Decision Making - ex 4

Filippo Brajucha, Youssef Hanna

November 2023

1 RCPSP

1.1 Table

	default		**smallest**	
	Makespan	**Time**	**Makespan**	**Time**
rcpspData1.dzn	90	265ms	90	259ms
rcpspData2.dzn	53	857ms	- (54)	-
rcpspData3.dzn	- (82)	-	- (75)	-

1.2 Observations

What do you observe? Is searching on the smallest (earliest) start times is always a good idea?

Running the **RCPSP** model, we can observe that it gives good results for the default solver (it succeeds in compiling both the first and second datasets), while the resolution with sorting by earliest start time is definitely worse, since that it times out for 2 datasets out of 3.
Examining the data, we can observe that the file *rcpspData1.dzn* has the same execution time for both solver, so we can think that it's already ordered, and the solution doesn't need difficult operations such as postponing.
The main difference between the two heuristics it's that the **smallest** one tries to find the solution following the earliest start times of the activities, it can be a good solution if the ordered activities fit good, but it can't be an optimal solution if the activities have to be postponed. If the solver needs to postpone an activity ordered with starting time have to go back by all the collection of activities and have to restart.
This is the reason why the default method is generally better in the *rcpspData2.dzn*. We can't assume anything for the *rcpspData3.dzn* because of timeouts for both strategies.

2 JSP

2.1 Table

	default		smallest	
	Makespan	**Time**	**Makespan**	**Time**
jobshop1.dzn	663	80msec	- (669)	-
jobshop2.dzn	- (826)	29s 501msec	- (921)	-

2.2 Observations

What do you observe? Is searching on the smallest (earliest) start times is always a good idea?

In the **Job Shop Problem** we have a set of jobs that need to be processed by a set of machines, and the goal is to minimize the completion time. A Job is a sequence of activities, and each activity in a job needs a distinct machine. And there are as many activities as there are machines Resources can run one activity at a time and cannot do Overlap (we don't care about capacity anymore). So two activities are bound by the **Disjunctive constraint**. Also present in this problem is the **Precedence constraint**, which forces one activity to end before another begins. Finally to do scheduling we want to optimize the common cost function: **Makespan** (the end of scheduling), is the completion time of the last activity, and the optimal makespan is the smallest one.

As can be seen from the table only deafult search allows us to find the optimal solution. In contrast, with smallest search not even the solution is found in the predefined time limit(5 mins).

When we need to find an optimal solution to an optimization problem, we find a solution and backtrack afterwards, but before we do this to find a better solution, we add a **bounding constraint**. So let us imagine the value of the current solution is **V**, we backtrack with an additional constraint to find a better solution in the same search tree, but with a cost less than that of the current solution. If it fails it means **V** that was the best solution, if not we have found **V1** that is better than the previous solution, and in this case backtrack is done again with **V1** until it fails. The idea is to go back to the last assignment try to assign a better value, if it fails, we go back to the still previous one and so on. So if we consider all our activities in a sequence I do backtrack. In the case of smallest heuristics we choose the variable with the least value in the domain (the job that the one that ends first). The **PRB algorithm** will always force the backtracking to backtrack to the root. First of all this behavior will not initiate much propagation, because going back to the root, the other variables do not do propagation. Then if every if every time I go to the root the time taken to backtrack and create new branches all the time will be very high.

Coupled with the heuristics of EST if the **postponed scheduler** is used you better the final solution. Where when you backtrack to the root, it freezes that variable and we wait for propagation to change the domain of that variable and put it in another part of the scheduling. So if we used just the smallest without we would not find the optimal solution, or it would have a very large cost anyway. The problem with **Minizinc** is that it does not use the postponed scheduler, and in this case the backtracking has a very hard time going from one branch to another by significantly changing the value of the variables (which is done by the default search). We can say that the **smallest search** strategy in Minizinc is not efficient and is incomplete, because it cannot prove it is the efficient solution.

Finally, we can say that the **default strategy** works better because the initial search is not strongly constrained to branches that do not lead to the optimal solution.