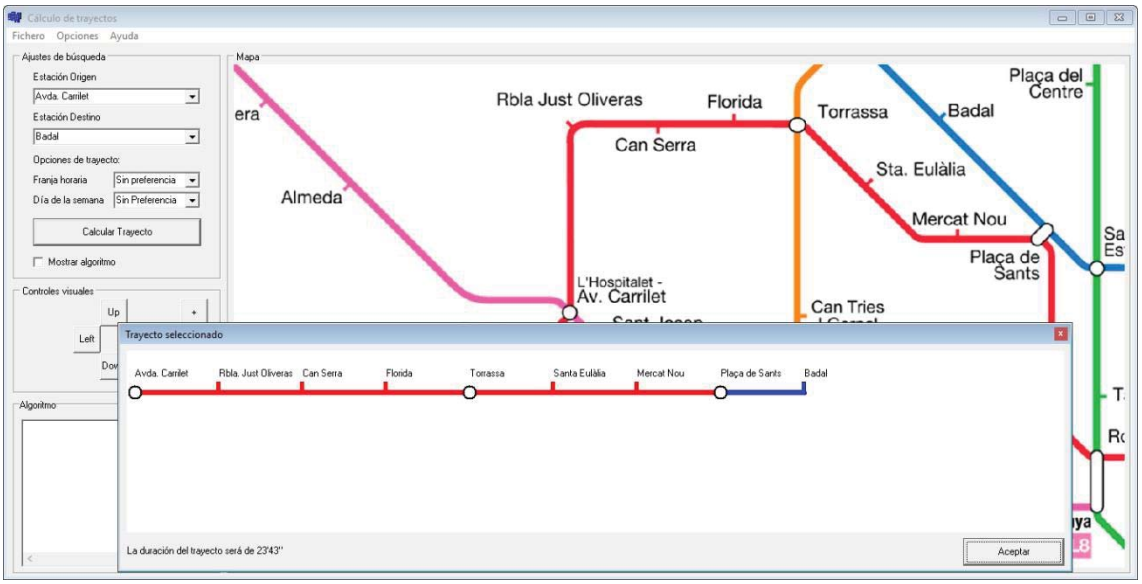
(Línea 1) Glòries - Marina - Arc de Triomf - Urquinaona - Catalunya - Universitat - Urgell - Rocafort - Hostafrancs - Plaça de Sants
(Línea 5) Plaça de Sants - Sants Estació

El coste total del recorrido, incluyendo el transbordo realizado en la estación de *Plaça de Sants*, es de 26 minutos y 57 segundos, tal y como aparece en la ventana emergente de la aplicación.

4.2.3. Camino de coste mínimo entre una estación de origen de transbordo y una estación destino simple

Este caso de prueba es similar al anterior en cuanto a la realización de una concatenación de caminos para obtener la ruta solución completa. Sin embargo, aquí es la estación de destino la que se ve modificada para poder aplicar el algoritmo de búsqueda A*. Para ello, se escoge cuál de las estaciones de transbordo adyacentes a la estación de destino dada por el usuario otorga un menor coste al camino si éste atraviesa cada una de estas estaciones. La función de evaluación para las estaciones adyacentes al destino se calcula de forma distinta del caso anterior, puesto que en este caso no se posee información acerca del coste real acumulado desde la estación de origen hasta cada una de ellas. Ese coste real, *g(n)*, debe ser sustituido por un valor heurístico, al que después se añadirá el valor heurístico hasta el destino para completar la función de evaluación necesaria para escoger cuál de las dos estaciones deberá contener el camino solución.

Como muestra de este caso de prueba se ha ejecutado la aplicación para calcular el camino de coste mínimo entre las estaciones de *Avda. Carrilet* y *Badal*. Las estaciones de transbordo pertenecientes a la línea 5 (línea de Metro a la que pertenece la estación *Badal*) que más cerca están de la estación de destino son *Collblanc* y *Plaça de Sants*. Entre estas dos estaciones se escoge *Plaça de Sants*, por aportar un menor coste al total del camino entre las estaciones de origen y destino.



Captura 4.3 Búsqueda del camino de coste mínimo con estación de destino simple.

La ruta obtenida, una vez se ha escogido cuál de las estaciones de transbordo más cercanas al destino es la más prometedora, contiene las siguientes estaciones:

(Línea 1) Avda. Carrilet - Rbla. Just Oliveras - Can Serra - Florida - Torrasa - Santa Eulàlia - Mercat Nou - Plaça de Sants
(Linea 5) Plaça de Sants - Badal

El coste de este camino es de 23 minutos y 43 segundos, incluyendo el transbordo en *Plaça de Sants* realizado durante el recorrido.

4.2.4. Camino de coste mínimo entre dos estaciones simples

Durante la ejecución de este caso de prueba deben combinarse los aspectos anteriores, esto es, el de averiguar cuál de las dos estaciones de transbordo más cercanas al origen aporta un mejor coste al camino solución y el de obtener la estación de transbordo más cercana al destino que cumpla la misma condición. En este caso, por tanto, se deben realizar cuatro comprobaciones, una por cada combinación de posibilidades. Una vez que se obtiene el par de estaciones de transbordo que simulan ser las estaciones de origen y destino para el algoritmo A*, se comienza la ejecución de este algoritmo de búsqueda. El resultado debe concatenar tres segmentos de camino:

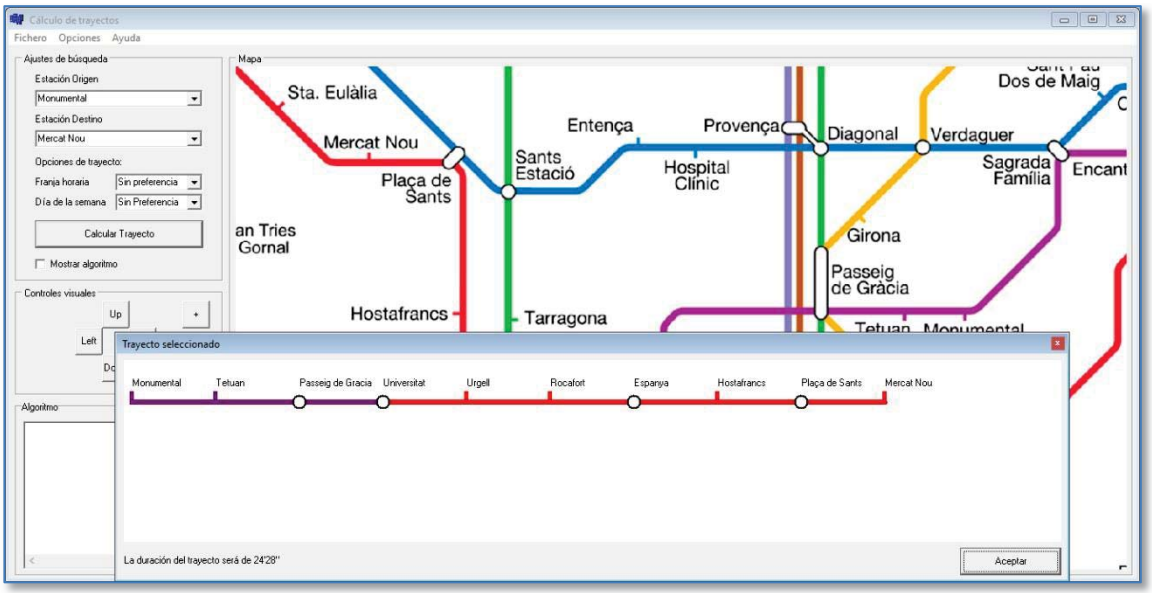
- El camino desde la estación de origen dada por el usuario hasta la primera estación de transbordo del recorrido.
- El camino entre la primera estación de transbordo hasta la última estación de transbordo del recorrido (resultante de la aplicación del algoritmo A*).
- El camino desde la última estación de transbordo del camino solución hasta la estación de destino, dada por el usuario.

En la Captura 4.4 se puede observar el resultado de la aplicación de este caso de prueba sobre la Red de Metro de Barcelona entre las estaciones de *Monumental* y *Mercat Nou*. El algoritmo es capaz de identificar cuál de las dos estaciones de transbordo más cercanas (*Plaça de Sants* y *Torrassa*) pertenece a la misma línea de Metro que la estación destino y, por lo tanto, resultará más cómoda a la hora de que el usuario realice el recorrido, tal como se ha descrito a lo largo de este capítulo.

El camino solución devuelto por este sistema entre las estaciones de Ríos Rosas y Retiro posee un coste total de 24 minutos y 28 segundos está constituido de la siguiente manera:

(Línea 2) Monumental - Tetuan - Passeig de Gràcia - Universitat

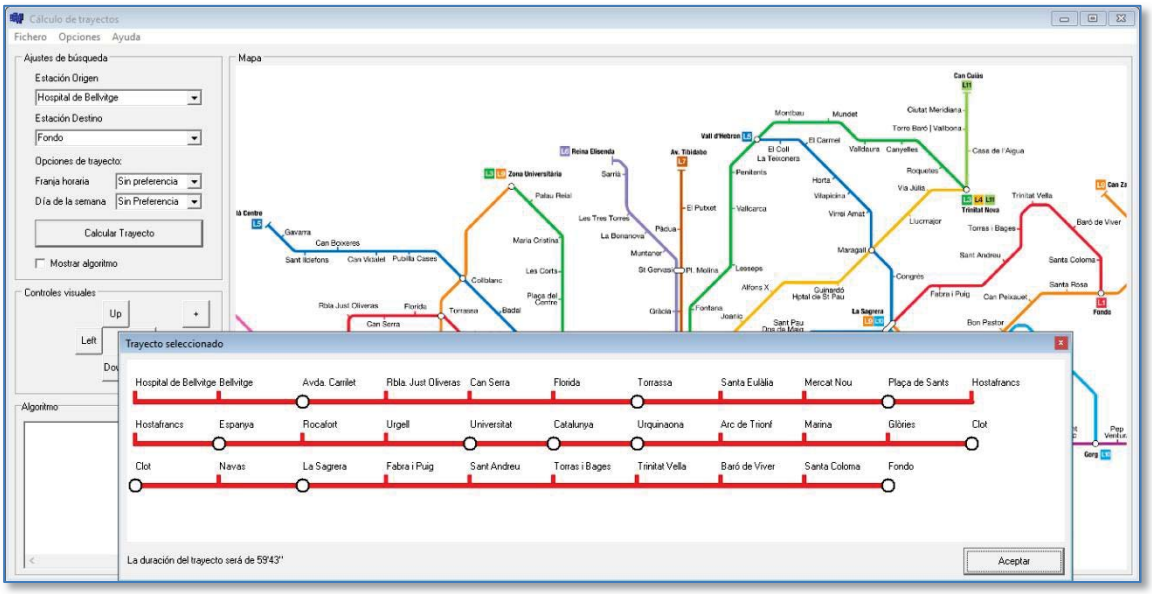
(Linea 1) Universitat - Urgell - Rocafort - Espanya - Hostafrancs - Plaça de Sants - Mercat Nou



Captura 4.4 Búsqueda del camino de coste mínimo entre dos estaciones simples.

4.2.5. Camino de coste mínimo entre estaciones de la misma línea

Como se ha mencionado anteriormente, cuando el algoritmo de búsqueda se encuentra sobre la misma línea de Metro que la estación destino, se deja de aplicar el algoritmo A* para continuar el camino hasta el final sin analizar ninguna decisión más, ya que la realización de cualquier transbordo siempre conlleva un coste adicional. De este modo, si la estación de origen de la búsqueda ya se encuentra en la misma línea que la estación de destino, no será necesaria la elección entre las posibles alternativas, por lo que el algoritmo A* no entrará en ejecución y se devolverá automáticamente la ruta que conecta las estaciones de origen y destino de la búsqueda a través de la línea de Metro común.



Captura 4.5 Búsqueda del camino de coste mínimo entre dos estaciones pertenecientes a la misma línea de Metro.

En el ejemplo de ejecución de la Captura 4.5 se puede observar que el algoritmo no abandona en ningún momento la línea a la que pertenecen ambas estaciones (*Hospital de Bellvitge* y *Fondo*), por lo que se cumple con la condición establecida en este proyecto para el cálculo de rutas.

El camino solución, que supone la ruta de menor coste, está formada como se indica a continuación:

(Línea 1) Hospital de Bellvitge - Bellvitge - Avda. Carrilet - Rbla. Just Oliveras - Can Serra - Florida - Torrassa - Santa Eulàlia - Mercat Nou - Plaça de Sants - Hostafrancs - Espanya - Rocafort - Urgell - Universitat - Catalunya - Urquinaona - Arc de Triomf - Marina - Glòries - Clot - Navas - La Sagrera - Fabra i Puig - Sant Andreu - Torras i Bages - Trinitat Vella - Baró de Viver - Santa Coloma - Fondo

La realización de esta ruta a través de la línea 1 conlleva un coste total de 59 minutos y 43 segundos.

4.2.6. Camino de coste mínimo entre dos estaciones en una franja horaria de mayor frecuencia de trenes

A ciertas horas del día, en las que existe un aumento considerable del número de usuarios de Metro de Barcelona, se establece una mayor cantidad de trenes circulando por la red. Esta situación implica que. al existir una mayor frecuencia de trenes, los viajeros esperarán menos tiempo, de media, a dichos trenes en sus respectivos andenes.

Desde el punto de vista del algoritmo de búsqueda empleado, este aspecto se traduce en una reducción de los costes asociados a la realización de transbordos durante el recorrido.

Si anteriormente se establecía en cinco minutos la penalización por realización de cada transbordo durante la construcción del camino, en una franja horaria en la que exista una mayor frecuencia de trenes esta penalización se reduce a tres minutos de media.

Las franjas horarias que se han considerado de mayor frecuencia de trenes coinciden con los intervalos horarios de entrada y salida de trabajadores, esto es, de las 7 h hasta las 9 h y de las 17 h a las 20 h. Por lo tanto, si el usuario del sistema desarrollado selecciona indicar como característica del recorrido que se realizará en franjas horarias de mayor frecuencia de trenes, el algoritmo de búsqueda deberá tener en cuenta el nuevo coste de la realización de transbordos, pudiendo resultar de este modo que una ruta calculada anteriormente sin especificar la franja horaria se vea modificada.

En la Captura 4.6 se ha calculado la ruta existente entre las estaciones de *Passeig de Gràcia* y *La Sagrera*, que se había obtenido previamente en el caso de prueba 4.2.1, para observar la diferencia en el coste total si el usuario establece que la franja horaria en la que se efectúa el viaje implica una mayor frecuencia de trenes (18 minutos y 13 segundos en el caso actual, frente a los 20 minutos y 13 segundos de la prueba anterior). El recorrido de estaciones existente en este trayecto no se ha visto modificada en este caso con respecto al caso de prueba 4.2.1.

Captura 4.6 Búsqueda del camino de coste mínimo en una franja horaria de mayor frecuencia de trenes.

4.2.7. Camino de coste mínimo entre dos estaciones en un día no laborable

Durante las jornadas no laborables, la frecuencia de trenes a lo largo de la Red de Metro se ve reducida a causa de una menor afluencia de viajeros. Esto repercute directamente a aquellos viajeros que decidan emplear este medio de transporte en días no laborables, situación que se quiere representar en este trabajo mediante la selección del domingo como día de la semana en el que el usuario de la aplicación quiere calcular la ruta de menor coste. Al existir una menor frecuencia de trenes en toda la red, los tiempos de espera aumentan cada vez que se debe emplear un nuevo tren para continuar el recorrido, esto es, la realización de transbordos bajo estas condiciones se vuelve más costosa que en los casos anteriores. Concretamente, cada transbordo realizado costará tres minutos más que en otros casos.

En la Captura 4.7 se muestra un ejemplo de ejecución en el que la búsqueda se condiciona a viajar en días no laborables y, por tanto, el coste total de la ruta calculada se verá incrementado con cada transbordo realizado durante la misma. Para poder comparar los resultados con los de ejecuciones anteriores, se evalúa también entre las estaciones de *Passeig de Gràcia* y *La Sagrera*, para poder observar de este modo cómo el coste total del camino se ve incrementado de manera proporcional al número de transbordos realizados. Al igual que en casos anteriores, el camino trazado no se ha visto modificado, pero sí su coste total, que debido a la realización de un transbordo durante el recorrido se ha incrementado hasta los 23 minutos y 13 segundos.

Captura 4.7 Búsqueda del camino de coste mínimo en días no laborables.

4.3. Comparativa de resultados

Todos los resultados obtenidos mediante la aplicación desarrollada en este proyecto de fin de carrera han sido comparados con los devueltos por el sistema elaborado para la página web de Metro de Barcelona[5] para conocer si los caminos trazados por esta aplicación se correspondían con estos últimos.

[5] Búsquedas realizadas en la página web del Metro de Barcelona: https://www.tmb.cat/es/home

5. CONCLUSIONES Y LINEAS FUTURAS

Este proyecto propone el desarrollo de una aplicación que permita el cálculo de caminos de coste mínimo sobre un grafo de grandes dimensiones y de manera en que el tiempo necesario para su construcción no sea excesivo. Mediante la aportación de mayor conocimiento a la ejecución del algoritmo, como es la clasificación de los nodos del grafo en nodos de *decisión y* nodos *simples,* es posible realizar la búsqueda del camino de coste mínimo con mayor eficiencia que otros algoritmos de búsqueda en los que se estudian todos y cada uno de los nodos del grafo para obtener el camino solución. La diferencia con el tiempo requerido por otros algoritmos de búsqueda es más notable cuanto más distantes están los nodos en el grafo del problema, puesto que el árbol de búsqueda generado aumenta exponencialmente su tamaño con cada paso realizado durante la ejecución. Si además, la proporción de nodos de decisión con respecto a la totalidad de nodos es baja, el número de nodos del grafo que son *visibles* al algoritmo de búsqueda empleado también es reducido, del mismo modo que el espacio de búsqueda del problema se ve disminuido.

Por lo tanto, en el caso de que el grafo del problema estuviera constituido únicamente por nodos de decisión, se tardaría lo mismo en calcular el camino de coste mínimo mediante el método empleado en esta aplicación que mediante el resto de algoritmos de búsqueda conocidos. Es decir, el tiempo empleado en resolver el problema es, a lo sumo, el mismo que el necesario para resolverlo mediante cualquier otro algoritmo de búsqueda existente. En la mayoría de los casos, cuando exista al menos un nodo en el grafo que no sea de decisión, el tiempo de ejecución será menor que el que conlleva resolver el mismo caso de prueba con otros algoritmos de búsqueda.

Los resultados obtenidos con el sistema desarrollado, aplicado a la Red de Metro de Barcelona, se han podido comparar con los caminos calculados por la herramienta perteneciente a la propia empresa de Metro. Los caminos resultantes son coincidentes en la mayoría de los casos, difiriendo en aquellos que se ven afectados por la política de continuidad sobre la línea de la estación de destino una vez que se llega a ella. Esto es, en algunos casos, la solución dada por Metro de Barcelona conlleva la realización de más transbordos a partir del momento en el que el algoritmo de búsqueda ya se sitúa sobre la línea de destino, mientras que en el sistema propuesto no se realizan más decisiones durante la ejecución si se ha alcanzado esta línea. De este modo, se consigue reducir aún más el tiempo de cálculo del camino de coste mínimo, puesto que el conocimiento adicional aportado provoca que el algoritmo de búsqueda no deba tomar tantas decisiones durante su ejecución como otros algoritmos, incluido el empleado por Metro de Barcelona.

Existen algunas diferencias más llamativas entre los resultados obtenidos mediante la experimentación y los mostrados por Metro de Barcelona en cuanto al coste de las rutas calculadas como solución de las búsquedas realizadas, siendo estos caminos idénticos. Esto se debe a la falta de una información completa acerca de los costes entre las estaciones de Metro, lo que ha obligado a realizar ciertas aproximaciones en los cálculos, de modo que en algunas ocasiones el error con respecto a las soluciones mostradas por la aplicación de Metro de Barcelona se ha acumulado provocando una diferencia mayor en el coste total del camino calculado. Sin embargo, los caminos calculados sí son caminos de coste mínimo teniendo en cuenta los costes, ligeramente mayores, de los tramos de Metro.

Al igual que la existencia de la clasificación de los nodos ayuda a reducir el tiempo de ejecución del algoritmo de búsqueda, también se consigue disminuir el espacio necesario para almacenar toda la información acerca del grafo que representa al problema. Es decir, el tamaño de la base de datos empleada es menor gracias a que el algoritmo tan solo necesita, en la mayoría de los casos, la información sobre los nodos en los que se toman decisiones.

El sistema propuesto es capaz de calcular caminos de coste mínimo sobre cualquier grafo, siempre y cuando la información acerca de la estructura del mismo esté definida de la manera que se describe en el capítulo 3. Esto es, se describa el modo en el que están conectados los nodos del grafo y se aporte la información de costes necesaria para la elección de alternativas sobre los nodos de decisión.

Algunas de las clases de la implementación del sistema se han creado con aspectos relacionados con la Red de Metro, como la existencia de líneas que agrupan en conjuntos los diversos tramos o aristas del grafo. En caso de querer emplear este sistema para calcular caminos de coste mínimo en otro tipo de grafos que no constituyan este tipo red de transporte público, sería necesaria una pequeña modificación en estas clases. En otros casos no sería necesario modificar la implementación, puesto que el sistema ha sido diseñado de tal forma que pueda aplicarse a otros tipos de grafos sin tener que modificar, por ejemplo, el algoritmo de búsqueda.

Por otro lado, sería posible ampliar el área de trabajo de la aplicación de modo que se trabaje con varias bases de datos, cada una de las cuales contendría la información acerca de una red de metro. Así, al iniciar la aplicación, se seleccionaría el mapa de Metro sobre el que se desea realizar la búsqueda. Este aparecería representado en el área de dibujo de la aplicación y, a partir de ahí, se procedería de la misma manera que se ha descrito anteriormente. Para trabajar con esta ampliación tan solo sería necesario modificar la clase que se comunica directamente con la base de datos para que pueda recibir como entrada la base de datos con la que deberá conectarse.

Gracias a la modularidad del sistema, también es posible modificar el algoritmo de búsqueda empleado para calcular los caminos de coste mínimo. Puesto que toda la información acerca de la ejecución de cada paso del algoritmo se encuentra independizado del resto de la aplicación, se posible sustituir la clase completa por otra en la que se especifique algún otro algoritmo de búsqueda para comprobar las diferencias entre los diversos algoritmos o, simplemente, mejorar algunos aspectos del ya existente.

No solo modificando algunas características de la implementación de las clases se pueden obtener nuevas funcionalidades del sistema. Si se modifican los costes almacenados en la base de datos se pueden estudiar otros comportamientos de la realización de la búsqueda de caminos en grafos. Por ejemplo, si todos los costes de las aristas fueran iguales entre sí y, a su vez, de valor 1, la búsqueda de caminos de coste mínimo se reduciría a la búsqueda del camino más corto, esto es, se consideraría mejor aquel camino que poseyera un menor número de aristas.

Referencias

[1] Clarke, J. et al. ***Reformulating Software Engineering as a Search Problem***. *IEE Proceedings - Software*, Volume 150 Issue 3 (2003) 161- 175.

[2] Dijkstra, E.W. ***A Note on Two Problems in Connexion with Graphs***. (1959)

[3] Gross, J.L. & Yellen, J. ***Graph theory and its applications***, Chapman & Hall/CRC. (2006)

[4] Hart, P.E., Nilsson, N.J. & Raphael, B.. ***A Formal Basis for the Heuristic Determination of Mínimum Cost Paths***. *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, (1968) 100-107.

[5] Hogg, T., Huberman, B.A. & Williams, C.P. ***Phase transitions and the search problem***. *Artificial Inteligence* 81 (1996) l-15.

[6] Holland, J.H. ***Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence***, University of Michigan Press. (1975)

[7] Kaindl, H. & Khorsand, A. ***Memory-bounded bidirectional search.*** *En Proceedings of the twelfth national conference on Artificial intelligence vol. 2,* American Association for Artificial Intelligence, (1994) 1359-1364.

[8] Knuth, D.E., Moore, R.W. ***An Analysis of Alpha-Beta Pruning***, Defense Technical Information Center. (1975)

[9] Koenig, S. & Simmons, R.G. ***Solving Robot Navigation Problems with Initial Pose Uncertainty Using Real-time Heuristic Search***. (1998)

[10] Korf, R.E. ***Depth-First Iterative-Deepening: An Optimal Admissible Tree Search***. *Artificial Intelligence* 27, (1985) 97-109.

[11] Korf, R.E. ***Encyclopedia of Artificial Intelligence***, Wiley. (1990)

[12] Ruinar, V. ***Encyclopedia of Artificial Intelligence***, Wiley. (1990)

[13] Manzini, G. ***BIDA*: An Improved Perimeter Search Algorithm***. *Artificial Intelligence* 75 (1995) 347-360.

[14] Moore, E.F. ***The Shortest Path Through a Maze***, Bell Telephone System. (1959)

[15] Neumann, J.V. ***Zur theorie der gesellschaftsspiele***. (1928)

[16] Nilsson, N.J. ***Principies of Artificial Intelligence***, Birkháuser. (1982)

[17] Nilsson, N.J. ***Problem-Solving Methods in Artificial Intelligence***, McGraw-Hill. (1971)

[18] Palma Méndez, J.T. & Marín Morales, R. ***Inteligencia artificial:métodos, técnicas y aplicaciones***, McGraw-Hill Interamericana de España. (2008)

[19] Pearl, J. ***Heuristics: Intelligent Search Strategies for Computer Problem Solving***, Addison-Wesley Pub. Co. (1984)

[20] Pohl, L. ***Encyclopedia of Artificial Intelligence***, Wiley. (1990)

[21] Real Academia Española: Algoritmo. ***Diccionario de la Lengua Española***. (2011)

[22] Reinefeld, A. & Marsland, T.A. ***Enhanced Iterative-Deepening Search***, Fachbereich Mathematik-Informatik, Universität Gesamthochschule Paderborn. (1993)

[23] Richards, D.J. & Hart, T.P. ***The Alpha-Beta Heuristic***. (1961)

[24] Rivin, I., Vardi, I. & Zimmerman, P. ***The n-Queens Problem***. *The American Mathemaíical Monthly* 101, (1994) 629-639.

[25] Rosenbloom, P,S. ***Encyclopedia of Artificial Intelligence,*** Wiley. (1990)

[26] Russell, S. ***Efficient Memory-Bounded Search Methods***. *University of California,* (1992)

[27] Russell, S.J. & Norvig, P. ***Artificial Intelligence: a modern approach***, Prentice Hall. (2003)

[28] Stockman, G.C. ***A Minimax Algorithm Better Than Alpha-Beta?*** *Artificial Intelligence 12*, (1979) 179-196.

[29] Tarjan, R.,. ***Depth-first search and linear graph algorithms***. 12th Annual Symposium on Switching and Autómata Theory, (1971) 114-121.