Name: Yanran Wang
ID: z5295965

COMP3411 Assignment 1

Part 1 - Search

Question 1: Search Algorithms for the 15-Puzzle

	UCS	IDS	A*	IDA*
Start10	2565	2407	33	29
Start12	Mem	13812	26	21
Start20	Mem	5297410	915	952
Start30	Mem	Time	Mem	17297
Start40	Mem	Time	Mem	112571

*Mem means run out of memory and Time means 5 mins without output

- ***UCS***

  It could be seen from the tests that UCS method has the worst time cost and space cost, ran out of memory on any start position larger than 10. UCS is a kind of Dijkstra's algorithm, and the problem is its implementation depends on the priority queue and for every expanded path will be inserted into the queue, which increases both time and memory cost.
  The time and space cost for UCS are both $O(b\hat{}\lceil c\hat{} * /\varepsilon\rceil)$.
  For time cost, C* is the cost of best solution and the cost of each step is $\lceil c\hat{} * /\varepsilon\rceil$.
  For space cost, when all moves share the same cost, $b\hat{}\lceil c\hat{} * /\varepsilon\rceil = b\hat{}d$.

- ***IDS***

  The IDS method has similar speed problem with UCS on time cost but is better at space cost as it's more memory efficient. It could be seen from the tests that for start position larger than 20 the method needed to run over 5 mins. As we said during lecture, this method "combines the benefits of depth-first and breadth-first", so it's preferred for large search space where depth is unknown. However, when dealing

with very large depths, it could give the answer, but the time cost would be immeasurable.

The time cost for IDS is $O(b\hat{}d)$ and the space cost for IDS is $O(bd)$.

- ***A****

The A* method is the second fastest method among these four and could be seen as an improvement of UCS method with better time cost, however there is a problem that A* is not that memory efficient since it stores all generated nodes into memory. It could be seen from the tests that for start positions 30 and 40 the A* method ran out of memory. A* uses the heuristic functions to seek for better path, which means A* could find priority nodes while Dijkstra only just explores all possible nodes.

The time cost for A* is $O(b\hat{}d)$ (worst case when heuristic = 0) and the space cost for A* is also $O(b\hat{}d)$.

- ***IDA****

The IDA* method is a low-memory variant of A* and IDS, it could be seen from the tests that only IDA* finished all 5 start positions without time or memory issues. However, it is not the fastest and in theory A* should be faster. IDA* method would perform a series of DFS but only care about the most possible nodes.

The time cost for IDA* is $O(b\hat{}d)$ (worst case when heuristic = 0) and the space cost for IDA* is $O(bd)$.

Name: Yanran Wang
ID: z5295965

Question 2: Heuristic Path Search for 15-Puzzle

	Start50		Start60		Start64	
IDA*	50	14642512	60	321252368	64	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116342	82	4432	94	190278
1.6	100	33504	148	55626	162	235848
Greedy	164	5447	166	1617	184	2174

Code change

```
F1 is 0.8 * G1 + 1.2 * H1,                    % changing code
```

The original code is *F1 is G1 + H1*.

Trade-off between speed and quality of solution

The speed and quality of these five algorithms basically depend on the heuristic evaluation function. IDA* needs the heuristic function to be admissible to guarantee the optimal solution and after w > 1 this condition will no longer be satisfied.

It could be seen from the table that IDA* solution has the best quality but requires the longest time to deal with while the Greedy algorithm requires the shortest time to run but its solution has the worst quality. The heuristic search rely more on heuristic function as the value of w becomes larger, making them become to be like the Greedy algorithm, as w increases, the speed of run will increase as less memory is needed while the quality of search will decrease since it becomes less optimal.

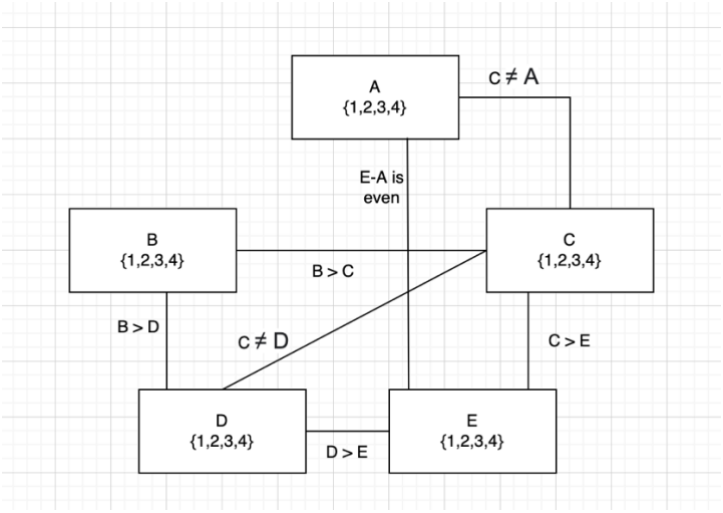
Name: Yanran Wang
ID: z5295965

Part 2

Question 1: Arc Consistency

**Arc would be directly described as constraints between nodes in the answer, e.g., C ≠ A is the Arc (C, A)*

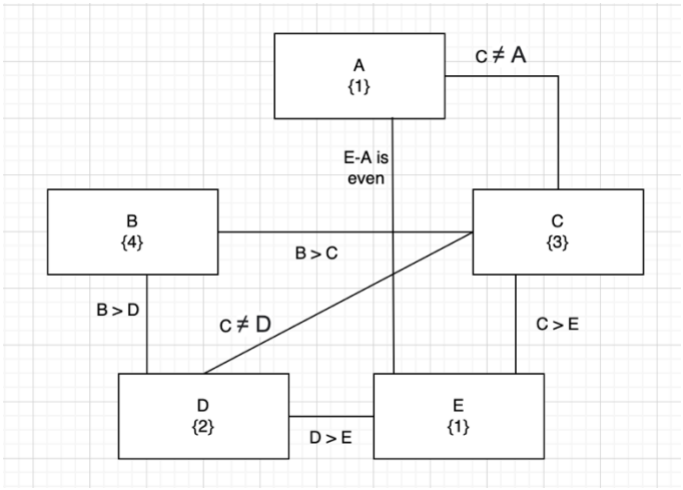
Constraint graph (original)



- Split the domain of A into 1, 2, 3 and 4.
- When A = 1, the domain of A in constraint graph would be {1}.
  Since C ≠ A, the domain of C would become {2, 3, 4}, and since E – A is even, the domain of E would become {1, 3}. Because of the constraint arc B > C, C cannot be 4, the domain of C becomes {2, 3}, and the domain of B becomes {3, 4}, and at the same time since B > D and D > E, the domain of D would become {2, 3}.
  Since C ≠ D, we could split the domain into 2 and 3, when C = 2, D = 3, E must be 1, and B = 4, which is a possible solution.
  When C = 3, D = 2, E must still be 1, and B = 4, which is also the possible solution.
- The rest of the solutions could be found by assigning A the value 2, 3 or 4 and repeating the same procedures with each constraint.

Name: Yanran Wang
ID: z5295965

Constraint graph (after arc consistency)

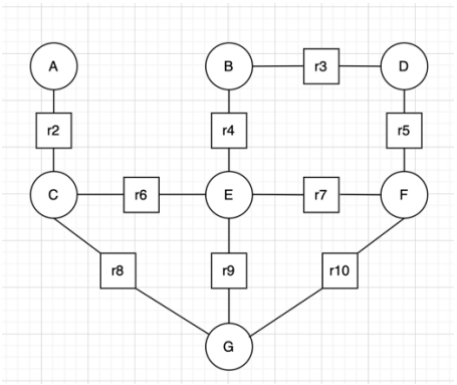


Name: Yanran Wang
ID: z5295965

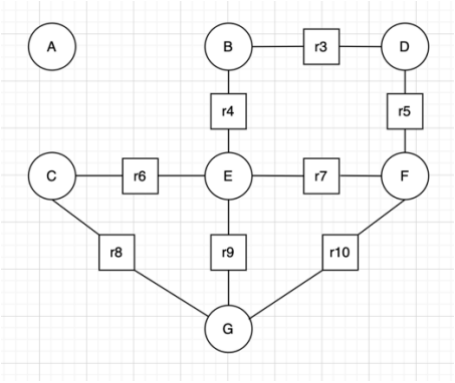
Question 2: Variable Elimination

(a) 1. Select variable A

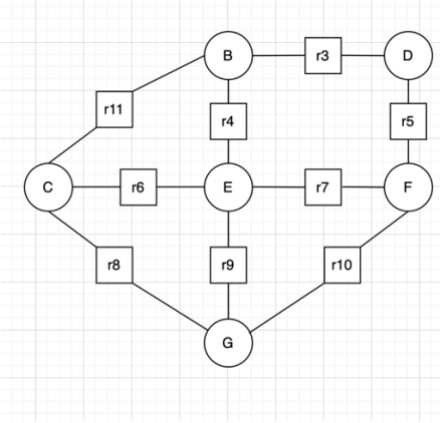
2. Enumerate constraint r1 which related to A and B



3. Enumerate constraint r2 which related to A and C

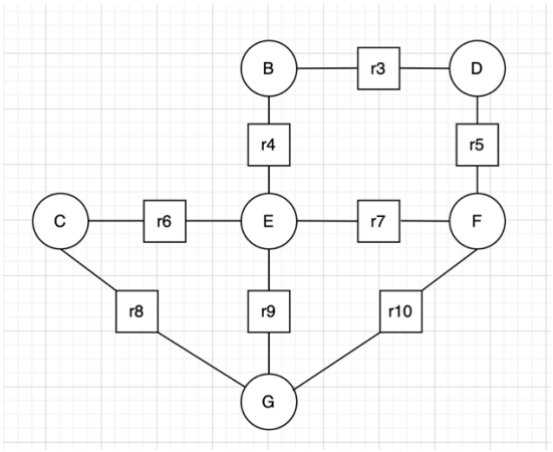


4. join these two constraints and project join onto its variables other than A

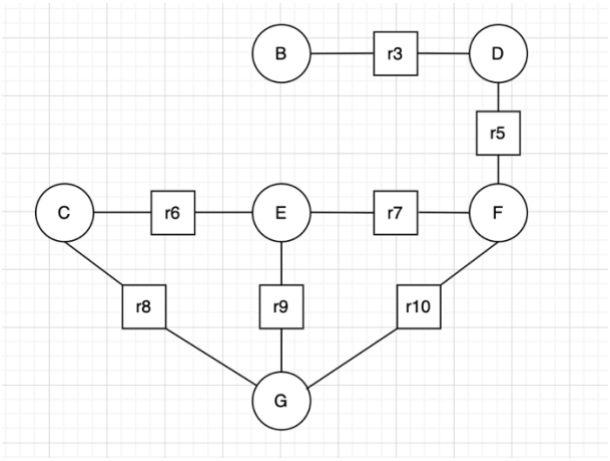


(b) 1. Select variable B

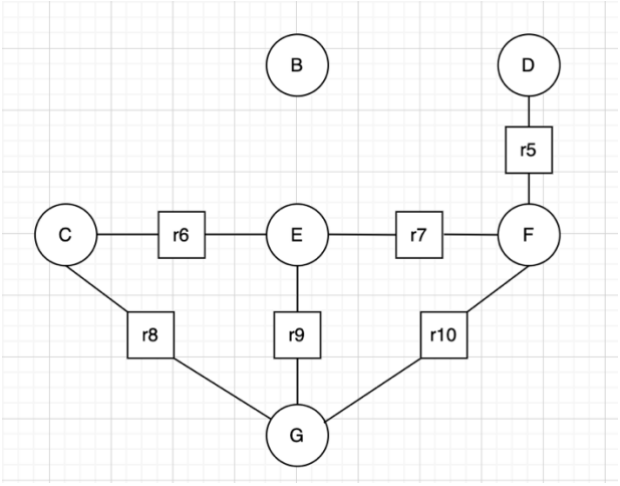
2. Enumerate constraint r11 which related to B and C



3. Enumerate constraint r4 which related to B and E



4. Enumerate constraint r3 which related to B and D



Name: Yanran Wang
ID: z5295965

5. join these two constraints and project join onto its variables other than B