COMP3411 ASSIGNMENT 1

Celine Lin z5311209

Part 1

Q1: Search Algorithms for the 15-Puzzle

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 uses lots of nodes, and its taking too many memory for searching for more complex puzzle
- IDS* tooks a lot of time for complex puzzle
- A* is efficient for simple puzzles, it will be taking lots of node/memonry for complex puzzle
- IDA* is the most optimal algorithm for both simple and complex puzzle

Q2: Heuristic Path Search for 15-Puzzle

a.

	start50	start60	Start64
IDA*	50 14642512	60 321252368	64 1209086782
1.2	52 191438	62 230861	65 431033
1.4	66 116342	82 4432	94 190278
1.6	100 33504	148 55626	162 235848
Greedy	164 5447	166 1617	184 2174

b. This is an example of heuristic.pl for $w = 1.6$, this is by chaning $F1$ by using the equation $F1 = (2 - w) * G1 + w * H1$

```
% Iterative Deepening A-Star Search

% COMP3411/9414/9814 Artificial Intelligence, UNSW, Alan Blair

% solve(Start, Solution, G, N)
% Solution is a path (in reverse order) from Node to a goal
state.
% G is the length of the path, N is the number of nodes
expanded.
solve(Start, Solution, G, N)  :-
    nb_setval(counter,0),
    idastar(Start, 0, Solution, G),
    nb_getval(counter,N).

% Perform a series of depth-limited searches with increasing
F_limit.
idastar(Start, F_limit, Solution, G) :-
    depthlim([], Start, 0, F_limit, Solution, G).

idastar(Start, F_limit, Solution, G) :-
    write(F_limit),nl,
    F_limit1 is F_limit + 2,  % suitable for puzzles with
parity
    idastar(Start, F_limit1, Solution, G).

% depthlim(Path, Node, Solution)
% Use depth first search (restricted to nodes with F <=
F_limit)
% to find a solution which extends Path, through Node.
```

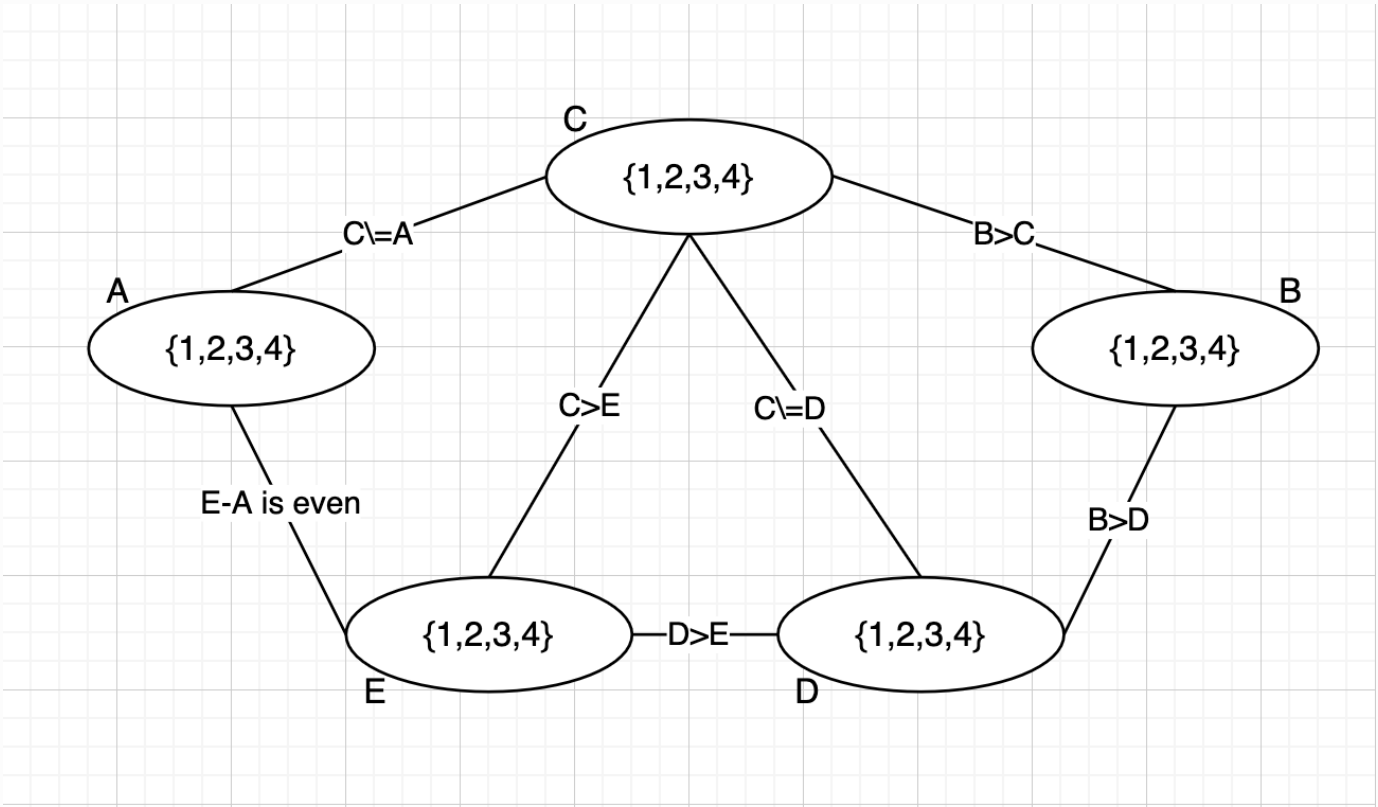
```
% If the next node to be expanded is a goal node, add it to
% the current path and return this path, as well as G.
depthlim(Path, Node, G, _F_limit, [Node|Path], G)  :-
    goal(Node).

% Otherwise, use Prolog backtracking to explore all successors
% of the current node, in the order returned by s.
% Keep searching until goal is found, or F_limit is exceeded.
depthlim(Path, Node, G, F_limit, Sol, G2)  :-
    nb_getval(counter, N),
    N1 is N + 1,
    nb_setval(counter, N1),
    % write(Node),nl,   % print nodes as they are expanded
    s(Node, Node1, C),
    not(member(Node1, Path)),      % Prevent a cycle
    G1 is G + C,
    h(Node1, H1),
    F1 is 0.4 * G1 + 1.6 * H1,
    F1 =< F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

d.

- IDA* tooks many nodes for more complex puzzle, compares with higher w, so that this takes lots of memory.
- By increasing the w in IDA*, memory decreases, and more efficient.
- Greedy is the most efficient algorithm, it is fast and takes the least memory.

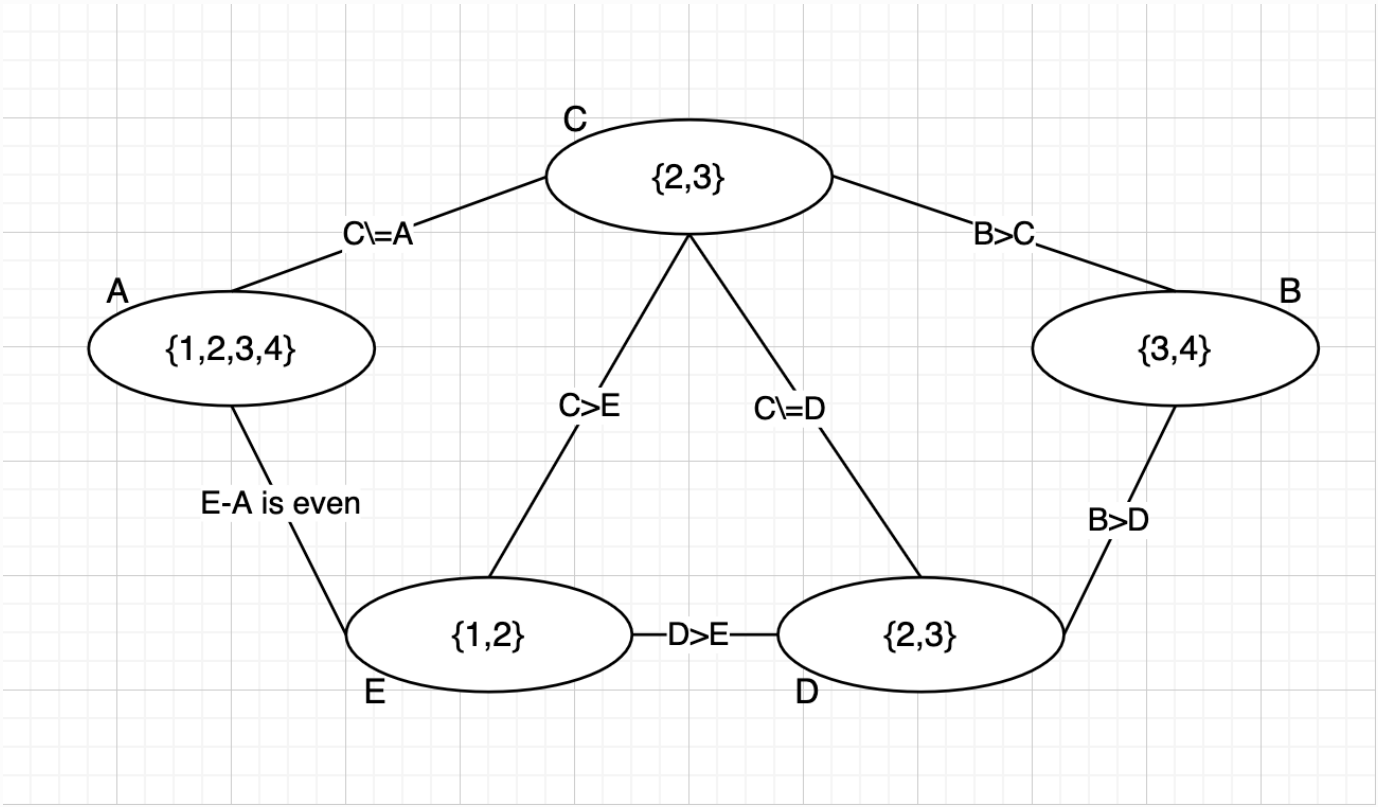
Part 2

Q1: Arc Consistency



1. Constraint: B>D, domain(B)= {2,3,4}, domain(D)={1,2,3}
2. Constraint: D>E, domain(D)={2,3}, domain(B)={3,4}, domain(E)={1,2}
3. Constraint: C>E and B>E, domain(C)={2,3}

∴ domain(A)={1,2,3,4}, domain(B)={3,4}, domain(C)={2,3}, domain(D)={2,3}, domain(E)={1,2}



Since E is the smallest variable, then let E=1. Similarily, B is the smallest variable, let B=4.

Case 1:

We chose C=2 and D=3, then A=3 or A=1.

Case 2:

We chose C=3 and D=2, then A=1.

∴ Solutions:

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

Q2: Variable Elimination

a.

For eliminating A, we need to remove $r_1(A, B)$ and $r_2(A, C)$, and create a new constraint $r_{11}(B, C)$ on variables B and C.

b.

For eliminating B, we need to remove $r_{11}(B, C)$, $r_3(B, D)$, and $r_4(B, E)$, and create two new constraint $r_{12}(C, D)$, on variables C and D, and $r_{13}(E, D)$, on variables E and D.