

CMPT 310
Assignment 2
Constraint Satisfaction Problem (CSP)
Jiadi Luo 301354107
Haoxuan Zhao 301385109

Instructions:

- You can do this assignment individually or in a team of two. If you are doing it in a group, only one submission per group is required.
- You may submit multiple times until the deadline. Grade penalties will be imposed for late submissions (see the course outline for the details).
- Always plan before coding.
- For coding questions - Use function-level and inline comments throughout your code. We will not be specifically grading documentation. However, remember that you will not be able to comment on your code unless sufficiently documented. Take the time to document your code as you develop it properly.
- We will carefully analyze the code submitted to look for plagiarism signs, so please do not do it! If you are unsure about what is allowed, please talk to an instructor or a TA.
- Make sure that in your answers, you clearly indicate the exact section you are answering.
- Check A2 rubric available on canvas.
- You will submit one .zip file.
 - o For theoretical questions - Your submission must be formatted as a single PDF file; other types of submissions (non-pdf, multiple files, etc.) will not be graded. Your name(s) and student number(s) must appear at the top of the first page of your submission.
 - o Make sure that in your answers, you clearly indicate the exact section you are answering.
- Please put all your files (pdf + code) in one folder. Compress these into a single archive named a2.zip and submit it on Canvas before the due date listed there.

Ques 1. Complex structures and allocation problems can be solved using CSP techniques. You are tasked to arrange several large developments in a bustling commercial area. You are to design an arrangement for the following buildings: A department store, two entirely fictional fast-food chains: WcDonald's and Burger Queen, and a Japanese style inn. The development area can be represented as a 3 by 3 grid (Three rows as 0,1,2 from left to right and three columns as 0,1,2 from top-down). Each development can occupy one cell of the grid. Unfortunately, there are practical constraints on the problem that requires consideration. For the following constraints, structure A is close to structure B if A is in a cell that shares an edge with B (Sharing a corner does **not** make the two buildings close).

- There is an underground hot spring in the cell (0,1) (You do not need to treat the hot spring as a variable)
- WcDonald's should be close to the Department Store
- Burger Queen should not be close to WcDonald's
- The Japanese style inn should be close to the underground hot spring
- Burger Queen should be close to the Department Store

- a) **[15 points]** Represent this problem as a CSP. We want your constraints to be:
- correct
 - precise and readable enough that someone could implement them without asking you for clarification (e.g. what does “close” mean in mathematical terms?)
 - reasonably formatted

Variables: {DC (Department store), WD (WcDonald’s), BQ (Burger Queen), JI (The Japanese style inn)} in total 4 variables.

Domain: Each cell can be represented as a 2-tuple (x, y), x is the row position index of the variable & y is the column position index of the variable. Therefore, the domain can be any cell except for (0, 1) which is already occupied by the underground hot spring. The domain can be represented as:

Domain (DC) = {(0, 0), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)}

Domain (WD) = {(0, 0), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)}

Domain (BQ) = {(0, 0), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)}

Domain (JI) = {(0, 0), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)}

Constraints: {C1, C2, C3, C4, C5, C6}

C1: All variables can’t overlap with each other:

$DC \neq WD \neq BQ \neq JI$

C2: WcDonald’s should be close to the Department Store

Given DS (x, y), WD (if existed) = (x-1, y) || (x, y-1) || (x+1, y) || (x, y+1)

C3: Burger Queen should not be close to WcDonald’s

Given WD (x, y), BQ (if existed) \neq (x-1, y) && (x, y-1) && (x+1, y) && (x, y+1)

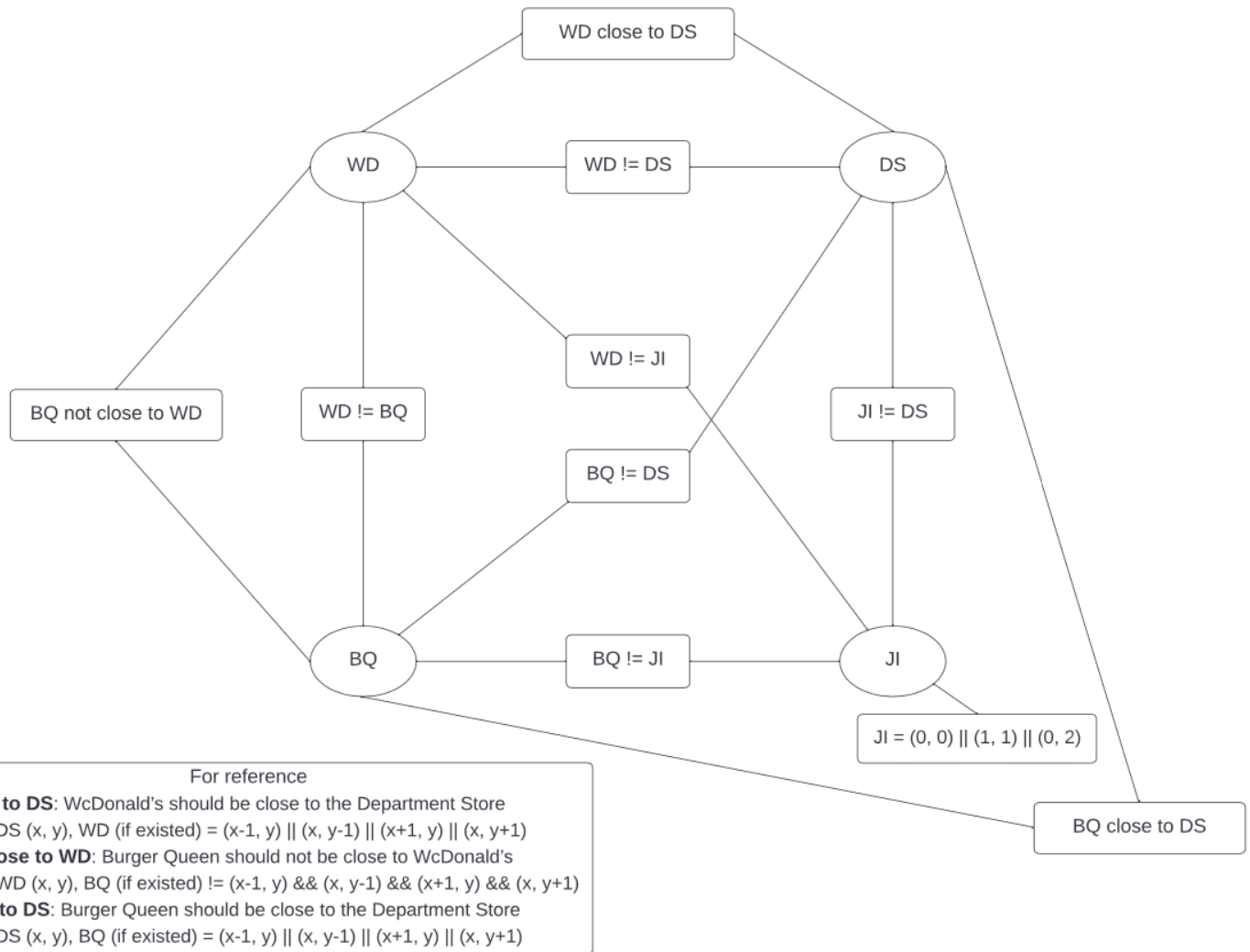
C4: The Japanese style inn should be close to the underground hot spring

$JI = (0, 0) \parallel (1, 1) \parallel (0, 2)$

C5: Burger Queen should be close to the Department Store

Given DS (x, y), BQ (if existed) = (x-1, y) || (x, y-1) || (x+1, y) || (x, y+1)

- b) **[5 points]** Draw a constraint graph for this problem. If a constraint/domain is too long to fit easily in the graph, use a label in the graph instead, indicating which constraint/domain the label refers to. Include unary constraints in the graph rather than reducing variable domains.



Ques 2. Consider a CSP, where there are six variables A, B, C, D, E, F.

A, B, C, D have the domain of $\{1, 2, 3, 4, 5, 6\}$, while E and F are in $\{1, 2, 3, 4\}$ domain. Suppose the constraints are:

$A > 1$, $|A - B| = 1$, $B \text{ odd}$, $A \neq C$, $B + C = 4$, $(B + D) \% 3 = 0$, $D \text{ even}$, $C \geq E$, $D - E \text{ odd}$, $E < F$, $E + F \text{ even}$.

a) [25 points] Show how search can be used to solve this problem, using the variable ordering A, B, C, D, E, F, G, H. To do this, you should write a program to

1. draw the search tree generated (see below)
2. report all solutions (models) found
3. report the number of failing consistency checks (i.e. failing branches) in the tree

You can use whatever programming language you like.

To draw the search tree, write it in text form **with each branch on one line**. For example, suppose we had variables X, Y and Z with domains {t, f} and constraints $X \neq Y$, $Y \neq Z$. The corresponding search tree, with the order X, Y, Z, can be written as:

```
X=t Y=t Failure!
      Y=f Z=t Solution!
            Z=f Failure!
X=f Y=t Z=t Failure!
      Z=f Solution!
            Y=f Failure!
```

Your tree output should follow this exact format. Do not use tabs for producing blank spaces.

Submit a zip file containing your codes and the result for the first part in the answer document. You can also print the answer to the second and third sections at the end of your code.

Answer for the first part:

```
A = 1 Failure!
A = 2 B = 1 C = 1 Failure!
      C = 2 Failure!
      C = 3 D = 1 Failure!
            D = 2 E = 1 F = 1 Failure!
                  F = 2 Failure!
                  F = 3 Solution!
                  F = 4 Failure!
            E = 2 Failure!
            E = 3 F = 1 Failure!
                  F = 2 Failure!
                  F = 3 Failure!
                  F = 4 Failure!
            E = 4 Failure!
      D = 3 Failure!
      D = 4 Failure!
      D = 5 Failure!
      D = 6 Failure!
    C = 4 Failure!
    C = 5 Failure!
    C = 6 Failure!
  B = 2 Failure!
  B = 3 C = 1 D = 1 Failure!
```

```
D = 2 Failure!
D = 3 Failure!
D = 4 Failure!
D = 5 Failure!
D = 6 E = 1 F = 1 Failure!
      F = 2 Failure!
      F = 3 Solution!
      F = 4 Failure!
      E = 2 Failure!
      E = 3 Failure!
      E = 4 Failure!
C = 2 Failure!
C = 3 Failure!
C = 4 Failure!
C = 5 Failure!
C = 6 Failure!
B = 4 Failure!
B = 5 Failure!
B = 6 Failure!
A = 3 B = 1 Failure!
      B = 2 Failure!
      B = 3 Failure!
      B = 4 Failure!
      B = 5 Failure!
      B = 6 Failure!
A = 4 B = 1 Failure!
      B = 2 Failure!
      B = 3 C = 1 D = 1 Failure!
            D = 2 Failure!
            D = 3 Failure!
            D = 4 Failure!
            D = 5 Failure!
            D = 6 E = 1 F = 1 Failure!
                  F = 2 Failure!
                  F = 3 Solution!
                  F = 4 Failure!
                  E = 2 Failure!
                  E = 3 Failure!
                  E = 4 Failure!
C = 2 Failure!
C = 3 Failure!
C = 4 Failure!
```

```

        C = 5 Failure!
        C = 6 Failure!
    B = 4 Failure!
    B = 5 C = 1 Failure!
        C = 2 Failure!
        C = 3 Failure!
        C = 4 Failure!
        C = 5 Failure!
        C = 6 Failure!
    B = 6 Failure!
A = 5 B = 1 Failure!
    B = 2 Failure!
    B = 3 Failure!
    B = 4 Failure!
    B = 5 Failure!
    B = 6 Failure!
A = 6 B = 1 Failure!
    B = 2 Failure!
    B = 3 Failure!
    B = 4 Failure!
    B = 5 C = 1 Failure!
        C = 2 Failure!
        C = 3 Failure!
        C = 4 Failure!
        C = 5 Failure!
        C = 6 Failure!
    B = 6 Failure!

```

Answer for second part:

Solutions in form of (A, B, C, D, E, F):

```
(2, 1, 3, 2, 1, 3)
```

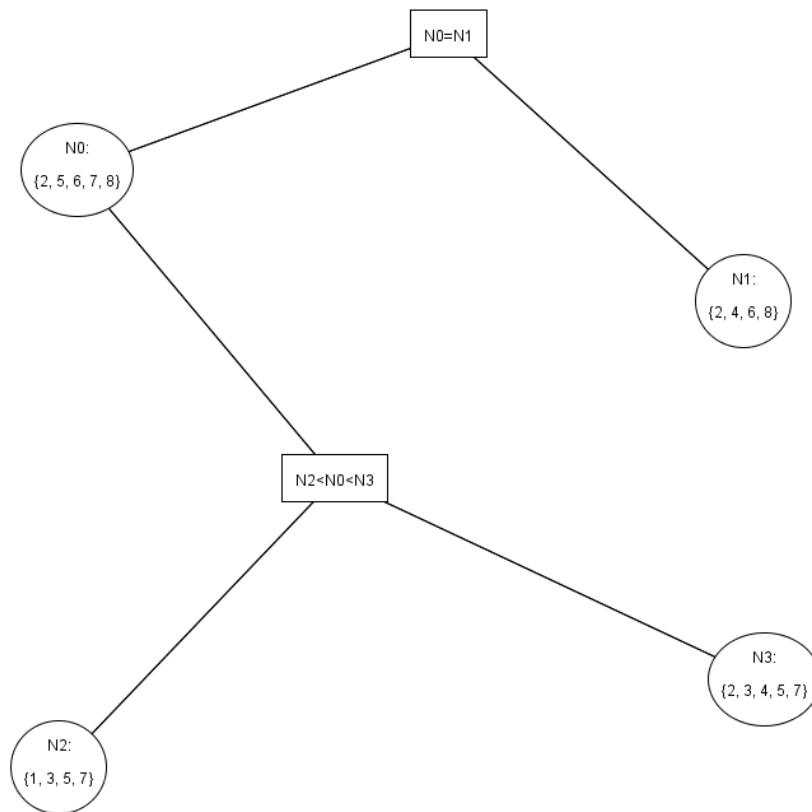
```
(2, 3, 1, 6, 1, 3)
```

```
(4, 3, 1, 6, 1, 3)
```

Answer for third part:

Number of failed Branch: 89

Ques 3. Consider the below graph



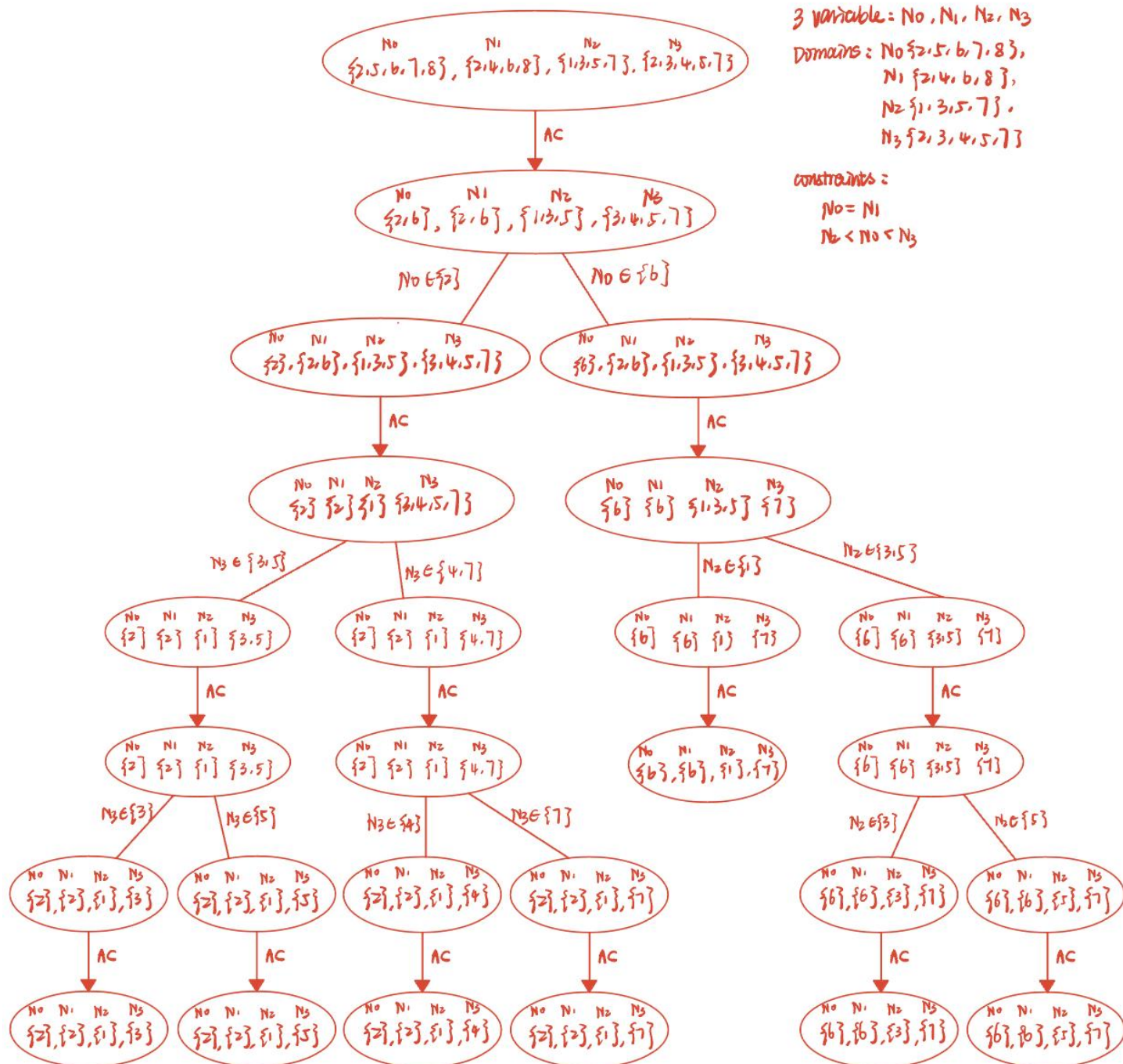
(a) [15 marks] Perform arc consistency on the graph and fill the table. Remove the arcs as given in the queue.

Queue	Arc removed for checking.	Domain of N0	Domain of N1	Domain of N2	Domain of N3	Add arc that needs to be rechecked
$\langle N0, (N0=N1) \rangle$ $\langle N1, (N0=N1) \rangle$ $\langle N0, (N2<N0<N3) \rangle$ $\langle N2, (N2<N0<N3) \rangle$ $\langle N3, (N2<N0<N3) \rangle$		2,5,6,7,8	2,4,6,8	1,3,5,7	2,3,4,5,7	
$\langle N1, (N0=N1) \rangle$ $\langle N0, (N2<N0<N3) \rangle$ $\langle N2, (N2<N0<N3) \rangle$ $\langle N3, (N2<N0<N3) \rangle$	$\langle N0, (N0=N1) \rangle$	2,6,8	2,4,6,8	1,3,5,7	2,3,4,5,7	
$\langle N0, (N2<N0<N3) \rangle$ $\langle N2, (N2<N0<N3) \rangle$	$\langle N1, (N0=N1) \rangle$	2,6,8	2,6,8	1,3,5,7	2,3,4,5,7	

$\langle N3, (N2 < N0 < N3) \rangle$						
$\langle N2, (N2 < N0 < N3) \rangle$ $\langle N3, (N2 < N0 < N3) \rangle$	$\langle N0, (N2 < N0 < N3) \rangle$	2,6	2,6,8	1,3,5,7	2,3,4,5,7	$\langle N1, (N0 = N1) \rangle$
$\langle N3, (N2 < N0 < N3) \rangle$ $\langle N1, (N0 = N1) \rangle$	$\langle N2, (N2 < N0 < N3) \rangle$	2,6	2,6,8	1,3,5	2,3,4,5,7	
$\langle N1, (N0 = N1) \rangle$	$\langle N3, (N2 < N0 < N3) \rangle$	2,6	2,6,8	1,3,5	3,4,5,7	
	$\langle N1, (N0 = N1) \rangle$	2,6	2,6	1,3,5	3,4,5,7	

(you can add more rows to the table if needed)

(b)) **[10 marks]** Use domain splitting to solve this problem. Draw your tree of splits and show the solutions. (It is sufficient to only label the arcs in your tree of splits)



Ques 4. [30 marks] Programming

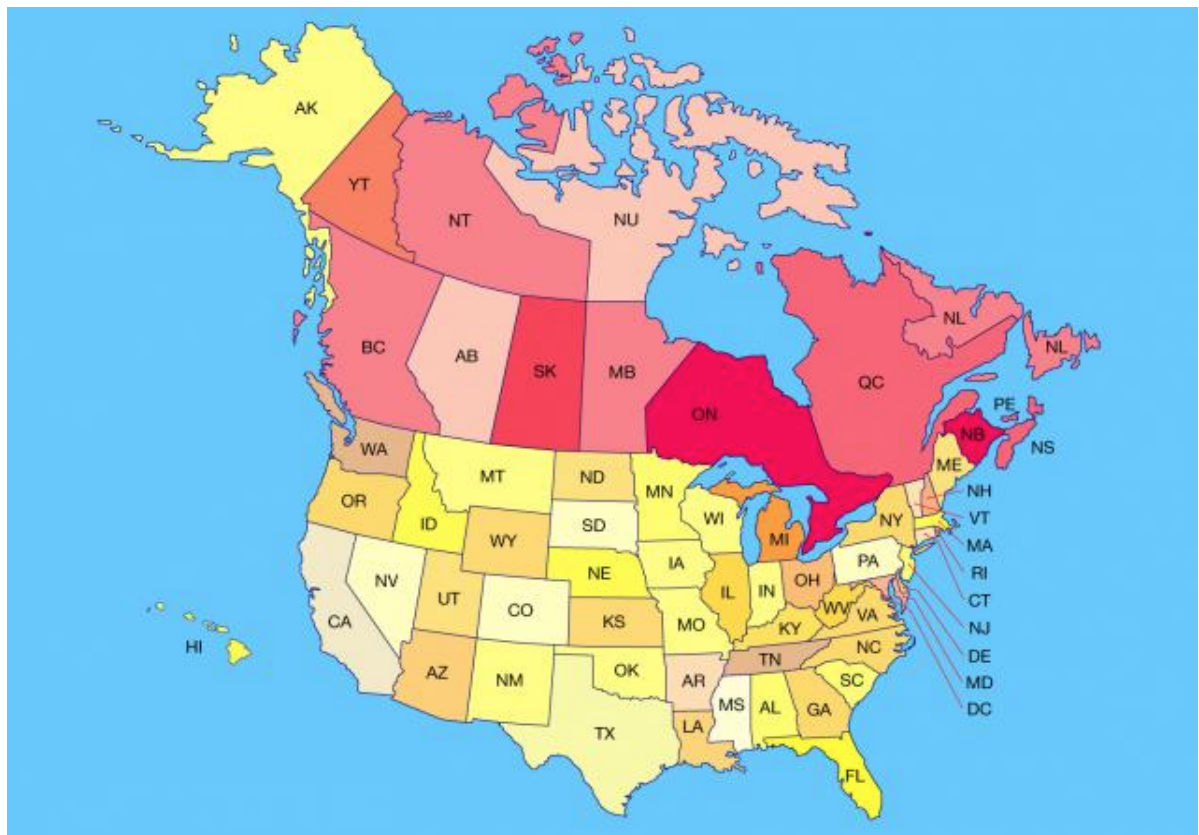
You can use whatever programming language you like.

If you code in Python, you can use the matplotlib library for plotting.

Required files :

- The adjacency list of the nodes is provided in the file “state_neighbors.txt”.

For this question, you will color the map of US and Canada with 4 colors, such that no adjacent states, provinces or territories have the same color.



This problem is NP-Complete, but it can be solved using local search or optimization algorithms. Here, you will implement the genetic algorithm for this purpose. Follow these steps in your algorithm:

Step 1: Generate the initial population

populationSize is one of the configurable parameters of the algorithm. Each member of the population is called a chromosome. Initially, all nodes of each chromosome are colored randomly.

Hint: You can use a simple array to represent the state of a chromosome.

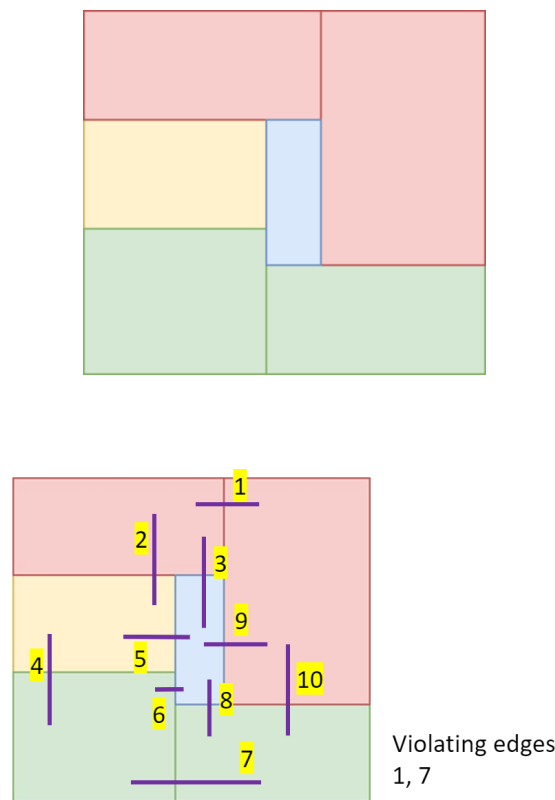
Step 2: Determine the fitness of each chromosome

The fitness function describes how good an answer is, and is used for choosing the parents. Suppose the $G = (V, E)$ is a graph with N nodes and M edges. if the color of node n_i is shown with $c(n_i)$, the coloring of the graph G would be $C(G) = \{c(n_0), c(n_1), \dots, c(n_{N-1})\}$.

function $\delta(i, j)$ is defined as:

for all $i, j \in E$, $\delta(i, j) = 1$ if $c(n_i) \neq c(n_j)$ and $\delta(i, j) = 0$ if $c(n_i) = c(n_j)$

For each chromosome, the fitness function is defined as $F(C(G)) = \frac{\sum_{i,j \in E} \delta(i, j)}{M}$ which is actually describing the proportion of the constraints that are satisfied. For example, in the coloring below, we have 6 nodes and 10 edges.



The fitness in this graph is equal to $\frac{8 \times 1 + 2 \times 0}{10} = 0.8$

Step 3: Choose the parents

In this step, you use the tournament selection¹ method. This method first randomly chooses k members of the population and then chooses the best of them. For example, if the number of initial population is 100 and $k = 4$, then 25 tournaments will be held and 25 chromosomes will be chosen, which are going to be the parents. Note that k is one of the configurable parameters of the algorithm.

¹ https://en.wikipedia.org/wiki/Tournament_selection

Step 4: Produce the next generation

In this step, crossovers are done on the parents, which produce new children by combining the genes of each two of the chosen parents. This is repeated until the number of children reaches *populationSize*. The pseudo code below is a simple example of this procedure:

```
for i = 1 to populationLength :  
    select x randomly from parents  
    select y randomly from parents where (y != x)  
    newChromosome[i] = crossover(x,y)  
end for;
```

There are different ways for defining the crossover function. Here, you divide each parent into 2 subgraphs, such that their intersection is empty and their union forms a new chromosome.

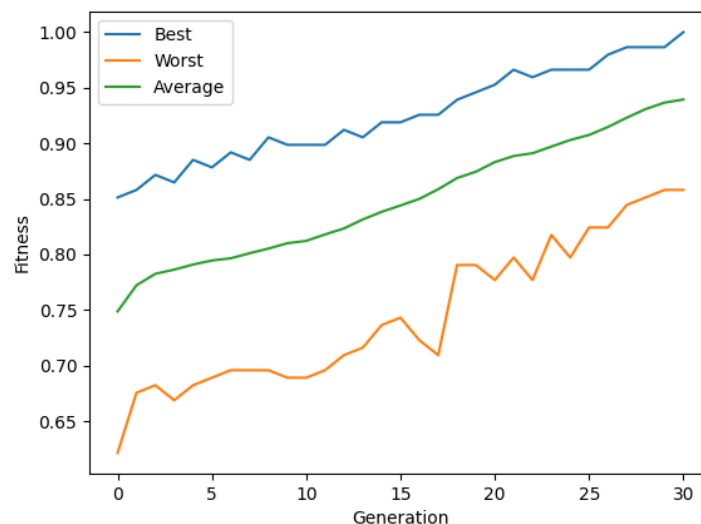
Step 5: Mutation

In this step, you randomly mutate a certain number of genes in the whole new population. Suppose that each chromosome has N colorable nodes, which are called genes. The total number of mutated genes is: $mutatedGenes = populationSize \times N \times mutationRate$ where *mutationRate* is a configurable parameter of the algorithm.

Step 6: Repeat

In this step, you replace the old population with the new one. The loop repeats at most *numberOfGeneration* times, which is one of the configurable parameters of the algorithm, until a solution is found. You should print the color of each node whenever you find the solution.

Store the best, the worst, and the average value of the fitness function for each generation and use them to plot these variables throughout generations:



Solve this problem using all the following parameters and discuss the effects of each parameter on the convergence.

numberOfGenerations = 50, 500, 5000

populationSize = 10, 100, 1000

tournamentSize = 2 for *populationSize* = 10 and 2, 5, 10 for *populationSize* = 100, 1000

mutationRate = 0.01, 0.02, 0.05, 0.1

Compile your findings in a table.

Sample output

```
Generation #0
Best fitness: 0.8581081081081081
Worst fitness: 0.6216216216216216
Average fitness: 0.7485405405405402

Generation #1
Best fitness: 0.8716216216216216
Worst fitness: 0.6554054054054054
Average fitness: 0.7722027027027025

Generation #2
Best fitness: 0.8716216216216216
Worst fitness: 0.6621621621621622
Average fitness: 0.7811689189189194

Generation #3
Best fitness: 0.8851351351351351
Worst fitness: 0.6621621621621622
Average fitness: 0.7876148648648641
```

....

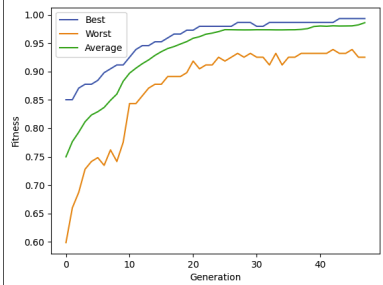
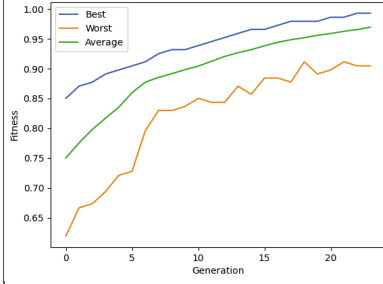
```
Answer:
AB: 3
BC: 1
MB: 2
NB: 1
NL: 3
NS: 0
NT: 2
NU: 1
```

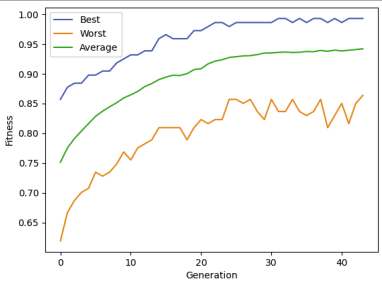
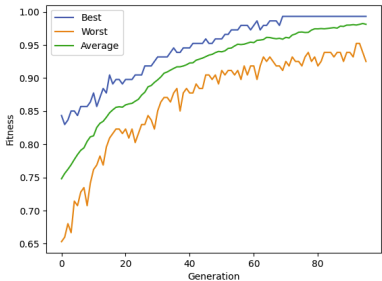
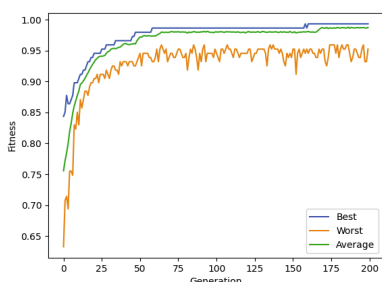
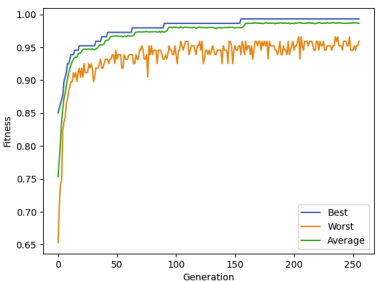
...

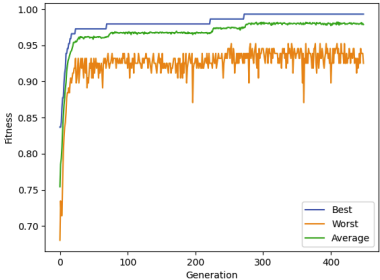
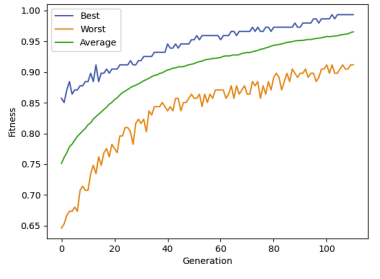
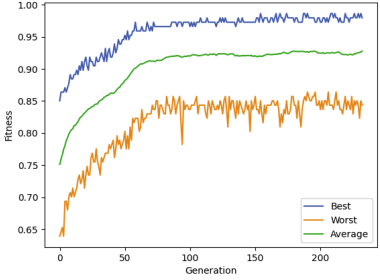
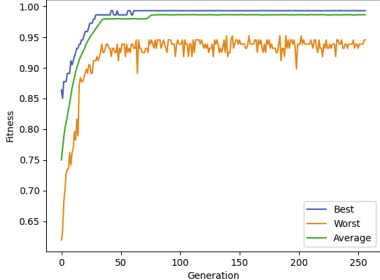
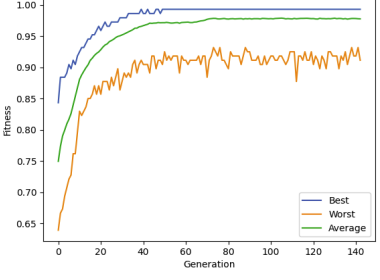
Once it found a solution, then it should show the graph.

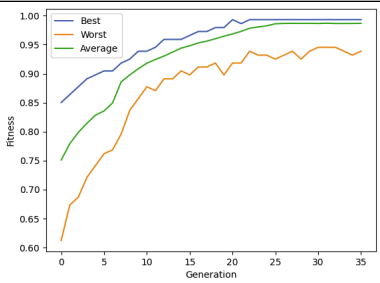
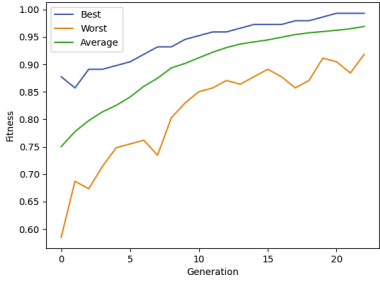
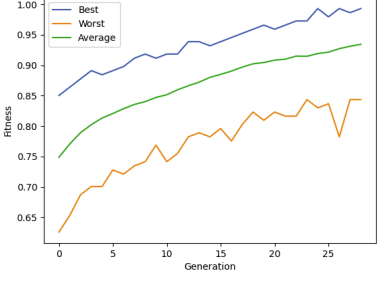
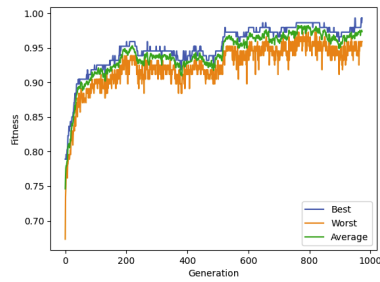
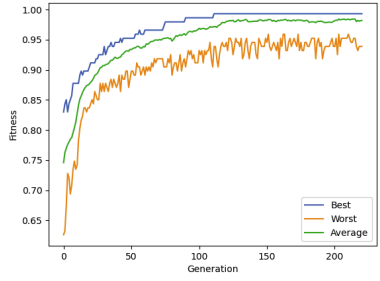
We used numbers to represent color. You can use numbers (0,1,2,3) or strings (red, green, blue etc).

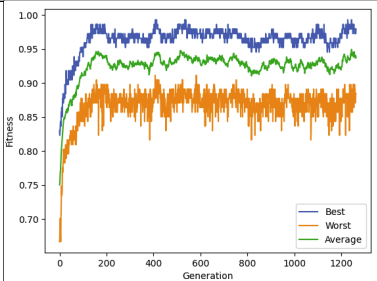
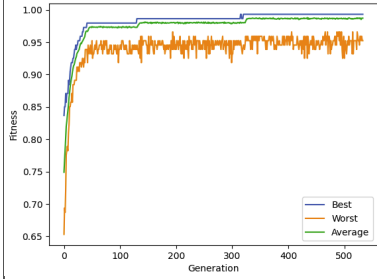
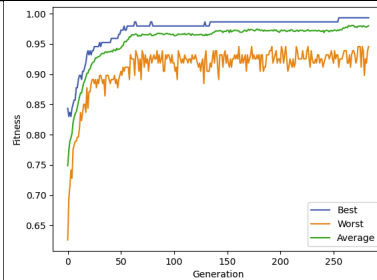
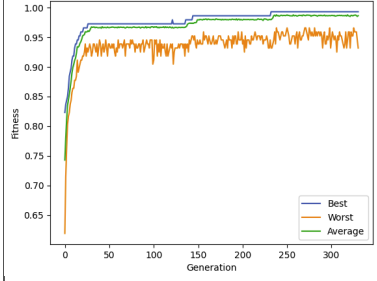
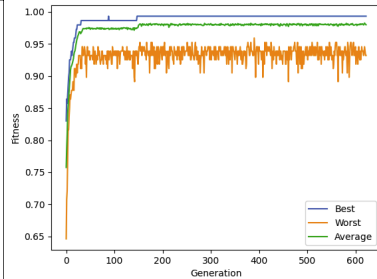
What to hand in for this question: code, table, graph

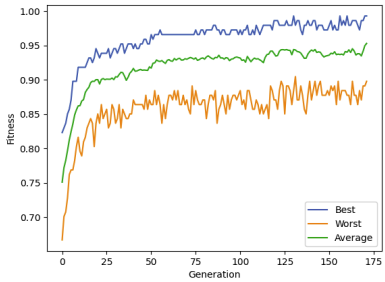
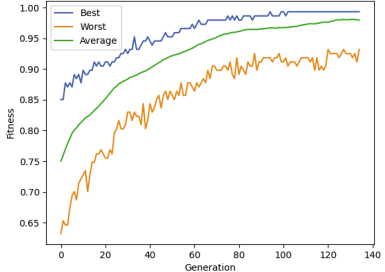
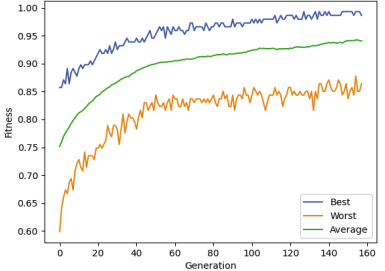
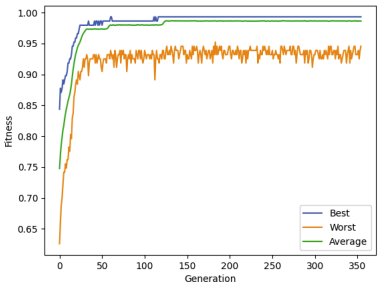
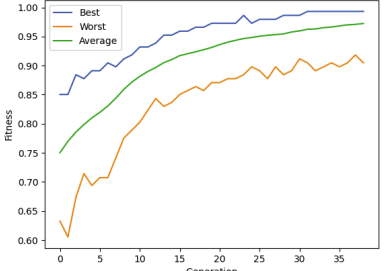
Number of Generations	Population Size	Tournament Size	Mutation Rate	Best Fitness	Executed Iterations	Execution Time (s)	Graph
50	10	2	0.01	0.8979591837	50	0.19133	Can't find a solution within 50 generations
			0.02	0.8707482993	50	0.19324	Can't find a solution within 50 generations
			0.05	0.8231292517	50	0.23357	Can't find a solution within 50 generations
			0.1	0.8367346939	50	0.19662	Can't find a solution within 50 generations
	100	2	0.01	0.9727891156	50	1.85273	Can't find a solution within 50 generations
			0.02	0.9319727891	50	1.86263	Can't find a solution within 50 generations
			0.05	0.8979591837	50	1.8579	Can't find a solution within 50 generations
			0.1	0.8571428571	50	1.99237	Can't find a solution within 50 generations
		5	0.01	0.9727891156	50	1.80618	Can't find a solution within 50 generations
			0.02	0.9727891156	50	1.82894	Can't find a solution within 50 generations
			0.05	0.9455782313	50	1.88758	Can't find a solution within 50 generations
			0.1	0.8979591837	50	1.88892	Can't find a solution within 50 generations
		10	0.01	0.9931972789	50	1.83496	Can't find a solution within 50 generations
			0.02	0.9795918367	50	1.86356	Can't find a solution within 50 generations
			0.05	0.9795918367	50	1.91006	Can't find a solution within 50 generations
			0.1	0.8979591837	50	1.93348	Can't find a solution within 50 generations
	1000	2	0.01	0.9591836735	50	16.96052	Can't find a solution within 50 generations
			0.02	0.9523809524	50	17.06312	Can't find a solution within 50 generations
			0.05	0.8843537415	50	17.52683	Can't find a solution within 50 generations
			0.1	0.8707482993	50	18.09331	Can't find a solution within 50 generations
		5	0.01	0.9863945578	50	17.23649	Can't find a solution within 50 generations
			0.02	0.9931972789	50	17.41878	Can't find a solution within 50 generations
			0.05	0.9591836735	50	17.58453	Can't find a solution within 50 generations
			0.1	0.9047619048	50	18.00073	Can't find a solution within 50 generations
		10	0.01	1	48	17.24679	
			0.02	1	24	8.36852	

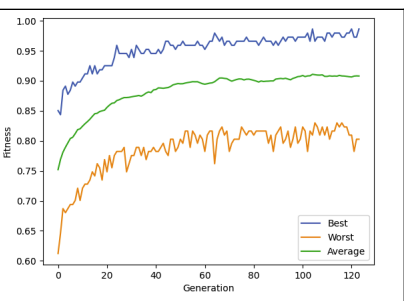
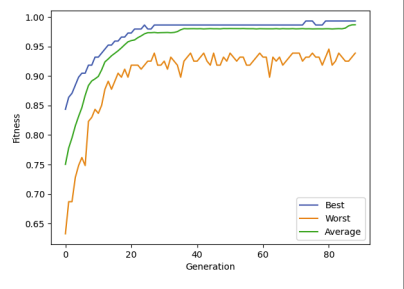
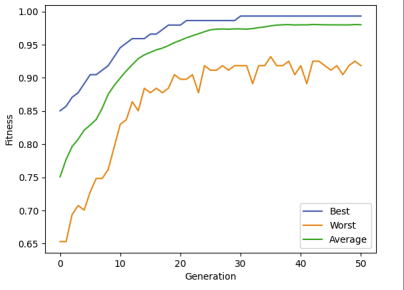
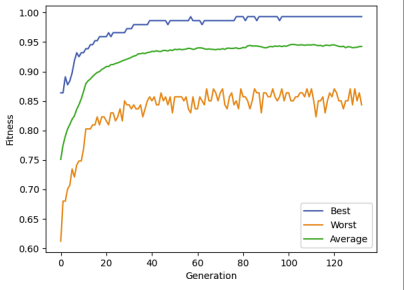
500	10	2	0.05	1	44	15.31128	
			0.1	0.9319727891	50	18.31601	Can't find a solution within 50 generations
			0.01	0.9795918367	500	1.92975	Can't find a solution within 500 generations
			0.02	0.8979591837	500	1.96532	Can't find a solution within 500 generations
			0.05	0.8299319728	500	1.98267	Can't find a solution within 500 generations
			0.1	0.8027210884	500	2.09447	Can't find a solution within 500 generations
	100	2	0.01	1	96	3.72687	
			0.02	0.9795918367	500	18.14991	Can't find a solution within 500 generations
			0.05	0.8707482993	500	18.19438	Can't find a solution within 500 generations
			0.1	0.8503401361	500	18.65922	Can't find a solution within 500 generations
		5	0.01	1	200	7.44545	
			0.02	0.9931972789	500	18.30228	Can't find a solution within 500 generations
			0.05	0.9591836735	500	18.33075	Can't find a solution within 500 generations
			0.1	0.8639455782	500	18.93038	Can't find a solution within 500 generations
		10	0.01	1	256	9.56116	

1000		0.02	1	449	20.16031	
		0.05	0.9795918367	500	18.4307	Can't find a solution within 500 generations
		0.1	0.9047619048	500	18.92186	Can't find a solution within 500 generations
	2	0.01	1	111	37.76587	
		0.02	1	233	79.3335	
		0.05	0.8979591837	500	170.83633	Can't find a solution within 500 generations
		0.1	0.8911564626	500	176.73157	Can't find a solution within 500 generations
	5	0.01	1	257	96.31763	
		0.02	1	143	52.93843	
		0.05	0.9727891156	500	171.78934	Can't find a solution within 500 generations
		0.1	0.8843537415	500	177.19309	Can't find a solution within 500 generations

		10	0.01	1	36	13.15024	
			0.02	1	23	8.55856	
			0.05	1	29	10.70809	
			0.1	0.9319727891	500	176.69396s	Can't find a solution within 500 generations
5000	10	2	0.01	1	973	3.69537	
			0.02	0.9047619048	5000	19.41559	Can't find a solution within 5000 generations
			0.05	0.8503401361	5000	19.55192	Can't find a solution within 5000 generations
			0.1	0.8639455782	5000	20.28684	Can't find a solution within 5000 generations
	100	2	0.01	1	221	7.83126	

	0.02	1	1263	45.50502	
	0.05	0.8639455782	5000	181.99014	Can't find a solution within 5000 generations
	0.1	0.8571428571	5000	187.72816	Can't find a solution within 5000 generations
5	0.01	1	534	19.04219	
	0.02	1	284	10.14872	
	0.05	0.9523809524	5000	183.45912	Can't find a solution within 5000 generations
	0.1	0.8775510204	5000	181.38406	Can't find a solution within 5000 generations
10	0.01	1	331	11.86891	
	0.02	1	623	22.62906	

1000		0.05	1	172	6.41526	
		0.1	0.9183673469	5000	180.46114	Can't find a solution within 5000 generations
	2	0.01	1	135	43.40453	
		0.02	1	158	50.4863	
		0.05	0.9115646259	5000	1736.87808	Can't find a solution within 5000 generations
		0.1	0.8639455782	5000	1710.67151	Can't find a solution within 5000 generations
	5	0.01	1	355	112.83611	
		0.02	1	39	12.5936	

10	0.05	1	124	40.73971	
	0.1	0.8979591837	5000	1738.7592	Can't find a solution within 5000 generations
	0.01	1	89	28.99584	
	0.02	1	51	17.06733	
	0.05	1	133	44.75184	
	0.1	0.9319727891	5000	1752.1159	Can't find a solution within 5000 generations

Discuss the effects of each parameter on the convergence:

- Number Of Generations: More likely to find a solution with larger Number Of Generations, may not find a solution if Number Of Generations is too small (50).
- Population Size: Fewer generations are needed to find a solution with a larger Population Size, but executed time increases for each generation run.
- Tournament Size: Fewer generations are needed to find a solution with a greater Tournament Size, may not find a solution if the Tournament Size is too small (2).
- Mutation Rate: A mutation rate around 0.01 - 0.05 produces fewer generations. A mutation Rate of 0.1 can't find a solution even in the 5000 generations test because the parents just keep mutating and deviating from our target fitness 1, and it also takes a longer execution time.