by Patrick
December 11, 2015

Any Scale Task Allocation (part 1)

1 Introduction

This is a problem presented to us by David and Jeremy. I think it goes as follows. We have a set of tasks that need to be performed. These tasks can be performed via a number of task variants. For example, we might have a task *sort* and a number of ways (task variants) that can do this, such as *bSort* or *mSort*. A task variant has a cost associated with it, and we might think of this as a load on a processor. We also have a number of processing elements, each with a given capacity, and this is a constraint on the number and make up of task variants allocated to that processor. Also, there is a restricted set of processors that a task variant can be allocated to. Processors are interconnected and each connection has a bandwidth, i.e. a capacity limit on inter-process communication. Finally, pairs of task variants need to communicate and in doing so require a specific amount of bandwidth. Therefore the first problem is to find an allocation of task variants to processors such that processor capacity and bandwidth is respected.

It is assumed that a task variant that has a high load on a processor produces more value than an equivalent task variant (i.e. one that can do the same task) with a lower load. Therefore, we want to produce the best quality service possible and we do this by maximising the sum of the loads on the processors.

2 A first model

Ciaran has already produced a model in miniZinc. I want to produce a model in choco3. Why? Because this will allow us more control over variable and value ordering heuristics, control over the use of specialised constraints and being able to limit search effort. Of course, not being a miniZincer, some of these capabilities might already be in miniZinc, so my only real justification is that the problem is interesting and I want to have some fun. So, here is a first stab at a naive model.

Decision variables For each task we have a set of task variants that can perform that task. Only one of these task variants will be assigned to an actual processor. Consequently task variants are constrained integer variables and their domains are processors. Assume that we have processors 1 to n, and that we have an "imaginary" processor 0. Therefore if we have task variants $tv_{1,1}$, $tv_{1,2}$ and $tv_{1,3}$ for a given task $task_1$, two of these will be assigned the value 0 (and will be allocated to our imaginary processor) and one of the task variants will be assigned to an actual processor. Therefore for $task_i$ with n variants the occurrence of 0 assigned to variants $tv_{i,1}, ..., tv_{i,n}$ is equal to $n - 1$.

Task variant to processor allocation A processor can be considered as a bin with a capacity. Given a vector of loads, where $load_i$ is the load demanded by task variant $tv_{i,j}$, we can view the allocation of task variants to processors as a bin packing problem. Fortunately, in choco3 there is a bin packing constraint. This is an implementation of Paul Shaw's "A Constraint for Bin Packing" [1]. I used it recently for a workforce allocation problem (2 weeks ago) and it worked really well.

Respecting bandwidth A link between two processors is again considered as a bin. Therefore we have a constrained integer variable $link_{xy}$ with domian $\{0..c\}$ for the link between processors x and y with bandwidth capacity c. Also we have constrained integer variables to represent pairs of task variant, call

these $ptv_{i,j}$. Therefore, when two task variants $tv_i = x$ and $tv_j = y$ (allocated to processors x and y) we constrain ptv_{ij} to be the pair (x, y) and add the communication demanded between these task variants to the bin $link_{xy}$. Again, this can be viewed as bin packing problem and we use Paul's constraint. Note that when two processors are not connected we generate a link with zero capacity. Also, we have a magic link of unbounded capacity. This link acts between the zeroth processor and all other processors and between each processor and itself.

3 Problem and solution

Below is a the first problem instance given to me by Ciaran. It has been reformatted so that I can read it in in java.

```
nTasks 5
nVariants 10
nProcessors 3
tasksToVariants
1 2
3 4
5 6
7 8
9 10
utilisations
1 2 1 2 1 2 1 2 1 2
capacities
3 3 3
links
0 2 8
2 0 0
8 0 0
bandwidths
0 1 0 0 2
1 0 1 1 0
0 1 0 1 1
0 1 1 0 3
2 0 1 3 0
permittedProcessors
1 2 3
1 2
1 2 3
1 2 3
1 2 3
1
1 2 3
1 2 3
1 2 3
1 2 3
sameProcessor
1 2
3 4
```

Below is a solution to the problem. I have sketched it out. On the left we have 4 processors, with processor zero being magic: unbounded capacity and linked to all other processors on an unbounded link. Within each processor box we have the task variants (I just called them V_i) allocated. On the right we have a small graph showing communication between tasks. Each vertex represents a task and the contents are the task variants. Red edges between red vertices signify tasks that must reside on the same processor. The constraint program delivers this solution quickly, a second or so.

4 Outroduction

What's to follow? Just now, my model has solved one small instance. This is one step, and it was bigger than I expected. Next wee steps are to use variable ordering over the decision variables, because at present all variable are decision variables. Then, I need to optimise, by maximising the sum of weights on processors.

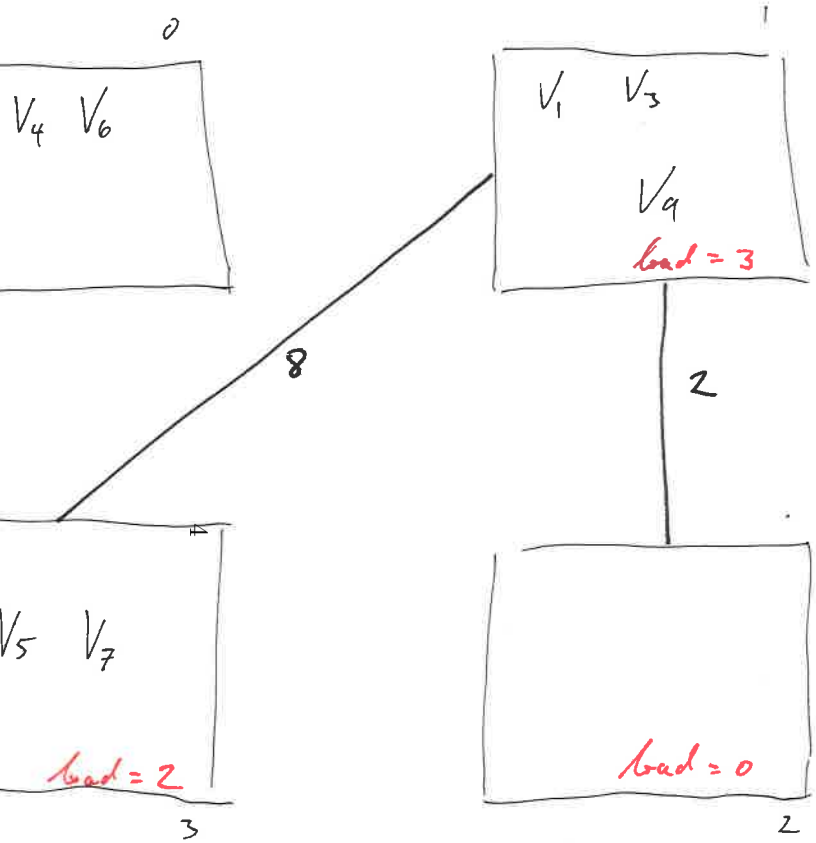
There may be symmetries, and we should probably hope a problem generator avoided these. In particular, if we have two task variants for a task and they are identical, we should delete one of these.

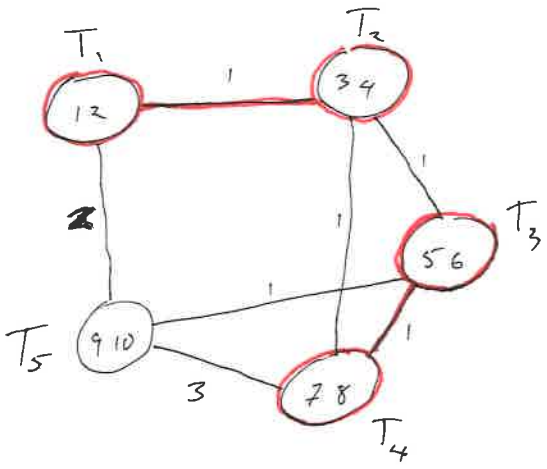
We can reduce constraints. In the case we have two tasks that must be allocated to the same processor, we might generate new composite tasks variants that can go on the intersection of the allowed processors and have connectivity that is the union of connectivity for both. This might simplify the model.

And finally, code and data and this note are in the following directory `http://www.dcs.gla.ac.uk/~pat/jchoco/anyScaleTaskAllocation/`

References

[1] P. Shaw. A constraint for bin packing. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, pages 648–662, 2004.





send processes