Assignment 1 – Search and Constraint Solving

1.1)

	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

All the algorithms are complete, optimal and have time complexity of `O(b ^ m)`

uscdijkstra is essentially a **BFS** search with weighted edges. As such, it shares **BFS's** exponential space time complexity of `O(b ^ m)` This is inefficient for large problems and can be seen with prolog generating a *Mem - out of global stack* for `start12 ... start40`

IDS is a repeated **DFS** depth-limited search. As such, it has much better space time complexity than **ucsdijkstra** of `O(b·m)` This can be seen with it being able to compute `start10 .. start20` However, the repeated search means for goals close to the target, it is slower than **ucsdijkstra** which can be seen in `start10`. **IDS** is an uninformed search and therefore expands a large number of unecessary nodes. The consequence of this is seen with the algorithm 'timing out' from `start30 ... start40`

A* is essentially an informed **ucsdijkstra**. This use of a heuristic means it is able to expand less nodes than **ucsdijsktra** and **IDS**. As a result, **A*** computes more solutions than **ucsdijkstra**, e.g. `start10 ... start20` However, like **ucsdjikstra** it has exponential space time which results in *Mem* for `start30 ... start40`

IDA* combines the aforementioned benefits of linear memory efficiency of **IDS** and the informed search of **A***. It's therefore able to compute solutions to `start10 ... start40`

1.2)

	start50	start60	start64
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

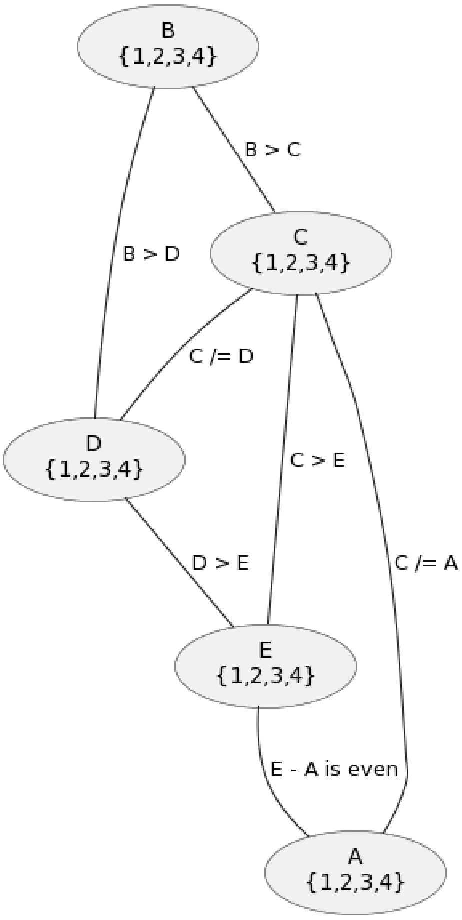
```
% Section of code changed for introducing Heuristic Path Search algorithm
< 43:    F1 is G1 + H1,
---
> 43:    F1 is (2 - 1.2) * G1 + (1.2 * H1), % where w = 1.2
```

The most optimal algorithm of the 5 presented is **IDA***. The most memory efficient is **Greedy**

By introducing the heuristic path search algorithm `(2 - w)·g(n) + w·f(n)`, we can make the **IDA*** algorithm more greedy or more optimal by altering the value of `w`. By increasing the value of `w`, we are making **IDA*** more greedy and therefore more memory efficient (increasing the speed). This can be seen with fewer nodes expanded in all successive increments of `w` in solutions `start50 ... start64` However, by making the algorithm more greedy we are reducing how optimal it is (lowering the quality). This can be seen with the length of the path explored exceeding the minimum number of moves. It's increasing with all successive increments of `w` in solutions `start50 ... start64`. The most extreme of this is seen in **Greedy**

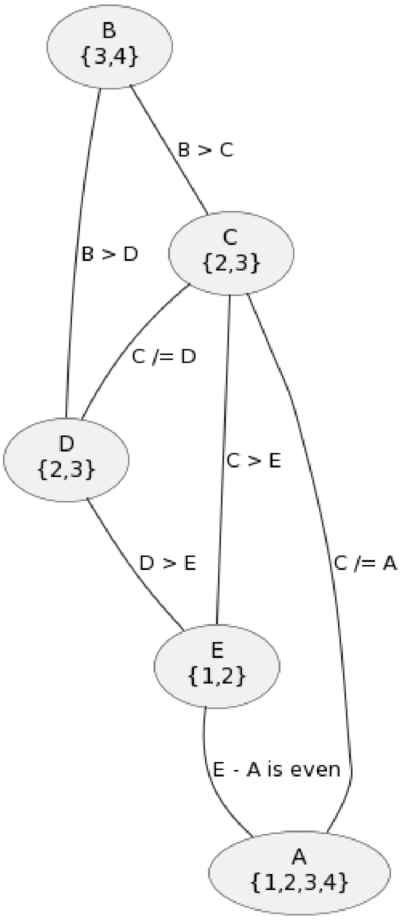
2.1)

Constraint Graph (next page ...)



Arc	Relation	Value(s) Removed
(B, C)	B > C	B = 1 & C = 4
(B, D)	B > D	D = 4
(D, E)	D > E	D = 1 & E = 4 & E = 3
(C, E)	C > E	C = 1
(B, C)	B > C	B = 2

Constraint Graph with Arc Consistency Finished (next page ...)



Domain splitting for C = 2

Arc	Relation	Value(s) Removed
(C, D)	C /= D	D = 2
(C, A)	C /= A	A = 2
(C, E)	C > E	E = 2
(B, D)	B > D	B = 3
(E, A)	E - A is even	A = 4

This gives two solutions:

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

Domain splitting for C = 3

Arc	Relation	Value(s) Removed
(C, D)	C /= D	D = 3
(C, A)	C /= A	A = 3
(D, E)	D > E	E = 2
(B, C)	B > C	B = 3

Arc	Relation	Value(s) Removed
(E, A)	E - A is even	A = 2 & A = 4

This gives one solution:

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

Therefore the union of these solutions gives solutions for this CSP.

2.2) Eliminating variable **A** would remove constraints *r1* and *r2*. A new constraint *r11* is created that exists between **B** and **C**. If we were to subsequently eliminate **B** this would remove constraints *r11*, *r4* and *r3*. A new ternary constraint would exist between **C**, **E** and **D**, say *r12*