Informed search algorithms

305450 Lecture 3-1

1

Informed search

- Idea: use an heuristic function *h(n)* for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
- Terminology
  - Heuristic – The word generally refers to a “rule of thumb”, something that may be helpful in some cases but not always. Generally held to be in contrast to “guaranteed” or “optimal”.
  - Heuristic function (evaluation function) - In search terms, a function that computes a value of a state (but does not depend on any path to that state) that may be helpful in guiding the search.

2

-หลักการของ informed search algorithms มีอยู่ว่าเราจะใช้ heuristic function ในการหาค่าประมาณของแต่ละโหนด

- ค่า heuristic จะประมาณค่าความเหมาะสมของแต่ละโหนดเพื่อที่ expand จะต่อไป

- นิยาม

  - heuristic เป็นคำที่ใช้อธิบายว่าบางสิ่งบางอย่างอาจจะมีประโยชน์ในบางกรณีแต่ก็ไม่เสมอไป

  - heuristic function หรือ evaluation function ในเรื่องของ search เป็นฟังก์ชั่นที่ใช้ในการหาค่าประมาณของ state ที่ช่วยแนะการหาคำตอบของปัญหาที่เรากำลังพิจารณา ซึ่งค่านี้ไม่เกี่ยวกับเส้นทางที่นำไปสู่ state นั้น

Informed search

- Implementation:
  - Pick "best" (measured by heuristic value of state) element of Q
  - Add path extensions anywhere in Q (it may be more efficient to keep the Q ordered in some way so as to make it easier to find the "best" element, e.g., decreasing order of desireability)
- Examples:
  - best-first search (greedy search)
  - A* search
  -

3

-Informed search จะเลือกค่า element ที่ดีที่สุดออกจากคิว

-แล้วใส่ค่าที่ expanded ออกมาที่ไหนก็ได้ในคิว

- เราจะเรียนสอง algorithms for informed search: best-first search และ A* search

Best-first search

- Evaluation function $f(n) = h(n)$ (heuristic)
- = estimate of cost from n to *goal*
-
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
-
- Best-first search expands the node that appears to be closest to goal
-

4

-ค่าการประมาณเป็นการประมาณจากโหนดหนึ่งๆไปยังปลายทาง

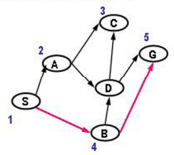
- เช่นเส้นตรงจากเมือง n ไปยังเมือง Bucharest

- best first search เลือก expand node ที่ดูเหมือนใกล้ที่สุด

Best-first search

- Pick “best” (by heuristic value) element of Q. Add path extensions anywhere in Q.

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5	(0 G B S) (4 D A S)	G,C,D,B,A,S



Heuristic Values

A=2 C=1 S=10
B=3 D=4 G=0

- Added paths in blue; heuristic value of node’s state is in front
- We show the paths in reversed order; the node’s state is the first entry

Classes of Seach

Class	Name	Operation
Any Path Uninformed	Depth-First Breadth-First	Systematic exploration of whole tree until a goal node is found.
Any Path Informed	Best-First	Uses heuristic measure of goodness of a node, e.g. estimated distance to goal.
Optimal Uninformed	Uniform-Cost	Uses path “length” measure. Finds “shortest” path.

-Breath first search, depth first search, best first search จะให้คำตอบที่เจอเป็นคำตอบแรก

-ส่วน UC จะให้คำตอบที่ดีที่สุด

Simple Search Algorithm

1. Initialize Q with search node (S) as only entry; set Visited = (S)
2. If Q is empty, fail. Else, pick some partial path N from Q
3. If state(N) is a goal, return N (we've reached a goal)
4. (Otherwise) Remove N from Q
5. Find all the children of state(N) not in Visited and create all the one-step extensions of N to each descendant.
6. Add all the extended paths to Q; add children of state(N) to Visited
7. Go to step 2.

- Critical decision:
  - Step 2: picking N from Q
  - Step 6: adding extensions of N to Q

7

Simple Search Algorithm

1. Initialize Q with search node (S) as only entry; ~~set Visited = (S)~~
2. If Q is empty, fail. Else, pick some search node N from Q
3. If state(N) is a goal, return N (we've reached a goal)
4. (Otherwise) Remove N from Q
5. Find all the children of state(N) not in ~~Visited and~~ create all the one-step extensions of N to each descendant.
6. Add all the extended paths to Q; ~~add children of state(N) to Visited~~
7. Go to step 2.

Don't use Visited for Optimal Search

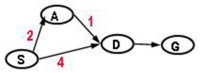
- Critical decision:
  - Step 2: picking N from Q
  - Step 6: adding extensions of N to Q

8

- ในการหาคำตอบที่ดีที่สุดของ UC นั้นจะใช้ visited list ไม่ได้เพราะจะทำให้เราพลาดค่าที่ดีที่สุดได้ ดูจากตัวอย่างในหน้าถัดไป

Why not a Visited list?

- For the any-path algorithms, the Visited list would not cause us to fall to find a path when one existed, since the path to a state did not matter.
- However, the Visited list in connection with optimal searches can cause us to miss the best path.



- **The shortest path from S to G is (S A D G)**
- **But, on extending (S), A and D would be added to Visited list and so (S A) would not be extended to (S A D)**

9

- ตัวอย่างนี้แสดงว่าการใช้ visited list ทำให้เราพลาดค่าที่ดีที่สุดในกรณีของ UC

Implementing Optimal Search Strategies

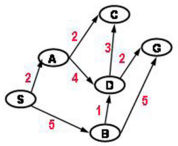
- Uniform Cost:
  - Pick best (measured by path length) element of Q
  - Add path extensions anywhere in Q

10

- UC เลือกค่าที่ดีที่สุดจากคิว ส่วนตัวที่ expanded มาจะใส่ที่ไหนในคิวก็ได้

Uniform Cost

- Like best-first except that it uses “total length (cost)” of a path instead of a heuristic value for the state
- Each link has a “length” or “cost” (which is always greater than 0)
- We want “shortest” or “least cost” path



Total path cost:	
(S A C)	4
(S B D G)	8
(S A D C)	9

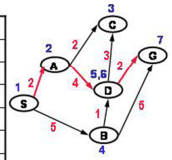
11

- การหาค่า cost ของ UC ให้เอาค่าที่ edge จากโหนดเริ่มต้นบอกกันไปเรื่อยๆถึงโหนดที่กำลังพิจารณา

Implementing Uniform Cost

- Pick best (by path length) element of Q. Add path extensions anywhere in Q

	Q
1	(0 S)
2	(2 A S) (5 B S)
3	(4 C A S) (6 D A S) (5 B S)
4	(6 D A S) (5 B S)
5	(6 D B S) (10 G B S) (6 D A S)
6	(8 G D B S) (9 C D B S) (10 G B S) (6 D A S)
7	(8 G D A S) (9 C D A S) (8 G D B S) (9 C D B S) (10 G B S)



- Added paths in blue; underlined paths are chosen for extension
- We show the paths in reversed order; the node’s state is the first entry

12

- ตัวอย่างของ UC without visited list to guarantee optimal solution

Why not stop on first visiting a goal?

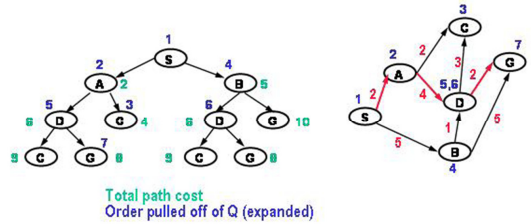
- When doing Uniform cost, it is not correct to stop the search when the first path to a goal is generated, that is, when a node whose state is a goal is added to Q.
- We must wait until such a path is pulled off the Q and tested in step 3. It is only at this point that we are sure it is the shortest path to a goal since there are no other shortest path that remain unexpanded.
- This contrasts with the non-optimal searches where the choice of where to test for a goal was a matter of convenience and efficiency, not correctness.
- In the previous example, a path to G was generated at step 5, but it was a different, shorter, path at step 7 that we accepted.

13

-ใน UC เราจะไม่หยุดเมื่อได้ค่าคำตอบครั้งแรก เนื่องจากอาจจะมีคำตอบที่ดีกว่าก็ได้ เพราะฉะนั้นต้องพิจารณาต่อไปจนกว่าจะหมด

-ซึ่งจะต่างจาก non-optimal search algorithms ที่จะให้ค่าคำตอบเมื่อเจอครั้งแรกเลย

Uniform Cost



UC enumerates paths in order of total path cost!

14

Classes of Searches

Class	Name	Operation
Any Path Uninformed	Depth-First Breadth-First	Systematic exploration of whole tree until a goal node is found.
Any Path Informed	Best-First	Uses heuristic measure of goodness of a node, e.g. estimated distance to goal.
Optimal Uninformed	Uniform-Cost	Uses path "length" measure. Finds "shortest" path.
Optimal Informed	A*	Uses path "length" measure and heuristic Finds "shortest" path

15

- อีกตัวอย่างของ optimal search algorithm คือ A*

Goal Direction

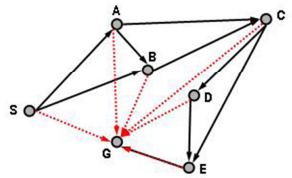
- UC is really trying to identify the shortest path to every state in the graph in order. It has no particular bias to finding path to a goal early in the search.
- We can introduce such a bias by means of heuristic function h(N), which is an estimate (h) of the distance from a state to the goal.
- Instead of enumerating paths in order of just length (g), enumerate paths in terms of f = estimated total path length = g + h.
- An estimate that always underestimates the real path length to the goal is called admissible.
- Use of an admissible estimate guarantees that UC will still find the shortest path.
- UC with an admissible estimate is known as A* search

16

-UC พยายามหา shortest path จากจุดเริ่มต้นไปที่ทุกๆโหนดในกราฟ โดยไม่สนใจว่า path นั้นจะไปถึงปลายทางได้เร็วกว่าหรือไม่

- เราสามารถพิจารณาความสำคัญของการไปถึงปลายทางโดยใช่ค่าการประมาณระยะทางจาก state หนึ่งๆไปถึงปลายทาง
- ให้ค่า g(N) เป็นค่า path cost ที่เกิดจากการบวกกันของ cost ของ edges จากโหนดเริ่มต้น ถึงโหนดปัจจุบัน
- ให้ค่า h(N) เป็นค่า heuristic ประมาณค่า path length จาก โหนดปัจจุบันไปถึงโหนดปลายทาง
- ให้ค่า f(N) ซึ่งเป็นการรวมกันของ g(N) and h(N)
- A* search จะใช้ค่า f(N) ในการเลือกโหนดที่ดีที่สุดในการ expand ซึ่งคือมีค่า f(N) น้อยที่สุด
- ค่าประมาณจำเป็นต้องน้อยกว่าค่าจริงเสมอเพื่อให้การหาคำตอบทำได้ถูกต้อง – admissible condition

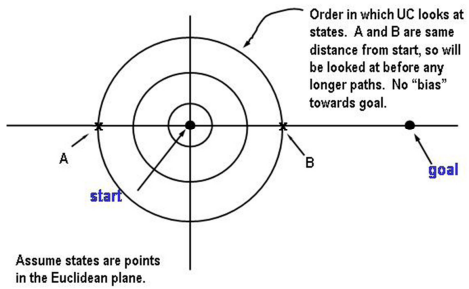
Straight-line estimate



17

- การประมาณค่า ระยะทางจากโหนดไปปลายทางโดยใช้ระยะทางเส้นตรง

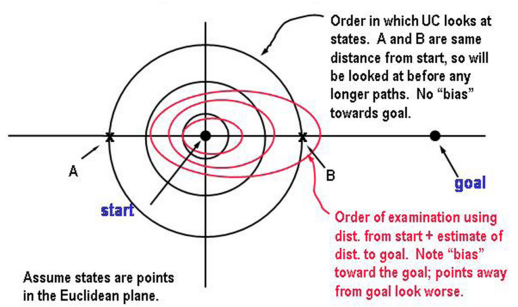
Why use estimate of goal distance?



18

- UC ให้ความสำคัญกับทุกโหนดเท่ากันหมด โดยไม่สนใจว่าจะไปถึง goal เร็วหรือช้ากว่ากัน

Why use estimate of goal distance?



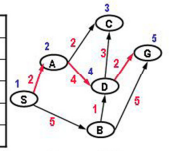
19

- แต่ถ้าเราพิจารณาระยะห่างจาก goal ด้วยเราจะไปถึงปลายทางได้เร็วขึ้น นี่คือ idea of A* search

A*

- Pick "best" (by path length + heuristic) element of Q. Add path extensions anywhere in Q.

	Q
1	(0 S)
2	(4 A S) (8 B S)
3	(5 C A S) (7 D A S) (8 B S)
4	(7 D A S) (8 B S)
5	(8 G D A S) (10 C D A S) (8 B S)



Heuristic Values

A=2	C=1	S=0
B=3	D=1	G=0

- Added paths in blue; underlined paths are chosen for extension
- We show the paths in reversed order; the node's state is the first entry

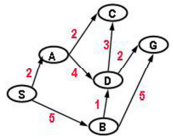
20

Not all heuristics are admissible

Given the link lengths in the figure, is the table of heuristic values that we used in our earlier best-first example an admissible heuristic?

No!

A is ok
B is ok
C is ok
D is too big, needs to be <= 2
S is too big, can always use 0 for start



Heuristic Values

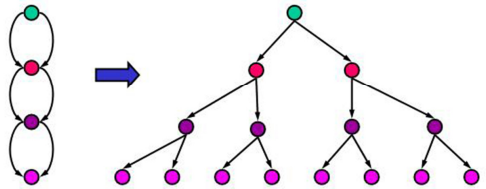
A=2	C=1	S=10
B=3	D=4	G=0

21

-การใช้ A* search นั้นต้องตรงตามเงื่อนไขของ admissible นั่นคือค่าการประมาณ heuristic จะต้องมีค่าน้อยกว่าค่าจริง

-เช่นกรณีของ D ค่าจริงจาก D to G เท่ากับ 2 ดังนั้นค่าประมาณควรน้อยกว่าหรือเท่ากับสอง

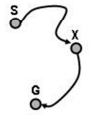
States vs Paths



22

Dynamic Programming Optimality Principle (and the Expanded list)

- Given that path length is additive, the shortest path from S to G via a state X is made up of the shortest path from S to X and the shortest path from X to G. This is the "dynamic programming optimality principle".



23

-ถ้าเราให้ว่าค่า นั้นมีค่าบวกรวมกันได้ ค่า shortest path จาก S ไป G โดยผ่าน X นั้นจะประกอบด้วยค่า shortest path จาก S ไป X และ จาก X ไป G

-สิ่งนี้เราเรียกว่า dynamic programming

Dynamic Programming Optimality Principle (and the Expanded list)

- Given that path length is additive, the shortest path from S to G via a state X is made up of the shortest path from S to X and the shortest path from X to G. This is the "dynamic programming optimality principle".
- This means that we only need to keep the single best path from S to any state X; if we find a new path to a state already in Q, discard the longer one.
- Note that the first time UC pulls a search node off of Q whose state is X, this path is the shortest path from S to X. This follows from the fact that UC expands nodes in order of actual path length.
- So, once we expand one path to state X, we don't need to consider (extend) any other paths to X. We can keep a list of these states, call it Expanded. If the state of the search node we pull off of Q is in the Expanded list, we discard the node. When we use the Expanded list this way, we call it "strict".
- Note that UC without this is still correct, but inefficient for searching graphs.

24

-นั่นหมายความว่าเราจำเป็นต้องเก็บแค่ค่า best path เพียงอันเดียวจาก S to X ถ้าเราพบอีก path หนึ่งที่นำไปสู่ state ที่อยู่ในคิวอยู่แล้ว เราไม่จำเป็นต้องพิจารณา

- Note ว่าครั้งแรกที่ UC เอาโหนดออกจากคิวนั้นจะเลือกทางที่เป็น shortest path จาก S ไป X อยู่แล้ว

- ดังนั้นเมื่อเราได้ expand path หนึ่งจาก S to X เราไม่จำเป็นต้องพิจารณาทางอื่นๆที่ไปถึง X เราสามารถเก็บค่าโหนด พวกนี้ไว้ได้ซึ่งเรียกว่า Expanded list

Simple Search Algorithm
(Uniform Cost)

1. Initialize Q with search node (S) as only entry;
2. If Q is empty, fail. Else, pick least cost search node N from Q
3. If state(N) is a goal, return N (we've reached the goal)
4. (Otherwise) Remove N from Q.
5. -
6. Find all the children of state(N) and create all the one-step extensions of N to each descendant.
7. Add all the extended paths to Q;
8. Go to step 2.

25

- This algorithm is used for uniform cost without visited list เพราะจาก slide 9 เราพบว่าการใช้ visited list อาจจะทำให้เราพลาด ที่ดีที่สุดได้

Simple Search Algorithm
(Uniform Cost + strict Expanded list)

1. Initialize Q with search node (S) as only entry; set Expanded = ( )
2. If Q is empty, fail. Else, pick least cost search node N from Q
3. If state(N) is a goal, return N (we've reached the goal)
4. (Otherwise) Remove N from Q.
5. if state(N) in Expanded, go to step 2, otherwise add state(N) to Expanded.
6. Find all the children of state(N) (Not in Expanded) and create all the one-step extensions of N to each descendant.
7. Add all the extended paths to Q; if descendant state already in Q, keep only shorter path to the state in Q.
8. Go to step 2.

26

-การใช้ Uniform cost กับ strict expanded list จะมีการเปลี่ยนแปลงจาก simple search algorithm ใน slide 25 ที่ใช้กับการหาค่า uniform cost search without expanded list ดังนี้

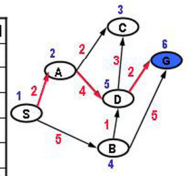
-ใส่ค่าโหนดใน expanded list เมื่อโหนดนั้นถูก expandeds

- ใน step 5 ถ้าโหนดที่กำลังพิจารณาอยู่ใน expanded list แล้วก็ไม่ต้องสนใจ กลับไปทำ step 2
- ใน step 7 ถ้าโหนดที่กำลังพิจารณามีอยู่แล้วใน Q ให้เลือกเก็บโหนดที่มีค่า path cost น้อยกว่า

Uniform Cost (with strict Expanded list)

- Pick best (by path length) element of Q. Add extensions anywhere to Q.

	Q	Expanded
1	(0 S)	
2	(2 A S) (5 B S)	S
3	(4 C A S) (6 D A S) (5 B S)	S,A
4	(6 D A S) (5 B S)	S,A,C
5	~~(6 D B S)~~ (10 G B S) (6 D A S)	S,A,C,B
6	(8 G D A S) ~~(9 C D A S)~~ ~~(10 G B S)~~	S,A,C,B,D



27

(6 D B S) – select shorter - step 7

(9 C D A S) – ignore as C is already in the expanded list – step 5

(10 G B S) – select shorter i.e., (8 G D A S) – step 7

A* (without strict Expanded list)

- **Let g(N) be the path cost of n, where n is a search tree node, i.e. a partial path.**
- **Let h(N) be h(state(N)), the heuristic estimate of the remaining path length to the goal from state(N).**
- **Let f(N) = g(N) + h(state(N)) be the total estimated path cost of a node, i.e. the estimate of a path to a goal that starts with the path given by N.**
- **A* picks the node with lowest f value to expand**
- **A* (without expanded list) and with admissible heuristic is guaranteed to find optimal paths – those with smallest path cost.**

28

-ให้ค่า g(N) เป็นค่า path cost ที่เกิดจากการบวกกันของ cost ของ edges จากโหนดเริ่มต้น ถึงโหนดปัจจุบัน

- ให้ค่า h(N) เป็นค่า heuristic ประมาณค่า path length จาก โหนดปัจจุบันไปถึงโหนดปลายทาง

- ให้ค่า f(N) ซึ่งเป็นการรวมกันของ g(N) and h(N)

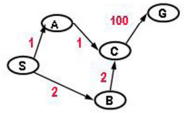
- A* search จะใช้ค่า f(N) ในการเลือกโหนดที่ดีที่สุดในการ expand ซึ่งคือมีค่า f(N) น้อยที่สุด

- A* search without expanded list จะให้ค่าคำตอบที่ดีที่สุด (shortest path) ถ้าหากว่า heuristic ถูกต้องตามเงื่อนไข admissible

A* and the strict Expanded list

- The strict Expanded list (also known as a Closed list) is commonly used in implementations of A* but, to guarantee finding optimal paths, this implementation requires a stronger condition for a heuristic than simply being an underestimate.
- Here's a counterexample: The heuristic values listed below are all underestimates but A* using an Expanded list will not find the optimal path. The misleading estimate at B throws the algorithm off, C is expanded before the optimal path to it is found.

	Q	Expanded
1	(0 S)	
2	(3 B S) (101 A S)	S
3	(94 C B S) (101 A S)	B, S
4	(101 A S) (104 G C B S)	C, B, S
5	(104 G C B S)	A, C, B, S



Heuristic Values

A=100 C=90 S=0

B=1 G=0

Added paths in blue; underlined paths are chosen for extension.

We show the paths in reversed order; the node's state is the first entry.

29

-โดยปกติ A* ที่ไม่ใช้ strict expanded list สามารถหาค่า optimal solution ได้ ถ้าหากว่าค่า heuristic มีค่าน้อยกว่าค่าที่เป็น cost จริง (ที่เรียกว่า admissible)

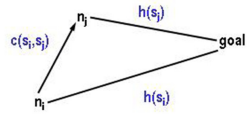
-แต่ถ้าใช้ strict expanded list ต้องมีเงื่อนไขเพิ่มขึ้นเนื่องจากเงื่อนไขของ admissible อย่างเดียวอาจไม่ทำให้ได้ค่า optimal solution เสมอไป

-อย่างเช่นในตัวอย่างนี้ เราจะเห็นได้ว่าเราควร expand ค่า (101 A S) ต่อถึงแม้ว่าจะได้ค่า G มาแล้วก็ตาม (เพราะจากที่เราเคยอธิบายไปว่า search algorithms ที่ให้ค่า optimal solution จะไม่หยุดเมื่อได้ path แรกมาแต่จะหาต่อไปเรื่อยๆจนกว่าจะได้ path ที่ดีที่สุด)

-ดังนั้นในตัวอย่างนี้เราควร expand ค่า (101 A S) ต่อไปด้วย ซึ่งจะทำให้ได้ค่าคำตอบสุดท้ายเป็น (102 G C A S) แต่เนื่องจากว่าเราใช้ strict expanded list ซึ่งค่า C ได้ถูกเก็บไว้แล้ว ดังนั้นตัว search algorithm จึงหยุด และได้คำตอบเป็น (104 G C B S) ซึ่งไม่ใช่ optimal solution

Consistency

- To enable implementing A* using the strict Expanded list, H needs to satisfy the following consistency (also known as monotonicity) conditions.
  - $h(s_i) = 0$, if n_i is a goal
  - $h(s_i) - h(s_j) \cdot c(s_i,s_j)$, for n_j a child of n_i
- That is, the heuristic cost in moving from one entry to the next cannot decrease by more than the arc cost between the states. This is a kind of *triangle inequality*. This condition is a highly desirable property of a heuristic function and often simply assumed (more on this later).



30

-เงื่อนไขที่เพิ่มเติมเข้ามาเรียกว่า Consistency (กรุณาอย่าสับสนกับ consistency ใน CSP)

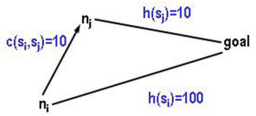
- Consistency condition ในเรื่องของ A* search นั้นมีเงื่อนไขอยู่ว่า

  - ค่า heuristic ของ goal state มีค่าเป็น 0 เสมอ, และ

  - ค่าความแตกต่างระหว่าง heuristic ของ state Si กับ state Sj จะต้องมีค่าน้อยกว่าหรือเท่ากับ ค่า cost ระหว่าง states ทั้งสอง Note ว่า state Sj เป็นลูกของ Si

Consistency Violation

- A simple example of a violation of consistency.
- $h(s_i) - h(s_j) \cdot c(s_i,s_j)$
- In example, 100-10 > 10
- If you believe goal is 100 units from n_i, then moving 10 units to n_j should not bring you to a distance of 10 units from the goal.



31

-รูปนี้เป็นตัวอย่างของการให้ค่า heuristic ที่ไม่ตรงตามเงื่อนไขของ consistency

-จากตัวอย่างจะเห็นได้ว่าความแตกต่างของค่า heuristics ระหว่างสองโหนดต่างกัน 90 ซึ่งมากกว่าค่า cost ระหว่างทั้งสองโหนดซึ่งคือ 10 ดังนั้นไม่ถูกต้องตาม consistency condition

A* and the strict Expanded list

- Note that consistency of the heuristic is only necessary for optimality when we want to discard paths from consideration, for example, because a state has already been expanded. Otherwise, plain A* without using an expanded only requires only that the heuristic be admissible to guarantee optimality.
- This illustrates that A* without an Expanded list has no trouble coping with the example we saw earlier that showed the pitfalls of using a strict Expanded list. This heuristic is not consistent but it is an underestimate and that is all that is needed for A* without an Expanded list to guarantee optimality.
- The extension of A* to use a strict expanded list is just like the extension to uniform-cost search. In fact, it is the identical algorithm except that it uses f values instead of g values. But, we stress that for this algorithm to guarantee finding optimal paths, the heuristic must be consistent.

32

- Consistency ของ heuristic จำเป็นก็ต่อเมื่อเราต้องการหาค่า optimality โดยสามารถลดค่าของจำนวน paths ที่ต้องพิจารณา เช่นดูว่า state ไหนที่ expanded ไปแล้วบ้าง มิฉะนั้นค่า heuristic with admissible ก็เพียงพอที่จะการันตีว่าเราจะได้คำตอบที่ optimal

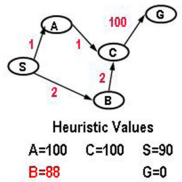
- ตัวอย่างที่ผ่านมาใน slide ที่ 21 เราจะเห็นได้ว่า A* search ที่ไม่ใช้ Expanded List ไม่มีปัญหาในการหาค่า optimal solution แม้ว่าค่า heuristics จะไม่ถูกต้องตามเงื่อนไขของการ consistency แต่ยังไงก็ตามค่า heuristic ต้องน้อยกว่าค่า cost จริงจากโหนดนั้นไปยัง goal (ซึ่งคือถูกต้องตาม admissible condition)

- การใช้ A* ร่วมกับ Strict Expanded List นั้นใช้ algorithm เหมือนกันกับกรณีของ uniform cost search เพียงแต่ว่าค่าที่ใช้ในการคิด cost ของแต่ละโหนดจะเป็น f แทนที่จะเป็น g (กลับไปทวน slide 11, 20 สำหรับการหาค่า f and g) แต่เน้นว่าการหาค่าคำตอบที่ดีที่สุดในกรณีของนั้น ค่า heuristics ต้องถูกต้องตาม consistency condition

A* and the strict Expanded list

If we modify the heuristic in the example we have been considering so that it is consistent, as we have done here by increasing the value of h(B), then A* (even when using a strict Expanded list) will work.

	Q	Expanded
1	(90 S)	
2	(90 B S) (101 A S)	S
3	(101 A S) (104 C B S)	A, S
4	(102 C A S) (104 C B S)	C,A,S
5	(102 G C A S)	G,C,A,S



Heuristic Values

A=100 C=100 S=90

B=88 G=0

33

h(S) – h(B) ≤ c(S,B)

90 – h(B) ≤ 2

90- 2 ≤ h(B)

88 ≤ h(B)

- ใน step ที่ 4 เนื่องจาก C ไม่ได้อยู่ใน strict expanded list เพราะฉะนั้นเรา expand A to C
- และเนื่องจาก (102 C A S) มีค่า cost น้อยกว่า (104 C B S) เราเอาตัวมากออกจากคิว เก็บเฉพาะตัวที่มีค่าน้อยกว่า
-

Dealing with inconsistent heuristic

- What can we do if we have an inconsistent heuristic but we still want optimal paths?
- Modify A* so that it detects and corrects when inconsistency has led us astray:
- Assume we are adding $node_1$ to Q and $node_2$ is present in Expanded list with $node_1$.state = $node_2$.state.
- Strict –
  - do not add $node_1$ to Q
- Non-Strict Expanded list –
  - If $node_1$.path_length < $node_2$.path_length, then
    - Delete $node_2$ from Expanded list
    - Add $node_1$ to Q

34

-ที่นี้จะมีปัญหาว่าถ้าหากค่า heuristics ที่ให้มาไม่ถูกต้องตาม consistency condition แล้วเราจะหาค่าคำตอบที่ดีที่สุดได้อย่างไร

- กรณีนี้ให้ปรับเปลี่ยน algorithm ของ A* search ดังนี้

-เดิม – เราจะไม่ใส่โหนดที่ expanded แล้วลงใน Q (see step 5 of algorithm in slide 26)

- เราแก้เป็นว่าให้เลือกเก็บโหนดที่มีค่า path length น้อยกว่าแทน และลบโหนดเดิมที่อยู่ใน Expanded List ออก