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	895ms	- (669)	-
jobshop2.dzn	- (881)	-	- (921)	-

2.2 Observations

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

Observing the data emerged for **JSP** model we can assume really similar things that we already discussed in RCPSP model. The **default** heuristic is the best option because of some problems linked to the postponing of the activities. The execution of *jobshop2.dzn* failed for both search heuristics, instead of *jobshop1.dzn* that complete the execution rapidly only for the **default** heuristic.
It's clear that the way it is ordered the input file is crucial for the default computation, so, looking at the **smallest** one, we can be sure that sort out the data isn't a good idea.
Looking at the results for the timed out **makespan** we can observe that the **default** heuristic is better also for the *jobshop2.dzn*. So it's clear that the **smaller** heuristic, also defined as *greedy solution*, isn't a good method for this type of exercise.

Focus on what happens when searching for the optimal solution after finding an initial solution (perhaps this is what you tried to explain in the first sentence but it was not clear, also because backtracking and restarting are two differente things). Why is the search inefficient to find the optimal solution? There are the thngs we discussed when we motivated the schedule or postpone search strategy

Main idea: 1) firdt branch schedule an activity ai with minimum ESTi, schedule it at its ESTi, 2) On backtracking, postpone ai and update ESTi, with re-scheduling of ai.

This is the setTimes strategy and it works because it use the normal starting time of the tasks (ESTi), modified only by other activities. The scheduling changes based on ESTi changes