Inteligencia Artificial 25/11/2012
Universidad Europea Miguel de Cervantes

Estudio de Estrategias de Búsqueda Informada y No Informada

Javier Montero Isusi

Índice

Contenido

1. Introducción

En este documento se va a tratar de explicar, con un ejemplo básico en grafos (el problema del Viajante), las propiedades de las estrategias de búsqueda, sus ventajas e inconvenientes, y una comparación entre ellas.

En el problema del viajante, una persona desea ir de una ciudad determinada a otra, también establecida. Según la estrategia de búsqueda que usemos, llegará de forma por el mejor camino posible, lo conseguirá tras haber recorrido muchas ciudades, o simplemente puede que no llegue nunca.

Un problema de este calibre en el mundo real sería mucho más complicado de tratar, ya que entran muchos más factores en el trayecto a parte de la distancia entre las ciudades (peajes, gasolineras más cercanas, posibles accidentes en las carreteras, etcétera). Como realmente para este estudio son irrelevantes **eliminaremos este problema abstrayéndolo** para poder trabajar con ello.

También es importante abstraer las acciones posibles a tomar en una ciudad, porque de nuevo no son importantes para este estudio (simplemente nos interesa **a qué ciudades podrá viajar desde esa localidad**, no nos interesa cosas como: hotel en el que se hospedará, monumentos que visita, tiempo que emplea en cada ciudad, etcétera).

Para comprender este documento es necesario explicar algunos conceptos:

- Un **estado inicial** es el estado en el que comienza un **agente** el viaje. Por ejemplo, el estado inicial para nuestro agente puede ser la ciudad de Craiova.
- Se deberá de realizar una descripción de las **acciones** que el agente pueda tomar. La formulación que utilizaremos será con una **función sucesor.**
    - Teniendo un estado particular x, ($SUCESOR - FN(x)$ devolverá un conjunto de pares compuestos que contendrán $< acción, sucesor >$, donde cada acción será una de las acciones posibles en el estado x, y cada **sucesor** será un estado que podrá alcanzarse desde x, aplicando la acción.
    - La función sucesor desde nuestro estado $En(Craiova)$ será:
        - $\{< Ir(Dobreta), En(Dobreta) >, <$
          $Ir(Rimnicu\,Vilcea), En(Rimnicu\,Vilcea) >, <$
          $Ir(Pitesti), En(Pitesti) >\}$
- Teniendo un estado inicial y una función sucesor, podremos con ello definir un **espacio de estados** que reúna todos los estados alcanzables desde el estado inicial. El espacio de estados formará un grafo, en el que los nodos son estados y los arcos entre los nodos son acciones.
- Un **camino** en el espacio de estados, será una secuencia de estados conectados por una secuencia de acciones.

- **Test objetivo:** determinará si un estado es un estado objetivo. En este caso será un condicional tipo *IF- SI* que comparará los nodos una vez sea abierto.
- **Costo del camino:** asigna a cada camino un coste numérico. **No será tenido en cuenta** cuando utilicemos estrategias de búsqueda **no** informada.
- **Costo individual:** será el coste asignado a un arco entre dos nodos en el grafo.
- **Solución** de un problema, es un camino desde el estado inicial al estado objetivo. La calidad de la solución se mide por la función costo del camino, y una **solución óptima** tiene el costo mínimo del camino entre todas las soluciones.

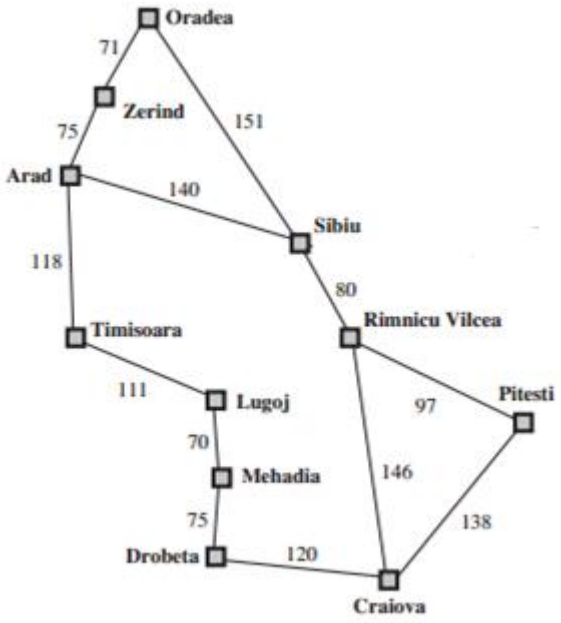
2. Problema del Viajante

Estado inicial: cualquier estado puede designarse como estado inicial. Tendremos once posibles estados iniciales.

Función sucesor: será el conjunto de ciudades posibles a las que ir desde nuestro estado actual, lo que posteriormente llamaremos *nodos hijo (una* lista donde almacenaremos todos los nodos a los que podamos acceder).

Test objetivo: comprobación de haber llegado al estado final, el estado **objetivo**.

Función de coste: será la unión de costes individuales entre el estado inicial y el estado final.



Espacio de estados del proyecto.
Mapa de carreteras de parte de Rumanía.

3. Estrategias de búsqueda implementadas

Existen múltiples estrategias de búsqueda, como clasificación tendremos:

- Estrategia de búsqueda no informadas
- Estrategia de búsqueda informadas

Se diferencian en si reciben información adicional sobre los estados o simplemente se basan en la otorgada por la enunciación del problema.

3.1 Estrategias de búsqueda no informadas

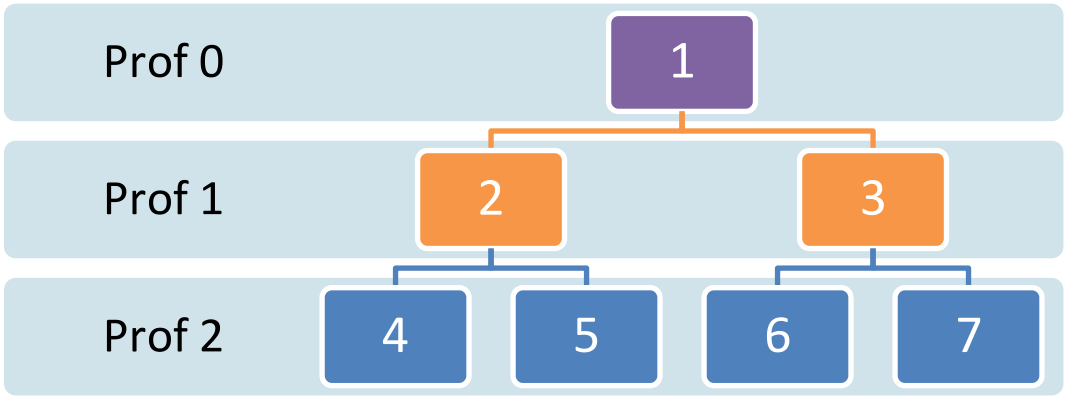
En este apartado se reúnen las estrategias de búsqueda comprendidas con el nombre de **búsqueda no informada** o **búsqueda a ciegas.** El término expresa que ellas no tienen información adicional acerca de los estados más allá de la que provee la definición del problema (su colocación en un grafo).

Todo lo que podrán hacer con esta información será generar sucesores (nodos hijo) y distinguir un estado objetivo del que no lo es.

3.1.1 Búsqueda Primero en Anchura (BPA)

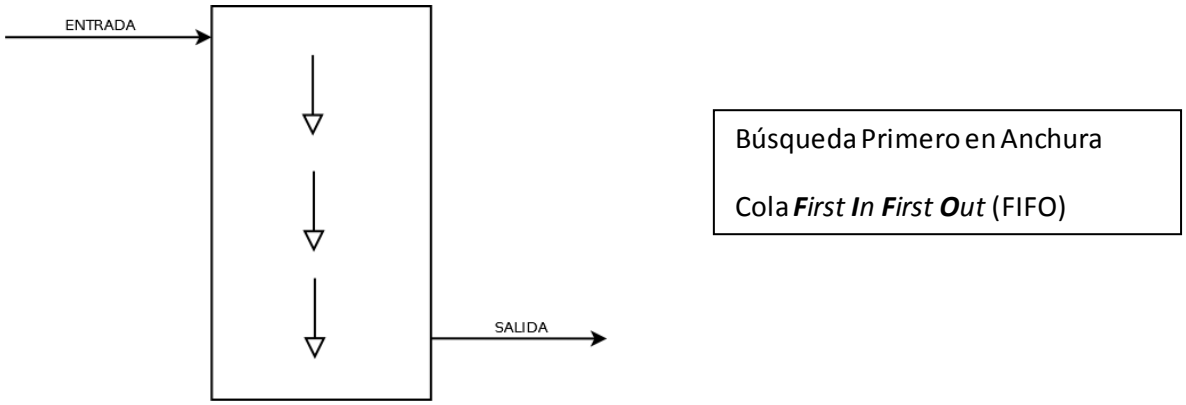
La búsqueda primero en anchura selecciona para su expansión el nodo no expandido **más superficial** del árbol de búsqueda, hasta dar con la solución. Se expanden todos los nodos de una misma profundidad antes que expandir cualquier nodo del próximo nivel.

En primera instancia se expande el nodo raíz (o nodo padre), a continuación se expanden todos los nodos hijo (sucesores) del nodo raíz, después los sucesores de los nodos hijo, etc…



Búsqueda Primero en Anchura (BPA) en un árbol binario.

Se implementará con una **cola FIFO** en la que se pondrán todos los nuevos sucesores a la cola, por lo que expandiremos aquellos que tengan menor profundidad, ya que son aquellos que entraron primero (los nodos menos profundos del árbol). En la imagen a continuación, los primeros a *salir* serán los primeros que hayan *entrado.*



Es similar a la **búsqueda de coste uniforme** (la cual no está expuesta en el documento), pero esta expande el nodo con el costo más pequeño del camino $g(n)$. Es completo y óptimo si el costo de cada paso excede de una cota positiva ϵ.

Propiedades de la Estrategia de Búsqueda

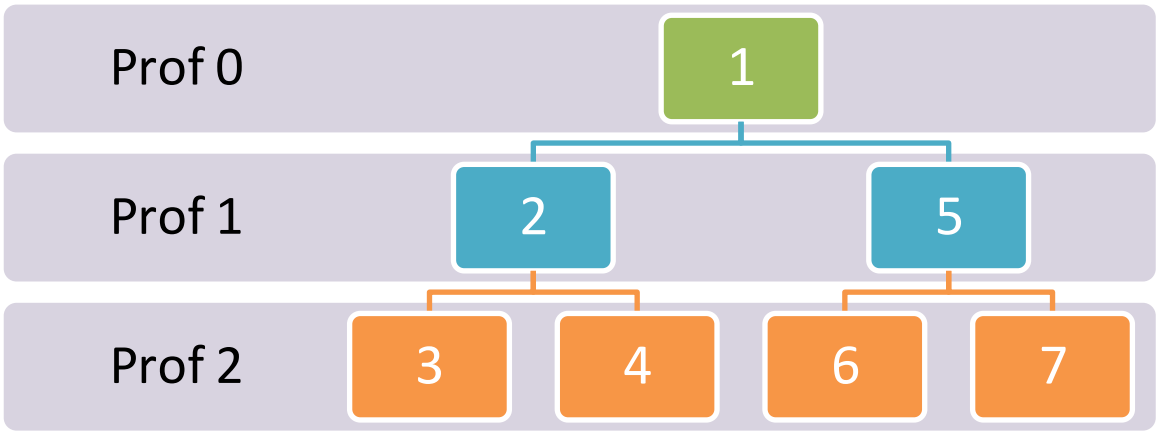
Estrategia	Completo	Óptimo	Complejidad Temporal	Complejidad Espacial
BPA	Sí	Sí	$O(b^d)$	$O(b^d)$

3.1.2 Búsqueda Primero en Profundidad (BPP)

La búsqueda primero en profundidad siempre expandirá el nodo más profundo en la frontera actual del árbol de búsqueda. La búsqueda procede inmediatamente al nivel más profundo del árbol de búsqueda. Cuando estos nodos son expandidos, **son quitados de la frontera**, en nuestro caso, de Abiertos. De esta forma *retrocederemos* al no haber encontrado la solución hasta el siguiente nodo que todavía tenga sucesores inexplorados.

Para implementar esta estrategia de búsqueda utilizaremos una cola **LIFO**, una pila, dónde los nodos últimos nodos agregados a la cola serán los primeros que expandirá en su futuro más próximo.

Tiene unos requisitos de memoria bastante menores de memoria en comparación a la estrategia de búsqueda primero en anchura, al no necesitar almacenar todos los nodos del árbol.



Búsqueda Primero en Profundidad (BPP) en un árbol binario.

El inconveniente que tiene la búsqueda primero en profundidad, es que se puede hacer una elección equivocada, y obtener un camino muy largo, o infinito. Por eso **cuando tengamos una rama infinita la estrategia de búsqueda no será ni completa ni óptima**. Sin ramas infinitas será completa y óptima, teniendo un mejor rendimiento que la Estrategia primero en anchura.

Estrategia	Completa	Óptima	Complejidad Temporal	Complejidad Espacial
BPP	No	No	$O(b^m)$	$O(b * m)$

3.1.3 Búsqueda Primero en Profundidad Limitada (BPL)

Esta estrategia de búsqueda se asemeja mucho a la Búsqueda Primero en Profundidad con la diferencia de que en esta, **limitamos la profundidad máxima**. Haciendo esto, lograremos evitar ramas infinitas, la mayor desventaja del BPP.

Aunque no todo son ventajas ya que el algoritmo se vuelve aún *más incompleto*, debido a que puede que escoja el camino adecuado y vaya explorando nodos, pero debido al **límite** que hemos impuesto de profundidad no consiga obtener una solución que en el caso de la Búsqueda Primero en Profundidad no hubiera habido ningún problema y **hubiera llegado**.

Técnicamente en el espacio de estados con el que trabajamos, se puede decir que el rendimiento del BPL será siempre menor o igual al que tenga el BPP, esto es debido a que no nos está aportando ninguna propiedad positiva y que serán todo desventajas respecto al BPP.

Tiene en la BPL mucha importancia el factor de ramificación, y la profundidad máxima que tenga nuestro árbol, es por ello que *limitando* mucho la profundidad (haciendo que sea muy superficial) no consigamos que llegue a la solución.

Es por ello conveniente que cuando evaluemos esta estrategia se haga en función del límite que hayamos impuesto antes de ejecutar el algoritmo. En el apartado 5.2.3 se trata esta cuestión.

Estrategia	Completo	Óptimo	Complejidad Temporal	Complejidad Espacial
BPL	No	No	$O(b^l)$	$O(b*l)$

3.2 Estrategias de Búsqueda Informadas

En las anteriores Estrategias de búsqueda podíamos decir que estas recorrían el árbol ciegamente, simplemente podía acceder a un nodo, expandirlo y comprobar si era la solución para a continuación continuar con los demás nodos abiertos. Para estas estrategias de búsqueda será necesario explicar qué es la función **Heurística** que nos dará una *referencia* en el camino**.**

Se le define como $h(n)$, que es el coste estimado del camino más corto desde el nodo n a un nodo objetivo. En el ejemplo de las ciudades rumanas, para poder estimar el coste del camino más barato entre dos ciudades, tendríamos que decir que el coste más barato sería hacer el recorrido en línea recta.

La función heurística busca transmitir un conocimiento adicional del problema al algoritmo de búsqueda.

3.2.1 Búsqueda voraz Primero el Mejor (BPM)

La búsqueda voraz o como indico, búsqueda primero el mejor realiza en cada iteración del bucle un intento de acercarse al nodo objetivo, con la opinión de que este sea el que nos conduzca a él por el camino óptimo. Evalúa los nodos utilizando la función heurística únicamente: $f(n) = h(n)$.

Tomaremos las distancias entre las ciudades **en línea recta**, en este trabajo se ha hecho calculando con una regla métrica la distancia que hay entre todas las ciudades.

Como busca siempre en los hijos de un nodo expandido, el nodo que está más cercano a la ciudad, **no nos dará siempre la solución óptima** , ya que puede que al expandir este nodo veamos que se alejan del estado final o que toma un camino que al final haga que sea más largo del que podíamos haber tomado con otro nodo, que en un principio fuese más lejano en primera instancia, pero que al calcular la distancia teniendo en cuenta a sus hijos viéramos que esa sí que fuera la solución óptima, de ahí lo de algoritmo voraz , o avaro, que en cada paso trata de ponerse tan cerca del objetivo como sea posible). Explicación continuada en el 5.2.4

Tampoco es completo, ya que se asemeja al de búsqueda primero en profundidad en el modo que prefiere seguir un camino hacia el objetivo, pero volverá atrás cuando llegue a un callejón sin salida. Esto nos da que puede haber una rama infinita con lo que nunca llegaríamos al objetivo. También puede dar que por la orientación de las ciudades, nunca lleguemos al objetivo, ya que puede ser que el nodo padre esté más cercano que los hijos por lo que dejará de expandir nodos.

La complejidad temporal y espacial en el peor de los casos será $O(b^m)$, dónde m es la profundidad máxima del espacio de búsqueda. Dependerá mucho del problema en particular y de la calidad de la heurística.

Algoritmo	Completo	Óptimo	Complejidad Temporal	Complejidad Espacial
BPM	No	No	$O(b^m)$	$O(b^m)$

4. Lenguaje de programación escogido

El lenguaje en el que se implementarán las estrategias de búsqueda será Java.

Las ventajas que podremos aprovechar en este proyecto serán:

- Mayor experiencia en la resolución de problemas con este lenguaje de programación.
- Es complicado que se puedan cometer errores de pérdida de memoria.

4.1 Implementación de la práctica

Para este proyecto se han tenido que crear varias clases abstrayendo las propiedades teóricas que tenemos tanto en las estrategias de búsqueda como en el tratamiento de árboles y nodos.

Propiedades recogidas de las estrategias de búsqueda

- Búsqueda Primero en Anchura
  - Implementada la solución ABIERTOS-CERRADOS para evitar repeticiones.
  - Cola FIFO ← Selecciona el nodo no expandido más superficial en el árbol de búsqueda.
  - Implementado como un ArrayList.
- Búsqueda Primero en Profundidad

    - Implementada solución CERRADOS para evitar repeticiones. Para representar ABIERTOS maneja el *Array* de Nodos abiertos.
    - Pila (LIFO) ← Selecciona para la expansión el nodo no expandido más profundo en el árbol de búsqueda.
    - Implementado como una Pila
- Búsqueda Primero en Profundidad Limitada
    - De forma idéntica a la Búsqueda Primero en Profundidad
    - Se impone un límite de profundidad a un BPP
- Búsqueda Primero el Mejor
    - Implementada la solución ABIERTOS-CERRADOS para evitar repeticiones.
    - Implementada la heurística para buscar el camino más corto
    - Implementado con un ArrayList donde se almacenan los nodos.

Ha sido fundamental el reactualizar la información de los hijos debido a que la información que un nodo tiene es quienes son sus nodos hijos, pero estos no contienen ninguna información, por ello se les debe de actualizar cada vez que accedamos a nuevos nodos (en cada iteración).

La solución y el número de nodos se recoge en cada iteración como un Array de enteros, posteriormente se vuelve a la clase Main donde se hace el tratamiento de: solución encontrada y número de nodos abiertos. La ruta es mostrada por pantalla.

Problemas en la implementación

Un caso que se nos puede dar en el uso de ABIERTOS y CERRADOS es la repetición de nodos en abiertos. En este proyecto se ha buscado seguir ese objetivo por los requisitos que tenía este proyecto. Como ejemplo tenemos que se *ignore* un nodo debido a que ya le añadimos a abiertos anteriormente, haciendo de él una rama, con esto tendremos que si el algoritmo de BPP profundiza en una rama **no se encontrará a ese nodo anteriormente añadido a ABIERTOS** por lo que se tendría que acabar toda la rama para poder acceder a ese nodo. Un ejemplo es, un viaje de SIBIU a ZERIND.

SIBIU-RIMNICUVILCEA-CRAIOVA-DROBETA-MEHADIA-LUGOJ-**TIMISOARA**-PITESTI-**ARAD**-ZERIND

Desde **TIMISOARA** tendría que haber accedido a **ARAD** directamente, pero como **ARAD** ya estaba abierto, terminó esa rama. Posteriormente acude a la última rama abierta (ajena a la que llegó a **TIMISOARA**), que era PITESTI, cuando expandió PITESTI llegó al final de esa rama ya que PITESTI era un nodo hoja (sin más hijos), al final llega a la que tenía **ARAD**, y de ahí llega a **ZERIND**

4.1.1 Pseudocódigos implementados

En este apartado se recogen los pseudocódigos que recogen el funcionamiento de las distintas estrategias de búsqueda implementadas.

Búsqueda Primero en Anchura (BPA)

```
Abiertos ← (nO) /*El primer nodo por expandir es el nodo padre)
      Cerrados ← () /* No hay nodos visitados aún
      Mientras Abiertos != NULL
            Añadimos n a Cerrados /*N pasa a considerarse visitado
            Eliminamos nodo de Abiertos(n) /*Pop de abiertos
            Si n es meta devolver solución
            Expandimos n colocando sus hijos en Abiertos
```

```
            Eliminar de Abiertos cualquier nodo que este en cerrados
            Si abiertos != NULL n <- abiertos (0)
      Return ERROR ( abiertos esta vacío)
```

Búsqueda Primero en Profundidad (BPP)

```
      Pila (NO lleno) /*El primer nodo por expandir es el nodo padre)
      pila.push(nodo de inicio)
      Cerrados () /* No hay nodos visitados aun
      Mientras pila != NULL{
      Eliminamos nodo de Abiertos(n) //Pop de la pila
      Añadimos n a Cerrados /*N pasa a considerarse visitado
      Si n es meta // Nodo final
            Devolver solución
      Expandimos n colocando sus hijos en la pila //Utilizando PUSH
      Eliminar de pila cualquier nodo que este en cerrados
      Eliminar de pila cualquier nodo repetido
      Si Pila != NULL
      n = pila(tamano_pila - 1) // El -1 se debe a que en Java el tamaño cuenta desde 1.
      Si no
      Devolver error
}
Return ERROR //No deberíamos llegar aquí
```

Búsqueda Primero en Profundidad Limitada (BPL)

```
      Pila (NO lleno) /*El primer nodo por expandir es el nodo padre)
      pila.push(nodo de inicio)
      Cerrados () /* No hay nodos visitados aun
      Mientras pila != NULL{
            Eliminamos nodo de Abiertos(n) //Pop de la pila
            Añadimos n a Cerrados /*N pasa a considerarse visitado
            Si n es meta // Nodo final
                  Devolver solución
            Si la nodo. Profundidad < limite
                        Expandimos n colocando sus hijos en la pila //Utilizando PUSH
            Eliminar de pila cualquier nodo que este en cerrados
            Eliminar de pila cualquier nodo repetido
            Si Pila != NULL
            n = pila(tamano_pila - 1) // El -1 se debe a que en Java el tamaño cuenta
desde 1.
            Si no
            Devolver error
      }
      Return ERROR //No deberíamos llegar aquí
```

Búsqueda de Primero el Mejor (BPM)

```
      1. nodo_actual ← nodo_inicio
      2. coste_del_camino ← 0
      3. CERRADOS = ()
      4. ABIERTOS = nodo_actual
      5. BUCLE
            SI ABIERTOS ← VACIO, RETURN ERROR
            Añadimos Nodo_actual a cerrados
            Si nodo_actual es meta devolver SOLUCION CON TODA LA RUTA
            Añadimos todos los hijos de Nodo_actual a ABIERTOS
            Eliminación de nodos repetidos(en cerrados) en ABIERTOS
            nodo_actual =n /* Sera el nodo con menos heurística, sacado tras un bucle*/
            Si ABIERTOS <- VACIO o NODO ANTERIOR = N DEVOLVER ERROR
```

4.1.2 Ejecución práctica

Para probar la aplicación realizada acceda con su consola (cmd) al directorio de la práctica (practica1inteligenciaArtificial) y utilice el siguiente comando:

java –jar Practica1.jar

5. Exposición y razonamiento de los resultados

5.1 Exposición de resultados

En este apartado revisaremos los datos aportados por la aplicación realizada, **sin tener en cuenta el tiempo que ha tardado en ejecutarse** ya que no es un dato *relevante.* Primero porque normalmente las limitaciones que tendremos serán de memoria, por lo que esto no nos afectará en la mayoría de los casos y segundo porque depende del equipo dónde se ejecute la aplicación. Las pruebas de ejecución han sido realizadas en un portátil con un Intel Centrino Duo 2x2.00 Gb, 2 Gb RAM, utilizando el SO Linux, Distribución Debian Wheezy (Testing) y tercero, porque depende de la forma que hayamos implementado las estrategias de búsqueda.

Para cincuenta (50) ejecuciones tendremos, con los nodos origen y destino **aleatorios**, estos datos como resultado:

	BPA	**BPP**	**BPL**	**BPM**
Soluciones Óptimas	50	50	50**	46
Nodos abiertos	334*	318*	322*	198*
Nodos ab por ej.	6	6	6	3
Tiempo empleado	12	97	125	64

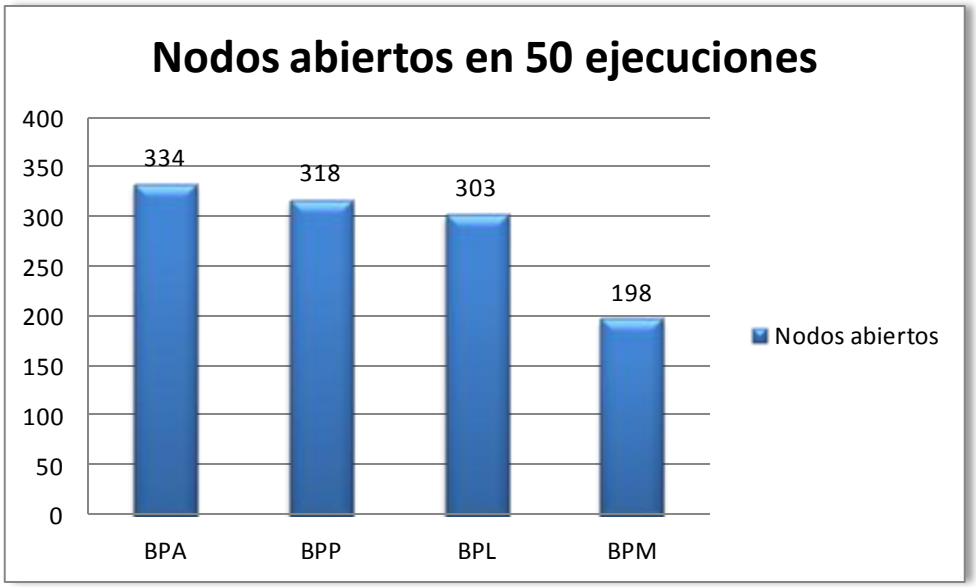
*Es importante recordar que, al ser aleatorios existe una desviación al desconocer las ciudades origen y destino en cada ejecución, y de forma fortuita pueden repetirse muchos casos dónde las ciudades ORIGEN-DESTINO sean las mismas, o que sea un nodo hijo del Nodo origen.

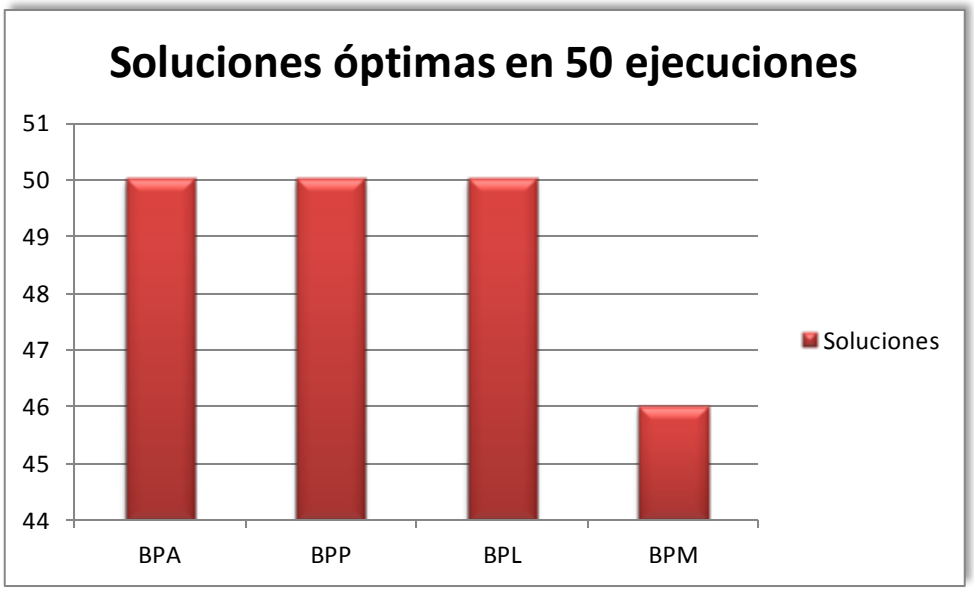
**Para casos dónde la profundidad límite es 5

Comparativa gráfica:

BPA= **Búsqueda Primero en Anchura** BPP= **Búsqueda Primero en Profundidad**

BPM= **Búsqueda Primero el Mejor.** BPL= **Búsqueda Primero en Profundidad**





5.2 Razonamiento de los resultados

Tras haber recogido los datos resultantes de las ejecuciones de las diferentes estrategias de búsqueda procedo a comentar si los resultados son los esperados.

5.2.1 Búsqueda Primero en Anchura

Partiendo del conocimiento que hemos adquirido sobre las complejidades de esta estrategia de búsqueda, podemos ver que sí que cumple con lo esperado.

- Es el algoritmo que más nodos abre.
    - Su explicación reside en su forma de resolución del problema, que emplea mucho tiempo en buscar la solución explorando todos los nodos ordenándolos en profundidad.
- Alcanza la solución óptima
    - Al explorar todos los nodos está garantizado que alcance la solución. Llegaría a la solución óptima **siempre** ya que independientemente haya una solución o múltiples, el algoritmo realizado invariablemente recogerá la solución **más superficial**.

5.2.2 Búsqueda Primero en Profundidad

- Abre un menor número de nodos que el BPA
    - Debido a que se hacen pruebas con múltiples casos esto es lo normal, ya que en el caso de que un árbol tuviese una solución fuese muy superficial y el BPP no la recogiese en primera instancia, el BPA tendría una menor complejidad temporal. El BPA siempre recorrerá todos los nodos por lo que normalmente tendrá más complejidad espacial.

- Alcanza la solución óptima
    - Ante la **no existencia de ramas infinitas** tendremos siempre la solución al alcance por lo que nunca habría incidencias y en este caso sí que sería óptimo y completo.

5.2.3 Búsqueda según Profundidad (BPL)

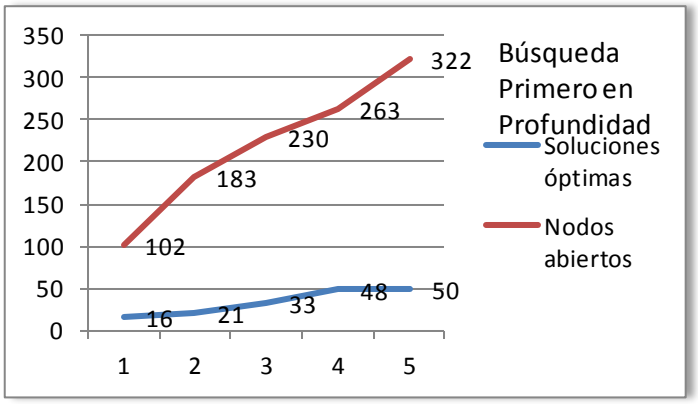
Tras haber expuesto previamente los datos comparándolos con las demás estrategias de búsqueda vimos sus datos son similares a los de una Búsqueda Primero en Profundidad (BPP) , tras ya haber comentado las propiedades del BPL en el apartado 3.1.3 , tratamos en este apartado de buscar la relación entre profundidad y soluciones encontradas. Esto está **íntimamente relacionado con el árbol** que tengamos. En la tabla expuesta en este apartado es posible percibir que la proporción soluciones—nodos se mantiene, aunque algo distorsionado por la desviación ya comentada.

Profundidad	Soluciones óptimas	% incremento soluciones	Nodos abiertos	% incremento nodos
1	16	100%	102	100%
2	21	179%	183	131%
3	33	225%	230	206%
4	48	258%	263	300%
5	50	316%	322	313%

Es por ello que no podemos garantizar en términos de eficacia (hablando en este caso de eficacia como llegar al nodo solución/final, cumplir para lo que ha sido diseñado) un buen resultado. En este grafo – árbol se dará:

$$Eficacia\ BPL \leq Eficacia\ BPP$$

Esto se debe a que en este proyecto **no hay casos en los que se aprovechen las ventajas del BPL** (evasión de las ramas infinitas), y simplemente el BPL no está aportando en su ejecución nada positivo en términos de eficacia y eficiencia.



Aunque en este estudio todas las comparativas entre estrategias de búsqueda se han realizado sobre 50 ejecuciones, para probar realmente el porcentaje de soluciones que recoge el BPL con un nivel determinado de profundidad, se probado con un 1.000.000 de ejecuciones, simplemente de forma experimental. Y los resultados son los siguientes:

Ejecuciones	Profundidad	Soluciones óptimas	% Sol.	Nodos abiertos	Nodos medios	Tiempo ejecutado(Min)
1000000	5	983520	98,35%	6069871	6,069871	7,61
1000000	6	983482	98,35%	6213838	6,213838	7,66
1000000	7	975194	97,52%	6383103	6,383103	7,40
1000000	8	991843	99,18%	6478646	6,478646	7,70
1000000	9	1000000	100%	6491545	6,491545	8,21

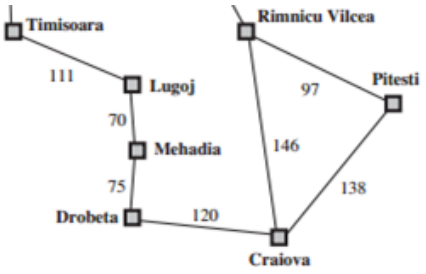
Podemos ver que un 98,35% de las veces el caminante pasa por más de cinco ciudades las cuales son nodos hijos de otras. En el caso que tuviésemos **ramas infinitas** sería adecuado hacer un estudio de este tipo (teniendo en cuenta la desviación) para saber en qué profundidad nos interesa poner el límite para llegar a alcanzar un porcentaje de soluciones óptimas aceptable de acuerdo a nuestras condiciones, de nuevo, habrá que tener en cuenta cómo es el árbol, su factor de ramificación, número de nodos, etcétera.

5.2.4 Búsqueda Primero el Mejor (BPM)

- Abre muy pocos nodos
    - Aproximadamente un 40% de nodos menos que el BPA, BPP y BPL. Esto es debido a que tenemos una buena heurística y que por tanto, por su forma de buscar la solución, será el número de nodos mínimo que recorramos de media para alcanzar la solución, aunque hay que tener en cuenta los casos excepcionales, cuando no alcanzamos las soluciones.
- No siempre alcanza la solución
    - Debido al diseño del grafo, existen viajes en los que, hay ciudades más cercanas al nodo objetivo que las ciudades circundantes, es por ello que se queda en un estado

bloqueado y finaliza sin encontrar la solución. El caso más claro es el viaje de LUGOJ a RIMNICU VILCEA. Como podemos ver en la imagen, LUGOJ está más cercano a RIMNICU VILCEA que TIMISOARA o MEHADIA, por lo que **siempre** que realicemos un viaje entre estas ciudades, no encontrará la solución.

- Esto es debido a que trabajamos con la heurística.



6. Conclusiones

La realización de este proyecto se realizó leyendo los apuntes de Inteligencia Artificial y como material complementario fue utilizado el libro detallado en la bibliografía (Inteligencia Artificial: Un Enfoque Moderno). En una primera fase se realizó una fase de análisis de los requisitos, posteriormente se investigó sobre las estrategias de búsqueda más propicias para ser implementadas. Posteriormente se fue realizando la práctica en Java y esta memoria de forma simultánea.

Este proyecto puede aportar conocimiento a cualquiera que desee iniciarse en el mundo de las Estrategias de Búsqueda, conociendo sus propiedades y cual nos pueda interesar más según los requisitos del problema dónde nos interese aplicarla. Estas estrategias de búsqueda nos sirven como una secuencia de acciones encontradas por un agente para resolver un problema complejo, dónde ninguna acción simple pudiera resolverla.

Esto nos podrá en el futuro llegar a manejar cualquier estado posible en muchos entornos, como es el ajedrez dónde con un par de movimientos seremos capaces de saber si esta partida estará perdida o por el contrario vamos a ganarla. Aún parece quedar lejos, pero llegará el día en que los computadores puedan almacenar todos los estados posibles y pueda haber un algoritmo suficientemente eficiente como para poder responder en un instante muy corto de tiempo. Es sólo un ejemplo pero las estrategias de búsqueda son muy útiles en campos como: planificación de la estrategia militar, inteligencia artificial, robótica, juegos y compra/venta de activos financieros

7. Bibliografía

1. Apuntes tomados en la asignatura Inteligencia Artificial impartida por Oscar J. Prieto
2. RUSSELL,S.J ; NORVIG, P. INTELIGENCIA ARTIFICIAL UN ENFOQUE MODERNO