Constraint Programming Assignment 2: Solver Comparison

2407682H

November 6, 2022

1 Introduction

Both the CSP and ILP models expect pre-defined variables describing the environment including a graph G represented as an adjacency matrix and the number of nodes n in the graph. An adjacency matrix allows the neighbourhood of a vertex to be retrieved more effectively than with other graph representations. The maximum time for the game is defined as T, the initial fire locations as a list f of node indexes, and an integer *budget* representing the budget of firefighters at each turn. The MiniZinc model makes use of the global $count(A, v, c)$ constraint which ensures that an array A contains no more than c occurrences of v.

2 MiniZinc CSP Model

The decision variables are two n-element arrays, b and d, with each element of each list taking a value in the range $0..T$ representing the time a node is burned or defended respectively. This is a more compact representation than an $n \times T$ matrix as in the $0-1$ ILP encoding and allows the constraint of remaining within budget to be implemented by simply requiring $count(d, t, budget)$ for $2 \leq t \leq T$. The MiniZinc model is implemented slightly differently so that the initial fires begin at $t = 1$ and no nodes are defended at $t = 1$, and this constraint is enforced by $count(d, 1, 0)$. Initial fire locations can also be set by constraining $b[i] = 1 \forall i \in f$. The model ensures that burned nodes cannot be defended and that defended nodes cannot be burned by constraining $d[x] = 0$ if $b[x]! = 0$ and $b[x] = 0$ if $d[x]! = 0 \forall 1 \leq x \leq n$. The final constraint controls the spread of the fire throughout the graph. It is a conditional constraint that iterates over each node x and iterates over the neighbours y of each node, if y is burning and x is neither burning nor defended, then x can either burn or be defended at time $b[y] + 1$. Finally, the model's solution is calculated by maximising $count(b, 0)$[1], which means that they are either defended or saved. The model makes use of Google's OR-Tools and its FlatZinc solver for use with MiniZinc. This solver was chosen for its efficiency compared to MiniZinc's default solvers such as Gecode which were too slow when approaching this problem.

3 PulP ILP Model

The PulP ILP model is implemented with the constraints in the week 4 lecture material, and works slightly differently to the MiniZinc CSP model. The first difference is that the decision variables b and d are implemented here as $b_{x,t}$ and $d_{x,t}$ for $x \in V, 1 \leq t \leq T$, which are 1 if node x is burned or defended at time t and 0 otherwise. This representation is essentially an $n \times T$ matrix allowing each time step to be represented as a row of the matrix. In this encoding the time begins at $t = 0$ and therefore the values of $b_{x,0} = 0 \forall x \in V$. The rest of the constraints can be implemented in a nested for-loop iterating over each t in $1..T$. For each time step the sum of the defended nodes at time $t - 1$ is subtracted from the sum of the defended nodes at time t and this value cannot exceed the budget. Within the earlier for-loop is another for-loop iterating over the nodes $x \in V$. In this nested for loop the initial fire locations are enforced as $b[x][0] = x$ if x is in f. Burning nodes continue burning and defended nodes continue being defended, and this is enforced by ensuring that $b[x][t] \geq b[x][t-1]$ and $d[x][t] \geq d[x][t-1]$. The inability to defend burning nodes and vice versa is enforced by the constraint $b[x][t] + d[x][t] \leq 1$, as at most one value may be 1 and therefore the other will be 0. In the final nested for-loop the neighbourhood of x, $y \in N(x)$ is iterated over and the constraint $b[x][t] + d[x][t] \geq b[y][t-1]$ dictates the spread of the fire by ensuring that a node with a burning neighbour can either be defended or burned at the following time step. This model makes use of PulP's default CBC (COIN-OR Branch-and-Cut) solver which was chosen for convenience and due to the fact that it was found that many PulP-compatible solvers require a commercial license for problems of this scale.

4 Benchmark Instances and Experimental Pipeline

The graphs used for experimentation were obtained from a conveniently provided benchmark set [5]. Three sets of benchmark graphs were used with different generation strategies, BBGRL [2], GBRL [4], and GEN [3, 1] and the maximum nodes of graphs was limited to 50 due to computational limitations. The experimental pipeline is a Python script set up in such a way that, given a directory of benchmark instances the code will iterate over each instance and extract the relevant information for the graph G and the values of n, T, and f provided the instance has a given number of nodes, for example during experimentation nodes were limited to 50. This node limitation and the firefighter budget for each model can be changed in the python script "firefighter.py", in the experiments here the budget was set to 1. Functions that govern the use of the MiniZinc CSP solver and the PulP ILP solver are then passed G, n, t, and F one after the other and the times reported from each solver are recorded for each benchmark instance. Times are recorded in seconds

[1] Here count is a function that returns the number of occurrences in the array.

using Python's built-in datetime function. The function handling the MiniZinc CSP solver loads the model from the "firefighter.mzn" file and sets the given values within the model before calling the solver, in this case Google's OR-Tools. The PulP solver function is simply the implementation of the model's constraints before calling the solver on the model. A Python script was chosen for this reason, as implementing the PulP model in Python allowed for it to be implemented within the pipeline script. In both cases the timeout limit was set to 5 minutes or 300 seconds, as anything above this threshold for a graph of 50 nodes seems unreasonable. Experiments were run on a laptop with a 4-core Intel i5 2.5Ghz processor and 32GB RAM, this limited the size of the graphs that were experimented on due to resource limitations, and may have impacted the accuracy of results as other processes that could not be accounted for may have been running during the time that the experiments were taking place.

5 Results

Figure 1 shows the time comparisons for each solver for each set of benchmark instances. Each point on the graph represents a benchmark instance and its position indicates the time taken by each solver with 300 seconds being the timeout for each. GBRL was the only set of instances to result in a timeout for either solver, with one timeout for the CBC solver and 2 for the OR-Tools solver. The figure also shows that OR-Tools performed worse on this set of instances than any other, whereas the CBC solver had mixed performance over all instance sets. Overall the OR-Tools solver vastly outperforms the CBC solver for most instances, with many solutions being found in only a few seconds, although there are a few instances of the GBRL set in which both solvers perform identically and some GBRL and BBGRL instances where the CBC solver outperforms the OR-Tools solver. In conclusion the plot appears to show that the OR-Tools solver with the CSP model is a far better choice in terms of time taken to find a solution than the CBC solver, however due to the limited amount of instances tested further experimentation with larger graphs from the benchmark sets may show that the CBC solver with ILP modelling could also be a good or better choice.

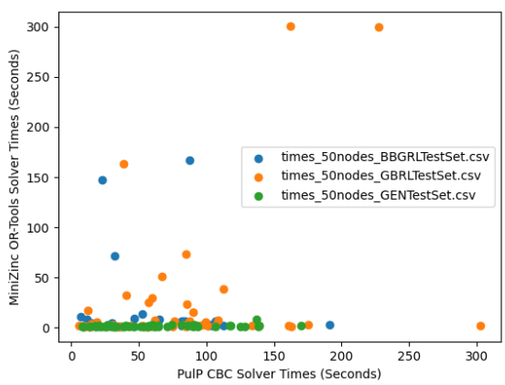

Figure 1: Performance comparison for each solver in seconds.

References

[1] Barabási, A.-L., and Albert, R. Emergence of scaling in random networks. *science 286*, 5439 (1999), 509–512.

[2] Blum, C., Blesa, M. J., García-Martínez, C., Rodríguez, F. J., and Lozano, M. The firefighter problem: Application of hybrid ant colony optimization algorithms. In *Evolutionary Computation in Combinatorial Optimisation* (Berlin, Heidelberg, 2014), C. Blum and G. Ochoa, Eds., Springer Berlin Heidelberg, pp. 218–229.

[3] Erdős, P., and Rényi, A. On random graphs i. *Publicationes mathematicae 6*, 1 (1959), 290–297.

[4] García-Martínez, C., Blum, C., Rodriguez, F., and Lozano, M. The firefighter problem: Empirical results on random graphs. *Computers Operations Research 60* (2015), 55–66.

[5] Ramos, N., de Souza, C. C., and de Rezende, P. J. The firefighter problem on graphs - benchmark instances, 2018. *www.ic.unicamp.br/~cid/Problem-instances/ffp*.