My interests in computer science can be captured in a sentence: “Computer Science enable people to solve problems in a brand-new way”. When I first learnt programming, I realised that it is really marvellous that it could create many amazing functions, some of which can only accomplished by computers: the Four Colour theorem, the first mathematical theorem proved with the help of computer, and etc. Computing is such a wonderful way for people to solve problems which people were not able to solve before and this is the reason why I am keen on this subject.

I once attended a summer camp where we were asked to solve the Sudoku problems using algorithms. At first, I attempted to use the computer algorithms to try numbers in each square and hence get a feasible solution by backtracking. Thus, the ones came up in my mind were the Depth First Search and the Breadth First Search. Considering the Sudoku as a graph with 81 vertices and there were edges between them representing their relationships. For instance, all the grids in the same row, column or 3x3 sub-grid would have edges between their corresponding vertices. Each time, assigning a value between 1 and 9 to the vertex and the edges which connected them together could not be assigned the same colour. Then, we made a function to check the correctness of the Sudoku and saved the valid ones in the stack (DFS) or the queue (BFS). For example, in the first blank, try integer 1. If it is correct, it will go to the next blank. Else, it will backtrack to the previous one. By doing this recursively, we could work out the correct answers. DFS tries to go as far as possible whereas BFS discovers all the vertices at a certain distance before moving to the further vertices. Although the time complexity of BFS and DFS are both O(|V|+|E|), in different situations of the graph, their running time will be different. BFS depends on the width of the graph whereas DFS depends on the depth of the graph. Additionally, BFS usually takes more memory space as it needs to store more possibilities. In the case of Sudoku, the number possibilities are really large that the width of the graph is large. After testing, the DFS always takes less time than BFS.

Unfortunately, Sudoku has been proved to be a NP complete problem. Though we may validate the feasibility of a solution quickly, it still takes a long time to find one. Some harder instances, for example, the hardest Sudoku in the world with only 17 clues, which was also proved using computers that trying every possible scenario for every possible configuration, are very time-consuming. Thus, in order to speed it up, I tried to avoid some unnecessary vertices by pruning. I tried to do the constraint programming which is similar to human ways that does the eliminations. We needed to make a constraints function that each grid must contain no more than one value and should be between 1 and 9. In addition, the constraints functions for each row, column and 3x3 box that the value in each grid must be unique. For some simple Sudoku without guesses, it could work really fast by itself. For some harder Sudoku, we could combine this method with DFS and BFS that each time the candidate Sudoku was popped out from the stack or queue, we did the constraint programming. As a result, if some grids had only one possibility, then we were able to directly put the corresponding number in. Therefore, we can reduce large amount of possibilities we need to test. Just like the branches of the tree were cut, so the width of the graph reduced and the time taken reduced a lot. Also, through amounts of different Sudoku, we surprisingly found that with pruning, BFS ran quicker than DFS.

Apart from these methods, we could also use stochastic search to solve Sudoku with genetic algorithms. It is an optimization strategy that mimics natural selection. Thus, our algorithms will generate solutions to the Sudoku and apply this principle of survival of the fittest to them.

In this case, we will have a fitness function to count the number of duplicates in each row, column and box. The purpose of the fitness is to be as low as possible with an optimum value of zero. At first, it will initialize the populations by randomly generate values such that each row is a correct permutation of the values from 1 to 9. Randomly select two parent Sudoku from the population and choose various rows from two parents Sudoku to create one child. Then, the mutation operates by picking a row, and then picking two values within that row to swap randomly for the possibility of 5% each time. At the end, check the fitness of the child, if the fitness is zero, then stops, else do the steps again and get into the next generation until there is a Sudoku’s fitness is zero. It works, but it may not be effective as graph colouring method.

Through testing different methods, we found that in different situations, the speed of algorithms varies. So we can collect the time taken for each algorithm to solve different amounts of blanks of Sudoku. It could be possible for us to use machine learning to create mathematical models to find relationship of the algorithms and the number of blank grids and let the program to choose the most suitable solution for different situations.

Apart from computer science, I am also interested in Maths and Physics. I achieved a gold certificate in the UK Senior Mathematical Challenge 2018 and 2019. I have also achieved the First Place Certificate for the Problem-Solving Round in the High School Team Maths Competition in 2018, which helped me to better understand mathematical concepts in computer science like the Asymptote and big-O notations.

I enjoyed the understanding from dedicating myself to these projects and take this commitment into university. Also, I love the fact that computer science can be applied to our daily life and make our life more convenient. I cannot wait to learn more at a higher level.