Formulating k-Colouring Problems as Constraint Satisfaction Problems

Group Members

Yikhei Chan - chanyikh

Arseniy Ivanov - ivanovar

Christine Murad - muradch1

Type of Project

Constraint Satisfaction Problem

Roles

Yikhei:

- Encoding the CSP in Python and writing the Methods Part of the report (major role)
- Providing heuristic functions (minor role)
- Assisted in deciding type of analyzation and independent variable (major role)

Arseniy:

- Providing FC and GAC propagator (minor role)
- Performing the propagation technique analysis and doing the write up for it in the Evaluation Section (major role)
- Assisted in deciding type of analyzation and independent variable (major role)

Christine:

- Performing the Variable ordering analysis and doing the write up for it in the Evaluation section (major role)
- Creating the test graphs (minor role)
- Writing the Introduction, Obstacles/Limitations, and Conclusion section (minor roles)
- Assisted in deciding type of analyzation and independent variable (major role)

Project Motivation/Background

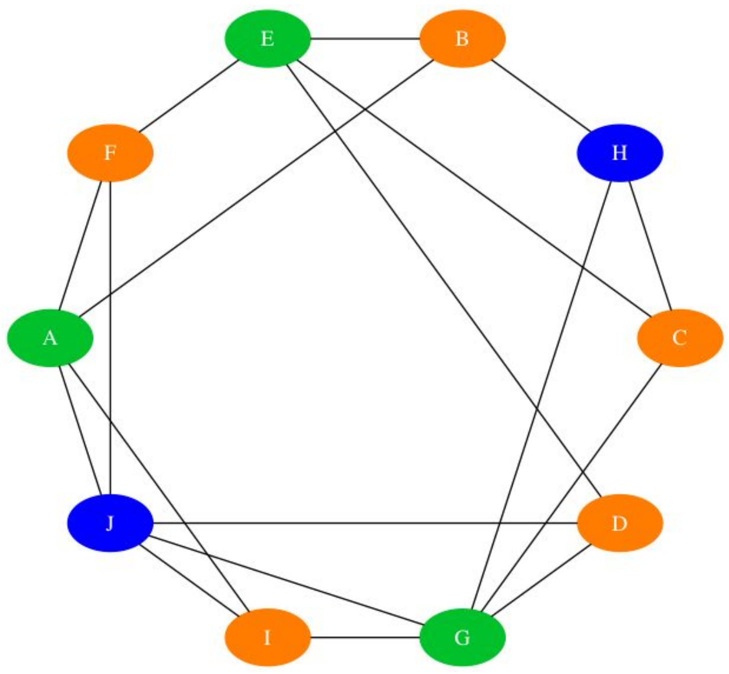

Figure 1. An example of a 3-coloured graph of 10 vertices and 18 edges.

What we are trying to accomplish in this project is to encode an NP-complete problem as a Constraint-Satisfaction problem. In particular, we wanted to encode k-Colouring as a Constraint-Satisfaction problem. k-Colouring naturally fits as a Constraint Satisfaction problem, as the main idea of k-Colouring is to be able to colour a graph so that no two nodes that are connected by an edge are coloured with the same colour. This idea of nodes that are attached by an edge not having the same colour composes itself into a constraint easily, and we can compose this problem using this constraint. Along with encoding k-Colouring as CSP, we wanted to analyze how different propagators and variable-ordering heuristics changed the amount of time finding a solution took, and how many variable assignments are attempted to find a solution to this. We looked at solving a k-Colouring using Forward Checking and Generalized Arc Consistency, and we looked at using the Degree Heuristic and the Minimum Remaining Values Heuristic to analyze our k-Colouring CSP.

Methods

The CSP takes a set of vertices V and a set of edges E which assemble a graph that we are going to colour on. We will have |V| number of variables and each variable has domain size of k. All variable share the same domain. A colour is represented using natural number. For example, variables in 3-colouring problem will have domain D = {0, 1, 2} where 0, 1, 2 are colours. We will also have |E| number of binary constraints. We create a constraint for each edge e = (v1, v2) in E such that v1 and v2 cannot have the same colour assigned. Binary constraints are preferred over higher-order constraints since k-colouring problem requires no 2 adjacent vertices have the same colour.

# of constraints	9	10	11	12	13	14	15	16	17	18	19
Min. degree	0	0	0	1	1	1	1	2	2	3	3
Max. degree	4	4	4	4	5	5	5	5	5	5	5

Table 1: Test graphs, properties

Evaluation and Results

We decided to evaluate our k-Colouring CSP by comparing the number of variable assignments made and CPU time taken to find a solution. We use two propagating techniques and two variable ordering heuristics to perform our comparisons.

Variable Ordering Heuristics

We analyzed our k-Colouring CSP using the Degree Heuristic and the Minimum Remaining Values Heuristic. We wanted to evaluate and compare how well each heuristic ran as you increased the number of constraints in the graph, to evaluate which heuristic runs better for this CSP.

We created 11 graphs to test on - each graph had a fixed node size of 10, but each test contained one extra constraint than the test before it - starting with 9 constraints and ending with 19 constraints (properties of these test graphs are defined in Table 1). We ran all 11 tests using a simple back-tracking search, with a k value of 3 (3 colours) - first with the degree heuristic, and then with the minimum remaining values heuristic. This is a graph depicting the number of assignments made for both heuristics with a k value of 3:

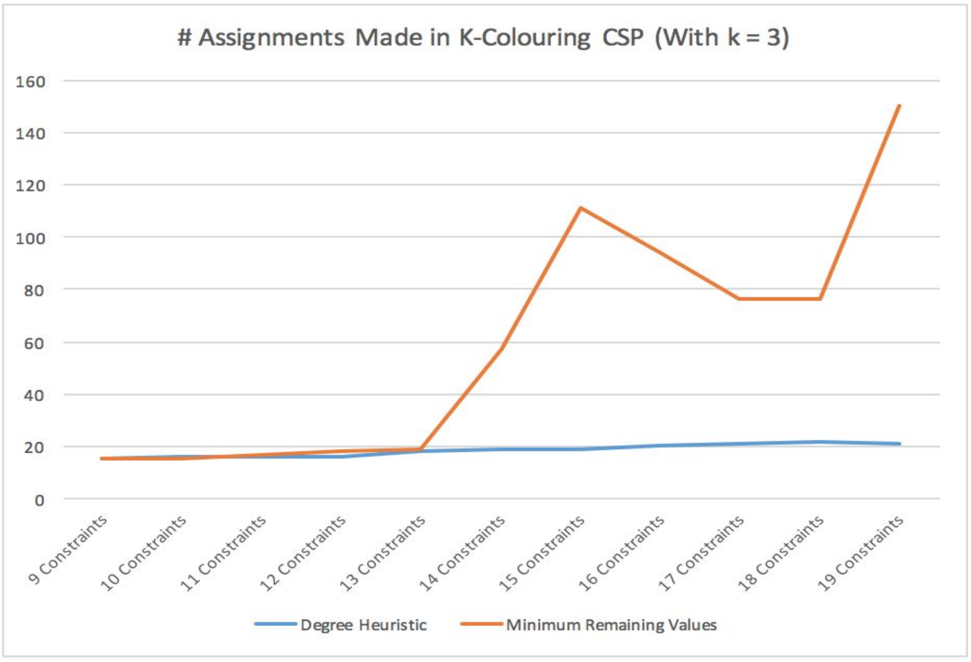

Figure 2: Number of assignments made with ordering heuristics, k = 3

After that, we increased our k-value to 4 and ran the same tests on both heuristics. Here is a graph depicting the number of assignments made for the graphs with a k-value of 4.

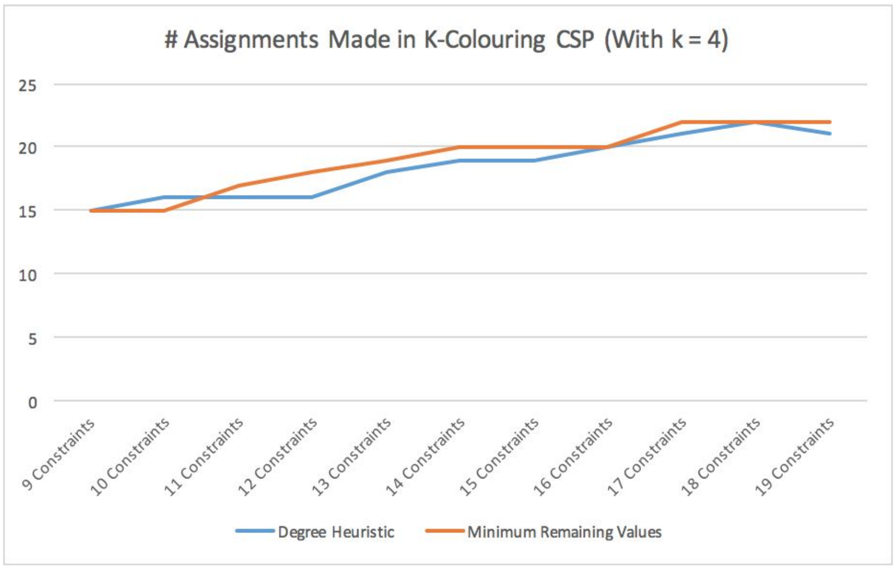

Figure 3: Number of assignments made with ordering heuristics, k = 4

As we can see, the degree heuristic performs much better as the number of constraints increase, when we're using a k-value of 3. With the minimum remaining values heuristic, the increase in number of assignments is quite drastic once we hit 14 constraints, while using the degree heuristic, it increases by no more than 1, sometimes even 0 assignments, as a constraint is added.

However, this changes when we increase our k-value to 4. As soon as we increase our k-value to 4, the number of assignments made for both the degree heuristic and the minimum remaining values heuristic become almost identical.

We see that the less values we have per variable, the faster degree heuristic performs compared to the minimum remaining values heuristic. The k-value increases the number of colours assignable to each variable - the colours are the values for each variable. As we give each variable one more value that is assignable, each variable domain becomes less constrained - there are more legal values for each variable. It is interesting that there is such a drastic change by just adding one more possible colour to each variable.

Propagation Techniques

To evaluate the effects of the propagators on the problem, two propagation algorithms were used: Forward Checking (FC) and Generalized Arc Consistency (GAC); and a simple backtracking search (BT) was used as a baseline. The test graph from the previous part was used; however, the CSP variable ordering was now fixed to sequential variable ordering, and the propagators varied. The number of assignments made under each propagation technique was recorded.

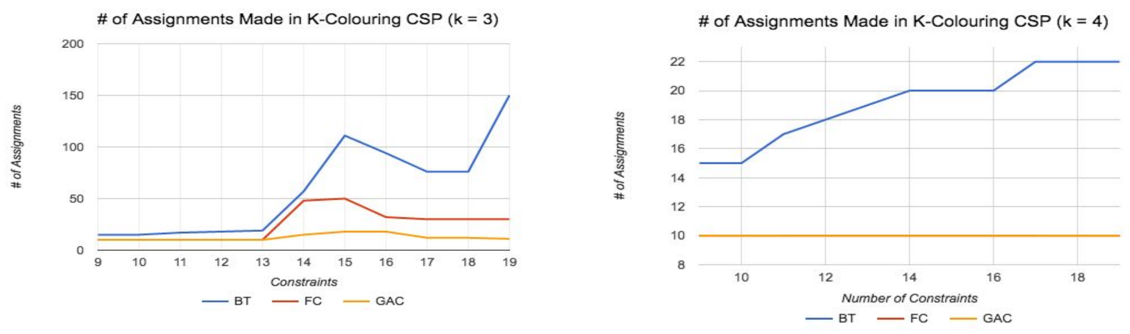

Figure 4: # of Assignments Made for (a) k = 3; and (b) k = 4.

We observed that, in all cases, BT was outperformed by FC and GAC. For k = 3 with fewer than 13 constraints, as well as for every case with k = 4, 10 assignments were made in total under FC and GAC, indicating that applying propagation algorithms to the given graph was sufficient to reduce the domains of the variables to a solution set.

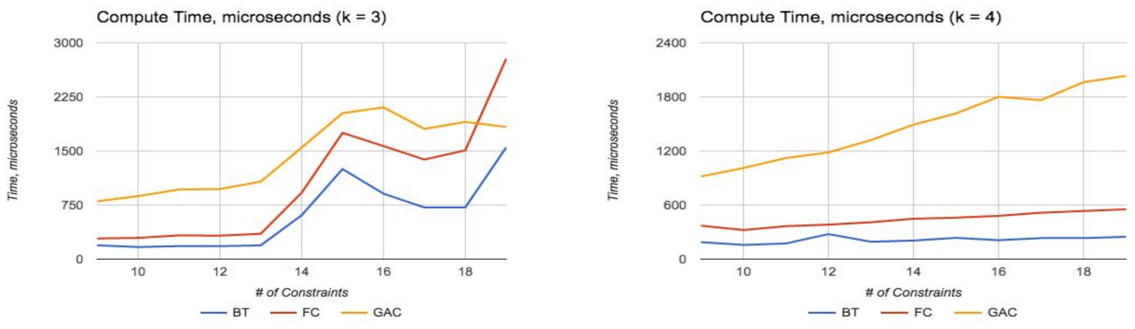

Figure 5: Compute Time to solve, (a) k = 3; and (b) k = 4.

While the number of assignments made to solve a k-Colouring CSP above can be considered a primary metric for the performance of a propagator, we observed that the compute time for each of the propagators was exhibited significantly different behaviours. In particular, BT performed best in terms of computation time on all of the problems. GAC performed and scaled better in highly constrained problems with k = 3, while FC performed and scaled better in k = 4 problems.

Overall, the performance of propagators was observed to be closer connected to the internal properties of the graph in the CSP and the k-value than to the explicit number of constraints. This experiment shows that performing a single pass of GAC on a k-colouring CSP may overall decrease the space of variable assignments, while FC achieves the best time/assignments ratio in iterative solving of lightly-constrained instances of the problem.

Limitations/Obstacles

One obstacle we had to overcome was deciding what our independent variable should be when analyzing our heuristics. Increasing the number of constraints is not the only thing that comes into play to change the number of assignments made for each heuristic - it is also between which variables constraints are made that would have a large effect. In our testing, we kept the degree of each variable relatively equal - if we had biased which variables constraints were placed between, we might have a different analyzation altogether. As we observed in the propagator evaluation, again we find that it is more than the number of constraints that determines how much time and assignments are made when finding a solution to the k-Colouring problem.

Conclusions

In our analyzation, we found that the Degree Heuristic performed much better on number of assignments made compared to the Minimum Remaining Values heuristic, on a k-value of 3, but that it balanced out when we used a k-value of 4. We also found that back-tracking search had the best computing time for our k-Colouring problems, but that Forward Checking and Generalized Arc consistency outperformed back-tracking search on number of assignments made - GAC performing the best with a k-value of 3. One thing we realized when encoding our test problems, as mentioned before, was that the way the constraints were created was an important part of how our evaluations performed. Our independent variable for our analyzations was “number of binary constraints in the graph”. As mentioned in our limitations, however, which variables the constraints apply to and the degree of each variable has a large effect for the way our metrics could have run. Further work on this problem could be to test more closely on the degree of each variable, and analyzing the number of assignments made when a graph has consistent degrees for each variable, and when the degree for different variables is drastically different.