A* Algorithm

Basics of A* algorithm

- ◊ You are in the middle of a search and have a set of potential paths P1 .. Pn to explore.
  - » **How do you select the best path to extend?**

Basics of A* algorithm – 2

- ◊ You are in the middle of a search and have a set of potential paths P1 .. Pn to explore.
  - » **How do you select the best path to extend?**
    - > **For the last node on each path have two costs**
  - » **What are they?**

Basics of A* algorithm – 3

- ◊ You are in the middle of a search and have a set of potential paths P1 .. Pn to explore.
  - » **How do you select the best path to extend?**
    - > **For the last node on each path have two costs**
  - » **What are they?**
    - > **(1) the real cost of following the path**
      - **g(n) where n is the last vertex in the path**

Basics of A* algorithm – 4

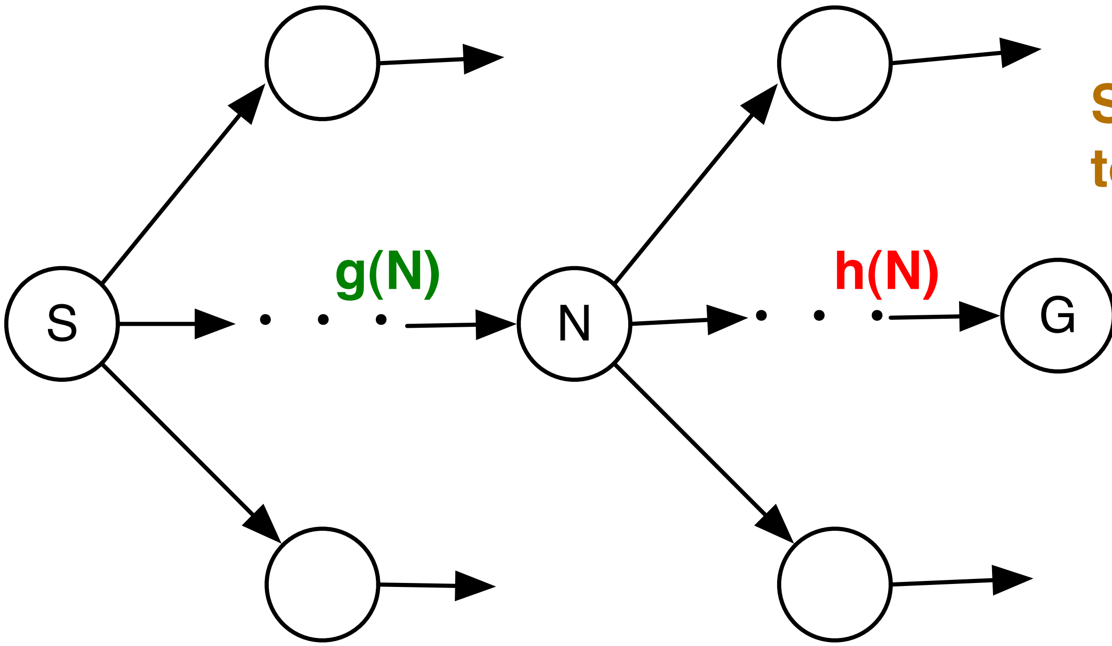
◊ You are in the middle of a search and have a set of potential paths P1 .. Pn to explore.

- » **How do you select the best path to extend?**
  - > **For the last node on each path have two costs**
- » **What are they?**
  - > **(1) the real cost of following the path**
    - – **g(n) where n is the last vertex in the path**
  - > **(2) a heuristic estimate of the cost of the optimal extension of the path to the goal vertex**
    - • **h(n) where n is the last vertex in the path**

Basics of A* algorithm – 5

- ◊ You are in the middle of a search and have a set of potential paths P1 .. Pn to explore.
  - » **How do you select the best path to extend?**
    - > **For the last node on each path have two costs**
      - – **(1) the real cost of following the path**
        - • **g(n) where n is the last vertex in the path**
      - – **(2) a heuristic estimate of the cost of the optimal extension of the path to the goal vertex**
        - • **h(n) where n is the last vertex in the path**
  - » **The estimated cost for the full path to the goal is**
    - > **$f(n) = g(n) + h(n)$**

Basics of A* algorithm – 3

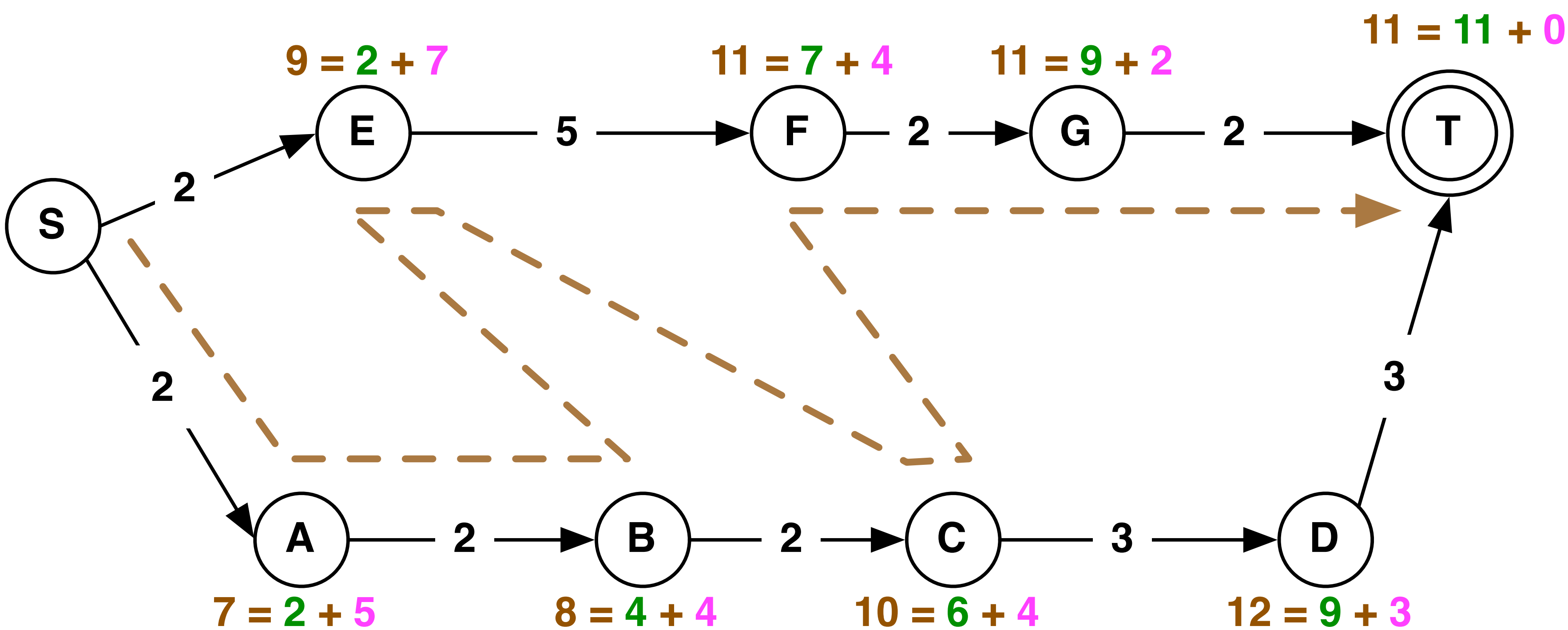


S .. G is the solution path
total estimated cost is
$f(N) = g(n) + h(N)$

S .. N is the known path
g(N) is its real cost

N .. G is the path yet to be found
h(N) is its estimated cost

Bratko Figure 12.2



f(n) in mocha = g(n) in clover + h(n) in magenta

Put "write('Case1 '), S=[NIP], write(S), nl,” just before "goal”
in expand case 1 to see the sequence in which the path is expanded.

A* data structures – leaf node

◊ A leaf is a single node tree – **l (N , F / G)**

Lower case L

A* data structures – leaf node – 2

- A leaf is a single node tree – **l (N** , F / G **)**
    - **N is a node in the state-space**

A* data structures – leaf node – 3

◊ A leaf is a single node tree – **l (N , F / G)**

» **N is a node in the state-space**

» **G = g(n) is the cost of the path to N**

A* data structures – leaf node – 4

◊ A leaf is a single node tree – **l (N , F / G)**

» **N is a node in the state-space**

» **G is the cost of the path to N**

» **F is f(N) = G + h(N)**

A* data structures – tree

◊ A tree – **t (N , F / G , Sub-trees)**

A* data structures – tree – 2

- A tree – **t (N ,** F / G , Sub-trees**)**
  - **N is a node in the state-space**

A* data structures – tree – 3

- A tree – **t** (N , F / **G** , Sub-trees**)**
  - » **N is a node in the state-space**
  - » **G = g(n) is the cost of the path to N**

A* data structures – tree – 4

- A tree – t (N , F / G , Sub-trees)
  - » **N is a node in the state-space**
  - » **G = g(n) is the cost of the path to N**
  - » **F is the updated value of f(N)**
    - > **f-value of the most promising successor of N**

A* data structures – tree – 5

◊ A tree – t (N , F / G , Sub-trees)

» N is a node in the state-space

» G = g(n) is the cost of the path to N

» F is the updated value of f(N)

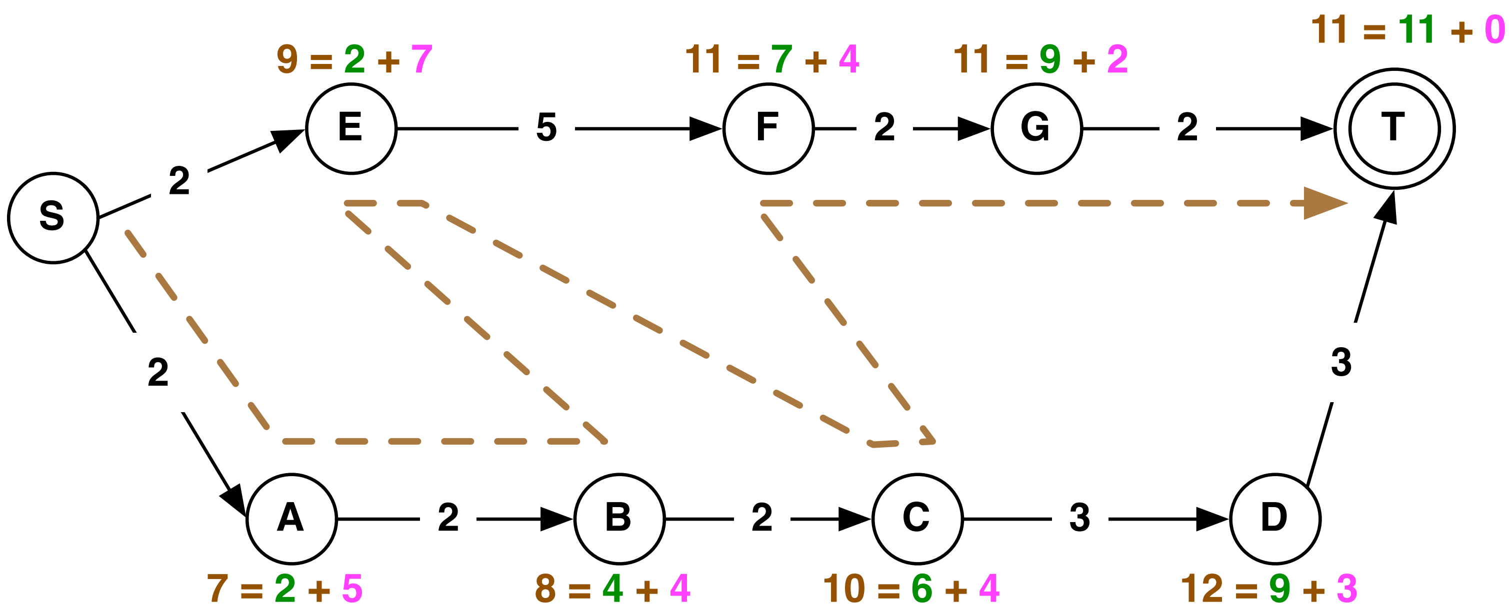
> f-value of the most promising successor of N

» Sub-trees is a list of the sub-trees from N

Example for Figure 12.2

When S is expanded, the existing tree is represented as

t (S , 7 / 0 , [l (A , 7 / 2) , l (E , 9 / 2)])

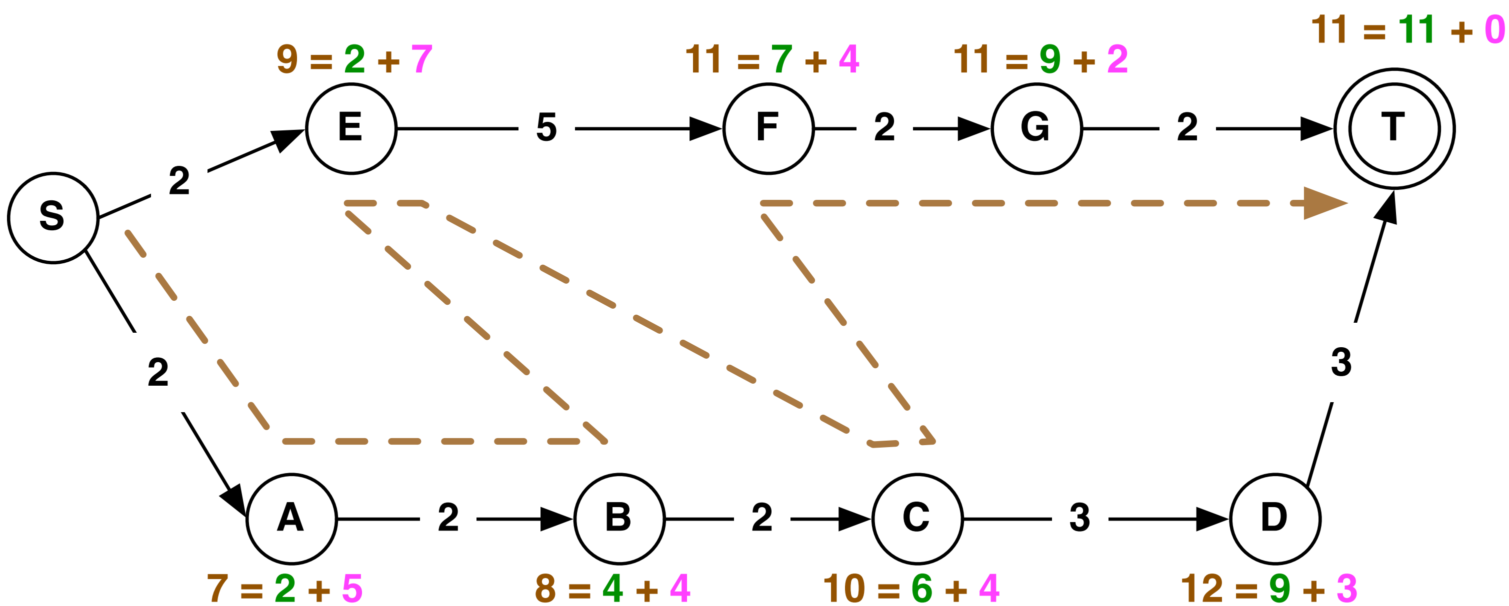


f(n) in mocha = g(n) in clover + h(n) in magenta

Example for Figure 12.2 – 2

t (S , 7 / 0 , [l (A , 7 / 2) , l (E , 9 / 2)])

The most promising node to expand is A



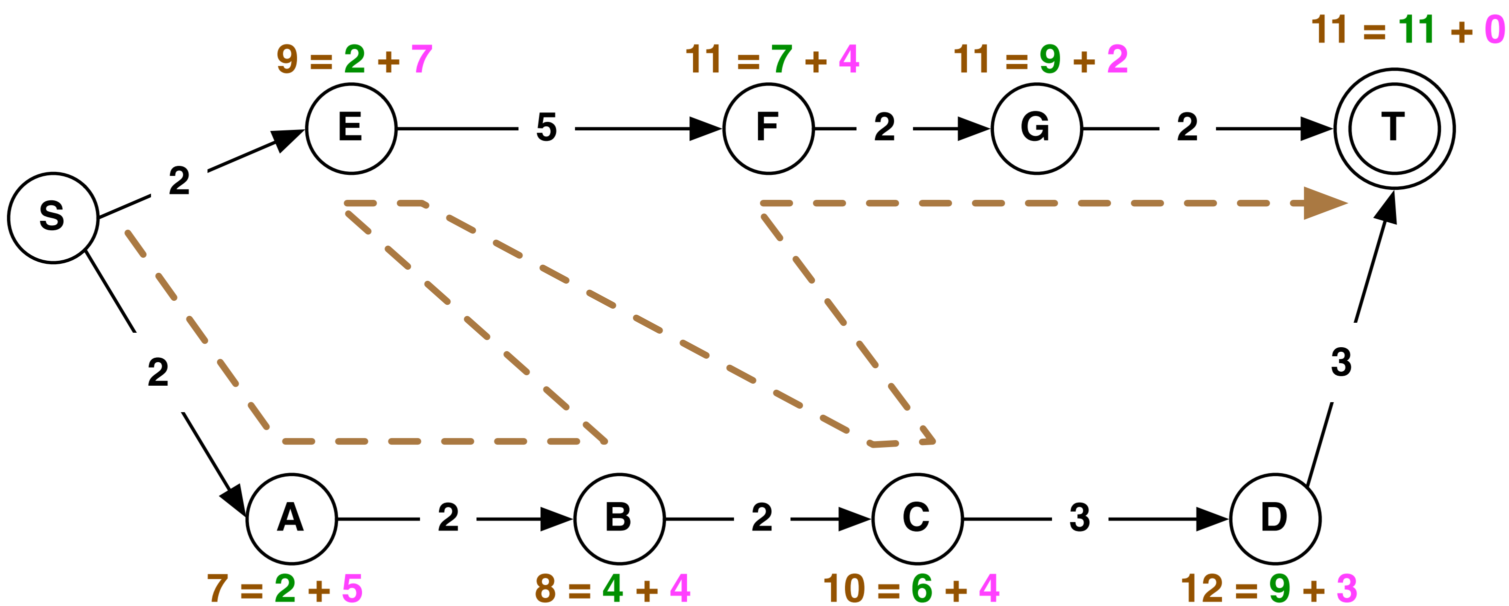
f(n) in mocha = g(n) in clover + h(n) in magenta

Example for Figure 12.2 – 3

After S and A have been expanded we have

t (S , 9 / 0 , [l (E , 9 / 2)] , t (A , 10 / 2 , [t (B , 10 / 4 ,
[l (C , 10 / 6)])])])

Updated – E is the most promising successor



f(n) in mocha = g(n) in clover + h(n) in magenta

Generalization of f-value definition

- For a single node we have

  - $f(N) = g(N) + h(N)$

Generalization of f-value definition – 2

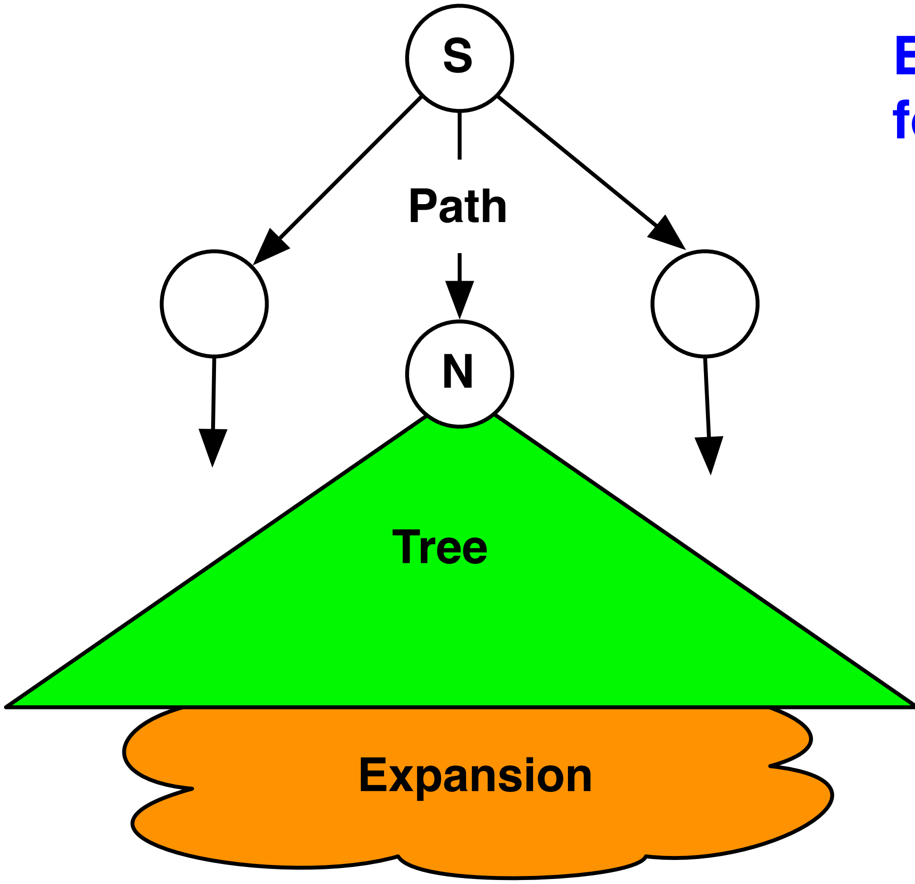
◊ For a single node we have

» $f(N) = g(N) + h(N)$

◊ For a tree with root node N we have, where the S_j are sub-trees of N

» $f(T) = \min(f(S_j))$

Expand parameter diagram



Expand is the main routine for the A* algorithm

Tree1 = Tree + Expansion

Nodes at boundary of expansion have f > Bound

Expand parameters for A*

- ◊ Prolog implementation for A* with the main routine
  - » **Expand (Path, Tree, Bound, Tree1, Solved, Solution)**
- ◊ Where
  - » **Path – between start and start of subtree Tree**

Expand parameters for A* – 2

- ◊ Prolog implementation for A* with the main routine
  - » **Expand (Path, Tree, Bound, Tree1, Solved, Solution)**
- ◊ Where
  - » **Path – between start and start of subtree Tree**
  - » **Tree – subtree at the end of Path**

Expand parameters for A* – 3

- ◊ Prolog implementation for A* with the main routine
  - » **Expand (Path, Tree, Bound, Tree1, Solved, Solution)**
- ◊ Where
  - » **Path – between start and start of subtree Tree**
  - » **Tree – subtree at the end of Path**
  - » **Bound – cost stops tree expansion**

Expand parameters for A* – 4

- Prolog implementation for A* with the main routine
  - **Expand (Path, Tree, Bound, Tree1, Solved, Solution)**
- Where
  - **Path – between start and start of subtree Tree**
  - **Tree – subtree at the end of Path**
  - **Bound – cost stops tree expansion**
  - **Tree1 – Tree expanded until f(N) > Bound**

Expand parameters for A* – 5

- ◊ Prolog implementation for A* with the main routine
  - » **Expand (Path, Tree, Bound, Tree1, Solved, Solution)**
- ◊ Where
  - » **Path – between start and start of subtree Tree**
  - » **Tree – subtree at the end of Path**
  - » **Bound – cost stops tree expansion**
  - » **Tree1 – Tree expanded until f(N) > Bound**
  - » **Solved – “yes” when goal is found**

Expand parameters for A* – 6

- Prolog implementation for A* with the main routine
  - **Expand (Path, Tree, Bound, Tree1, Solved, Solution)**
- Where
  - **Path – between start and start of subtree Tree**
  - **Tree – subtree at the end of Path**
  - **Bound – cost stops tree expansion**
  - **Tree1 – Tree expanded until f(N) > Bound**
  - **Solved – “yes” when goal is found**
  - **Solution – path to goal when it is found**

Admissibility

» **What does admissible mean?**

Admissibility – 2

- What does admissible mean?
  - Acceptable or valid

Admissibility – 3

» **What does admissible mean?**

> **Acceptable or valid**

– **Especially as evidence in a court of law**

Admissibility of a search algorithm

- When would a search algorithm be considered to be admissible?

Admissibility of a search algorithm – 2

» When would a search algorithm be considered to be admissible?

> If it is guaranteed to find an optimal solution

Admissibility of A*

» **Is A* admissible?**

Admissibility of A* – 2

» Is A* admissible?

> Yes, with necessary conditions

Admissibility of A* – 3

» **Is A* admissible?**

  > **Yes, with necessary conditions**

» **What are those conditions?**

Admissibility of A* – 4

» **Is A* admissible?**

> **Yes, with necessary conditions**

» **What are those conditions?**

> **$h(N) \leq h^*(N)$ for all nodes in the state space**

Admissibility of A* – 5

- Is A* admissible?
  - Yes, with necessary conditions
- What are those conditions?
  - $h(N) \le h^*(N)$ for all nodes in the state space
- What is h*(N)?

Admissibility of A* – 6

» **Is A* admissible?**

> **Yes, with necessary conditions**

» **What are those conditions?**

> **$h(N) \leq h^*(N)$ for all nodes in the state space**

» **What is $h^*(N)$?**

> **The actual cost of the minimum cost path from N to the goal**

Admissibility of A* – 7

» **Is A* admissible?**

> **Yes, with necessary conditions**

» **What are those conditions?**

> **$h(N) \leq h^*(N)$ for all nodes in the state space**

» **What is $h^*(N)$?**

> **The actual cost of the minimum cost path from N to the goal**

Pick an h(N) that is optimistic

Trivial Optimistic h(N)

» **What is a trivial optimistic h(N)?**

Trivial optimistic h(N) – 2

- **What is a trivial optimistic h(N)?**
  - **h(N) = 0**

Trivial optimistic h(N) – 3

» **What is a trivial optimistic h(N)?**

> **h(N) = 0**

» **What is the problem with this choice?**

Trivial optimistic h(N) – 4

» What is a trivial optimistic h(N)?

  > $h(N) = 0$

» What is the problem with this choice?

  > Gives poor guidance for a search

Trivial optimistic h(N) – 5

» **What is a trivial optimistic h(N)?**

> **h(N) = 0**

» **What is the problem with this choice?**

> **Gives poor guidance for a search**

> **All possible expansion nodes are equally "good"**

Optimal optimistic h(N)

- **What would be an optimal optimistic h(N)?**

Optimal optimistic h(N) – 2

- **What would be an optimal optimistic h(N)?**
  - **$h(N) = h^*(N)$**

Optimal optimistic h(N) – 3

- What would be an optimal optimistic h(N)?
  - $h(N) = h^*(N)$

- What is the problem in getting the optimal h(N)?

Optimal optimistic h(N) – 3

» What would be an optimal optimistic h(N)?

> $h(N) = h^*(N)$

» What is the problem in getting the optimal h(N)?

> Finding the optimal h(N) is the essence of the difficulty in finding a solution to a problem

Optimal optimistic h(N) – 4

» What would be an optimal optimistic h(N)?
  > h(N) = h*(N)

» What is the problem in getting the optimal h(N)?
  > Finding the optimal h(N) is the essence of the difficulty in finding a solution to a problem

In practice finding h(N) that minimizes the space that is searched and is admissible is the main difficulty

Distance between states

◊ Many heuristics depend upon distance between states

Distance between states – 2

◊ Many heuristics depend upon distance between states

> **For example in the travelling salesman problem it is the distance between cities**

Distance between states – 3

◊ Many heuristics depend upon distance between states

> **For example in the travelling salesman problem it is the distance between cities**

> **In the tile-puzzle it is the distance the tiles are from the goal position**

L	A	T	E
Y	O	U	R
M	I	N	D
P	A	R	

Common distance heuristics

» **What are two common distance heuristics?**

Common distance heuristics – 2

» What are two common distance heuristics?

> Euclidean distance

> Manhattan distance

Euclidean distance

» **What is Euclidean distance?**

Euclidean distance – 2

- ◊ The Euclidean distance between point (X1,Y1) and point (X2, Y2)
  - » **Is the straight line distance between the points based on Euclidean geometry**

$$D = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

Manhattan distance

» **What is Manhattan distance?**

Manhattan distance – 2

- The Manhattan distance between point (X1, Y2) and point (X2,Y2)

  » **Is the sum of the horizontal and vertical distances between the two points.**

$$D = abs(X1 - X2) + abs(Y1 - Y2)$$

Manhattan distance – 3

- Manhattan is one of the boroughs in New York with rectangular blocks. To travel between two points you can only move parallel to one or the other of the X or Y “axes” along the streets

L	A	T	E
Y	O	U	R
M	I	N	D
P	A	R	

The empty square can only travel parallel to the axes