Constraint Programming Assignment 2: Solver Comparison

2675767C

1 Introduction

Both the Constraint Programming (CP) and Integer Linear Programming (ILP) models were designed to solve the Infectious Firefighter Problem (IFP). In this problem, the environment is represented by a graph G, with an adjacency matrix, the number of nodes n, the maximum time T, and the firefighter budget. The initial fire locations are given as a list of node indexes, and the firefighter budget represents the number of nodes that can be defended at each time step. The adjacency matrix representation was chosen for its efficiency in retrieving a vertex's neighbours. The goal of the problem is to maximize the number of unburned vertices by strategically deploying firefighters to defend nodes while the fire spreads.

The CP model was implemented using MiniZinc, while the ILP model was constructed using the PulP library. These two solvers were compared in terms of their performance on several benchmark graph instances. The MiniZinc model uses the global count(A, v, c) constraint to ensure that no more than c firefighters are assigned to defend at any time t. In contrast, the PulP ILP model uses binary decision variables for each node and time step, represented in an n × T matrix.

2 MiniZinc CSP Model

The CP model in MiniZinc uses two arrays, b and d, to represent the burn and defend status of each node at each time step. Each element in these arrays can take a value between 0 and T, where b[x] = t means node x is burning at time t, and d[x] = t means node x is defended at time t. This compact representation is more efficient than using an n × T matrix as in the ILP model. The CP model starts with an initial fire at t = 1, and no nodes defended at that time, which is enforced by the constraint count(d, 1, 0). The initial fire locations are set by assigning b[i] = 1 for each i ∈ f, where f is the set of initial fire locations. The key constraints in the model include: A node that is burned cannot be defended, and vice versa; The number of firefighters deployed at each time step is bounded by the budget, which is enforced by count(d, t, budget) for 2 ≤ t ≤ T; The fire spreads to adjacent nodes at each time step, modelled by a conditional constraint: if a neighbour node y is burning at time t, node x can either be burned or defended at time t + 1. The objective of the CP model is to maximize the number of saved nodes by maximizing count(b, 0), which counts the nodes that remain unburned.

3 PulP ILP Model

The ILP model is formulated using binary decision variables b[x][t] and d[x][t], where b[x][t] = 1 means node x is burned at time t, and d[x][t] = 1 means node x is defended at time t. The ILP model works on an n × T matrix representation, where the time steps are explicitly represented in the rows of the matrix. Key differences between the ILP and CP models include: Time step starting at t = 0: In the ILP model, b[x][0] = 0 for all x ∈ V, meaning no node is initially burned at time t = 0; Fire spread: The fire spreads from node x to node y if y is burning at time t-1. This is enforced by the constraint b[x][t] + d[x][t] ≥ b[y][t-1], ensuring that a node with a burning neighbor can either be defended or burned in the next time step; Defend and burn exclusivity: A node cannot be both burned and defended at the same time, enforced by the constraint b[x][t] + d[x][t] ≤ 1. The ILP model is implemented using PulP's default CBC solver, which was chosen for its accessibility and ease of use within the Python-based pipeline.

4 Benchmark Instances and Experimental Pipeline

The The study used benchmark graph instances generated with three strategies: BBGRL, GBRL, and GEN, limited to 50 nodes due to computational constraints. The experimental pipeline, implemented in Python, automated the testing process by extracting parameters like the adjacency matrix, number of nodes, fire locations, and time limit for each graph. Both CP and ILP solvers were executed on each graph, with the MiniZinc model solved using Google's OR-Tools and the PulP ILP model solved using the CBC solver. The pipeline recorded execution times and the number of nodes saved, with a 5-minute timeout for each solver. Experiments ran on a laptop with a 8-core Intel i7 processor and 32 GB of RAM, limiting the size and complexity of the graphs. This setup may have affected result accuracy due to interference from other processes.

5 Results

The results of the experiments are shown in Figure 1, which compares the time taken by each solver for each benchmark instance. The times are reported in seconds, and the 300-second timeout is indicated for each solver. The results show that the OR-Tools solver generally outperforms the CBC solver for most instances, with many solutions being found in just a few seconds. However, the CBC solver exhibited better

performance for certain benchmark sets, particularly the GBRL instances, where both solvers had mixed performance. In conclusion, the OR-Tools solver with the CP model appears to be the better choice in terms of time efficiency for most instances, especially for smaller graphs. However, further experimentation with larger graphs and additional benchmark instances is needed to confirm the relative performance of the solvers for more complex problem instances.

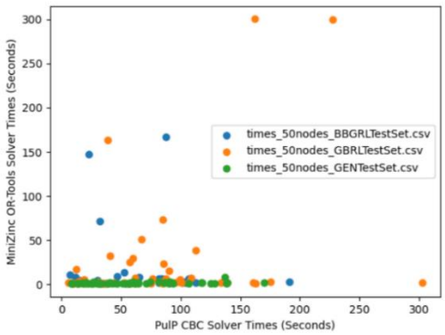

Figure 1: Performance comparison for each solver in seconds.