Part 1

Question 1

(a)

	start10	start12	Start20	Start30	Start40
UCS	**2565**	**Mem**	**Mem**	**Mem**	**Mem**
IDS	**2407**	**13812**	**5297410**	**Time**	**Time**
A*	**33**	**26**	**915**	**Mem**	**Mem**
IDA*	**29**	**21**	**952**	**17297**	**112571**

(b)

UCS: This algorithm is to first expand the unexpanded nodes with the lowest cost and when all the paths have the same cost, it is a normal breadth-first search. And it exhaustively expanding all nodes closer to the initial state than the goal. So the time efficiency will not be very high and the memory consumption is very fast.

IDS: This algorithm Use depth first search to look for a solution recursively, up to the specified depth limit (D). The time efficiency can be very bad, especially if the position of the target is opposite to the position chosen at the beginning of the algorithm, e.g., starting from the leftmost node of the tree, yet the answer is at the bottom node on the rightmost side of the tree. This is equivalent to expanding all possible nodes. However, the memory efficiency is not bad, linear space similar to DFS. (One interesting fact is that I ran "Start30" on this algorithm and it took me four days to get a result, and I got N = 19636420921.)

A*: This algorithm combines the advantages of Greedy Search and Uniform Cost Search, so the time efficiency is very high. However, he will Keeps all nodes is memory, so the memory efficiency is not high. When the depth is too large, it can easily lead to

memory overflow. For example, like in the table above, after starting from "start30", the limited memory of 1GB is not enough.

IDA*: This algorithm is a low-memory variant of A* which performs a series of depth-first searches first. And then cuts off each search when the sum f() exceeds some pre-defined threshold. So, it has high time efficient and memory efficient. According to the above table, this is the only algorithm that completes "Satrt40" within the specified time and memory space.

Question 2

	Satrt50		Satrt60		Satrt64	
IDA*	**50**	**14642512**	**60**	**321252368**	**64**	**1209086782**
1.2	**52**	**191438**	**62**	**230861**	**66**	**431033**
1.4	**66**	**116174**	**82**	**3673**	**94**	**188917**
1.6	**100**	**34647**	**148**	**55626**	**162**	**235852**
Greedy	**164**	**5447**	**166**	**1617**	**184**	**2174**

Before:

```
42      h(Node1, H1),
43      F1 is G1 + H1,
44      F1 =< F_limit,
```

After:

```
42      h(Node1, H1),
43      F1 is (2 - W) * G1 + W * H1,
44      F1 =< F_limit,
```

Where $0 \leq W \leq 2$.

We learned from the lecture that when w = 0, it's a Uniform Cost Search, and when w = 1, it's an IDA*, when w = 2, it's a Greedy Search.

Furthermore, the weight of the Uniform Cost Search algorithm in IDA* takes up half of its total weight. The cost from the starting point to the current node is always considered, so it is guaranteed to be the optimal solution. But the speed will be much slower, because it has to expand more nodes.

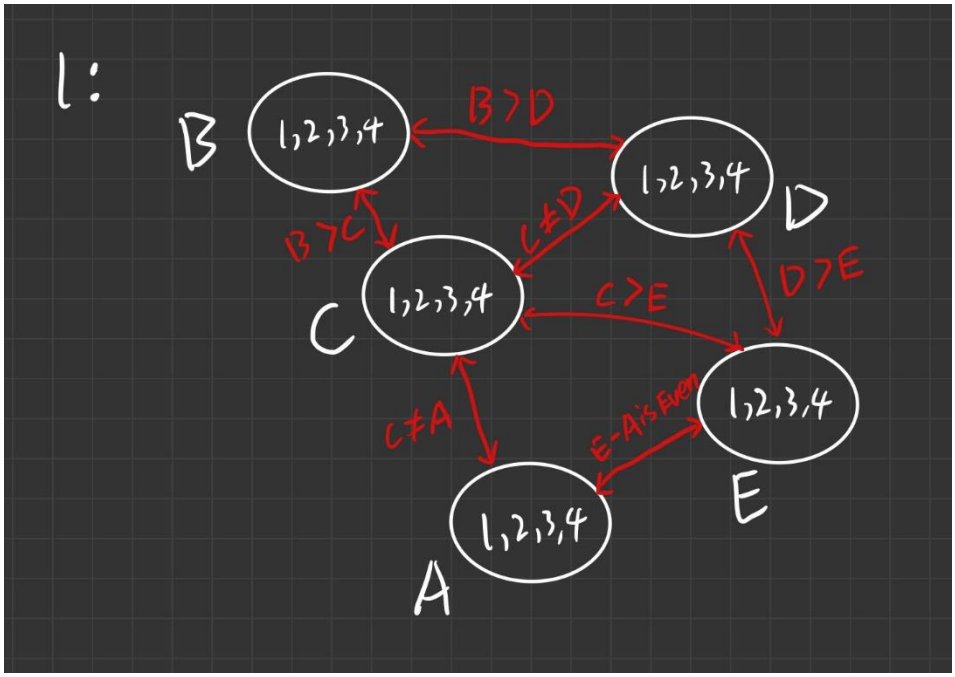
In the greedy algorithm, the only weight is the Heuristic function. That is, it focuses on the cost of the current node to the goal. Thus, although the goal can be found quickly, the length of the path is not guaranteed to be optimal.

Therefore, when we change the weight of W which affecting g() and h() in IDA* from 1.2 little by little to 1.6, we can clearly see from the results that the length of the path slowly becomes larger and the total number of states expanded gradually decreases.

Part 2

Question 1

The constraint network:



Setps:

Arc(B, C) delete 1 from domain B;

Arc(C, B) delete 4 from domain C;

Arc(C, E) delete 1 from domain C;

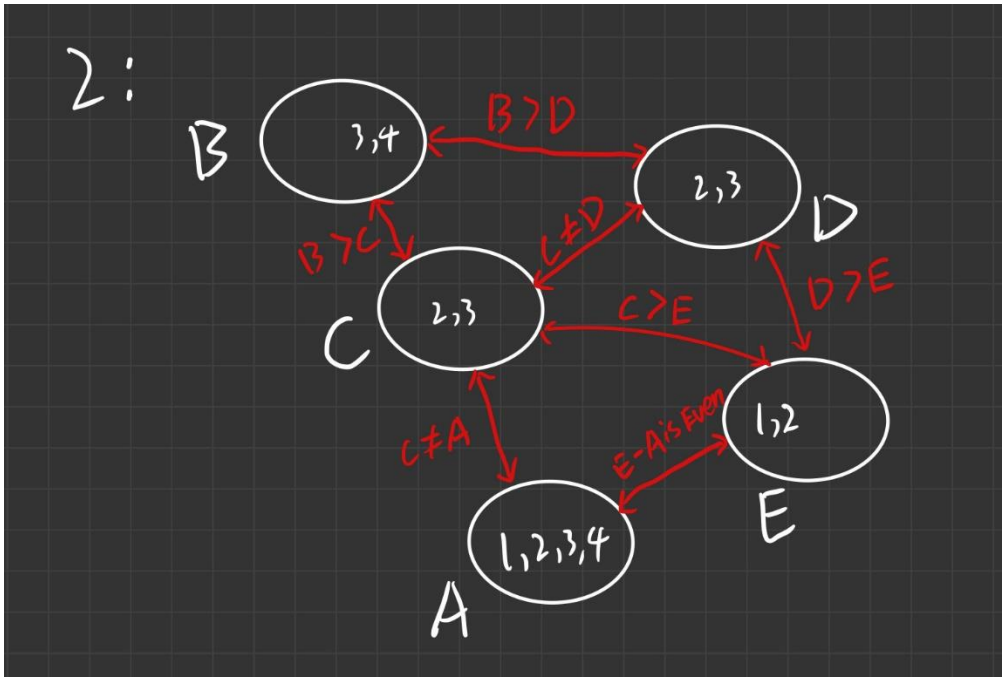
Arc(E, C) delete 3, 4 from domain E;

Arc(D, E) delete 1 from domain D;

Arc(B, D) delete 2 from domain B;

Arc(D, B) delete 4 from domain D.

After arc consistency has stopped we get:



Split the domain of C, with the case when C = 2:

Arc(A, C) delete 2 from domain A;

Arc(D, C) delete 2 from domain D;

Arc(B, D) delete 3 from domain B;

Arc(E, C) delete 2 from domain E;

Arc(A, C) delete 4 from domain A;

Results in:

$$\{1,3\} \in A, \{4\} \in B, \{2\} \in C, \{3\} \in D, \{1\} \in E$$

Given that there are two elements left in domain A,

the solution is A=1, B=4, C=2, D=3, E=1, and A=3, B=4, C=2, D=3, E=1.

If, we split the domain of C, with the case when C = 3:

Arc(A, C) delete 3 from domain A;

Arc(D, C) delete 3 from domain D;

Arc(B, C) delete 3 from domain B;

Arc(E, D) delete 2 from domain E;

Arc(A, C) delete 2, 4 from domain A;

Solution is A=1, B=4, C=3, D=2, E=1.

Question 2

(a)

If I eliminate variable A, then the constraints r1(A,B) and r2(A,C) should be removed.

A new constraint r11(B,C) should be formed.

(b)

If after part(a) I continue to eliminate variable B, then the constraints r3(B,D), r4(B,E), and r11(B,C) should be removed.

A new constraints r12(C,D) and r13(E,D) should be formed.