Part II Problem - Solving by Searching

Many problems can be translated into finding the best path in a graph. To solve them, we need to first define the corresponding graph and what do we mean by “best/good”.

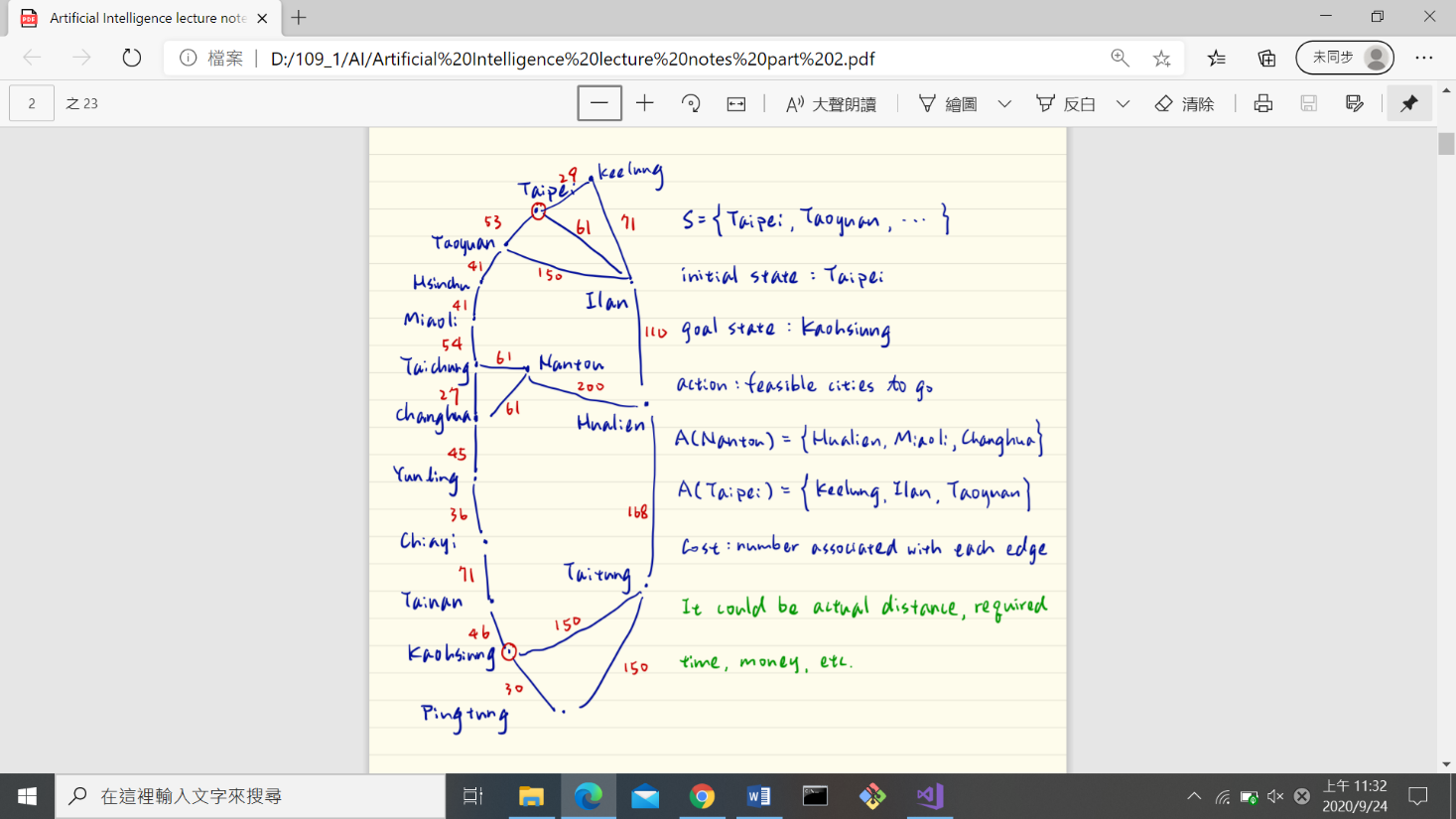
A problem can be defined by

* A set of states , where S0 is initial state.
* For each , a set of feasible actions at that state.
* Transition model which describes the consequence of doing action at state .

- Deterministic: Result of is deterministic

- Probabilistic: Result of follows a distribution

* Goal test: Determines whether a state is a goal state.
* Costs: is the cost/reward/utility when we take action at the state and then reach . The path cost is the sum of the step costs along the path. We assume to be non-negative.



Example: (Navigation) Jerry wants to go from Taipei to Kaohsiung, but he has very limited knowledge about the geography of Taiwan.

initial state: Taipei

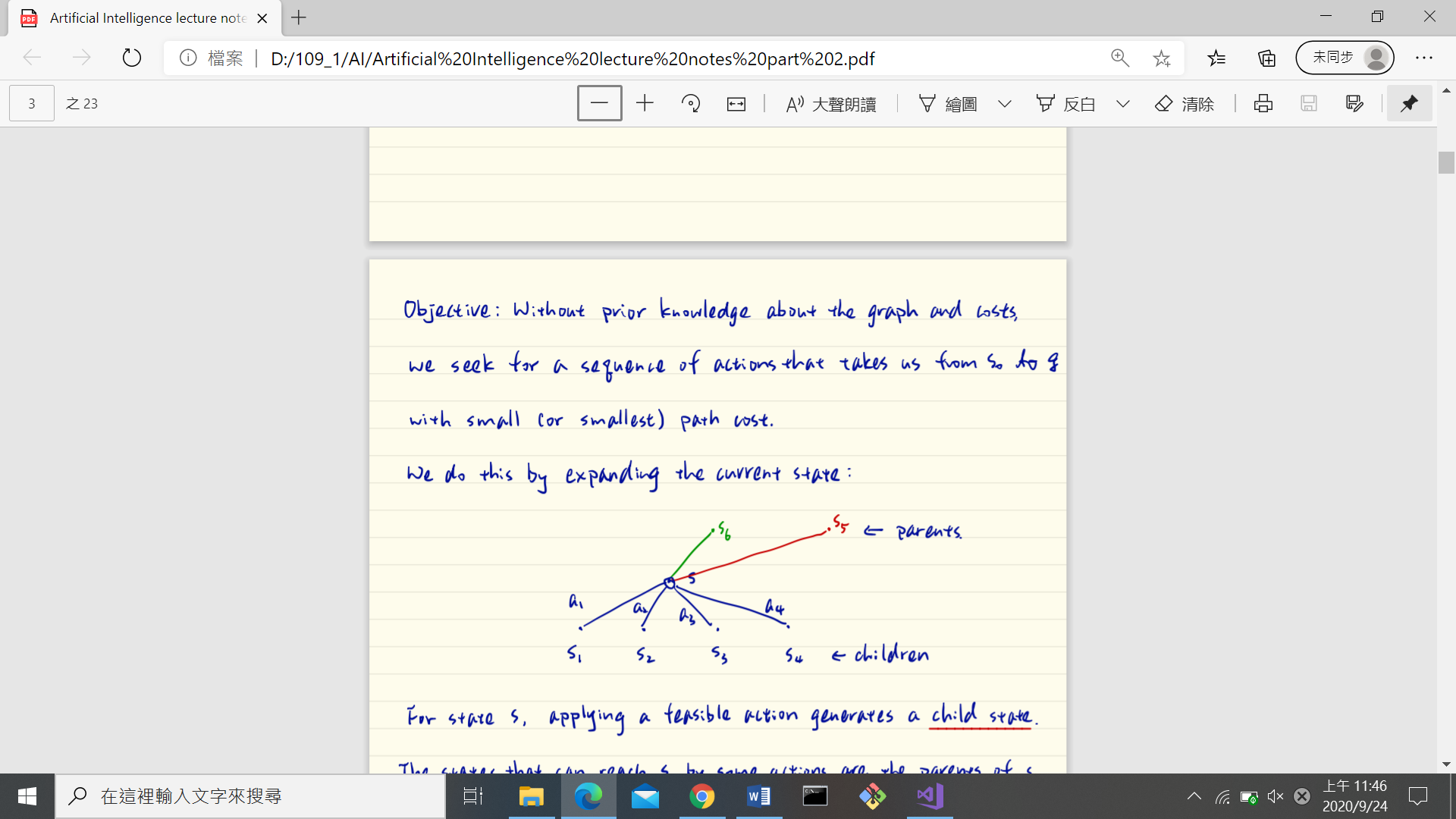
goal state: Kaohsiung

action: feasible cities to go

Cost: number associated with each edge (it could be actual distance, required time, money, etc.)

Objective: Without prior knowledge about the graph and cost, we seek for a sequence of actions that takes us from to with small (or smallest) path cost.

We do this by expanding the current state:



Once we visit state by doing at we learn and and this node is ready to be expanded.

The set of all such nodes is called the frontier.

The process of expanding nodes on the frontier continues until either a solution is found or there are no more nodes to expand.

This is called tree-search.

function

returns a or

Initial state

loop

if then

Choose an element in frontier according to strategy and expand

if found a goal

return corresponding path

else

remove node whose actions are all expanded from Frontier

add the newly visited nodes into Frontier

end

If we have unlimited computing power, we can fully expand the tree and find the best solution.

Unfortunately, we don’t have this luxury.

How to choose the next node to expand?

Performance metrics

Completeness: Does the algorithm always find a solution?

Optimality: Does the found solution always incur the smallest cost/largest reward?

Time complexity: How long does it take to find a solution?

Space complexity: How much memory is required?

We measure complexities of a problem by its branching factor b and depth d.

Branching factor: s - state

Depth: g - goal

Two categories:

1. Uninformed Search: No additional information -

Breadth-first search, uniform cost search, depth-first search, iterative deepening

2. Informed Search: Some problem-specific knowledge is available -

Best-first search (Greedy algorithm) and A\* algorithm

Uninformed Search

**Breadth-first search**: All the nodes are expanded at a given depth before any nodes at the next level are expanded

Any new path to a state already in the frontier or already explore is discarded.

BFS is **complete**.

It guarantees to find the shallowest goal. The shallowest solution may NOT have the lowest cost => **Not optimal**.

Time complexity:

Space complexity:

**Depth-First Search**: Always expand the deepest node in Frontier.

DFS is clearly not optimal. The deepest may NOT be the best.

DFS may NOT be complete unless we expand nodes only if they are NOT explored in the current path.

DFS only stores one path at a time together with unexpanded siblings for each node on the path.

Let

which may be much larger than d.

Time complexity:

Space complexity:

**Iterative Deepening DFS**:

Problem of DFS: m can be much larger than d. Why do we expand all the way to depth m?

Idea: run DFS as if the graph is -deep,

NOT optimal.

Complete if we expand nodes only if they are not explored in the current path.

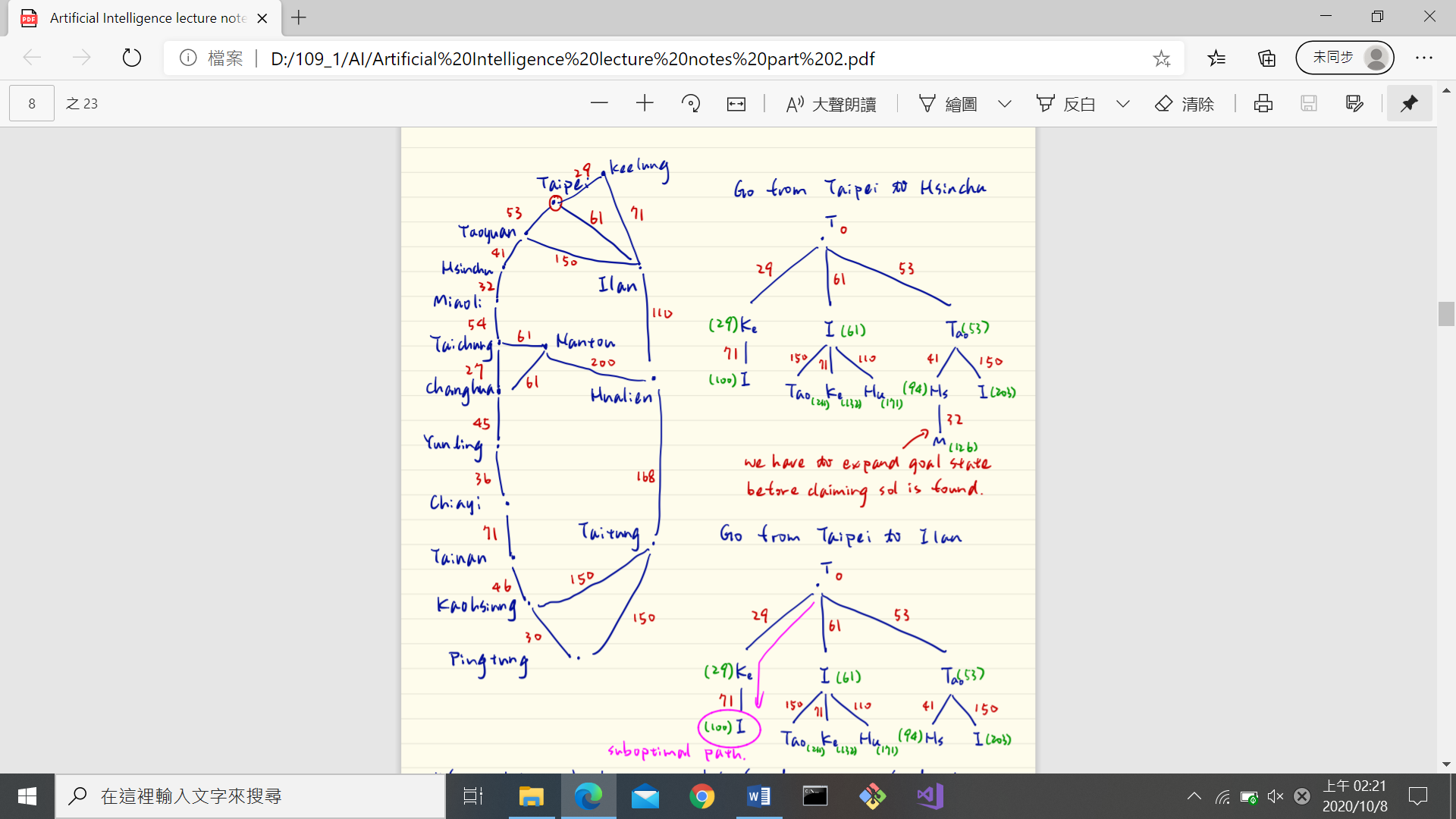
Time complexity:

Space complexity:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Breadth | Depth | Iterative |
| Complete | Y | Y/N | Y |
| Optional | N/Y(only if what we want is the shallowest) | N | N/Y(only if what we want is the shallowest) |
| Time |  |  |  |
| Space |  |  |  |

Uniform-Cost Search(UCS)

Always expand node in Frontier that has smallest **path** cost.



UCS is optimal because we only expand nodes that have accumulated lowest cost.

Cost:

It may not be complete if there is a path with an infinite sequence of zero actions.

It is complete if every action has

Let be the path cost of the optimal solution. Each expansion will increase the path cost by at least => the optimal path is **at most deep**.

* Time complexity
* Space complexity

can be much larger than d => UCS may have a much larger complexity than BFS.

Informed Search

For some problem, it is possible to have some problem-specific knowledge. Informed search exploits those knowledge by selecting nodes for expansion based on evaluation function f(n). An informed search simply expands the node n with the lowest f(n).

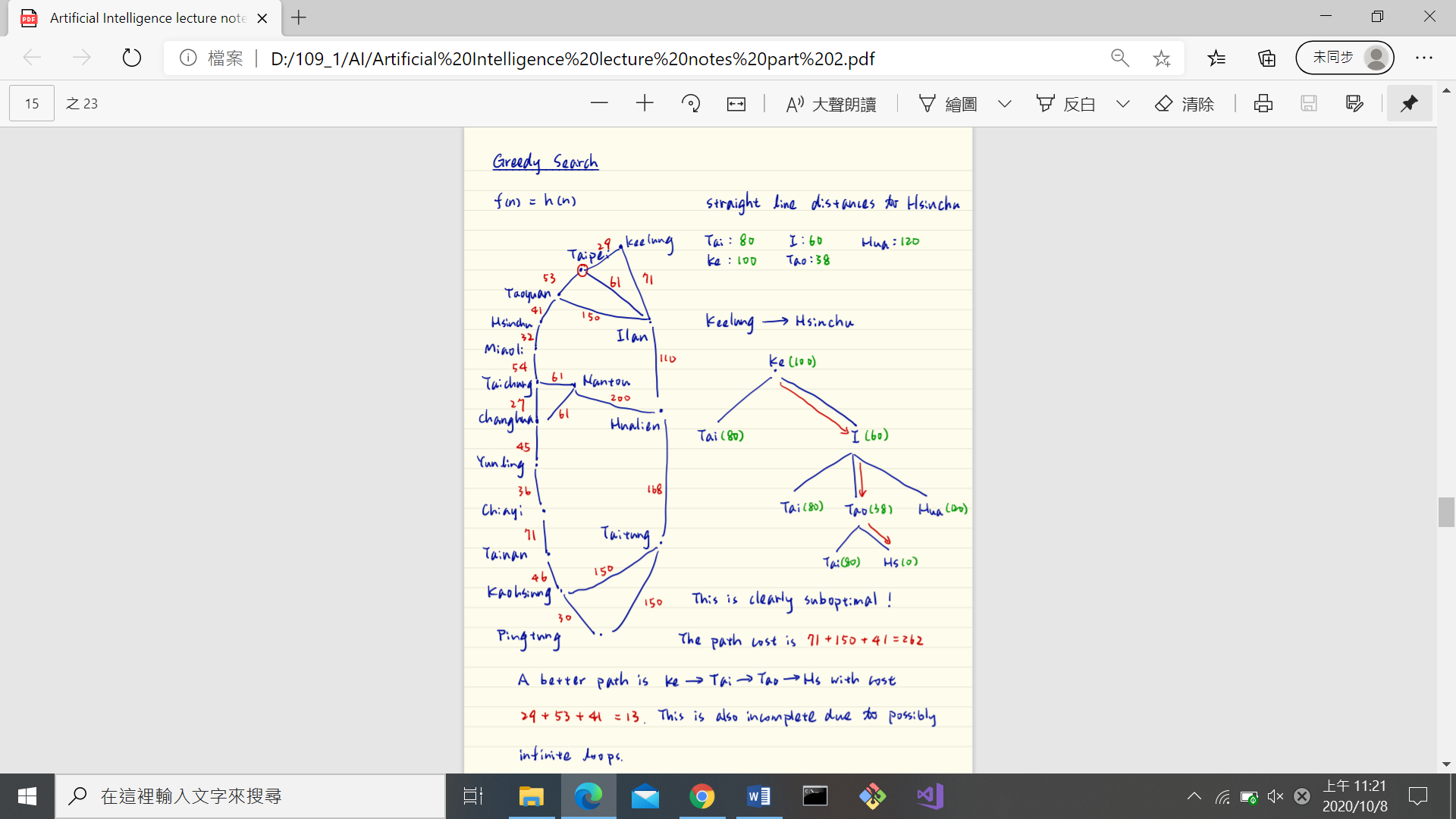
Def

In informed search, at each node n, the heuristic function h(n) reports an estimated cost of the cheapest path from n to a goal. h(g) = 0

Ex: Euclidean distance

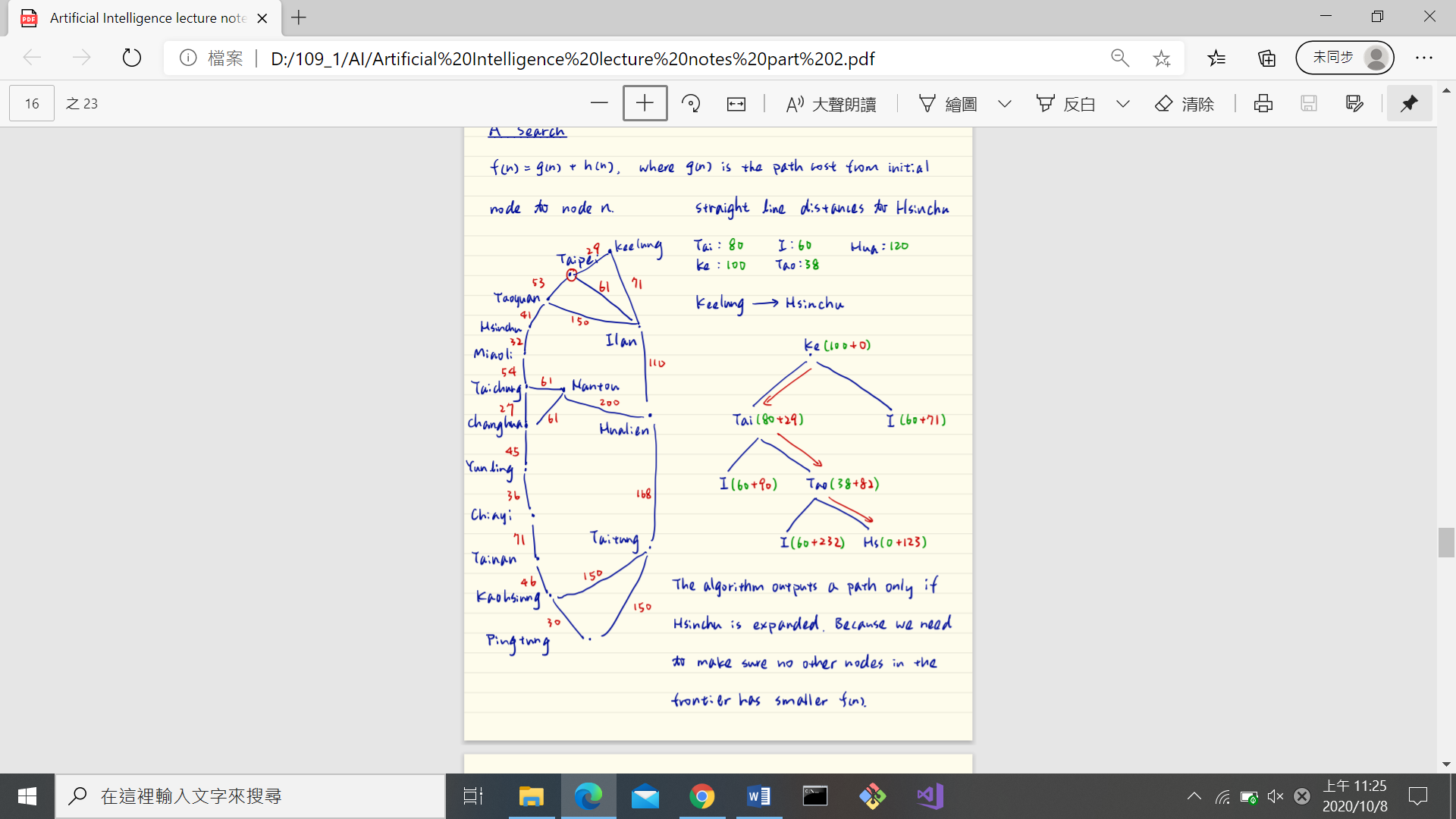
How to choose a good heuristic function is an art!

Greedy Search:



A\* Search:

where is the path cost from initial state to n.



Uniform-Cost Search is a special case of A\* Search with

Def: A heuristic function is admissible if it never overestimates the cost to the goal.

i.e. <= time cost

**: A\* Search is optimal if is admissible**

Pf:

Assume A is optimal goal node, B is suboptimal goal node, h is admissible.

If B is in frontier, some ancestor n of A is also in frontier.

We have

n must be expanded before B

: A\* is complete (pf: similar to UCS)

Time complexity: which is relative error in h and d is solution depth.

Space complexity: