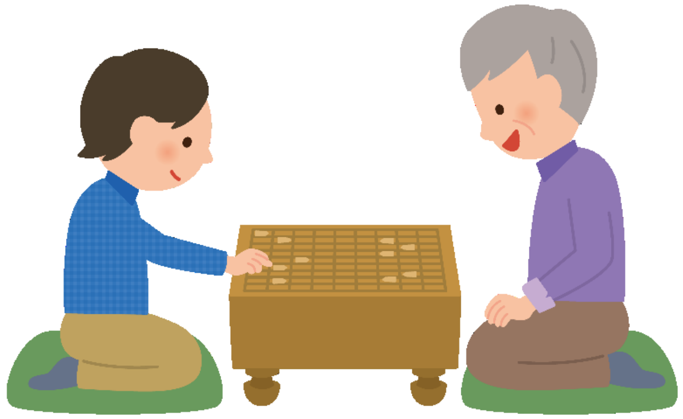
Heuristic Search

Topics of this lecture

- What are heuristics?
- What is heuristic search?
- Best first search
- A* algorithm
- Generalization of search problems

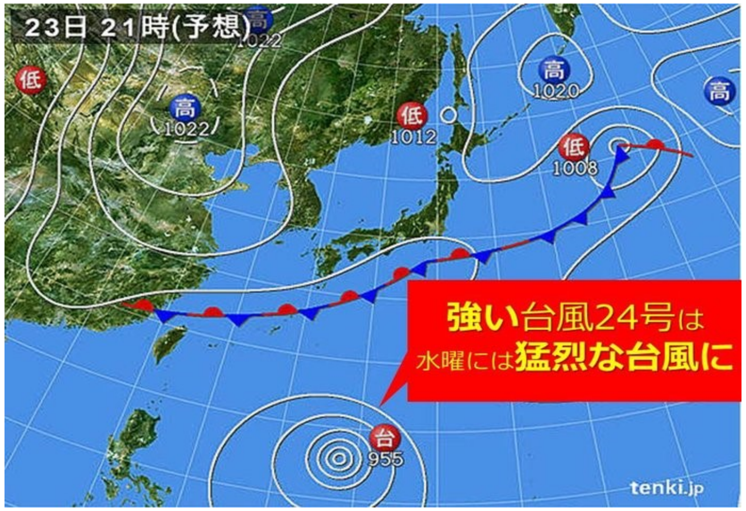
What are heuristics?

- Heuristics are know-hows obtained through a lot experiences.
- Heuristics often enable us to make decisions quickly without thinking deeply about the reasons.
- In many cases, heuristics are “tacit knowledge” that cannot be explained verbally.
- The more experiences we have, the better the heuristics will be.



Why heuristic search?

- Based on the heuristics, we can get good solutions without investigating all possible cases.
- In fact, a deep learner used in Alpha-Go can learn heuristics for playing Go-game, and this learner can help the system to make decisions more efficiently.
- Without using heuristics, many AI-related problems cannot be solved at all (may take many years to get a solutions).



Patterns memorized so far can be used to predict the move of the typhon. We do not have to calculate the results based on the raw data.

Best first search

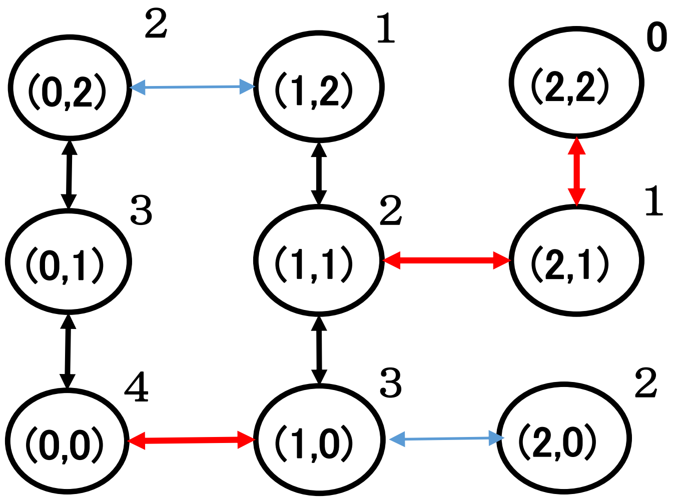
- The basic idea of best first search is similar to uniform cost search.
- The only difference is that the “cost” (or in general, the evaluation function) is estimated based on some heuristics, rather than the real one calculated during search.

Best first search

- Step 1: Put the initial node x0 and its **heuristic value** H(x0) to the open list.
- Step 2: Take a node x from the top of the open list. If the open list is empty, stop with failure. If x is the target node, stop with success.
- Step 3: Expand x and get a set S of child nodes. Add x to the closed list.
- Step 4: For each x' in S but not in the closed list, estimate its heuristic value H. If x' is not in the open list, put x' along with the edge (x,x') and H into the open list; otherwise, if H is smaller than the old value H(x'), update x' with the new edge and the new heuristic value.
- Step 5: Sort the open list according to the heuristic values of the nodes, and return to Step 2.

Example 2.6 p. 25

Steps	Open List	Closed List
0	{(0,0),4}	--
1	{(1,0),3} {(0,1),3}	(0,0)
2	{(2,0),2} {(1,1),2} {(0,1),3}	(0,0) (1,0)
3	{(1,1),2} {(0,1),3}	(0,0) (1,0) (2,0)
4	{(2,1),1} {(1,2),1} {(0,1),3}	(0,0) (1,0) (2,0) (1,1)
5	{(2,2),0} {(1,2),1} {(0,1),3}	(0,0) (1,0) (2,0) (1,1) (2,1)
6	(2,2)=target node.	(0,0) (1,0) (2,0) (1,1) (2,1)



The A* Algorithm

- We cannot guarantee to obtain the optimal solution using the “best-first search”. This is because the true cost from the initial node to the current node is ignored.
- The A* algorithm can solve this problem (A=admissible).
- In the A* algorithm, the cost of a node is evaluated using both the estimated cost and the true cost as follows:

$$F(x) \quad = \quad C(x) \quad + \quad H(x)$$

- It has been proved the A* algorithm can obtain the best solution provided that the estimated cost H(x) is always smaller (conservative) than the best possible value H*(x).

The A* Algorithm

- Step 1: Put the initial node x0 and its cost F(x0)=H(x0) to the open list.
- Step 2: Get a node x from the top of the open list. If the open list is empty, stop with failure. If x is the target node, stop with success.
- Step 3: Expand x to get a set S of child nodes. Put x to the closed list.

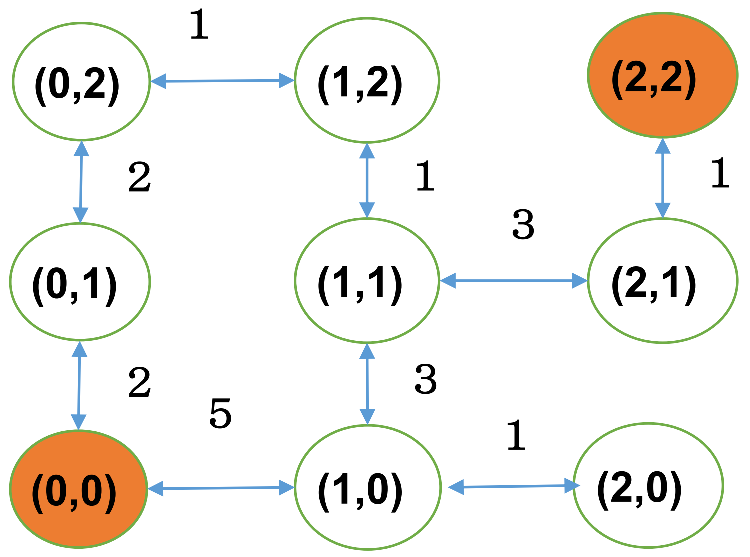
The A* Algorithm

- Step 4: For each x’ in S, find its cost

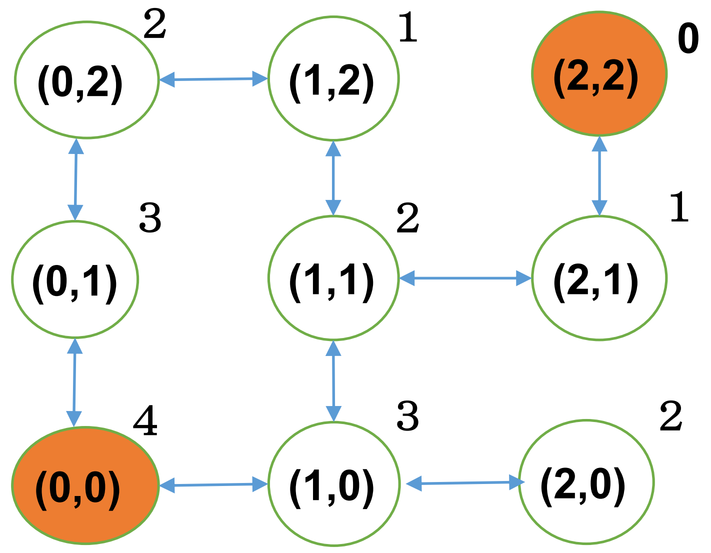
$$F = F(x) + d(x, x') + [H(x') - H(x)]$$

  – If x’ is in the closed list but the new cost is smaller than the old one, move x’ to the open list and update the edge (x,x’) and the cost.
  – Else, if x’ is in the open list, but the new cost is smaller than the old one, update the edge (x,x’) and the cost.
  – Else (if x’ is not in the open list nor in the closed list), put x’ along with the edge (x,x’) and the cost F to the open list.
- Step 5: Sort the open list according to the costs of the nodes, and return to Step 2.

Example 2.7 p. 27



True cost



Estimated cost

Example 2.7 p. 27

Steps	Open List	Closed List
0	{(0,0),--, 4}	--
1	{(0,1),5} {(1,0),8}	{(0,0),4}
2	{(0,2),6} {(1,0),8}	{(0,0),4} {(0,1),5}
3	{(1,2),6} {(1,0),8}	{(0,0),4} {(0,1),5} {(0,2),6}
4	{(1,0),8} {(1,1),8}	{(0,0),4} {(0,1),5} {(0,2),6} {(1,2),6}}
5	{(1,1),8} {(2,0),8}	{(0,0),4} {(0,1),5} {(0,2),6} {(1,2),6}} {(1,0),8}
6	{(2,0),8} {(2,1),10}	{(0,0),4} {(0,1),5} {(0,2),6} {(1,2),6}} {(1,0),8} {(1,1),8}
7	{(2,1),10}	{(0,0),4} {(0,1),5} {(0,2),6} {(1,2),6}} {(1,0),8} {(1,1),8} {(2,0),8}
8	{(2,2),10}	{(0,0),4} {(0,1),5} {(0,2),6} {(1,2),6}} {(1,0),8} {(1,1),8} {(2,0),8} {(2,1),10}
9	(2,2)=target node	{(0,0),4} {(0,1),5} {(0,2),6} {(1,2),6}} {(1,0),8} {(1,1),8} {(2,0),8} {(2,1),10}

Property of the A* Algorithm

- The A* algorithm can obtain the best solution because it considers both the cost calculated up to now and the estimated future cost.
- A sufficient condition for obtaining the best solution is that the estimated future cost is smaller than the best possible value.
- If this condition is not satisfied, we may not be able to get the best solution. This kind of algorithms are called A algorithms.

Comparison between A* algorithms

- Suppose that A1 and A2 are two A* algorithms. If for any node x we have H1(x)>H2(x), we say A1 is more efficient.
- That is, if H(x) is closer to the best value H*(x), the algorithm is better.
- If H(x)=H*(x), we can get the solution immediately.
- On the other hand, if H(x)=0 for all x (the most conservative case), the A* algorithm becomes the uniform cost search algorithm.

Some heuristics for finding H(x)

- For any given node x, we need an estimation function to find H(x).
- For maze problem, for example, H(x) can be estimated by the Manhattan distance between the current node the target node. This distance is usually smaller than the true value (and this is good) because some edges may not exist in practice.
- For more complex problems, we may need a method (e.g. neural network) to learn the function from experiences or observed data.

Generalization of Search

- Generally speaking, search is a problem for finding the best solution x* from a given ***domain*** D.
- Here, D is a set containing all "states" or candidate solutions.
- Each state is often represented as a n-dimensional state vector or ***feature vector***.
- To see if a state x is the best (or the desired) one or not, we need to define an ***objective function*** (e.g. cost function) f(x). The problem can be formulated by

Min f(x), for x in D

Optimization and search

- The result of search may not be the best one (e.g. depth-first search).
- However, search with an objective function often results in more desirable solutions (e.g. shortest-path search).
- The goal of heuristic search is to minimize the objective function with the minimum effort (e.g. best-first search)

→ the solution may not be optimal.

Various optimization problems

- If D is the whole n-D Euclidean space, the problem is un-constrained.
- If D is defined by some functions, the problem is constrained, and is often formulated by

Min f(x)
s. t. x in D

- All x in D are called feasible solutions, and D is usually defined a group of functions as follows:

D={x | gi(x) <= 0, i=1,2,…}

Various optimization problems

- If the objective function f(x) and all constrain functions are linear, the problem is a linear programming problem, and can be solved by using the simplex algorithm.
- If f(x) is quadratic (2nd order) and the constrain functions are linear, the problem is called a quadratic optimization problem.
- If f(x) is a convex function, and D is a convex set, we can easily find the "global optimal" solution.

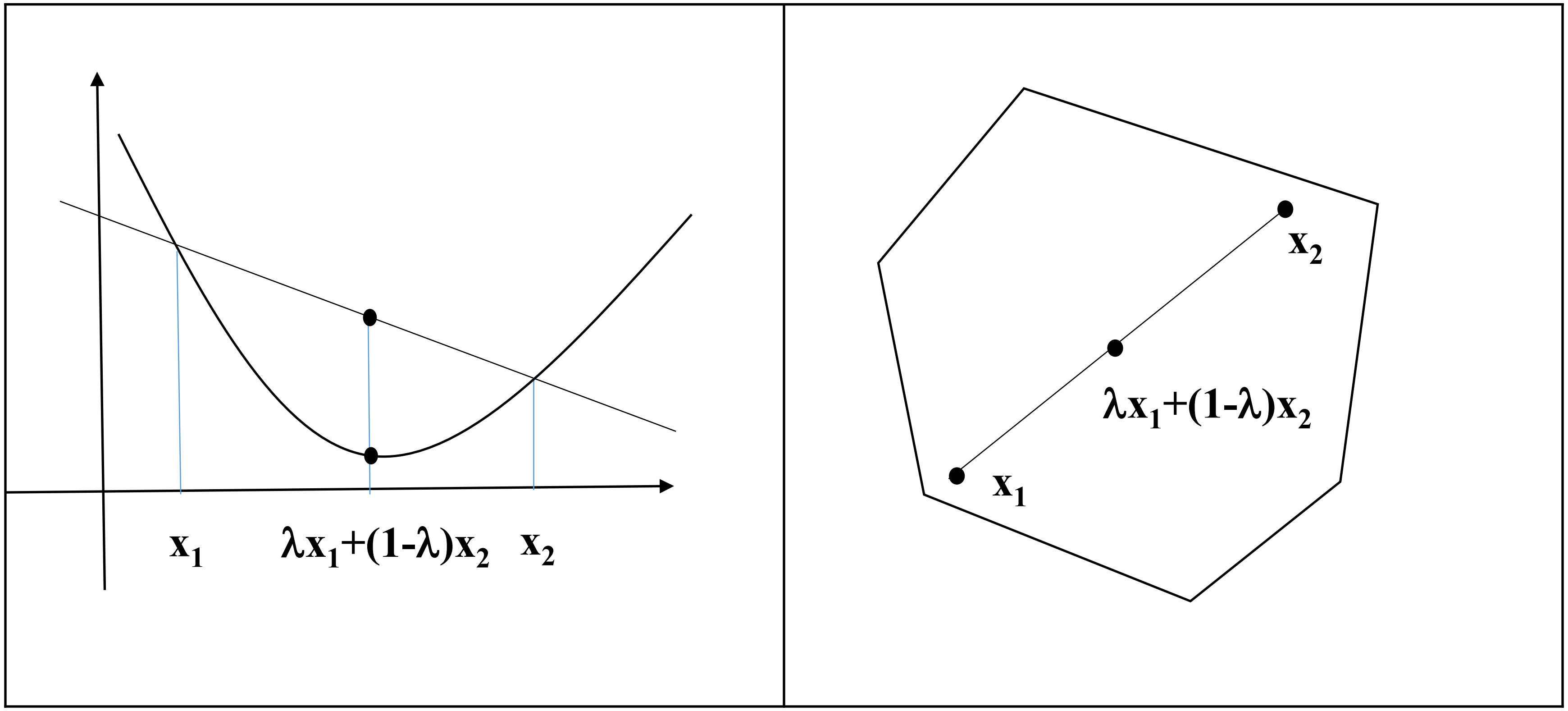
Local and global optimal solution

- If f(x)>=f(x*) for x in ***epsilon-neighborhood*** (see the first line of p. 31) of x*, x* is a local optimal solution

- If f(x)>=f(x*) for all x in D, it is a global optimal solution.

- The goal of search is to find the global optimal solution, but this is often difficult.
- In practice, we may just find some local but useful solution.

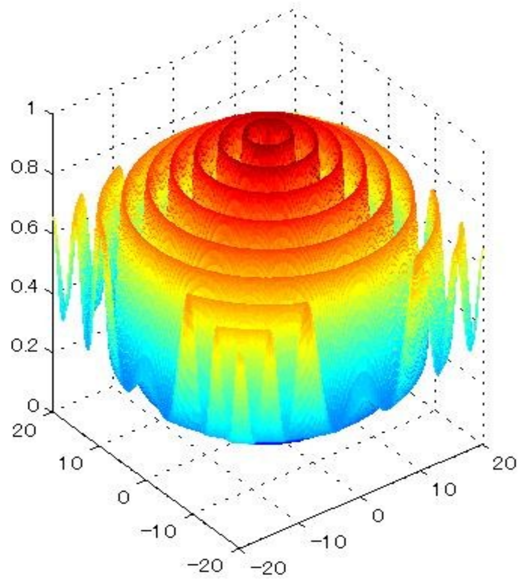
Convex function and convex set

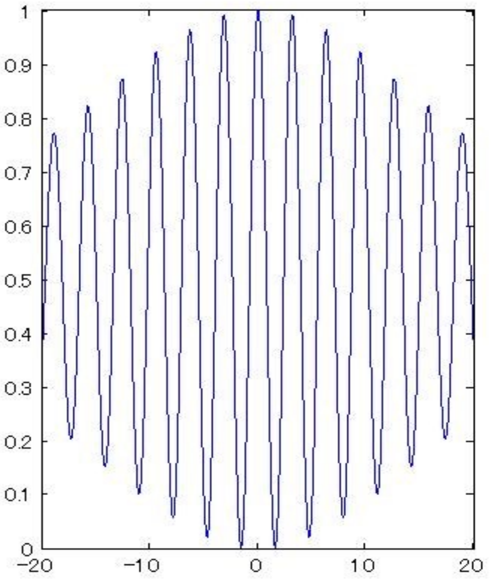


$$\forall \mathbf{x}_1, \mathbf{x}_2 \in D, \forall \lambda \in [0,1], f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \le \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

A simple but difficult problem

Schaffer function F6 and its segment for x2=0





$$f(\mathbf{x}) = 0.5 - \frac{(\sin\sqrt{x_1^2 + x_2^2})^2 - 0.5}{(1.0 + 0.001(x_1^2 + x_2^2))^2}$$

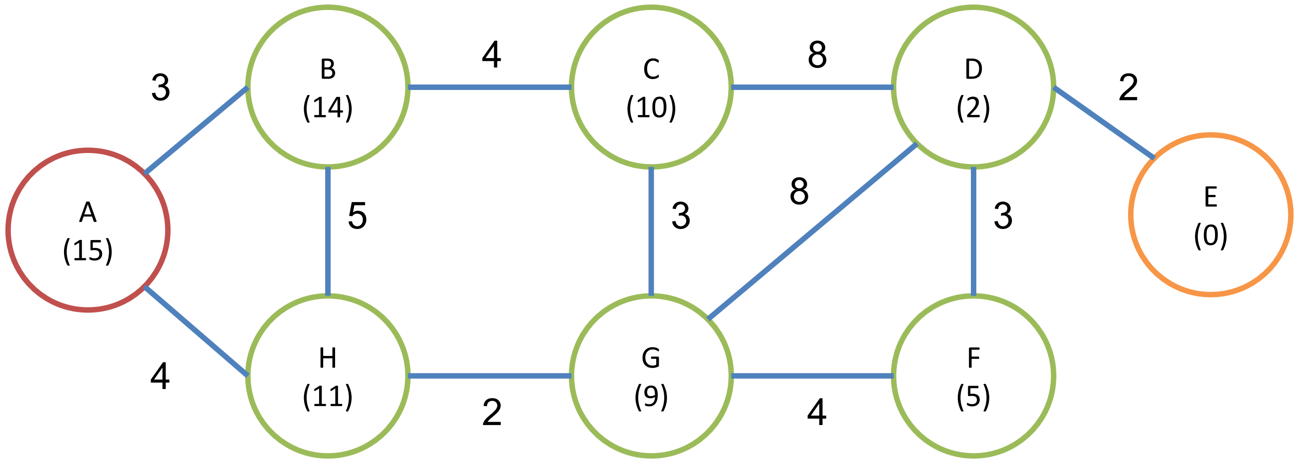
Homework for lecture 4 (1)

Ex. 2.4 (p. 29 in the textbook)

- For the search graph shown in Fig. 2.7, add an edge with cost 1 between nodes (0,1) and (1,1), find the shortest path between the initial node (0,0) and the target node (2,2) using A* algorithm.
- Summarize the search processing in the same as Table 2.9 in p. 28 of the textbook.
- Submit the result (in hardcopy) to the TA within the exercise class.

Homework for lecture 4 (2)

- Based on the skeleton problem, complete a program containing the uniform cost search, the best-first search, and the A* algorithm.
- Here, we suppose that A is the initial node, and E is the target.
- The cost of each edge and the heuristic value of each node are also given in the figure.



Quizzes

- What is result of uniform cost search?
- What are the advantage and dis-advantage of the best-first search algorithm, compared with the uniform cost search algorithm?
- What condition is needed to guarantee the best solution when we use the A* algorithm?
- Write the definitions of a location optimal solution and the global optimal solution.
  - Local optimal solution:
  - Global optimal solution: