Testing analysis

Andrei Ionita

December 4, 2023

1 Introduction and methodology

This document provides the results and observations that I came across after using automated testing to test a minizinc model that uses dpath to find the shortest road between two given nodes, with additional mandatory vertex constraints.

I implemented a python script to randomly generate numbers that would be used to make up a .dzn file, that would be used as the input to the minizinc model. Afterwards, I created a bash script that would continuously run the python script and then run the minizinc model on the provided, generated input. The limits I used for the bash script were the following:

1. A single run of the model has an upper limit of 30 sec before it is terminated.
2. The bash script repeats the process until 50 successful tests are generated.

The output of the minizinc model was then appended to a different .txt file together with all the other results. That file is what I used to analyse the model. Of course, the statistics are not based on only 50 successful runs of the model, that was just the number that I chose to represent a single batch of tests.

2 Types of input and their respective statistics

After a couple of tries using graphs with no more than 100 nodes and/or edges, and seeing that none of the inputs required 30 seconds of runtime, I decided to separate the testing and analysis into two parts:

1. Graphs with less than or equal to 100 nodes and edges(from this point onwards called small graphs).
2. Graphs with more than 100 nodes/edges, and less than 1000 nodes and edges(from this point onwards called large graphs).

The following sections will demonstrate my findings for both of these categories of inputs.

2.1 Small Graphs

Small note to begin with: The runtime for **all** of the tests was 0 ms.

Given this observation, the only conclusions I can draw are unrelated to time. An interesting and, at the same time, obvious finding is the fact that using `bounded_path` instead of `bounded_dpath` had a greater rate of success ('success' meaning that the model found a road that meets the constraints): 40% of the generated inputs had solutions for undirected graphs, while only around 25% of generated inputs had solutions using the dpath algorithm. (directed graphs) Given the nature of directed and undirected graphs, this result was somewhat to be expected, since there overall less roads between any two nodes in a directed graph than in an undirected one with the same edges.

However, the disadvantage of using `bounded_path` was the runtime itself. Compared to the 0 ms, instant runtime in the case of directed graphs, each run of the model took on average 7 seconds when using undirected graphs, most likely due to the sheer amount of possible roads. Therefore, automated testing was easier from the point of view of time spent in the case of the final algorithm - `bounded_dpath`.

Below, I plotted 4 batches of tests, so 200 successful model runs. More specifically, the following figure shows the types of graphs generated by the python script. Even given the fact that the generation is entirely random, I thought the results can be interesting.

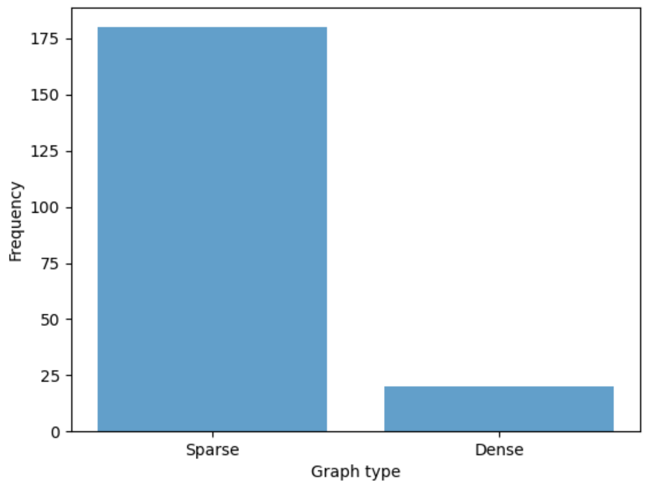

Figure 1: Figure showing distribution of graph types generated with a solution found by the model

To be precise, out of 200 random successful tests, 180 of them represent a sparse graph(less than half of possible edges), while only 20 of them represent dense graphs. While one would expect dense graphs to have a higher runtime, due to the low nature of the number of nodes and edges, as mentioned previously, the runtime was still always 0 ms.

2.2 Large Graphs

After playing around with small values (under 100) of edges and nodes, and seeing that the model had no problem finding a solution with respect to the constraints almost instantly, I decided to try testing the limits of the model's speed through higher values.