EasyChair Preprint

№ 1732

Towards Efficient Algorithms for Constraint Satisfaction Problems

Huu-Phuc Vo

October 21, 2019

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*—optimsation, 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 video-serving 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 at [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 [4], 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

Table I: Results for our Streaming Videos model. (*): *MiniZinc* 2.1.7, ($^\psi$): *MiniZinc* 2.2.1

Technology	CP		LCG		MIP		CBLS		SAT	
Solver	Gecode		Chuffed		Gurobi		OscaR.cbls		Lingeling	
Backend	Gecode		Chuffed		Gurobi		fzn-oscar-cbls		Picat-sat	
instance	score	time	score	time	score	time	score	time	score	time
warm_up$^\psi$	562.5	0.457	562.5	0.424	562.5	0.892	562.5	t/o	562.5	1.154
warm_up*	562.5	0.286	562.5	0.183	562.5	0.615	562.5	t/o	562.5	0.260
me_at_the_zoo$^\psi$	–	t/o	–	t/o	607.33	56.217	–	t/o	–	t/o
trending_today* $^\psi$	–	t/o	–	t/o	–	t/o	–	t/o	–	t/o
video_worth_spreading* $^\psi$	–	t/o	–	t/o	–	t/o	–	t/o	–	t/o
kittens* $^\psi$	–	t/o	–	t/o	–	t/o	–	t/o	–	t/o

used to guarantee that the total sizes of all stored videos in a cache do not exceed its maximum given capacity. In addition, another constraint computes the total number saving the time of all caches and requests. With all the empty cache, the unrequested videos will be stored in the cache. In the model, we add another constraint to iterate over the endpoint, cache, and video to allocate unrequested videos that are stored in the cache under the following conditions: (1) there is connection between endpoint and cache, (2) the considered video could be possible to store in the cache, (3) the considered cache is empty, and (4) the cache does not exceed its limit when storing the video. To avoid the duplication of stored videos, another constraint is defined. To all caches connecting to the endpoint, the `at_most` constraint restricts that each requested video could be stored only in one of those connected caches. The 2D-matrix `usedCache`, which represents the final result in the streaming videos problem, does *not* introduce the symmetries. Since each cache has different latency, swapping the cache rows in the `usedCache` might produce a non-optimal result. Similarly, swapping any number of columns which is corresponding to the stored videos in the `usedCache` solution might lead to a non-optimal result also.

b) Global constraint model: 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`.

IV. Experiments

The two models could be found at [1]. We have chosen the backends for Gecode, Chuffed, Gurobi, OscaR.cbls, and Lingeling. Table I gives the results for various instances on the Streaming Videos model. The time-out was 600000 milliseconds. The experiment is done using two different versions of *MiniZinc*, 2.1.7 and 2.2.1 as it is recently released. In the first experiment, all the instances are conducted using *MiniZinc* 2.1.7. The test results produced by *MiniZinc* 2.1.7, and *MiniZinc* 2.2.1 are marked by (*) and ($^\psi$), respectively. In order to run the test in all backends, the final score computation is done at the output phase to avoid the division computations such as $/$ and div which are not executable in *Chuffed* and *Gecode*. The models are tested using all five instances, with both *MiniZinc* 2.1.7 and *MiniZinc* 2.2.1. All the test results are shown in I. To the instance *me_at_the_zoo*, the backend `Gurobi` is the best one among the others since it could give the final score after 56.217 seconds while other backends timed-out. When testing with much bigger instances such as *trending_today*, and *video_worth_spreading*, all backends couldn't produce the final results after 600000 milliseconds. The instance *kittens* is the biggest and toughest instance that defeats all the backends.

[1] https://github.com/PhucVH888/streamingVideos

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

[1] Google. Streaming videos, 2017. Available from https://hashcode.withgoogle.com/2017/tasks/hashcode2017_qualification_task.pdf.

[2] Google. Streaming videos data, 2017. Available from https://hashcode.withgoogle.com/2017/tasks/qualification_round_2017.in.zip.

[3] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[4] Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. The MIT Press, 2005.