version 4

Modelling for Combinatorial Optimisation (Course 1DL451 = Part 1 of Course 1DL441) Uppsala University – Autumn 2018 Project Report

Huu-Phuc Vo

27th November 2018

Paper2 = Goal_1DL451_work_at_weekend

A Alternative Model

As the discussion in the presentation session on 20 September and the meeting on 17 October 2018, the `bin_packing_load` constraint could be used as an alternative model. The *`bin_packing_load(array[int] of var int: load, array[int] of var int: bin, array[int] of int: w)`* constraint requires that each item `i` with weight `w[i]` be put into bin `bin[i]` such that the sum of the weights of items in each bin `b` is equal to `load[b]`. In this problem, with the view point of video serving network, capacity `load[i]` must be no greater than given capacity `X` of each cache server. The weights of each item, `w[i]`, corresponds to the videos size `reqVid[i]`. Each *cache server* is corresponding to one *bin*, so `C` cache servers corresponds to `C` bins. While the videos that are not requested or exceed the capacity of cache servers will be stored in the data center. In the section *Task* of the problem description at page 1 requires to decide which videos to put in which cache server, and at page 4 of *File format* section saying that "it is not necessary to describe all cache servers: ...", so, it's not necessary to consider data center as a cache server in the scrope of the *Streaming Videos* problem.

new

2

but this unquoted sentence implies another conclusion than yours

A.1 Description

The *bin_packing_load* model includes constraints that consider the *caches* as *bins*, with maximum capacity and loading capacity. The *bin_packing_load MiniZinc* model could be found in Listing 1. The videos that are stored in data center are implicitly captured by parameter `reqVid`.

Listing 1: A *bin_packing_load MiniZinc* model for the Streaming Videos problem

```
1  include "globals.mzn";
2
3  int: V;% the number of videos
4  int: E;% the number of endpoints
5  int: R;% the number of request descriptions
6  int: C;% the number of cache servers
7  int: X;% the capacity of each cache server in MB
8
9  set of int: VID      = 1..V; % index range for videos
10 set of int: ENDPOINT = 1..E; % index range for endpointS
```

```
11 set of int: CACHE     = 1..C; % index range for cache servers
12 set of int: CACHE0    = 0..C; % index range for load of bin
13 set of int: CAP0      = 0..X; % index range for capacity of cache
   server
14 set of int: SIZE0     = 0..1000; % index range for videos

15
16 % videoSize[v] = the size of video v
17 array[VID] of SIZE0: videoSize;
18
19 % Ld: data center latency, K: number of connected caches
20 enum LDK = {Ld, K};
21
22 % endpoint[e,{Ld, K}] = the latency Ld, number of connected caches
   of endpoint e
23 array[ENDPOINT, LDK] of 0..4000: endpoint;
24
25 % Rv: requested video, Re: coming from endpoint
26 % Rn: number of requests
27 enum RVEN = {Rv, Re, Rn};
28
29 % request[r, {Rv,Re,Rn}] = requested video Rv, coming from endpoint
   Re,
30 %    number of requests Rn of request r
31 array[RREQ, RVEN] of int: request;
32
33 % number of distinct requests
34 int: realReq;
35
36 % index range for real requests
37 set of int: RREQ = 1..realReq;
38
39 % eConCache[e, c] = latency of endpoint e and cache c
40 array[ENDPOINT, CACHE] of 0..1000: eConCache;
41
42 % total number of requests
43 int: nReq = sum(re in RREQ)(request[ re, Rn ]);
44
45 % valid requested videos
46 int: nReqVid;
47
48 % reqVid[v] = size of valid requested videos
49 array[VID] of SIZE0: reqVid;
50
51 % videoInCap[v] = size of videos that less than or equal to max
   cache capacity
52 array[VID] of SIZE0: videoInCap;
53
54 % bin[v] = cache i-th that stores video v
55 array[VID] of var CACHE: bin;
```

not used anywhere?

CACHE0 = 0..C; % index range for load of bin → ?

CAP0 = 0..X; % index range for capacity of caches → value

SIZE0 = 0..1000; % index range for videos → sizes of

better for seq'l readability

max(4000, 1000), to be precise

same concept?

500? → what convention if e is not connected to c?

your datafiles use 0 instead of the latency of the connection of e to the data center, but this leads to errors: Ex

do we always have nReqVid ≤ realReq ≤ R?

number of

so are they all valid?

1..nReqVid?

requested & cachable?

RREQ?

so are they all cachable?

cachable?

there seems to be an unstated assumption that non-cachable & non-requested videos have size 0? same for videos only requested from endpoints not connected to caches?

+ what constrains videos to be cached in actually connected caches? → nothing? see Ex3

```
56
57 % load[c] = load of cache c-th
58 array[CACHE] of var CAP0: load;
59
60 % total saving time
61 var int: savingTime = sum(req in RREQ where request[req, Rn] > 0)(
62   let { int: rv = request[req, Rv];
63         int: re = request[req, Re];
64         int: rn = request[req, Rn];
65         int: ld = endpoint[re, Ld];
66         var int: lc = eConCache[re, bin[request[req, Rv]]];
67     } in (
68        (ld - lc)
69        *
70        (bool2int(reqVid[rv] > 0))
71        *
72        rn
73 ));
74
75 % find out which videos to assign to which bins
76 constraint bin_packing_load(load, bin, reqVid);
77
78 solve :: int_search(
79         bin ++
80         load
81         ,
82         first_fail, indomain_min, complete) maximize savingTime;
83
84 output [show((savingTime/nReq)*1000) ++ "\n" ++
85         show(sum(vi in VID)(bool2int(load[vi] > 0)))] ++
86        [ if vi = 1 then "\n\(b) " else "" endif ++
87          if (fix(reqVid[vi]) > 0 /\ fix(bin[vi]) = b)
88          then "\(vi) "
89          else "" endif
90          | b in CACHE, vi in VID];
91
92 % for test server
93 var int: nReq; nReq = sum(re in RREQ)(request[ re, Rn ]);
94
95 % output ["\nbin = " ++ show(bin) ++ "\nload = " ++ show(load)];
```

was videoInCap in v3: probably the source for its discrepancy in objective value with v1

very procedural explanation

c in CACHE?!

how could it not be fixed: it's a parameter, no?

compilation error

what purpose upon lines 42-43?

Search Annotations. In the alternative *bin_packing_load* model for Streaming Videos problem, the `::int_search` annotation is used to compute the final *score* with array of variables, which concatenate the `bin` array and `load` array. The next argument `first_fail` specifies that the variables are chosen in the order that appear. To those chosen variables, the assignment annotation `indomain_min` will assign the smallest video size in the `bin` and `load` domain. Ultimately, the strategy annotation `complete` is specified.

value ?

A.2 Implementation

In this model, the load of each cache is computed and guaranteed that its load can not exceed the cache capacity. Ultimately, search strategy is included to find the maximal score. An array of decision variables `array[CACHE] of var CAP0: load;` is declared to capture the capacity of the cache servers.

The variables `bin`'s index is ranged over the `VID` to indicate that which videos are stored in which bin. Similarly, the variable `load`'s index is also ranged over `CACHE` to capture the capacity of all cache. The parameters `reqVid` and `videoInCap` can be used alternatively without affecting the final results, but fewer variables usually means faster solving. Since the `videoInCap` captures all the videos that is no greater than the given capacity of cache servers. The unrequested videos are also stored in the `videoInCap` in this case. While the `reqVid` is stricter since it only stores the requested videos that are not greater than the given capacity of cache servers. Those parameters can be alternatively used because just the valid requested videos are used to compute the final scores. In other word, unrequested videos in `videoInCap` doesn't affect to final scores.

The prepocessing has been done in the following ways. First, the duplicated requested videos are aggregated and corresponding real requests are captured by parameter `realReq`. Second, the valid requested videos are preprocessed and stored in array `videoInCap`.

A.3 Bin Packing Model Evaluation

We have chosen the backends for Gecode, Chuffed, Gurobi, OscaR.cbls, and Lingeling. Table 2 gives the results for various instances at Table 1 on the Streaming Videos model[1, 2]. The time-out was 900000 milliseconds. To the instance *me_at_the_zoo* and `warm_up`, the backend `fzn-oscar-cbls`, *picat-sat*, *Gurobi*, *Chuffed*, and *Gecode* give the results in details at table 2. When testing with much bigger instances such as *trending_today*, and *video_worth_spreading*, all backends couldn't produce the final results after 900000 milliseconds. The instance *kittens* is the biggest and toughest instance that defeats all the backends, and ends up with the time-out.

Name	Videos	Cache Servers	Endpoints	Requests
warm_up	5	3	2	4
me_at_the_zoo	100	10	10	81
video_worth_spreading	10000	100	100	40317
trending_today	10000	100	100	95180
kittens	10000	500	1000	197987

Table 1: Instances of Streaming Videos model.

this stems from the error in line 40 and/or the unstated assumption of the ~~hidden~~ comment on top of page 3

see v3: no explanation is provided here in v4

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	562500	618	562500	443	562500	677	562500	2428	562500	734
me_at_the_zoo	607330	595	607330	414	607330	518	607330	2356	607330	736
video_worth_spreading	–	t/o	–	t/o	–	t/o	–	t/o	–	t/o
trending_today	–	t/o	–	t/o	–	t/o	–	t/o	–	t/o
kittens	–	t/o	–	t/o	–	t/o	–	t/o	–	t/o

Table 2: Results for the alternative *bin packing* model with *MiniZinc* 2.2.1

3?

Upon examining the produced solution, it turns out that some videos are cached at caches not accessible to the endpoints where their requests originate from: the results can be super-optimal.

these runtimes are basically as good (and sometimes ≠ better!?) than on the much tinier warm_up: this is suspicious

B Conclusion

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.

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`.