Decision Making with Constraint Programming

Exercise 4

Dessì Leonardo [0001141270]

Spini Riccardo [0001084256]

Corso di laurea magistrale in Informatica

Alma Mater Studiorum
Università di Bologna

December, 2023

1 | Resource Constrained Project Scheduling Problem (RCPSP)

	Default search		Smallest search	
	Time	**Makespan**	**Time**	**Makespan**
data1	0.34 s	90	0.33 s	90
data2	0.76 s	53	300 s	54
data3	300 s	82	300 s	75

The table above reports the objective value and the execution time of three instances for the Resource Constrained Project Scheduling Problem.
The RCPSP problem consists of a set of cumulative resources, a set of tasks with duration and resource requirements, and precedence constraints between some tasks. We want to execute each task in a way that minimizes the makespan (the line that defines the end of the scheduling), subject to precedences and cumulative resource constraints.
The constraints of this problem are:

- process p_i must start before process p_j with a preemptive right defined by the graph in figure 1.1.
- the resources are cumulative.

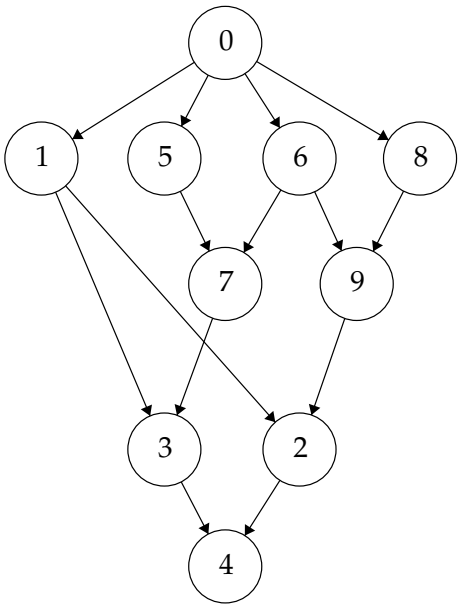

Figure 1.1: Precedence graph for *rcpspData1*

The goal of RCPSP is to minimize makespan. In our case, the makespan is defined as:

$$\max S_i + D_i$$

where i is between 1 and n_{task}.

- *smallest* is a heuristic that chooses the variables with the smallest value in their domain first. In our case, those with the smallest value are those that have the time window starting earlier and therefore the Earliest Start Time smaller.

As we can see from the results table, we can see that there are no sensitive differences between the two search methods for *data1*. This is probably caused by the small size of the instance. Therefore, we cannot say which of the two methods is better than the other. Calculating the failures we notice that *default search* has more than *smallest search*, but the latter has no perceptible improvement given the small instance.
In *data2*, on the other hand, we notice that there is a substantial difference in terms of time although the value of the objective function is very similar. Indeed, the *default search* immediately finds the optimal solution.
Regarding the results obtained in *data3*, for the same amount of time we obtained a better makespan with the smallest search.

Regarding the *smallest search*, when the solver finds a solution and wants to improve it, it has to go back to the root node completely to improve it and find a better objective value. The solver must return to the root node because with the branch-and-bound method each time a feasible solution is found, a new bounding constraint is placed that ensures that a future solution must be better than the one previously found. So if the first solution has a $makespan = n$, a $makespan < n$ constraint is added, bounding constraint propagation is carried out and most likely what happens is that the initial assumed value of the root node does not allow for a better solution than the one found. So this is the reason why backtracking goes all the way to the root node.
This process of backtracking and thus new reassignments to variables can lead to a large increase in time while still finding (if there were no time limit at 5 minutes) the smallest makespan value.
In branching by the smallest method, however, the problem is the weakness of constraint propagation in creating a new branch at the root, such as one in which $S_1 \neq v$, where v was the smallest original value in the domain. The problem arises when the domain of S_i is very large.
When reassigning values to the variables from the root node, the domain of the variables has changed very little, and so you will get a result very similar to the previous one.

Searching on EST with the smallest method can therefore take a long time because each time you go back to the root node the propagation of the constraints is very weak ($S_1 \neq v$) and you fall into suboptimal subtrees.
In conclusion, in our opinion having such discordant results we cannot say that smallest search is better than default search and vice versa, so we think it may depend on how the instances are structured.
Finally, a graphical representation was created with Python of the optimal solution of the *rcpspData1* problem. In addition, we noticed that task 0 does not consume any resources however it has its own duration and precedence constraints (Figure 1.1) that do not allow other tasks to overlap with task 0, that is why all drawn tasks start after 10

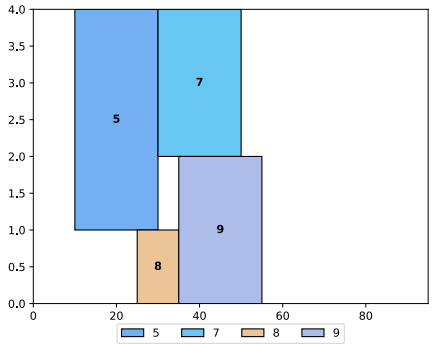

Figure 1.2: Resource 1 Cumulative chart for *rcpspData1*

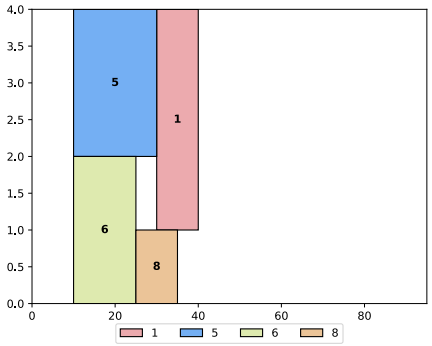

Figure 1.3: Resource 2 Cumulative chart for *rcpspData1*

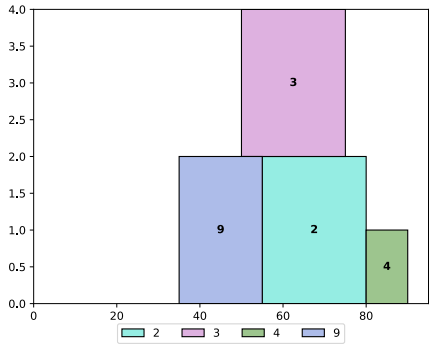

Figure 1.4: Resource 3 Cumulative chart for *rcpspData1*

2 | Job Shop Scheduling Problem (JSP)

	Default search		Smallest search	
	Time	**Makespan**	**Time**	**Makespan**
jobshop1	0.36 s	663	2.37 s	663
jobshop2	300 s	826	300 s	921

The table above reports the objective value and the execution time of two instances for the Job Shop Scheduling Problem.
In JSP problem we have a set of machines and a set of jobs, each composed of a sequence of tasks/orders, each requiring a different machine and machine requirements and machine-dependent durations of the job tasks. In this problem, unlike the RCPSP, we use a disjunctive constraint. This requires that the tasks do not overlap in time.
As in the previous exercise, our goal is to minimize the makespan.

As we can see from the table, the *default search* has better performance in terms of reaching the best objective value in the first and second instance, than the *smallest search* method. The reasons why the smallest search finds worse results are the same as before, which is because of the weakness of constraint propagation.

Finally, we implemented a graphical visualization in Python to display the result of *jobshop1* (Figure 2.1). Each color corresponds to a job and each row to a machine. Each job has a task in each machine, and the various tasks must be executed sequentially

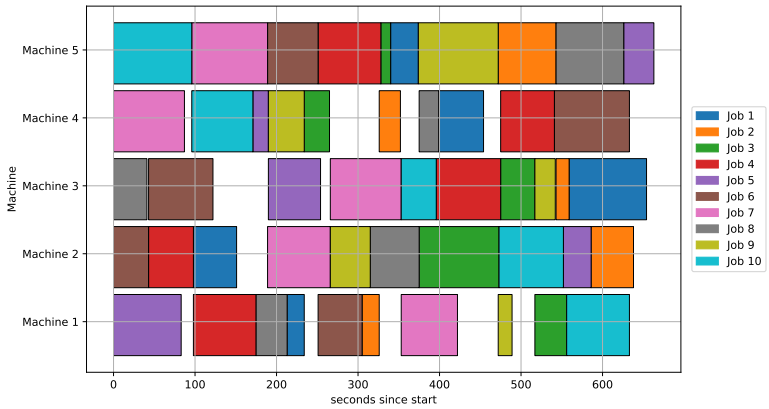

Figure 2.1: Best solution Gantt diagram of *jobshop1*

3 | Postponing

Both the *default method* and the *smallest search method* are inefficient. The postponing approach could be used to improve performance.
Therefore, it is better to use better constraint propagation by marking activity i as postponed. In this way, activity i will be scheduled only after the propagation has updated EST_i.
This is necessary because we want to explore different branching decisions. The postponing strategy is preferable because it allows us to delay some scheduling decisions, thus providing more flexibility in finding solutions. This approach allows us to explore the solution space more broadly, adapt to dynamic changes in the problem, and optimize resource allocation more fully.
To improve the results we think it would be useful to use a postponing strategy. The *smallest search* and *default search* may be problematic for finding the optimal solution due to the following reasons:

- Greedy decision-making: Prioritizing tasks with the shortest starting times may lead to a greedy approach where short-term gains are prioritized without considering the long-term consequences. This can result in not optimal scheduling decisions.

A postponing strategy, on the other hand, aims to delay certain tasks' scheduling to explore alternative sequences and resource allocations. This strategy is often beneficial for finding optimal solutions because:

- we suppose that postponing decisions allows for a more global consideration of the project schedule. It could enable the algorithm to explore different combinations of task sequences, resource allocations, and start times, potentially identifying more efficient schedules.

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

No, it is not always a good idea!
Now that we have seen that searching on the smallest EST is problematic because of bad branching decisions we have realized that this search fails to improve the default search consistently. So in general searching on the smallest EST can only be a good idea if you want to find a solution but not the optimal solution.

As mentioned above, when not only a solution but also an optimal solution has to be found and backtracking has to be done, the propagation of constraints where $S_1 \neq v$ is too weak and thus the search for small ESTs is too inefficient.