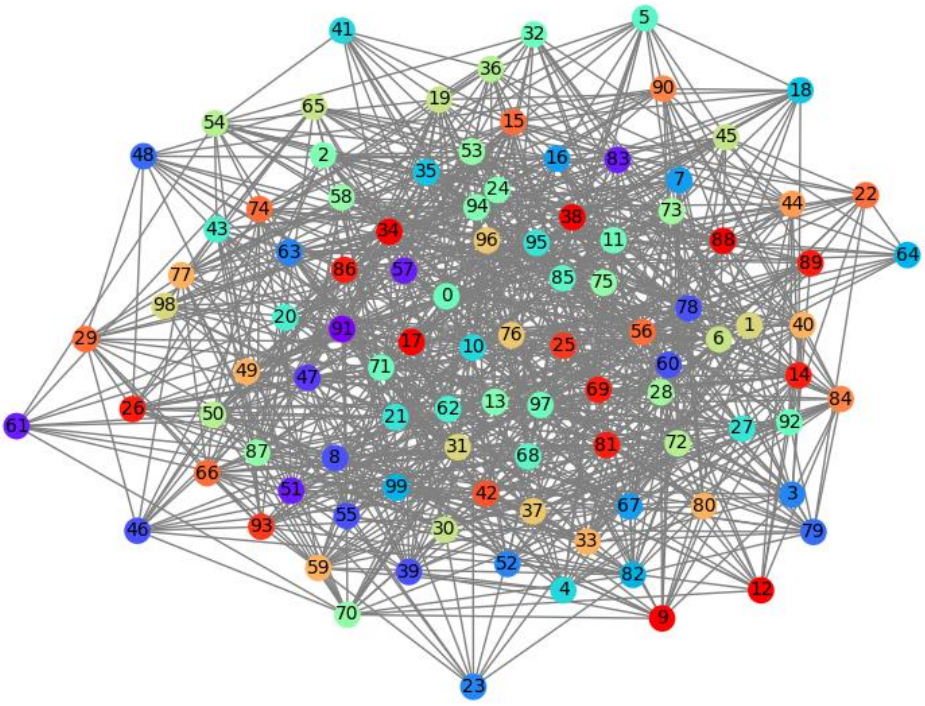
Ai Assignment 2 – Derek Kenny 20416526

Link to GitHub

Choosing a network graph topology:

For this assignment I have chosen to use the Python library, networkx, as it provides functionality for generating nodes and edges for graphs. These graph objects can then be serialised and saved as a .pickle file to be reused again. This allows for the running of experiments to have a control variable somewhat.

The networkx library can be used to generate an Erdos-Renyi grah which is a random graph that assigns edges between nodes based on a set probability. So for example, nx.erdos_renyi_graph(n, p) will generate a random graph of n nodes with the probability of an edge occurring between 2 given nodes being between 0 and 1. The generated graph looks like this with an initial number of conflicts (neighbours with the same colour) being 39. This is an undirected graph with nodes of equal weight.



The goal is to see if colouring of the graph can be carried out correctly, with no conflicts. Along with this, if no conflicts can be reached, then the number of colours to choose from should drop by 1 and the process should be repeated. This is an iterative process. The number of iterations is counted for each colour and graphed against the number of colours available. As the number of colours decreases, the number of iterations should increase as it becomes more difficult to find an optimal solution. Because it is hard to determine if there isn't a clear solution, a failsafe is added whereby the algorithm will stop if the number of iterations reaches 1000. This can be changed of course as a local optimum maybe found but not the actual optimal value.

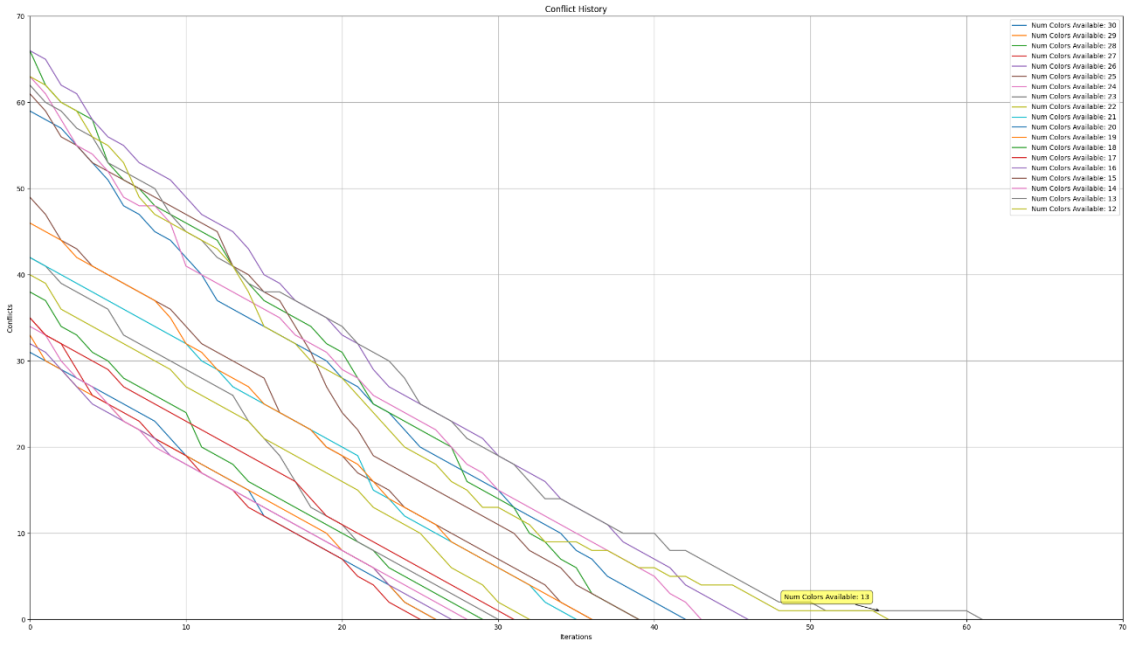
The approach I've chosen to optimise the colouring of the graph is a greedy local search approach.

- Colours are randomly assigned to nodes initially.
- Nodes chosen randomly if they have neighbours conflicting with them.
- These neighbours are then given a colour which produces the least conflicts with the selected node.
- The conflicts in the graph are recalculated.
- This approach is greedy, the best approach is identified locally to quickly optimise the graph.
- One node is selected at a time which is a slow process really.
- Previously assigned colours aren't considered which may lead to getting stuck in local minima.

Parameters:

- Graph of 100 nodes loaded from file system.
- 1000 iterations for each colour.
- Starting with 30 colours

Results Local Search



Analysis of Results

We can see that as we decrease the number of colours the number of conflicts initially starts very high. As well as this, the number of iterations to reach 0 conflicts tends to increase. The unpredictability of the graph is due to the randomness of selecting nodes which is why some lower numbers can perform slightly better despite the problem becoming harder. Due to the termination parameters (the algorithm gives up after the same number of conflicts appears repeatedly) it is possible that lower numbers are achievable. In this case, a local minimum may be hit, a problem which occurs for lower numbers due to the shape of the graph.

Problem 2

An issue with the algorithm above is its tendency to get caught in local minima. This comes down to producing random solutions using a greedy heuristic and not considering solutions which have already been considered. One thing that could be done to mitigate this is optimising the local search algorithm by using an array to store previous results. This is known as TABU search.

The process of performing tabu search is as follows:

- Colours are randomly assigned to nodes initially. Conflicts calculated; this is the initial "best" solution.
- For each node in the graph, its colour is changed to every possible colour except its current one, generating several potential solutions.
- Calculate the number of conflicts for each potential solution.
- The best solution is selected based on the least number of conflicts.
- Chosen solution is added to a tabu array so it can't be used in the future.
- The oldest entry is removed if the tabu list exceeds a certain size (10 in this case).
- If the current solution has fewer conflicts than the previous best solution, update the best solution to the current one. Number of conflicts are calculated in each iteration.
- The loop breaks after either 1000 iterations or when the number of conflicts reaches zero for a given number of colours.

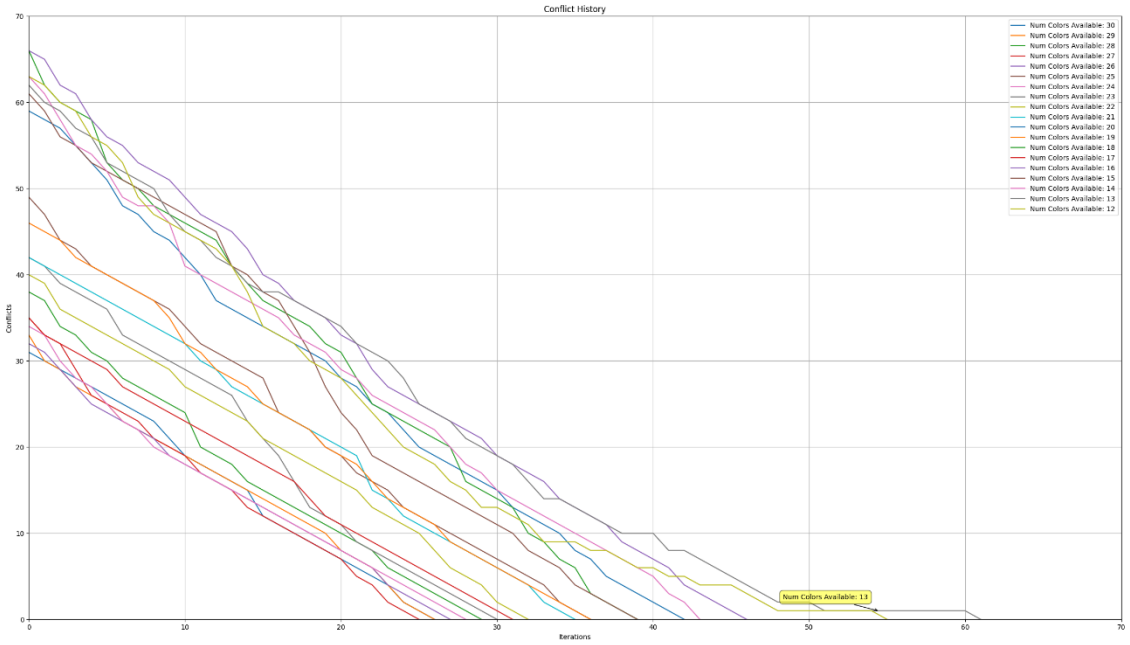
The goal of the experiment is to explore how this works in comparison to the basic local search algorithm I've used in problem 1 and how these two perform when edges are randomly added/removed to the initial graph (after 20 iterations for example).

Parameters:

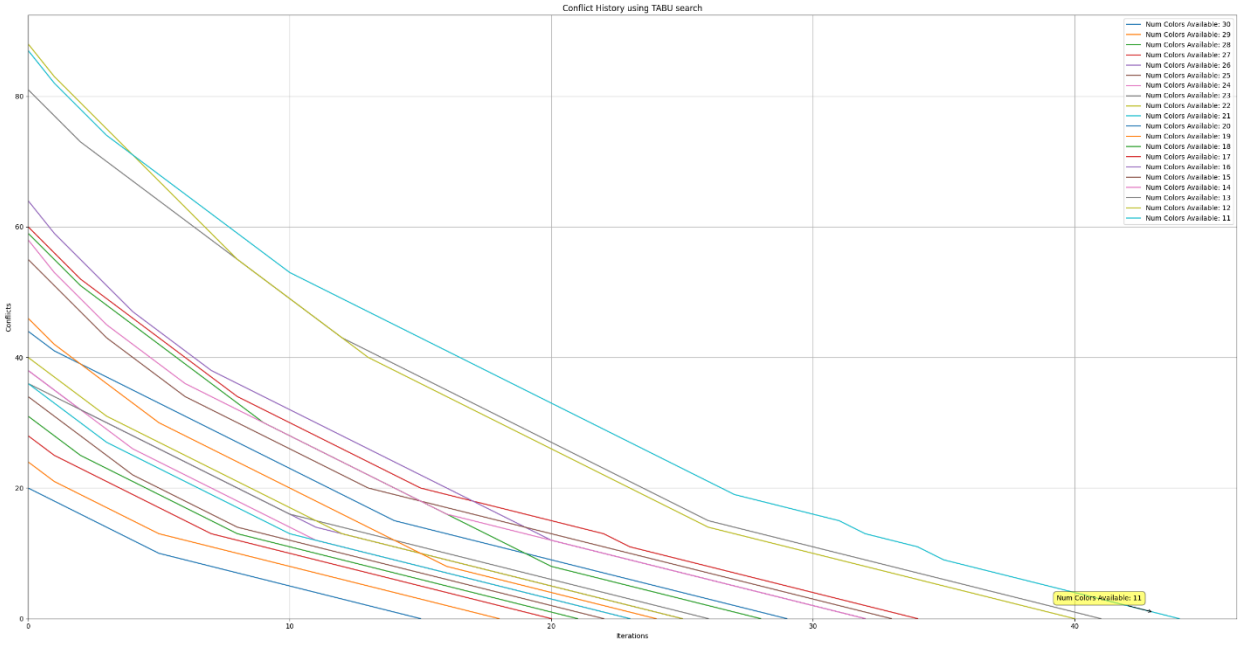
- Graph of 100 nodes loaded from file system.
- 1000 iterations for each colour.
- Staring with 30 colours.
- Max tabu array of 10.
- 20 nodes to be randomly added and 20 to be removed.

Initial Graph Results:

Local Search (Greedy)



TABU Search



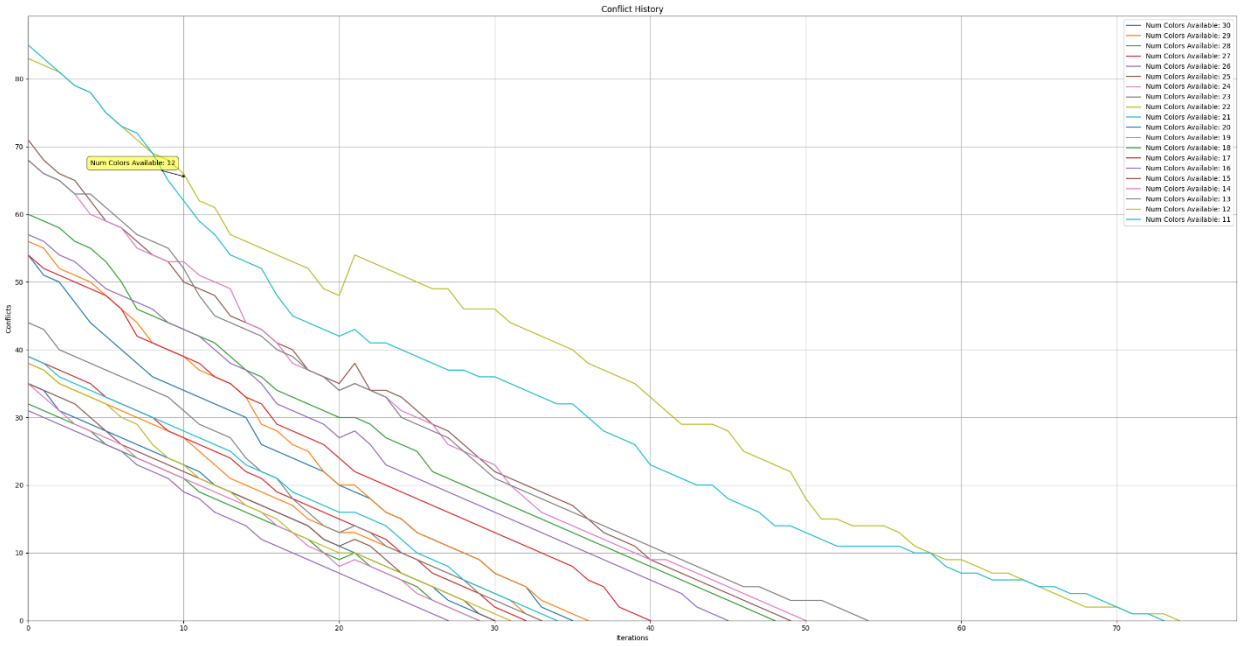
Analysis of Results

We can see that TABU search doesn't get stuck at a local minimum as often as the greedy local search. This is due to it considering all possible combinations for a random node as opposed to just trying to change the colours beside the node. With all combinations for a node considered the scope isn't as

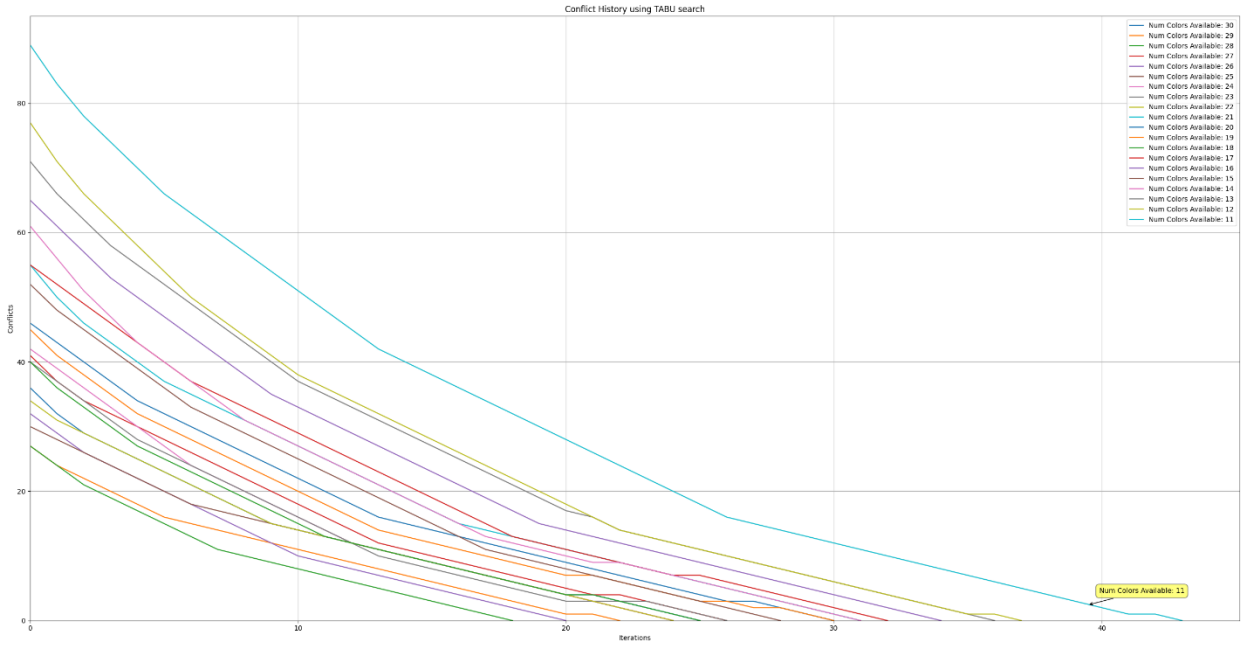
limited. This process is slower in terms of execution. However, based on the graphs above, the number of iterations is actually lower, converging on a point quicker due to the best option being chosen each time. This may matter more when the scale of the problem is much larger. For example, the number of nodes could be thousands. Increasing the size of the TABU array may be something to consider, especially for larger problems as the number of potential solutions grows.

To further test these two algorithms, I decided to introduce noise to the graph. After 20 iterations, the edges on the graph are randomly removed and edges are randomly added. This makes it more difficult for each algorithm to find the least number of conflicts as their work is undone. The results of which are below.

Local Search (Greedy)



TABU Search



<u>Analysis of Results:</u>

We can see that both algorithms do seem to suffer when the noise is introduced in each case where the number of colours changes. In the case of the greedy search, the effect is much more noticeable as we see a sharp increase in the number of conflicts after the 20-iteration mark. For the TABU search, there is no noticeable spike, indicating that it is more robust in tackling this problem due to its ability to consider all possible solutions for a given node. Results aren't as clean as the initial graph, with slight hiccups in performance (the graph remains the same over iterations) in comparison, producing lines which aren't as linear as before.

In summary:

The greedy approach works very quickly but produces a higher number of local optima, causing it to become stuck. It doesn't deal well with noise/changes to the problem space. For smaller graphs, greedy local search may work well and will achieve a result quickly, however, it may come at a cost of accuracy.

The TABU search approach adds more complexity and takes longer to run as a result. However, due to it considering many possibilities at once, it tends to escape from local optima, especially when it considers solutions it has already tried. It handles noise well for this reason, very adaptable. TABU search may be more useful for larger scale problems, especially for ones that may dynamically change size over time, it minimises the risk of getting stuck in local optima at a cost of speed.