Towards Efficient Algorithms for Constraint Satisfaction Problems

Huu-Phuc Vo
Towards Efficient Algorithms for Constraint Optimisation Problems

Huu-Phuc Vo
Dept. of Information Technology
Uppsala University, Uppsala, Sweden
Email: huu-phuc.vo@it.uu.se

Abstract—Constraint programming is pervasive and widely used to solve real-time problems which input data could be scaled up to the huge sizes, and the results are required to be given efficiently and dynamically. Many technologies such as constraint programming (CP), hybrid technologies, mixed integer programming (MIP), constraint-based local search (CBLS), boolean satisfiability (SAT) could have different solvers and backends to solve the real-time problems. Streaming videos problem is the problem that requires to decide which videos to put in which cache servers in order to minimise the waiting time for all requests with a description of cache servers, network endpoints and videos are given. In this paper, we model the streaming videos problem in two different ways. The first model is implemented using heuristics, and the global constraints will be used in the second model. The aim of the paper is to benchmark those technologies to evaluate the execution time and final scores of the two models using large instances of input data from Google Hash Code.

Index Terms—optimisation, constraint programming, modelling

I. INTRODUCTION

Nowadays, watching videos online is pervasive, especially watching videos from Youtube. When streaming videos from Youtube to a huge amount of people, who could be in the same city or from different continents, minimising the waiting time for all requests from clients are critical. In the context of the Streaming videos problem, the videoserving infrastructure includes remote data centers locating in thousands of kilometers away, cache servers which store copies of popular videos, and endpoints which each of them represents a group of users connecting to the Internet in the same geographical area. The expected solution is to decide which videos to put in which cache servers. The specification of the problem could be found in detailed in [1], and the data could be found at [2]. MiniZinc [3] is a constraint-based modelling language for satisfaction and optimisation problems such as Streaming videos problem with independent solving technologies which supports for diverse technologies’ solvers for instances constraint programming (CP), CBLS, MIP, and SAT. In this paper, the bin-packing approach, which is modelled in modelling language MiniZinc, will be used to solve the Streaming videos problem in two different ways: use the built-in global constraint bin_packing_load, and model the problem using a heuristic.

II. BACKGROUNDS

Given a description of cache servers, network endpoints, and videos, along with predicted requests for individual videos, the task is to decide which videos to put in which cache servers in order to minimise the average waiting time for all requests. In other words, the task is to maximise the average saving time for all given requests. The infrastructure of the video serving network includes the data center, cache servers, and endpoints [1]. The data center stores all videos. The sizes of videos, the maximum capacity of cache servers are in megabytes (MB). Each video can be put in 0, 1, or more cache servers. Each cache server has a maximum capacity. Every endpoint is connected to the data center, however, it may be connected to 0, 1 or more cache servers. Each endpoint is characterised by the latency of its connection to the data center, and by the latencies to each cache server that it is connected to. The predicted requests provide data on how many times a particular video is requested from a particular endpoint. The paper makes the following contributions: microbenchmarks that compare the CP, LCG, MIP, CBLS, and SAT’s bin_packing_load global constraint versus manual model.

III. MODELS

a) Manual model: In the model, two 2D-matrix arrays usedCache and vInDc are defined. The first 2D-matrix array usedCache represents decision variables of which videos will be put in which the corresponding cache. The domain value of each element of usedCache is \( \{0, 1\} \). In order to mark which videos are put in the data center because their sizes exceed the capacity of caches connecting to a corresponding endpoint, another 2D-matrix array vInDc is defined. The final score is computed in the output phase by dividing savingTime by total requests nReq, and then multiplying by 1000. The first precomputation calculates the total number of used caches in the 2D-matrix usedCache. While the second precomputation iterates over all the requests and gives the total number of all requests. Three functions are defined in this model. The first function selectedVideo checks whether the video already stored in any other caches. The second function hungryCache checks the spare capacity of a cache before storing a new video into a cache to make sure that the total capacity does not exceed the given maximum capacity of the cache. The last function emptyCache checks whether the given cache ca is empty or not. A constraint is
The bin_packing_load constraint could be used as an alternative model. The bin_packing_load constraint requires that each item with its weight be put into a bin such that the sum of the weights of items in each bin is equal to a load of that bin. In this problem, with the viewpoint of video serving network, capacity must be no greater than the given capacity of each cache server. The weights of each item correspond to sizes of videos. Each cache server is corresponding to one bin, so cache servers corresponds to the number of bins. While the videos that are not requested or exceed the capacity of cache servers will be stored in the data center.

The bin_packing_load model includes constraints that consider the caches as bins, with maximum capacity and loading capacity. The videos that are stored in the data center are implicitly captured by parameter reqVid.

### V. Conclusion and Future Work

In this project, the disadvantage of those backends is the division computation such as / and div, which can be avoided by putting the division computation in the output phase. The real question here is how can the MiniZinc model be improved to instantiate and give the result for the biggest data instance, kittens, whose size is up to 5.4 MB in text format. The Streaming video problem could be modelled by other modelling language and benchmarked with the same data instances to compare the performance and the efficiency with MiniZinc model.

### References