

# Problem-solving Agents

Alejandro Figueroa

# Problem-solving Agents

- Pensemos en un agente de vacaciones en la ciudad de Arad, Romania.
- El desempeño de este agente es multi-dimensional:
  - Mejorar su rumano
  - Visitar los sitios importantes
  - Disfrutar la vida nocturna
  - Evitar resacas
  - Etc....
- El problema de decisión es complejo e involucra muchos trade-offs y la lectura cuidadosa de las guías turísticas.

# Problem-solving Agents

- Supongamos además que este agente tiene el objetivo de ir a Bucarest al día siguiente, ya que tiene un ticket de avión no-reembolsable.
- Lo que haría que el objetivo de este agente es llegar a Bucarest antes que salga el avión.
- El objetivo del agente es alcanzar el objetivo y decidir como actuar ahora y en el futuro para lograrlo.
- La “formulación del problema” es el proceso de decisión qué acciones y estados hay que considerar para conseguir el objetivo.

# Problem-solving Agents

- Para el caso de nuestro consideremos que el agente hace movimientos de un pueblo grande a otro. Cada estado corresponde a estar en una ciudad en particular.
- Tres caminos salen de Arad: hacia Sibiu, Timisoara y el otro a Zerind. Ninguno llega al objetivo, salvo que el agente conozca la geografía de Romania. Con un mapa, el agente puede organizar su viaje através de las tres ciudades hacia Bucarest.
- En general, un agente con variadas opciones inmediatas de valor incierto puede decidir qué hacer mediante examinando primero acciones futuras que eventualmente conducen a estados de valor conocido.

# Problem-solving Agents

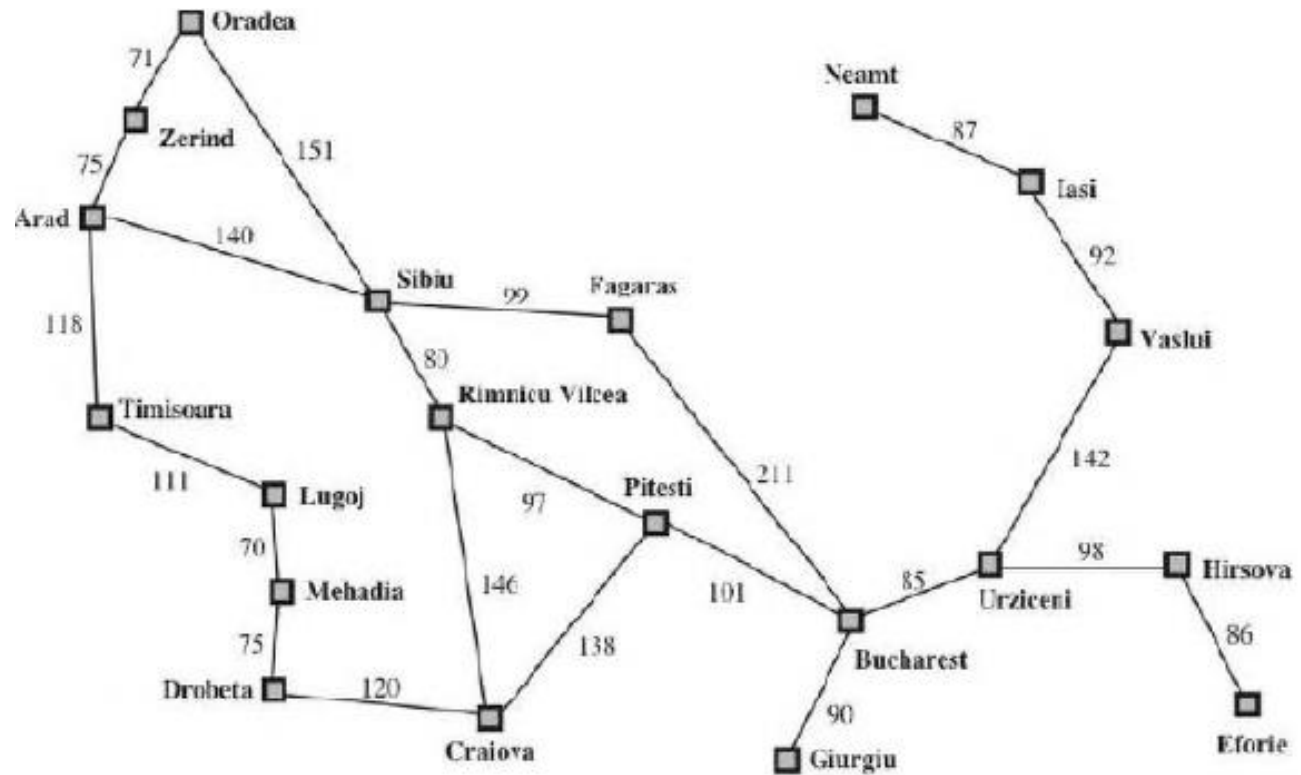
- Para que el agente pueda manejar por Rumania, debe existir un cartel en el mapa que le indique su presencia en las ciudades, o donde está.
- También asumimos que el ambiente es discreto, es decir en cada punto tenemos sólo un grupo de acciones finitas que se pueden hacer.
  - Esto es cierto porque una ciudad está conectada directamente con un grupo finito de ciudades.
- También sabemos que el ambiente es conocido, es decir que estados se puede alcanzar dado una acción determinada.
- También asumimos que el ambiente es determinista, ergo cada acción tiene sólo una salida.

# Problem-solving Agents

- Bajo estas condiciones, la solución es una secuencia fija de acciones.
  - En problemas más complejos, podríamos tener un árbol que hay que evaluar de acuerdo a las condiciones de ambiente que hayan en el momento.
- El problema de buscar una secuencia de acciones que lleguen a un objetivo es llamado en la IA “búsqueda (search)”. Un algoritmo de búsqueda toma un problema como entrada y retorna una solución en la forma de una secuencia de acciones.
- Tenemos un simple “formular, buscar, ejecutar”: Después de formular un objetivo, resolver un problema, el agente llama un algoritmo de búsqueda para encontrar la solución.

# Problem-solving Agents

- Un problema puede ser formulado formalmente por cinco componentes:
  - El estado inicial. Para el caso del ejemplo del taxi, tenemos  $In(Arad)$ .
  - Una descripción de las posibles acciones disponibles para el agente dado un estado “s” ( $ACTIONS(S)$ ). Por ejemplo, del estado  $Ir(Arad)$ , las acciones posibles son:  $\{GO(Sibiu), GO(Timisoura), GO(Zerind)\}$ .



# Problem-solving Agents

- Un problema puede ser formulado formalmente por cinco componentes (cont.):
  - Una descripción de lo que hace cada estado. A esto se le llama el “modelo de transición” (
    - $\text{Result}(s,a)$  = el resultado de aplicar la acción “a” al estado “s”. Al nuevo estado también se le llama “sucesor”.
    - $\text{Result}(\text{In}(\text{Arad}), \text{Go}(\text{Zeriyad})) = \text{In}(\text{Zeriyad})$
  - Estas tres componentes: estado inicial, acciones y modelo de transición define lo que se llama el “espacio de estados” de un modelo. Este espacio tiene forma de grafo, en el cual los nodos son estados y los arcos son acciones. Un camino en este grafo es un par de estados conectados por una secuencia de acciones.



# Problem-solving Agents

- Un problema puede ser formulado formalmente por cinco componentes (cont.):
  - Otra componente es la “prueba de objetivo” la cual determina si un estado actual es un estado final.
  - La ultima componente es el “costo de un camino” que es una función que asigna un costo a cada camino del grafo. Esta función normalmente refleja una métrica de desempeño. Para el agente taxi, puede ser el largo en kilómetros.
- Estas componentes definen la estructura de datos que recibe un algoritmo de “problema-solving”. Una solución es la salida que involucra una secuencia de acciones que van del estado inicial al final. La calidad de la solución es medida por la función de costo y la solución óptima es la de menor coste entre todas las soluciones posibles.

# Problem-solving Agents: Problemas de Ejemplo

- El 8-puzzle consiste de un tablero de 3x3 con ocho cuadrados deslizantes y un espacio en blanco. Cada cuadrado está numerado.
  - Un cuadrado blanco puede deslizarse al espacio vacío.
  - El objetivo es ponerlos en orden ascendente.

7		4
5		6
8	3	1

	1	2
3	4	5
6	7	8

# Problem-solving Agents: Problemas de Ejemplo

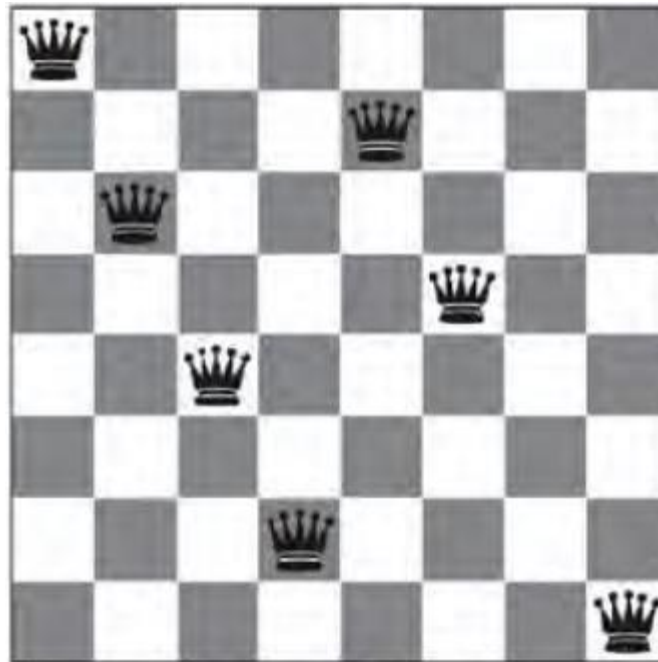
- Estados: Una descripción de estado implica la posición de cada uno de los cuadrados y del espacio en blanco.
- Estado Inicial: Cualquier estado puede ser utilizado como estado inicial.
- Acciones: La formulación más simple considera el movimiento del espacio en blanco hacia arriba, abajo, izquierda y derecha.
- Modelo de Transición: Dado un estado y acción, devuelve la nueva configuración.
- Test de objetivo: Revisa si el estado actual se alinea con el objetivo final del agente.
- El costo del camino: Cada paso cuesta 1, entonces el costo de un camino es el número de pasos que contiene.

# Problem-solving Agents: Problemas de Ejemplo

- El problema 8-puzzle pertenece a la familia de puzles de bloque deslizantes, que es NP-completo.
- En el caso de 8 cuadros, tenemos 440 estados alcanzables, pero si subimos a uno de 4x4, es decir 15 piezas, tenemos 1.3 trillones de estados.

# Problem-solving Agents: Problemas de Ejemplo

- El objetivo del problema de las 8-reinas es poner ocho reinas en un tablero de ajedrez tal que ninguna de ellas se ataque.
  - Una reina puede atacar cualquier pieza que esté en la misma diagonal, columna o fila.



# Problem-solving Agents: Problemas de Ejemplo

- Hay dos tipos de formulaciones:
  - Incremental
  - Estado completo
- En ambas formulación el costo del camino no importa, porque lo único que interesa es llegar al estado final.
  - Estados: Cualquier combinación de 0 a 8 reinas en el tablero.
  - Estado Inicial: No hay reinas en el tablero.
  - Acción: Poner una reina en una celda.
  - Modelo de transición: Devuelve el tablero con la reina agregada.
  - Prueba de objetivo: Las ocho reinas están en el tablero, ninguna se ataca.

# Problem-solving Agents: Problemas de Ejemplo

- En esta formulación, tenemos  $64 \times 63 \times 62 \times \dots \times 1$  posibles secuencias para investigar. Una mejor formulación prohibiría poner cualquier reina en un cuadrado que ya esté atacado:
  - Estados: todas las posibles combinaciones de  $n$  reinas ( $0 < n < 8$ ), una por columna en las “ $n$ ” que están más a la izquierda, sin que ninguna reina se pueda atacar.
  - Acción: Agregar una reina en la columna siguiente tal que no sea atacada por las reinas que ya han sido ubicadas.
- Esta reformulación reduce el problema de las 8 reinas de  $1.8 \times 10^{14}$  a sólo 2,057 combinaciones. Para el caso de las 100 reinas, las combinaciones bajan de  $10^{400}$  a  $10^{52}$ .

# Problem-solving Agents: Problemas de Ejemplo

- Pero un problema de  $10^{52}$  estados sigue siendo intratable.
- En 1964, Donald Knuth conjeturó que partiendo del número cuatro, una secuencia de factoriales, raíces cuadradas, y operaciones piso puede resultar en cualquier número entero deseado.
  - Por ejemplo, se puede alcanzar el 5 desde el cuatro con la siguiente fórmula:

$$\sqrt{\sqrt{\sqrt{(4!)!}}} = 5$$



# Problem-solving Agents: Problemas de Ejemplo

- La definición del problema es muy simple:
  - Estados: los números positivos.
  - Estado inicial: 4
  - Acciones: aplicar el factorial (sólo a números enteros), raíz cuadrada o operación piso.
  - Modelo de transición: De acuerdo a las definiciones matemáticas de las operaciones antes descritas.
  - Prueba de objetivo: El estado es o no el entero buscado.

# Problem-solving Agents: Problemas de Ejemplo

- Los problemas de ruteo están definidos en términos de ubicaciones específicas y transiciones mediante vínculos entre las ubicaciones.
- Este tipo de problemas están implícitos en diversas aplicaciones del mundo real:
  - Web
  - Sistemas internos dentro de los autos
  - Planificación de operaciones militares
  - Sistemas de planificación aérea

# Problem-solving Agents: Problemas de Ejemplo

- Consideremos el problema de viaje aéreo que debe ser resuelto por un sitio web de planificación de viaje:
  - Estado: considera una ubicación (e.g., aeropuerto) y la hora actual. También hay que tomar en cuenta factores históricos, como el costo de una acción (un segmento de vuelo), ya que puede depender de los tramos previos. Además si es un vuelo doméstico o internacional.
  - Estado Inicial: Esto lo especifica el usuario con su consulta.
  - Acciones: Tomar cualquier vuelo desde la ubicación actual, en cualquier tipo de asiento. Partiendo después de la hora actual, dejando tiempo para transfers entre aeropuerto si es necesario.
  - Modelo de transición: el estado resultante de tomar un vuelo va a ser el destino como ubicación actual y la hora de llegada como la nueva hora actual.
  - Prueba de Objetivo: ¿Estamos en el destino requerido por el usuario?
  - El costo del camino: Monetario, tiempo de espera, tiempo de vuelo, procedimientos de aduanas e inmigración, calidad del asiento, hora, tipo de avión, kms. Acumulados, etc.

# Problem-solving Agents: Problemas de Ejemplo

- Sistemas de recomendación de viajes comerciales tienen agentes de este tipo. Sin embargo, todo viajero sabe, que estos sistemas deben considerar planes de contingencia también de acuerdo a la probabilidad y costo del fallo en el plan original.
- Otro tipo de problemas, similar pero diferente al mismo tiempo, son los de “tours”. Como los problemas de ruteo, las acciones corresponden a viajes entre ciudades adyacentes. El “espacio de estados” es diferente. No sólo debe incluir la ubicación actual, sino que también el conjunto de ciudades que el agente ya ha visitado.

# Problem-solving Agents: Problemas de Ejemplo

- Así el estado inicial sería  $In(Bucarest)$ ,  $Visited(Bucarest)$ .
- Un estado intermedio podría ser  $In(Vaslui)$ ,  $Visited(\{Bucarest, Urziceni, Vaslui\})$ .
- La “prueba de objetivo” sería si el agente está en Bucarest y si ya ha visitado las 20 ciudades.
- En el caso del TSP (Traveling Salesperson Problem ), cada ciudad debe ser visitada exactamente una vez. El objetivo es encontrar el camino más corto para hacerlo.
  - Es un problema NP-duro

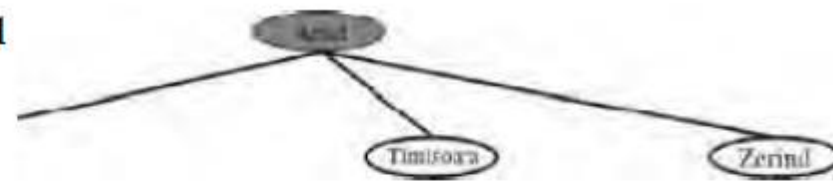
# Problem-solving Agents: Problemas de Ejemplo

- Formular los problemas de manera exitosa es el primer paso. El segundo es encontrar la(s) solución(es).
- Una solución es una secuencia de acciones, ergo los algoritmos de búsqueda trabajan considerando varias opciones.
- Las secuencias de acciones posibles forman un árbol con el estado inicial como raíz; las ramas con acciones, y los nodos corresponden a estados en el “espacio de estados”.

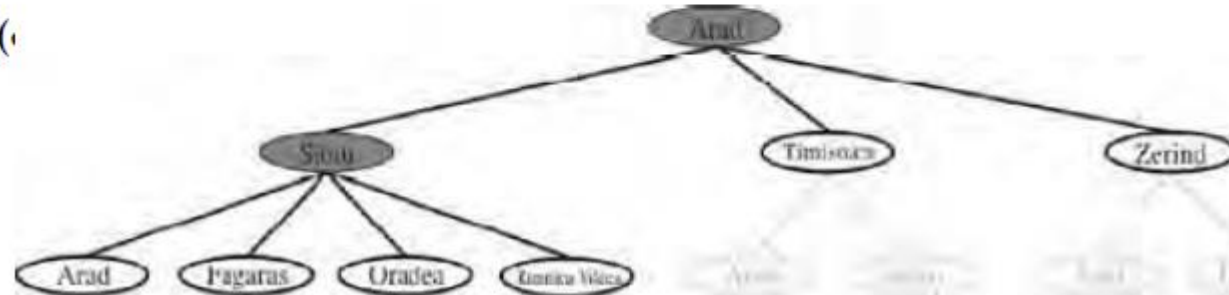
# Problem-solving Agents: Problemas de Ejemplo

**Después de expandir el árbol se tienen los siguientes estados. Hacia Arad por ejemplo.**

(b) After expanding Arad



(c)



# Problem-solving Agents: Problemas de Ejemplo

- El proceso de expandir los nodos continua hasta que no hayan más estados posibles o hasta que se haya encontrado una solución.
- La mayoría de las estrategia de búsqueda usan este esquema de árbol. Difieren primeramente en cómo deciden que estado explorar en el siguiente paso.
- Uno de los problemas que deben solucionar los algoritmos de búsqueda son los loops.
  - Arad->Sibiu->Arad
  - Pueden ser infinitos y nunca se encontrará una solución.
- Sin embargo, cuando los caminos tienen costos, es difícil que esos loops se den porque los costos del camino van creciendo sucesivamente. Lo cual los hace inviables para el agente.

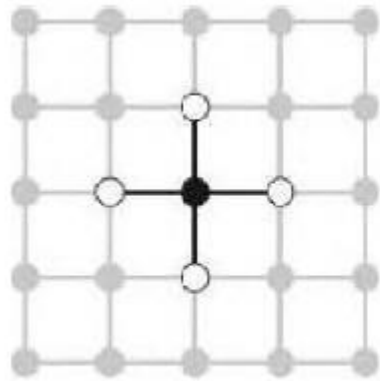


# Problem-solving Agents: Problemas de Ejemplo

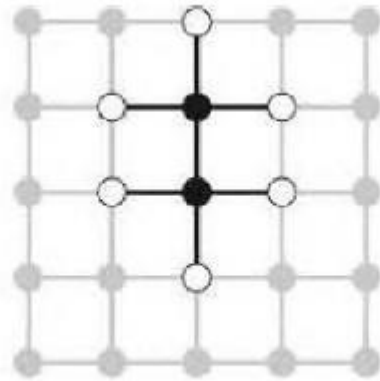
- En realidad, “loops” son un caso más especial de “caminos redundantes”. Estos existen cada vez que existen diversas alternativas de ir de un estado a otro.
  - Arad–Sibiu (140 k) y Arad–Zerind–Oradea–Sibiu (297 km).
- Caminos redundantes suelen aparecer cuando es possible revertir algunas acciones.

# Problem-solving Agents: Problemas de Ejemplo

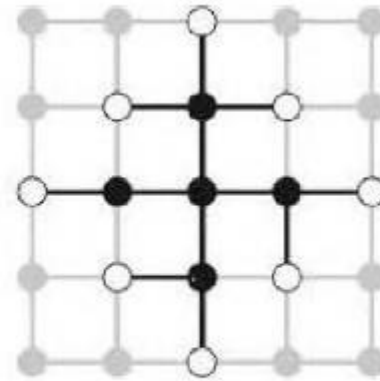
- Un tipo de problema típico en los juegos de computador es encontrar una ruta en una grilla rectangular.



(a)



(b)



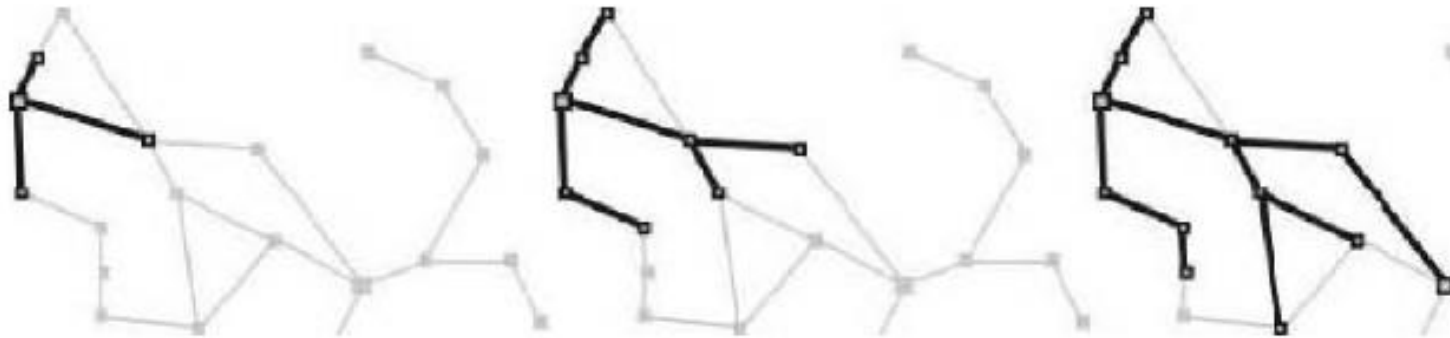
(c)

# Problem-solving Agents: Problemas de Ejemplo

- En una grilla como esa, cada estado tiene cuatro sucesores. Por ende, un árbol de búsqueda de profundidad “d”, que incluye estados repetidos, tiene cuatro hijos.
- Pada “d” igual a 20, tenemos  $2 \cdot d^2$  estados diferentes, y un total de un trillón de nodos.
  - Seguir sólo los nodos diferentes puede transformar el problema de una intratable a uno tratable.
  - Una forma de evitar caer en estados repetidos es recordar donde se ha estado. Para esto, se enriquece los algoritmo de búsqueda de árboles con un “conjunto explorado”, el cual recuerda todos los nodos que han sido expandidos.

# Problem-solving Agents: Problemas de Ejemplo

- El algoritmo tiene otra buena propiedad: la frontera separa el grafo de espacio de estados en regiones exploradas y no exploradas.
  - Entonces, todos los caminos desde el estado inicial a un estado no-explorado tiene que pasar por un estado en la frontera. (Un nodo frontera debe abrirse).



# Problem-solving Agents: Problemas de Ejemplo

- Los algoritmos de búsqueda necesitan una estructura de datos para mantener el rastro del árbol de búsqueda que está siendo construido. Por cada nodo “n” del árbol, tenemos una estructura que contiene cuatro componentes:
  - n.STATE: El estado del “espacio de estados” al cual corresponde este nodo.
  - n.PARENT: El nodo del árbol de búsqueda que generó este nodo.
  - n.ACTION: La acción que se le aplicó al padre para generar este nodo.
  - n.PATH-COST: El costo, tradicionalmente  $g(n)$ , del camino partiendo desde la raíz hasta el nodo.

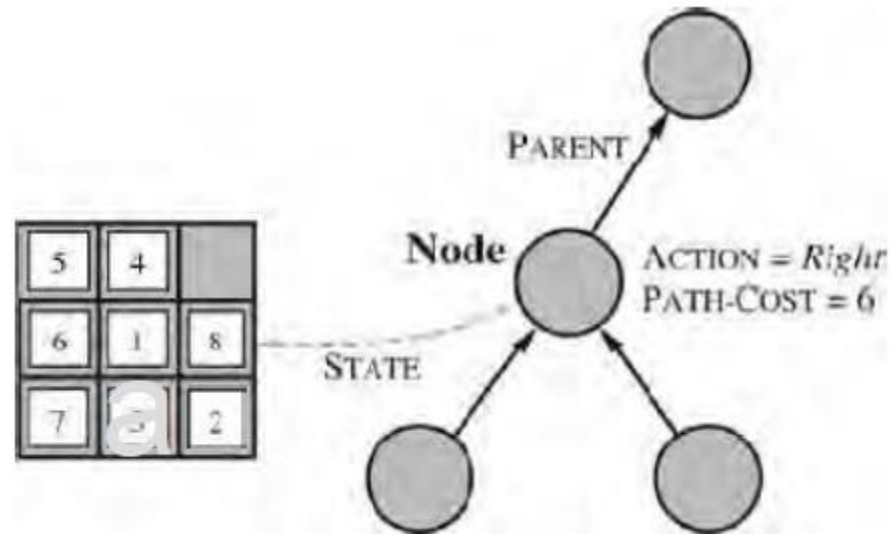
# Problem-solving Agents: Problemas de Ejemplo

- Dado las componentes para un nodo padre, es fácil ver cómo calcular las componentes necesarias para cada nodo hijo. La función Child-Node toma un nodo padre y una acción, retornando el nodo hijo resultante:

```
function CHILD-NODE(problem, parent, action) returns a node
  return a node with
    STATE = problem.RESULT(parent.STATE, action),
    PARENT = parent, ACTION = action,
    PATH COST = parent.PATH COST + problem.STEP COST(parent.STATE, action)
```

# Problem-solving Agents: Problemas de Ejemplo

- La estructura de datos nodo puede ser graficada de la siguiente forma:



# Problem-solving Agents: Problemas de Ejemplo

- Un nodo es una estructura de datos usada para representar el árbol de búsqueda. Un estado es una configuración del mundo. Los nodos están en un camino específico definidos por los punteros PADRES, en cambio los estados no.
- Dos nodos diferentes podrían contener el mismo estado del mundo si este puede ser generado por dos caminos diferentes (los costos también podrían ser diferentes).
- La frontera debe ser almacenada de tal forma que el algoritmo de búsqueda pueda fácilmente escoger el próximo nodo a expandir de acuerdo a la estrategia preferida.



# Problem-solving Agents: Problemas de Ejemplo

- La estructura de datos que se puede utilizar es la Cola. Recordar que las operaciones que se pueden hacer a una cola son:
  - Empty?(queue) retorna verdadero si la cola está vacía.
  - Pop(queue) remueve el primer elemento de una cola y lo retorna.
  - Insert(queue) agrega un elemento y devuelve la cola resultante.
- Las colas se diferencian por el orden en el cual se van insertando los nodos.
  - FIFO, LIFO, prioridad.
- El conjunto de estados explorados se puede implementar con una tabla hash.

# Problem-solving Agents: Problemas de Ejemplo

- El desempeño de los algoritmos se pueden medir de diversas formas:
  - **Complejitud**: ¿Garantiza el algoritmo encontrar una solución cuando hay una?
  - **Optimalidad**: ¿Encuentra el algoritmo la mejor solución?
  - **Complejidad en el tiempo**: ¿Cuánto tarda en encontrar una solución?
  - **Complejidad en el espacio**: ¿Cuánta memoria es necesaria para realizar la búsqueda?
- Para complejidad en el tiempo y espacio, se utiliza alguna referencia que normalmente es el tamaño del espacio de búsqueda.

# Problem-solving Agents: Problemas de Ejemplo

- Complejidad es usualmente vistas en tres cantidades:
  - $b$ =el factor de ramificación o número máximo de sucesores de cualquier nodo
  - $d$ =el número de pasos por el camino desde la raíz hasta el nodo que cumple el objetivo en el nivel más superficial.
  - $m$ =el largo máximo de un camino en todo el espacio de búsqueda.
- El tiempo se mide de acuerdo al número de nodos generados durante la búsqueda y el espacio en términos del máximo número de nodos almacenado en memoria.
- La efectividad del algoritmo debemos tomar en cuenta el costo de la búsqueda (recursos utilizados) y el costo del camino (calidad de la solución).

# Problem-solving Agents: Problemas de Ejemplo

- Hay diversos tipos de algoritmos para recorrer los árboles. El primer tipo se llama de búsqueda desinformada o ciega.
  - No hay información adicional del problema salvo la provista por la definición del problema.
  - Todo lo que pueden hacer es generar los sucesores y distinguir el estado final del resto de los estados.
  - Estas estrategias se distinguen por el orden mediante el cual los nodos son expandidos.
  - Ejemplos: BFS, DFS, Uniform-cost Search, Depth-limited search, Iterative Deepening Depth-first Search, Bidirectional Search.

# Problem-solving Agents: Problemas de Ejemplo

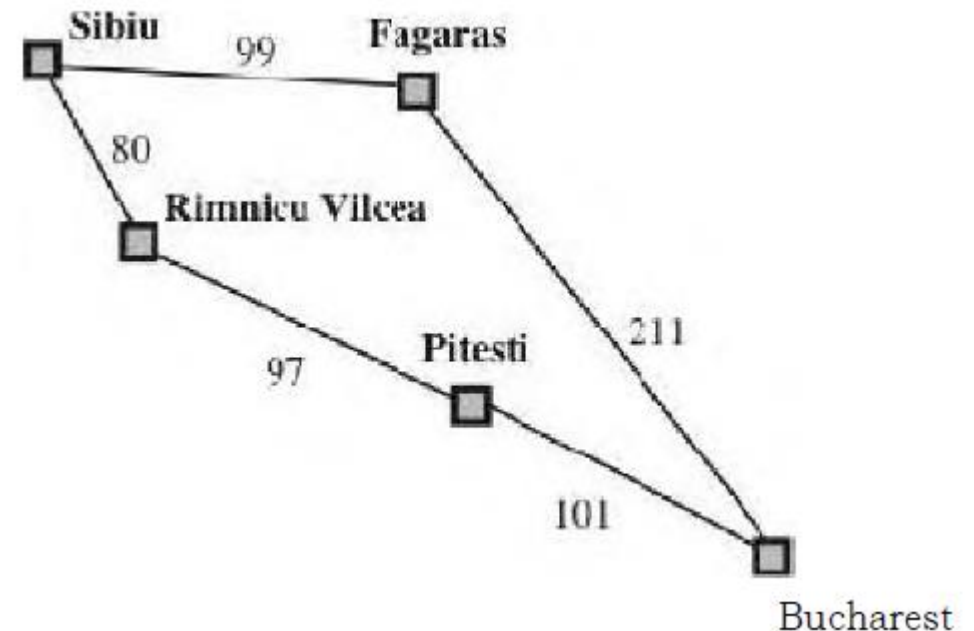
- Hay diversos tipos de algoritmos para recorrer los árboles. El primer tipo se llama de búsqueda desinformada o ciega.
- Uniform-cost Search expande el nodo que tiene el menor costo de camino (al contrario de BFS que abre el nodo menos profundo). Así va ordenando en la cola por coste.

Los sucesores de Sibiu son Rimnicu Vilcea y Fagaras con costos 80 y 99, respectivamente.

El nodo de menor coste es expandido a continuación, agregando Pitesti con  $80+97=177$ .

El siguiente nodo es Fagaras, que agrega a Bucarest con  $99+211=310$ . Acá se alcanza el objetivo.

Pero Uniform-cost Search sigue buscando, expandiendo Pitesti, el cual da a Bucarest con  $177+101=278$ .



# Problem-solving Agents: Problemas de Ejemplo

- Es claro que cuando se escoge un nodo para ser expandido (no cuando se genera), el camino óptimo a ese nodo ha sido encontrado.
- BFS y DFS también pueden ser utilizados para recorrer el árbol. Existe una variante de DFS, llamada DFS-limitado que permite avanzar en profundidad una cierta cantidad de niveles “l” fija. Los nodos en el nivel “l” son tratados como si no tuviesen sucesores.
  - Si bien esto combate caminos “infinitos”, introduce una nueva complejidad al problema cuando no conocemos la profundidad necesaria para encontrar el nodo solución más superficial del árbol.
  - Hay veces que podemos tener estimaciones, en el caso de Romania sabemos que la profundidad máxima es 19, porque tenemos 20 ciudades.

# Problem-solving Agents: Problemas de Ejemplo

- Es claro que cuando se escoge un nodo para ser expandido (no cuando se genera), el camino óptimo a ese nodo ha sido encontrado.
- BFS y DFS también pueden ser utilizados para recorrer el árbol. Existe una variante de DFS, llamada DFS-limitado que permite avanzar en profundidad una cierta cantidad de niveles “l” fija. Los nodos en el nivel “l” son tratados como si no tuviesen sucesores.
  - Si bien esto combate caminos “infinitos”, introduce una nueva complejidad al problema cuando no conocemos la profundidad necesaria para encontrar el nodo solución más superficial del árbol.
  - Hay veces que podemos tener estimaciones, en el caso de Romania sabemos que la profundidad máxima es 19, porque tenemos 20 ciudades.

# Problem-solving Agents: Problemas de Ejemplo

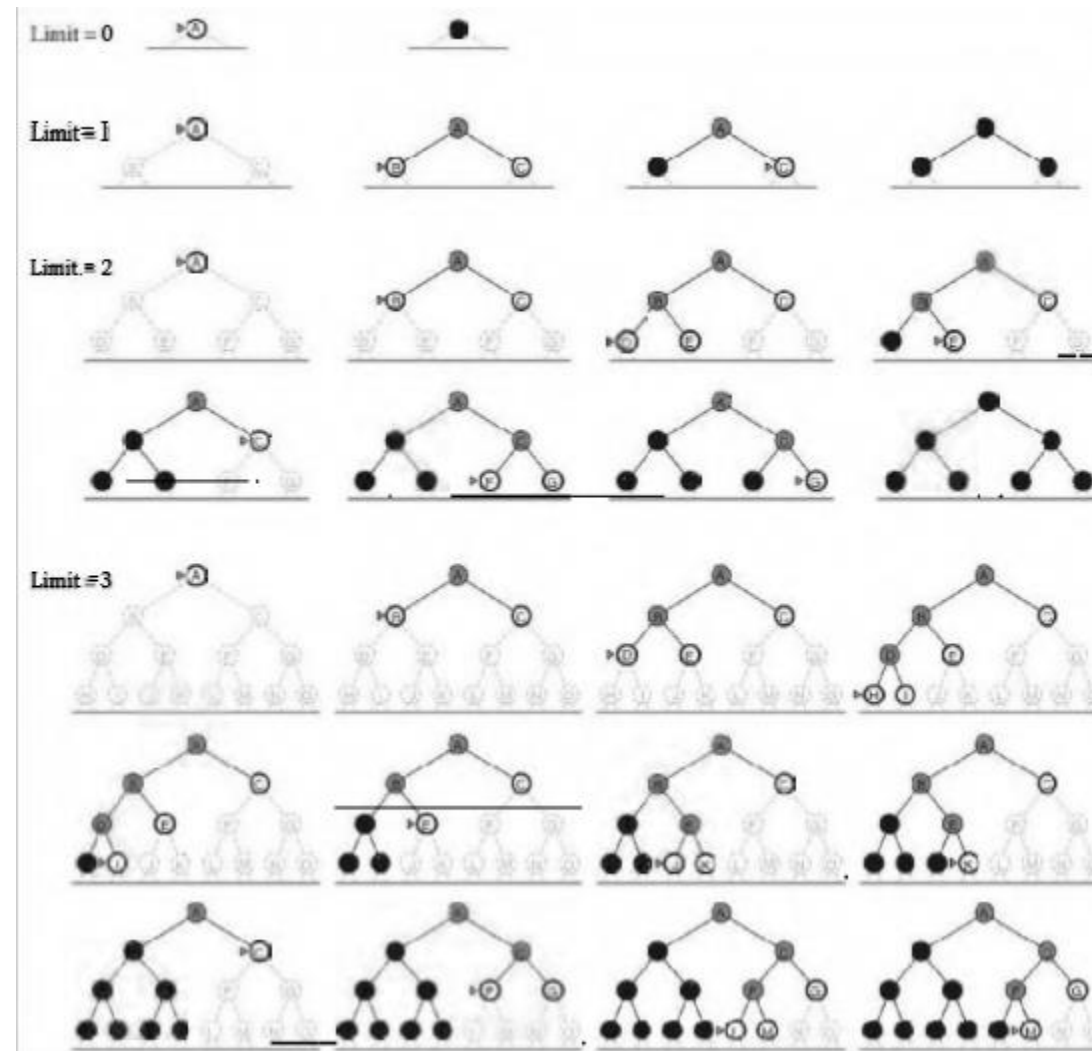
- Lo interesante de DFS-limitado, es que muchas veces podemos estimar el parámetro de profundidad muy bien.
  - Por ejemplo, si descubrimos que una ciudad puede alcanzar cualquier otra ciudad en a lo más nueve pasos.
  - Este parámetro se llama diámetro y normalmente es una estimación mucho más eficiente para la búsqueda.



# Problem-solving Agents: Problemas de Ejemplo

- Otra estrategia es “Iterative Deepening DFS”, la cual modifica el DFS original mediante la búsqueda de la mejor profundidad. Lo hace mediante el incremento sistemático del parámetro de profundidad: 0, 1, 2, .... Etc.
- Cuando se encuentra un objetivo, se tiene el objetivo en el nivel más superficial.
- IDDFS se parece a BFS en que ambos explorar todo un nivel para poder avanzar al otro.

# Problem-solving Agents: Problemas de Ejemplo



# Problem-solving Agents: Problemas de Ejemplo

- Otra técnica ciega es Bidirectional Search, la cual ejecuta dos búsquedas en paralelo: Una de forma tradicional que parte desde el estado inicial, y la otra desde el estado objetivo hacia atrás.
  - Cuando ambas búsquedas se encuentran entonces se encontró una solución.
  - Es posible que la solución encontrada sea subóptima. Es necesario agregar revisiones extras.
  - Buscar hacia atrás es problemático:
    - Se necesita calcular los predecesores de un estado.
    - Muchas veces no es claro lo que significa “el objetivo”.

# Problem-solving Agents: Estrategias Informadas

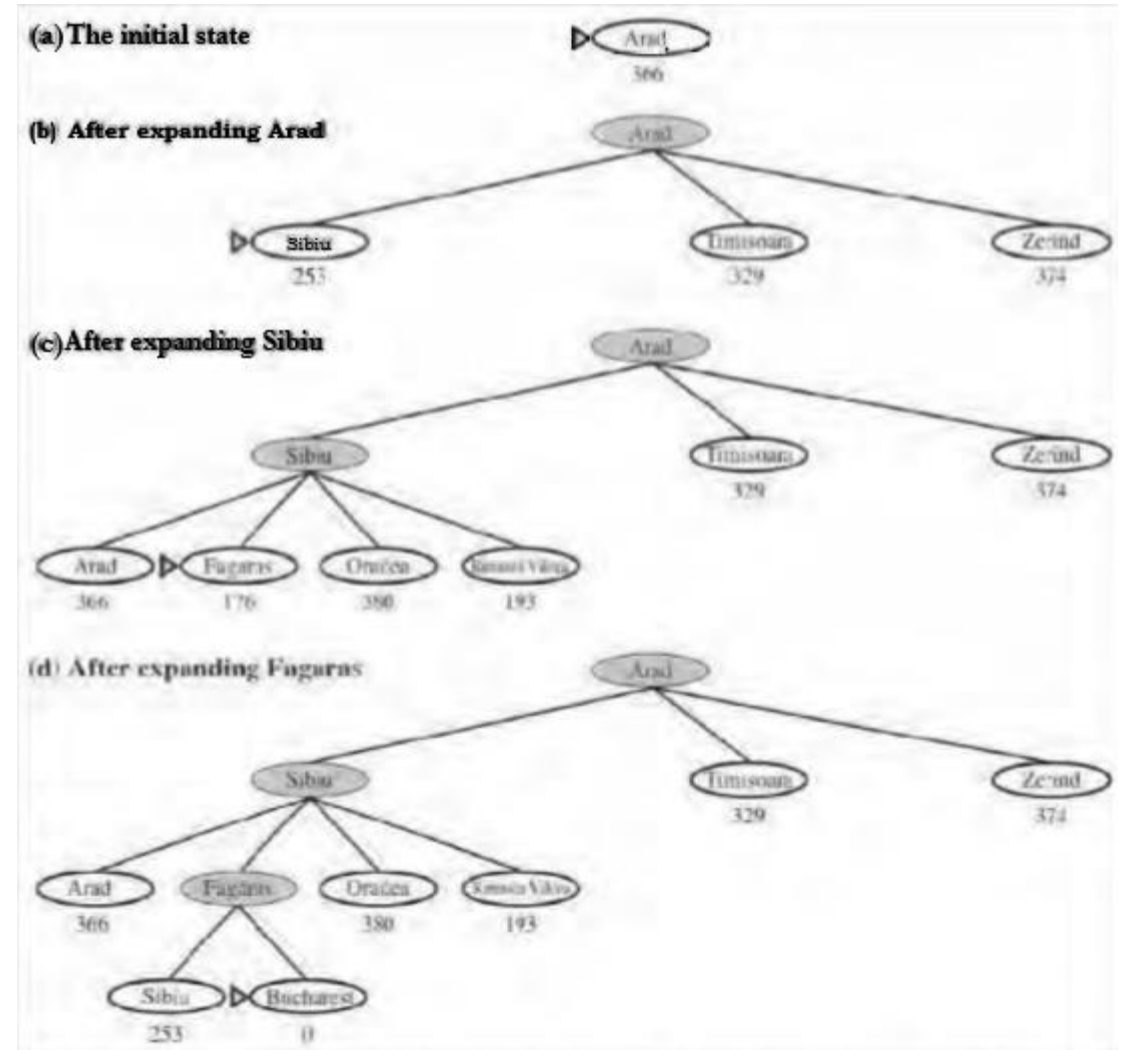
- La primera estrategia es Best-first Search. La cual selecciona el nodo a expandir de acuerdo a una función de evaluación  $f(n)$ .
  - Esta función se construye como una que estima los costos, de tal forma que la de más bajo coste se expande primero.
  - Es idéntico a Uniform-cost Search, salvo por el uso de  $f(n)$ . En los algoritmos modernos, esta función incluye una componente heurística  $h(n)$ . La cual estima el coste más barato del camino desde este nodo al objetivo.
  - Por ejemplo, para el problema de Rumania, la estimación puede ser la distancia en línea recta desde la ciudad al objetivo.
  - Las funciones heurísticas es la forma más común de darle conocimiento a los algoritmos de búsqueda. Son arbitrarias. No-negativas, específicas al problema, con sólo una restricción  $h(n)=0$  para los nodos objetivo.

# Problem-solving Agents: Estrategias Informadas

- Greedy best-first Search expande el nodo que está más cerca al objetivo con la idea de llegar rápido.
- Este algoritmo expande primero Sibiu porque es el más cercano a Bucarest, el siguiente será Fagaras porque es el más cercano, el cual después escoge Bucarest.
- Para este caso, el algoritmo no encuentra el óptimo. El camino propuesto tiene un costo de 32km superior a la alternativa: Rimnicu, Vilcea y Pitesti.
- Es un algoritmo incompleto porque no busca en todo el espacio de soluciones.

# Problem-solving Agents: Estrategias Informadas

Arad	366	<b>Mehadia</b>	241
Bucharest	0	<b>Neamt</b>	234
Craiova	160	Oradea	380
<b>Drobeta</b>	242	Pitesti	100
Eforie	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	Sibiu	253
Giurgiu	77	Timisoara	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
Iasi	226	Vaslui	199
Lgoj	244	Zerind	374



# Problem-solving Agents: Estrategias Informadas

- El siguiente algoritmo es A\*. Combina:
  - La función  $g(n)$ : el coste del alcanzar el nodo.
  - La función  $h(n)$ : el coste del alcanzar el objetivo desde el nodo.
- Entonces,  $f(n)=g(n)+h(n)$  que es el coste estimado de la solución más barata que pasa por “n”.
- A\* es óptimo y completo.
- Es parecido a UCS excepto que  $g+h$  en vez de sólo  $g$ .

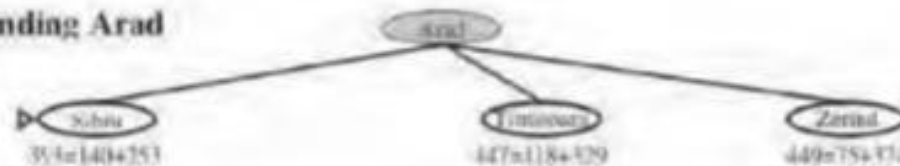
# Problem-solving Agents: Estrategias Informadas

- ¿Cuáles son las condiciones para que  $A^*$  se óptimo?
  - La función  $h(n)$  debe ser una heurística admisible, es decir que nunca sobrestime el costo de alcanzar el objetivo. Son por naturaleza optimistas, porque creen que el costo de resolver el problema es menor de lo que realmente es. Por ejemplo, la heurística de la línea recta. La ruta real va a tener un coste mayor que ese, es imposible que sea menor.

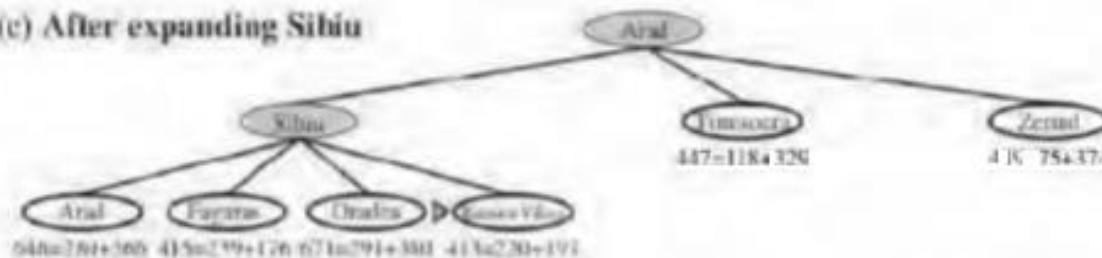
(a) The initial state



(b) After expanding Arad



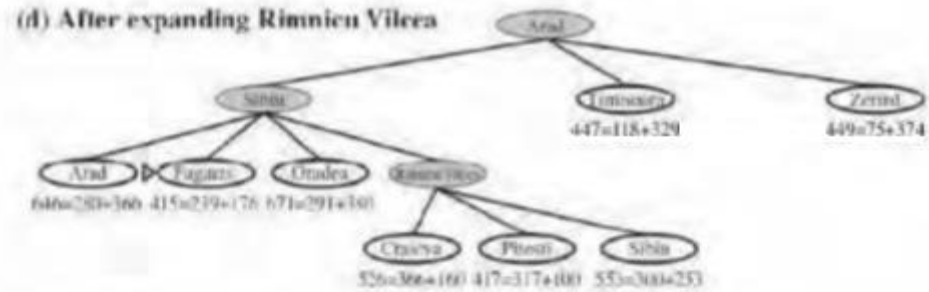
(c) After expanding Sibiu



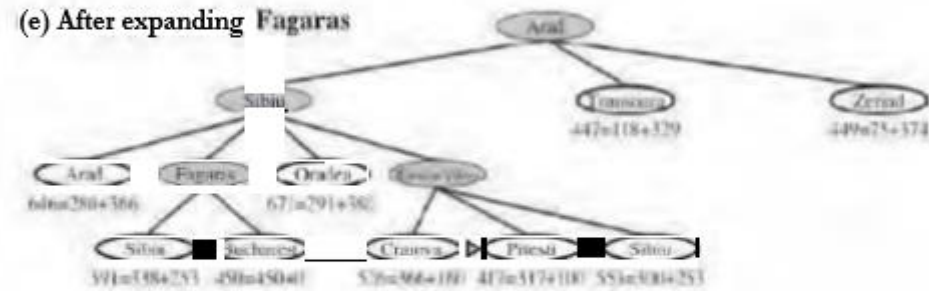


# Problem-solving Agents: Estrategias Informadas

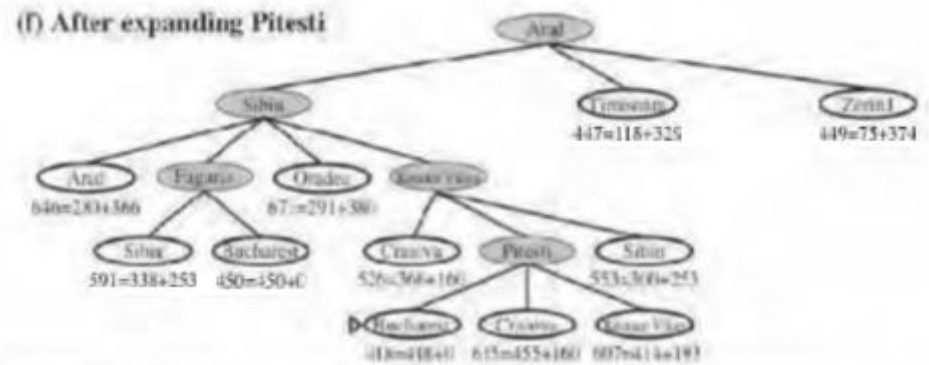
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti

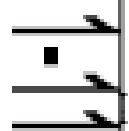


# Problem-solving Agents: Estrategias Informadas

- Otra condición importante para que A\* funcione bien es la de consistencia o monotocidad.
- Una heurística  $h(n)$  es monótona si para cada nodo  $n$  y su sucesor  $n'$  generado desde  $n$  por una acción  $u$ , el coste estimado de alcanzar el objetivo desde  $n$  no es más grande que el coste del paso de llegar a  $n'$  más el coste estimado de alcanzar el objetivo desde  $n'$ :  $h(n) \leq c(n,a,n') + h(n')$

# Problem-solving Agents: Funciones Heurísticas

- Si queremos encontrar el camino más corto utilizando A\* para el problema de las 8 piezas deslizantes, tenemos que idear una función que nunca sobre estime el número de pasos para llegar al objetivo.

7	2	4
5		6
8		1

Start State

	1	2
3	4	5
6	7	8

Goal State

# Problem-solving Agents: Estrategias Informadas

- Hay una larga historia de heurísticas de búsqueda para el problema de 15 piezas (por lo atractivo del problema y complejidad).
- Dos son las usadas comúnmente:
  - $H_1$ =El número de celdas mal ubicadas. Para la figura anterior, todas están mal ubicadas, ergo el valor inicial de  $h_1$  sería ocho, ya que tiene sentido que todas las celdas deban moverse al menos una vez.
  - $H_2$ =La suma de las distancias Manhattan de las celdas a su posición objetivo. Es admisible, porque todo lo que puede hacer es mover un paso más cerca al objetivo.
    - Para el ejemplo anterior, tenemos  $3+1+2+2+2+3+3+2=18$

# Problem-solving Agents: Estrategias Informadas

- Otra

# Referencias

- “Artificial Intelligence: A Modern Approach”, 3rd Edition, Capítulo 3.