Linear-Time Filtering Algorithms for the Disjunctive Constraint

Hamed Fahimi

Claude-Guy Quimper

Université Laval

Claude-Guy.Quimper@ift.ulaval.ca

hamed.fahimi.1@ulaval.ca

July 2014

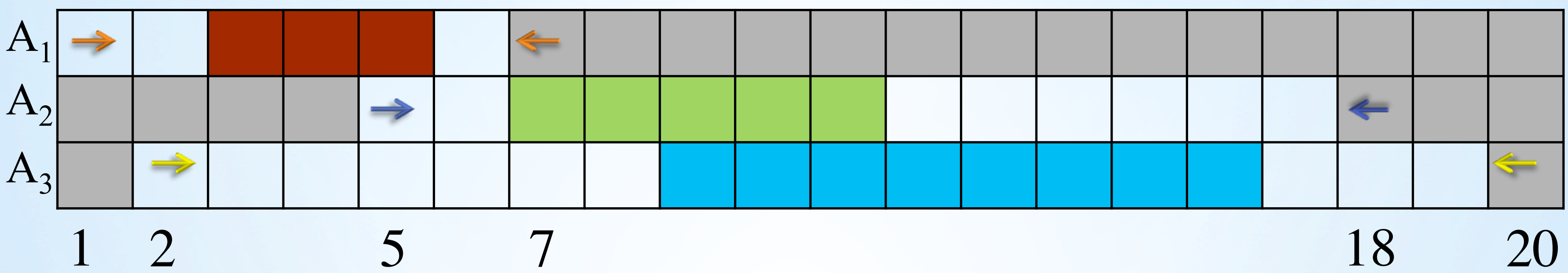
Disjunctive Constraint

Consider a set of n tasks, with known parameters:
The **release time** (r_i); The **deadline** (d_i); The **processing time** (p_i); and the unknown starting times $[s_1,\ldots,s_n]$.

Disjunctive Constraint

Consider a set of n tasks, with known parameters:

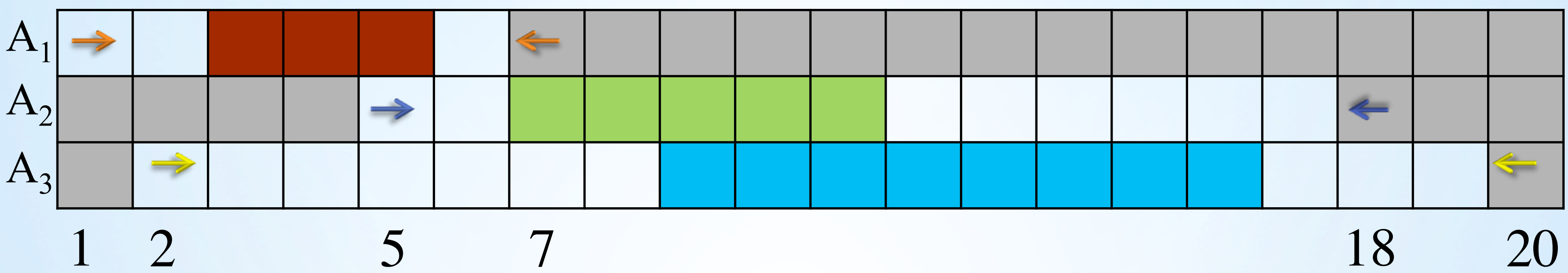
The **release time** (r_i); The **deadline** (d_i); The **processing time** (p_i); and the unknown starting times $[s_1, \ldots, s_n]$.



Disjunctive Constraint

Consider a set of n tasks, with known parameters:

The **release time** (r_i); The **deadline** (d_i); The **processing time** (p_i); and the unknown starting times $[s_1,\ldots,s_n]$.

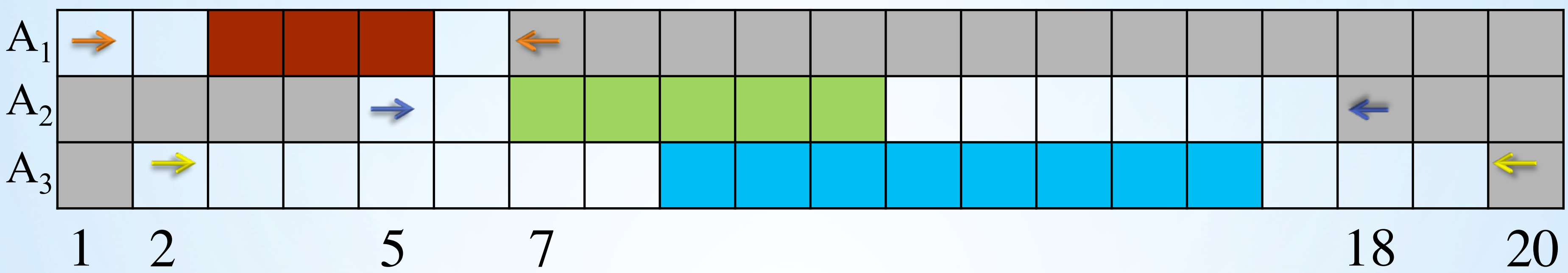


Constraint:

$$\text{DISJUNCTIVE}([s_1,\ldots,s_n]) \Leftrightarrow s_i + p_i \leq s_j \text{ or } s_j + p_j \leq s_i$$

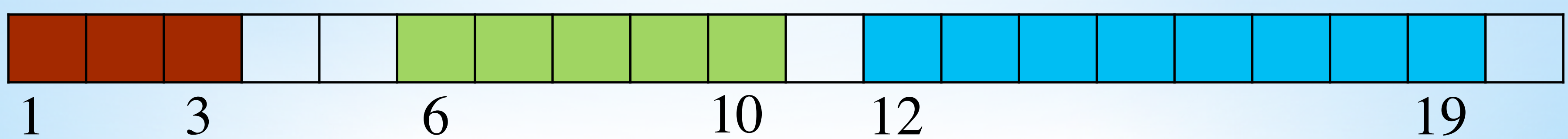
Disjunctive Constraint

Consider a set of n tasks, with known parameters:
The **release time** (r_i); The **deadline** (d_i) ; The **processing time** (p_i);
and the unknown starting times $[s_1,\ldots,s_n]$.



Constraint:

$$\text{DISJUNCTIVE}([s_1,\ldots,s_n]) \Leftrightarrow s_i + p_i \leq s_j \text{ or } s_j + p_j \leq s_i$$



- A feasible schedule!

Disjunctive Constraint

- It is NP-Complete to determine whether there exists a solution to the Disjunctive constraint.
- It is NP-Hard to filter out all values that do not lead to a solution.
- Nonetheless, there exist rules that detect in polynomial time some filtering of the domains of the tasks.
- Our goal is to improve some of these existing filtering algorithms for this constraint.

Preliminary

- We aim at designing algorithms with linear complexity.

- To achieve this goal, we assume that sorting can be done with a linear time algorithm, say *radix sort*.

Time-Tabling

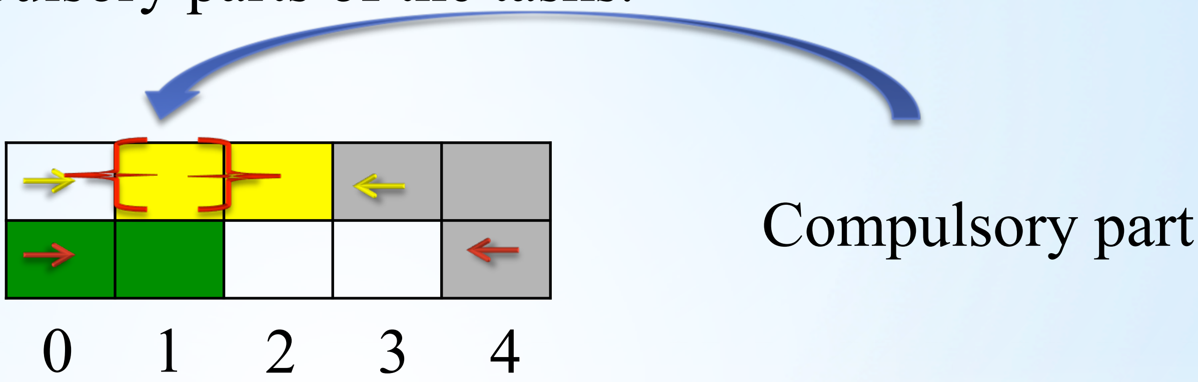
- If $lst_i < ect_i$ for a task, then the interval $[lst_i, ect_i)$ is called the *compulsory part* of *i*.

Time-Tabling

- If $lst_i < ect_i$ for a task, then the interval $[lst_i, ect_i)$ is called the *compulsory part* of *i*.
- The Time-Tabling technique filters the domains which are in conflict with the compulsory parts of the tasks.

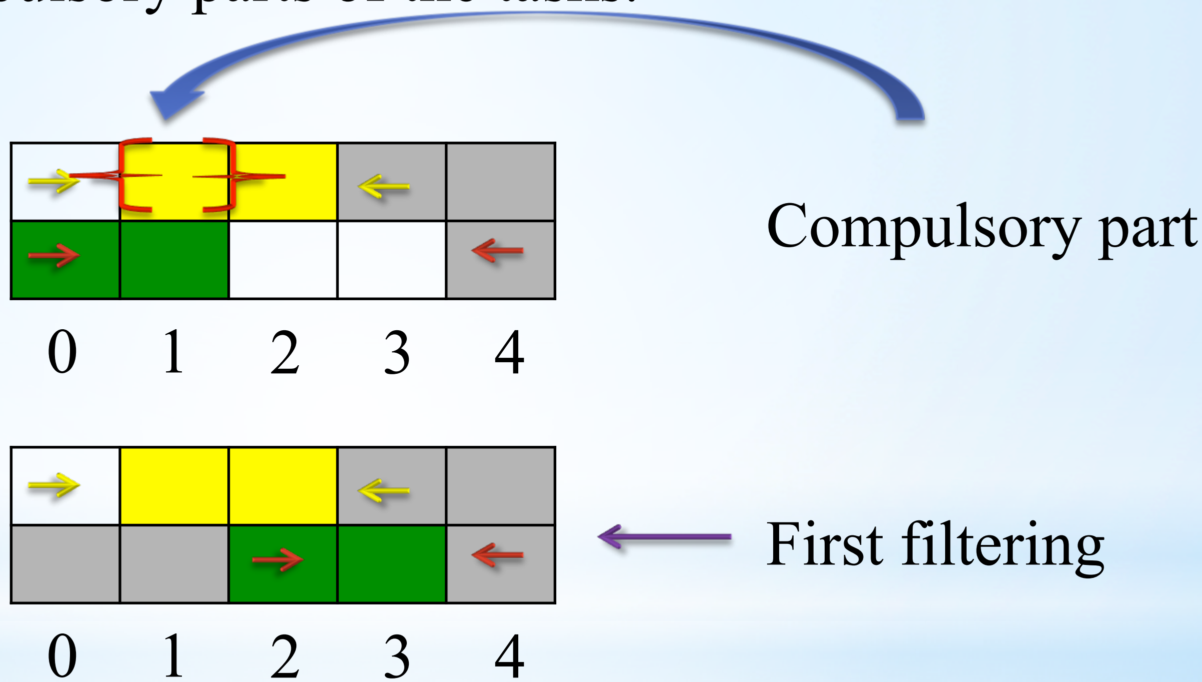
Time-Tabling

- If $lst_i < ect_i$ for a task, then the interval $[lst_i, ect_i)$ is called the *compulsory part* of *i*.
- The Time-Tabling technique filters the domains which are in conflict with the compulsory parts of the tasks.



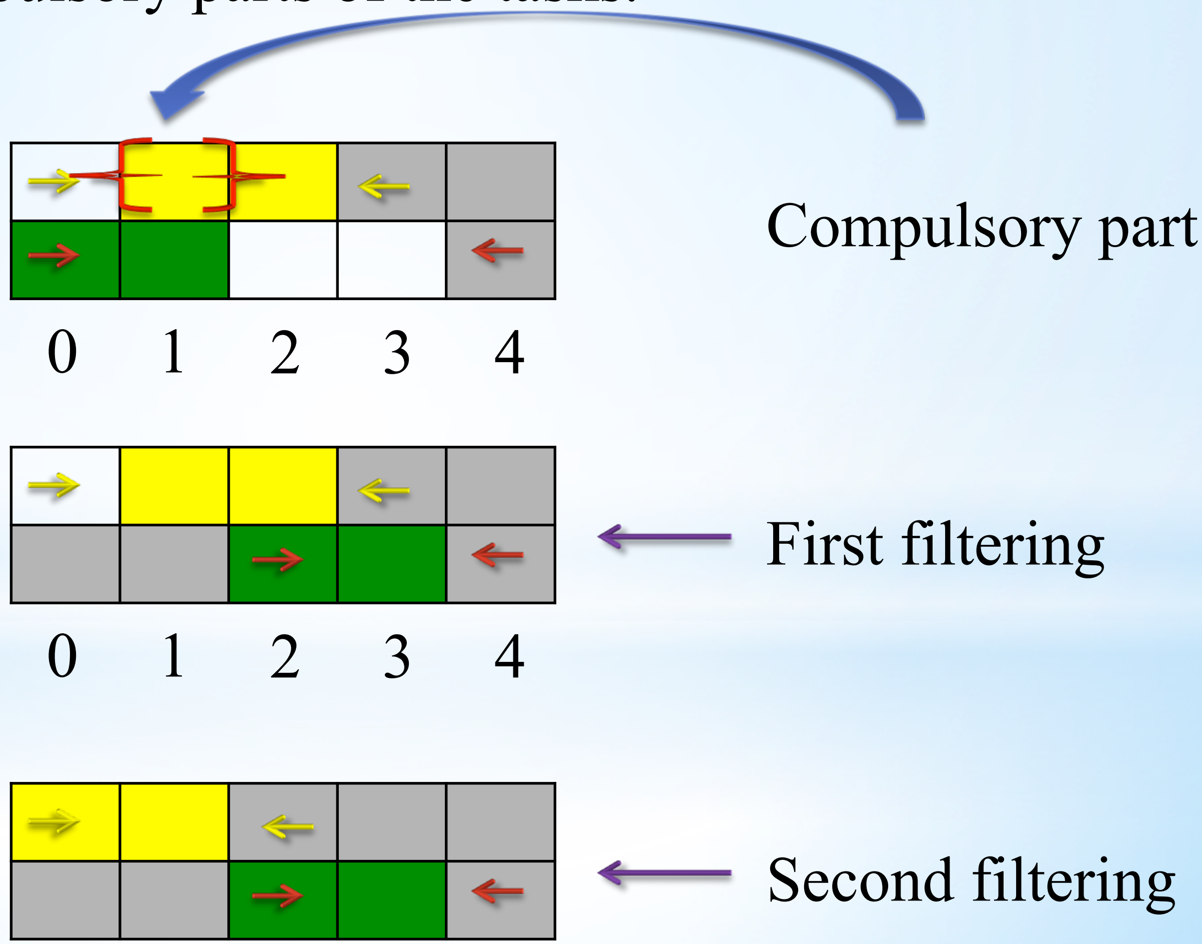
Time-Tabling

- If $lst_i < ect_i$ for a task, then the interval $[lst_i, ect_i)$ is called the *compulsory part* of *i*.
- The Time-Tabling technique filters the domains which are in conflict with the compulsory parts of the tasks.



Time-Tabling

- If $lst_i < ect_i$ for a task, then the interval $[lst_i, ect_i)$ is called the *compulsory part* of *i*.
- The Time-Tabling technique filters the domains which are in conflict with the compulsory parts of the tasks.

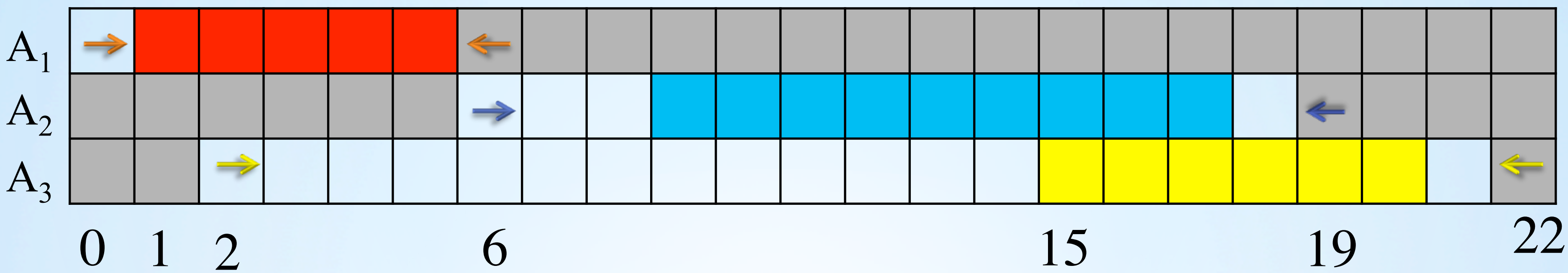


Time-Tabling

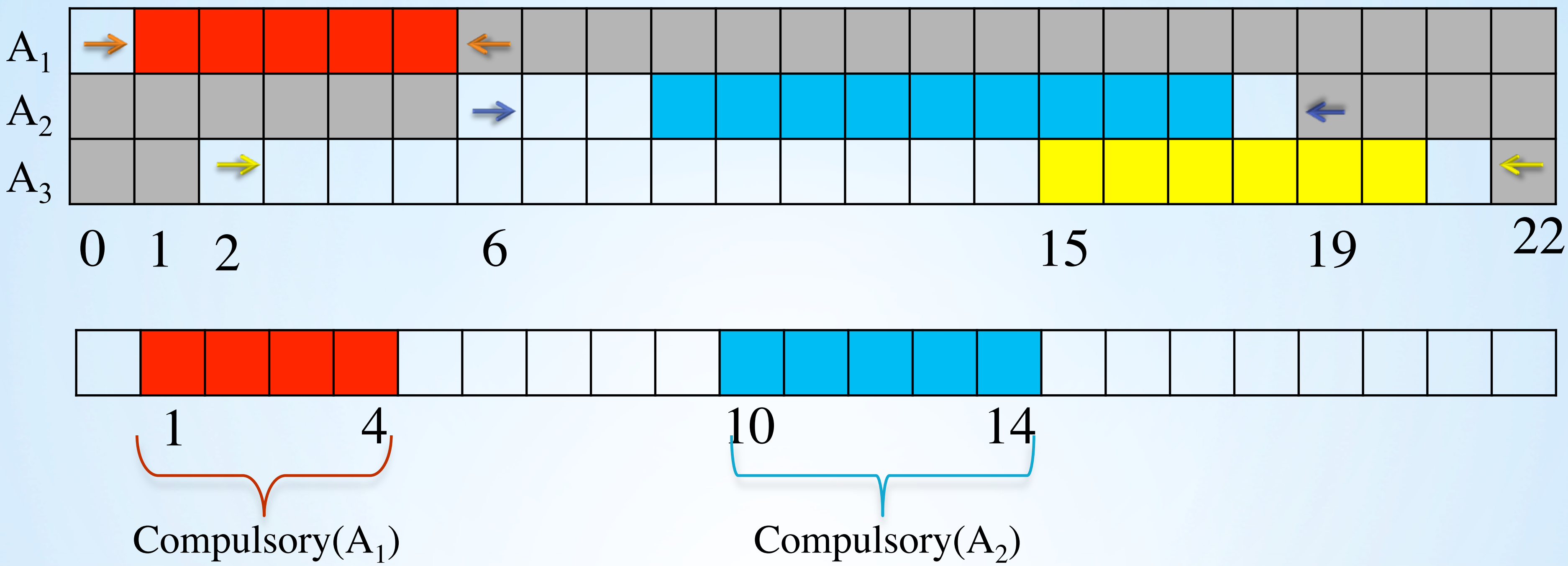
- Ouellet & Quimper presented an algorithm for Time-Tabling on a generalized case in O($n$log(n)).

- We took advantage of Union-Find to achieve an algorithm that admits a linear time implementation for the Disjunctive case.

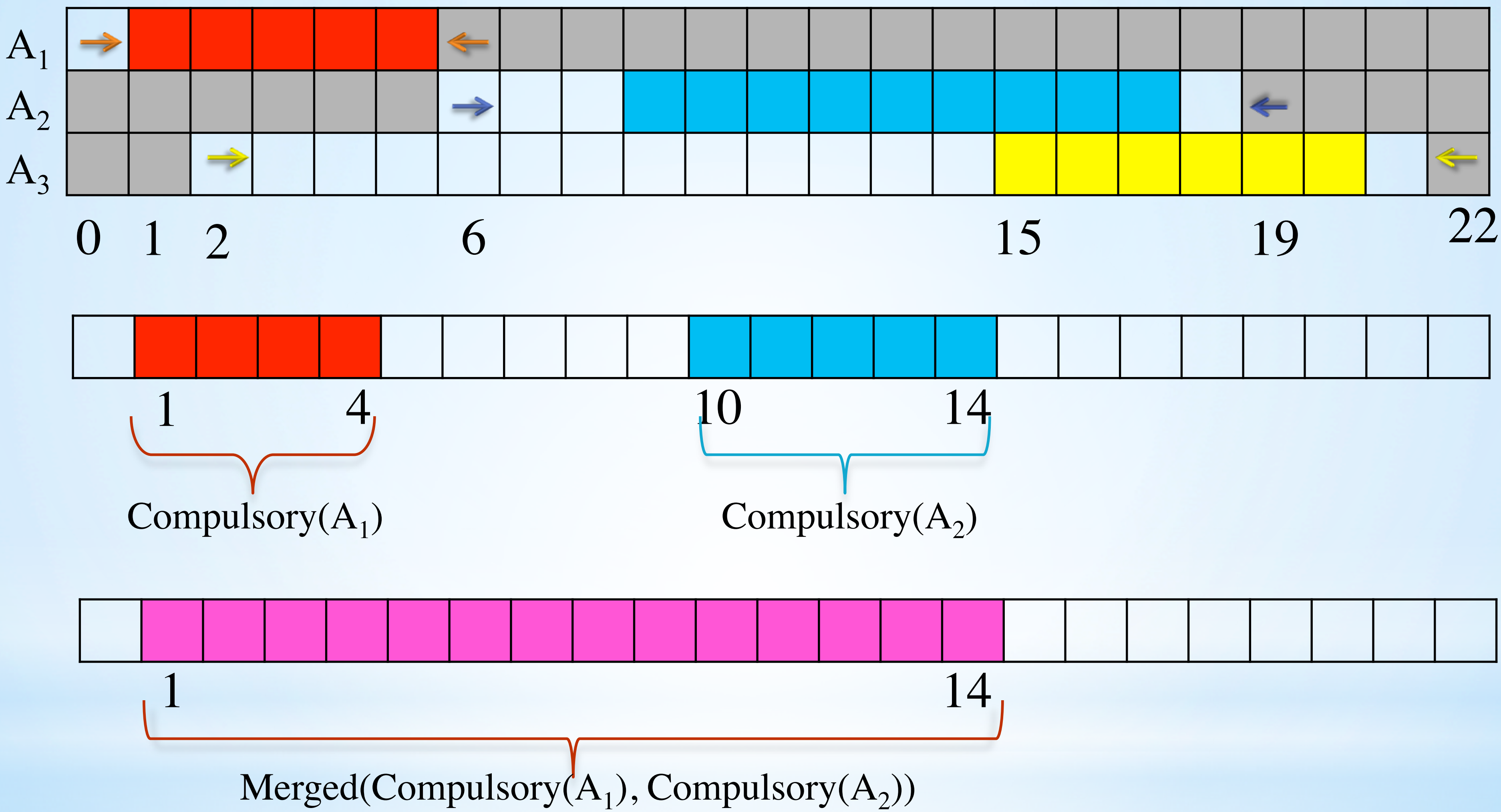
The strategy of our algorithm



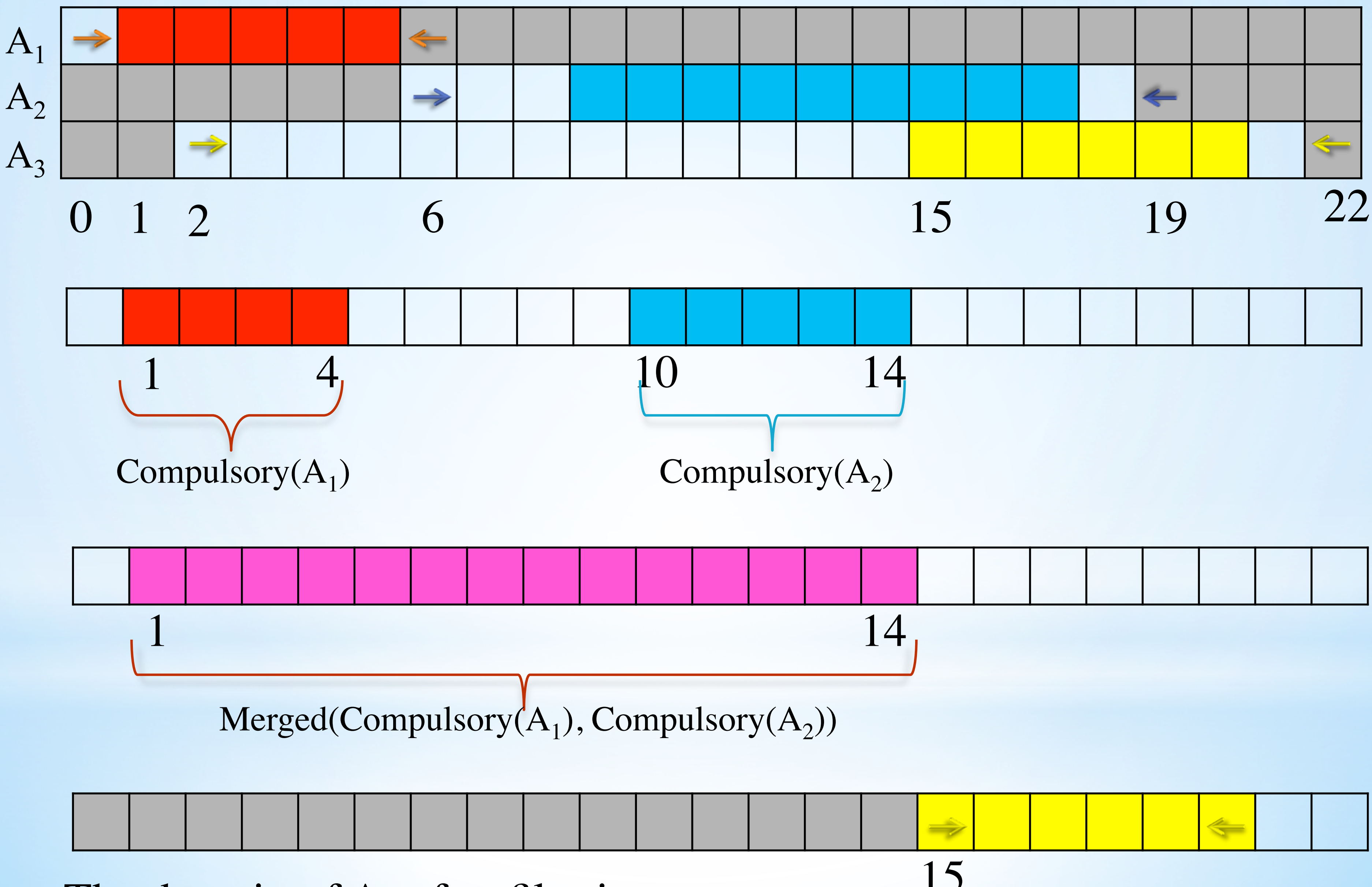
The strategy of our algorithm



The strategy of our algorithm



The strategy of our algorithm



- The domain of A_3 after filtering.

Time line

- This is a data structure that keeps track of when the resource is executing a task.

- It is initialized with an empty set of tasks $\Theta = \emptyset$.

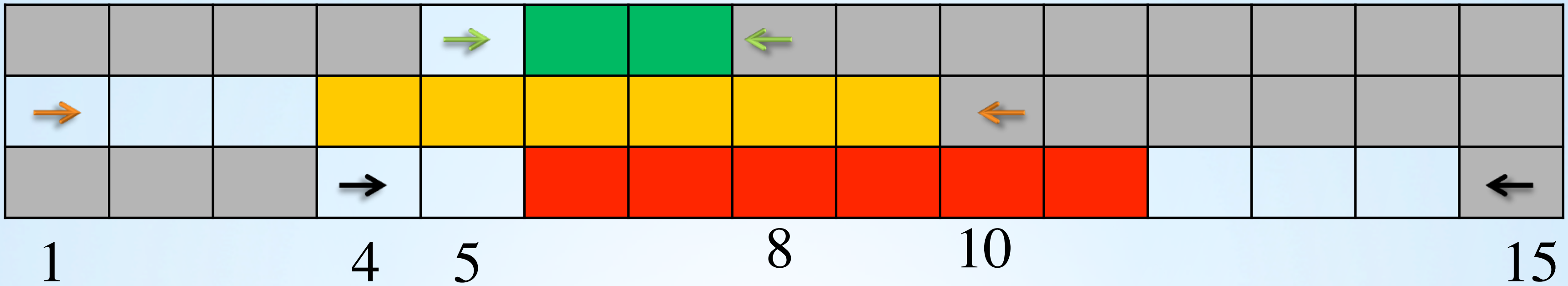
- It is possible to add a task to Θ in constant time. The task will be scheduled at the earliest time as possible with preemption.

- It is possible to compute the earliest completion time of Θ in constant time at any time!

Θ-Tree and Time line comparison

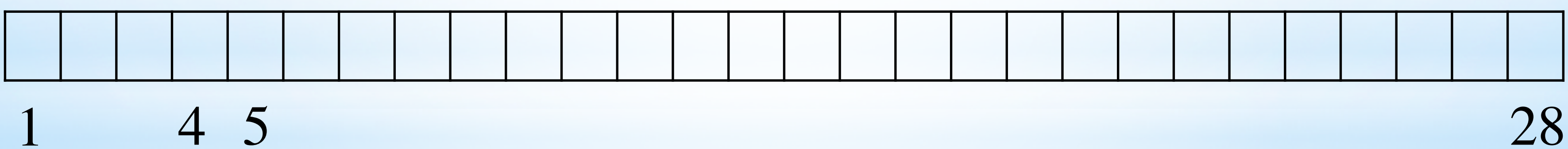
Operation	Θ-Tree (Vilím)	Time line
Adding a task to the schedule	$O(\log(n))$	O(1)
Computing the earliest completion time	O(1)	O(1)
Removing a task from the schedule	$O(\log(n))$ steps	Not supported !

Time line example



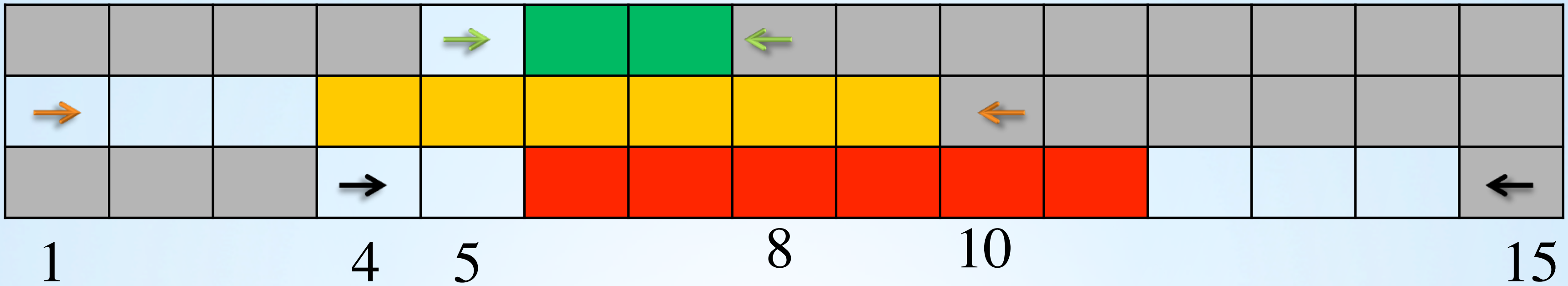
- Between each two consecutive time points, there is a capacity that denotes the amount of time that the resource is available through. The capacities are initially equal to the difference between the consecutive time points.

est_i	lct_i,	p_i
5	8	2
1	10	6
4	15	6



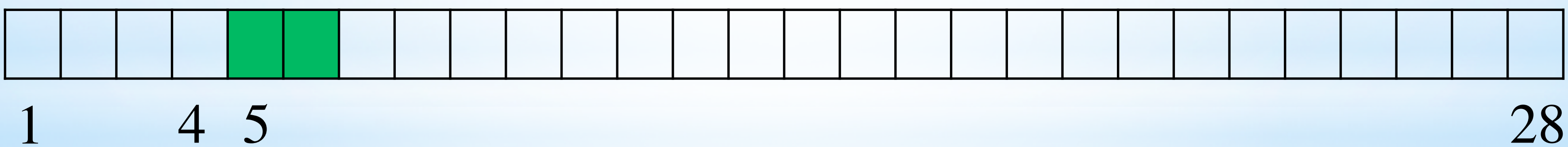
$$\{1\} \xrightarrow{3} \{4\} \xrightarrow{1} \{5\} \xrightarrow{23} \{28\}$$

Time line example



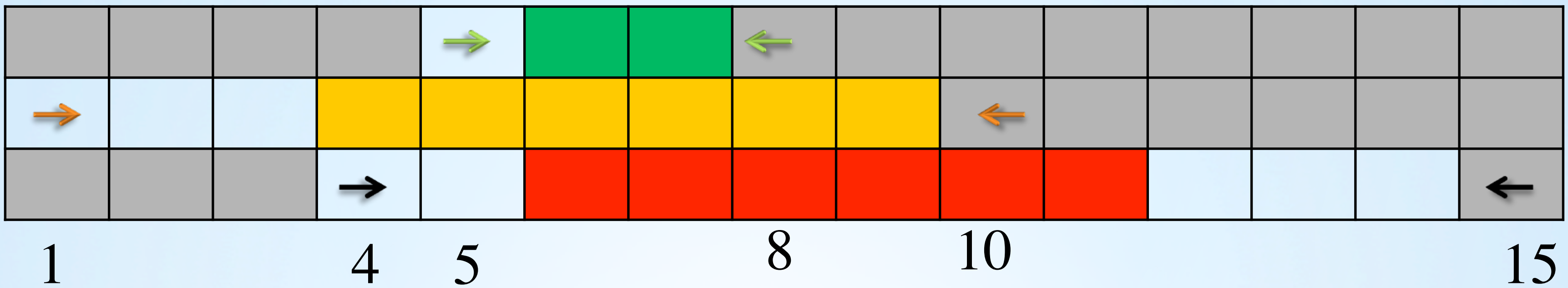
- We schedule the tasks, one by one. After scheduling, the free times will reduce.

est_i	lct_i,	p_i
5	8	2
1	10	6
4	15	6



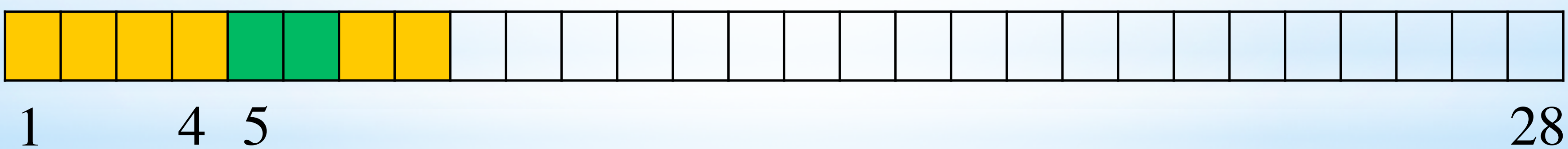
$$\{1\} \xrightarrow{3} \{4\} \xrightarrow{1} \{5\} \xrightarrow{21} \{28\}$$

Time line example



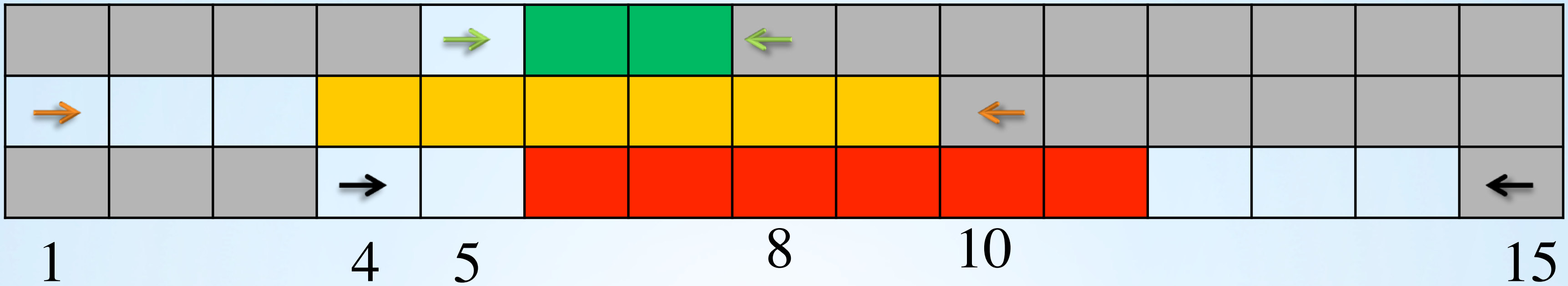
- We schedule the tasks, one by one. After scheduling, the free times will reduce.

est_i	lct_i,	p_i
5	8	2
1	10	6
4	15	6



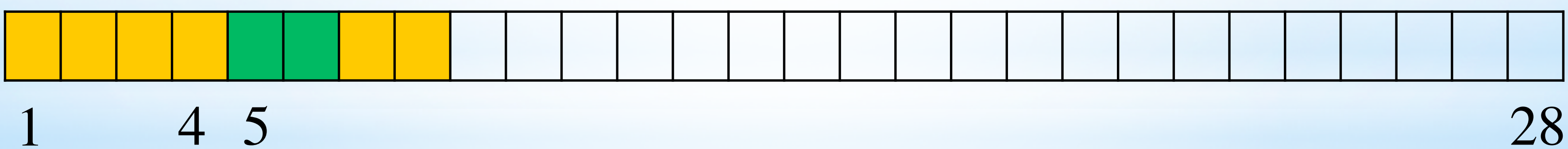
$$\{1\} \xrightarrow{0} \{4\} \xrightarrow{0} \{5\} \xrightarrow{19} \{28\}$$

Time line example



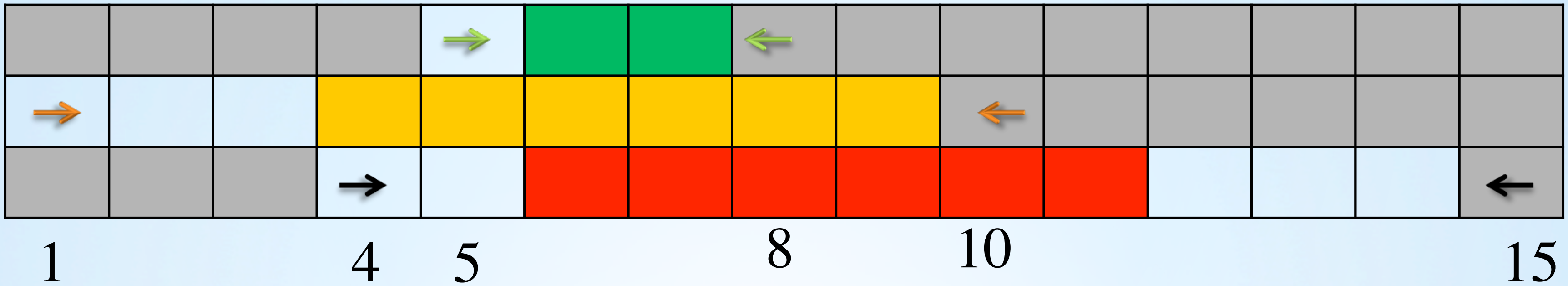
- Once a capacity equals null, the corresponding time points are merged by Union-Find.

est_i	lct_i,	p_i
5	8	2
1	10	6
4	15	6



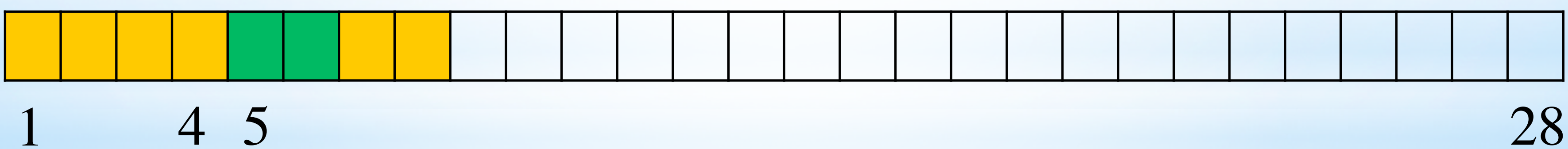
$$\{1\} \xrightarrow{0} \{4\} \xrightarrow{0} \{5\} \xrightarrow{19} \{28\}$$

Time line example



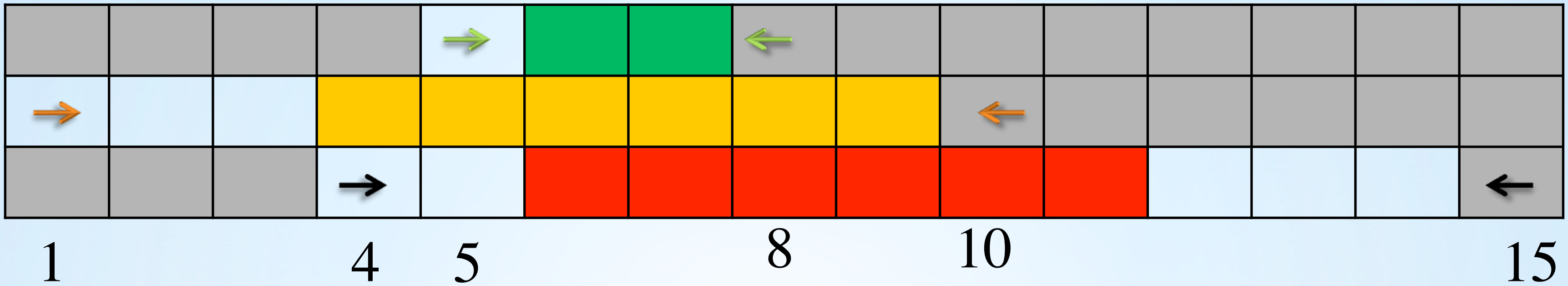
- Once a capacity equals null, the corresponding time points are merged by Union-Find.

est_i	lct_i,	p_i
5	8	2
1	10	6
4	15	6



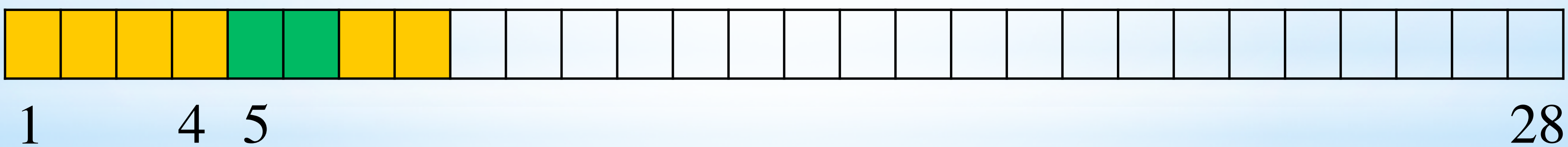
$$\{1,4,5\} \xrightarrow{19} \{28\}$$

Time line example



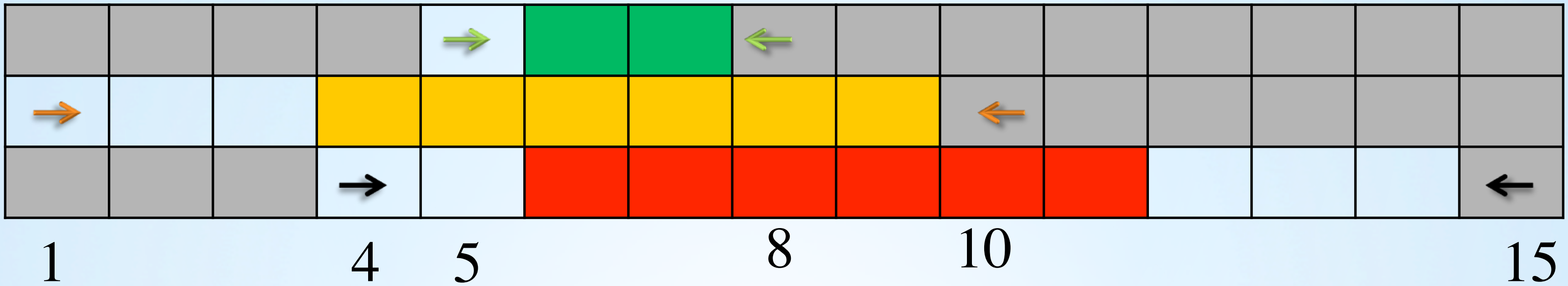
- That allows to run a linear search over the time line for periods that have free time. This search will jump over the occupied regions in constant time.

est_i	lct_i,	p_i
5	8	2
1	10	6
4	15	6



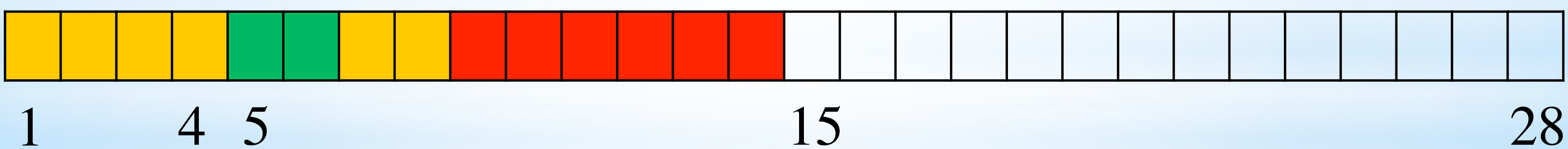
$$\{1,4,5\} \xrightarrow{19} \{28\}$$

Time line example



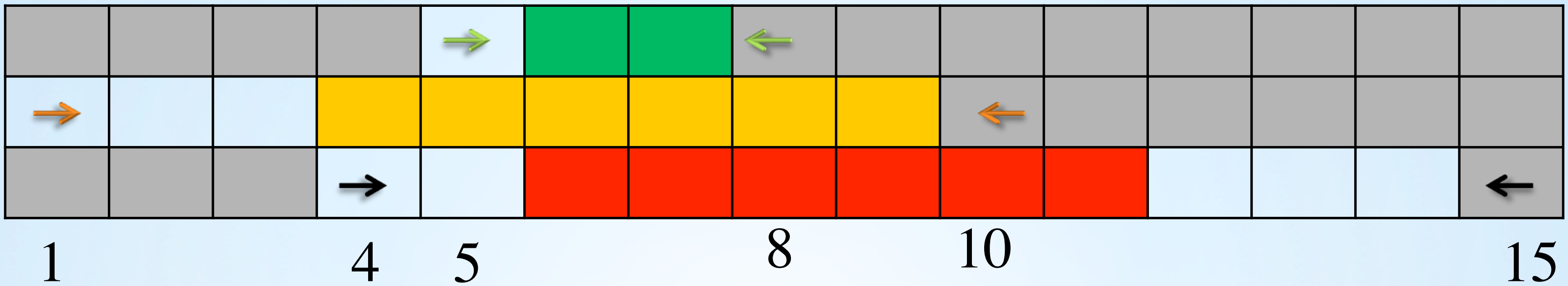
- That allows to run a linear search over the time line for periods that have free time. This search will jump over the occupied regions in constant time.

est_i	lct_i,	p_i
5	8	2
1	10	6
4	15	6



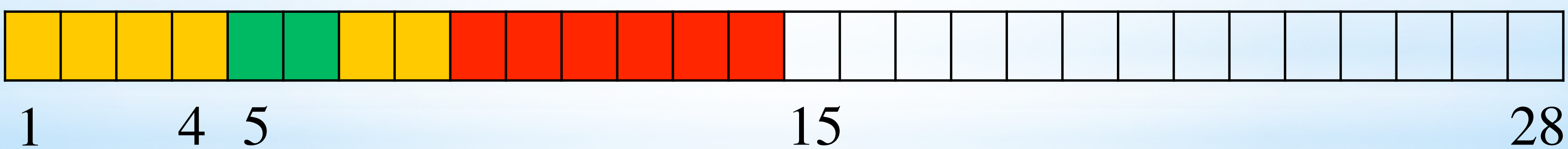
$$\{1,4,5\} \xrightarrow{13} \{28\}$$

Time line example



- That allows to run a linear search over the time line for periods that have free time. This search will jump over the occupied regions in constant time.

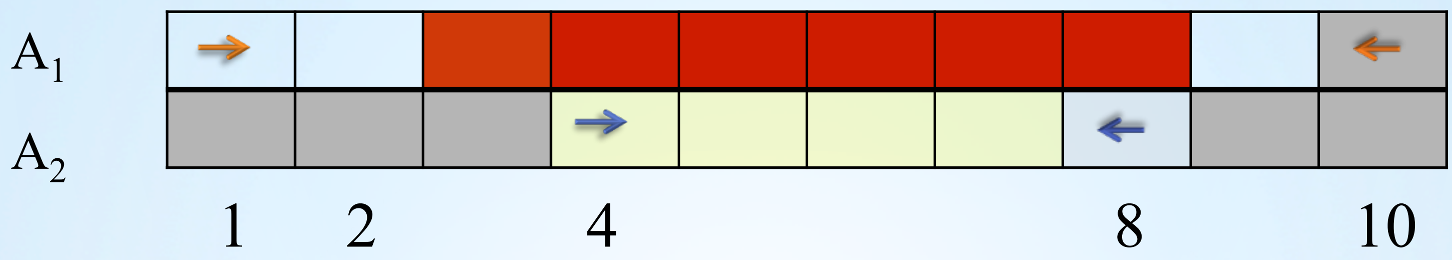
est_i	lct_i,	p_i
5	8	2
1	10	6
4	15	6



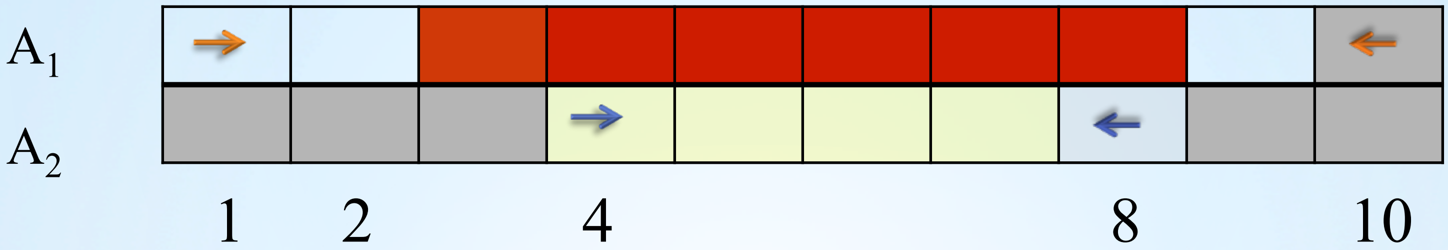
$$\{1,4,5\} \xrightarrow{13} \{28\}$$

- The earliest completion time is computed in constant time by 28-13 = 15.

Overload Checking



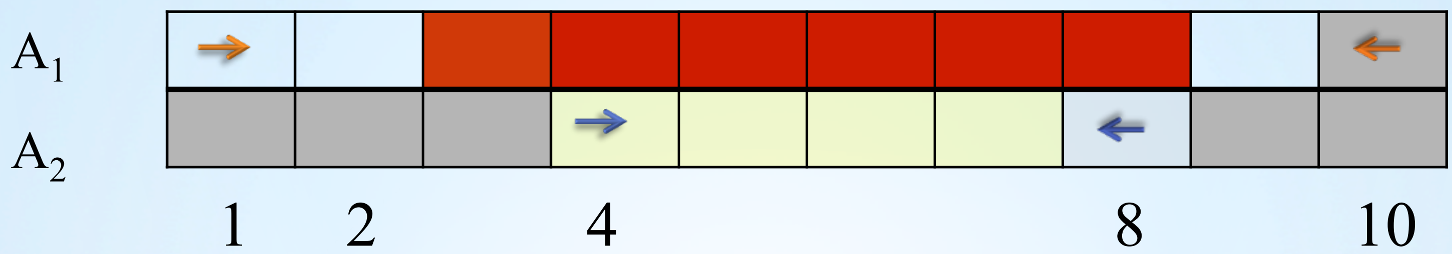
Overload Checking



- Using the idea of a Θ-Tree, Vilím presented the following algorithm for the overload check.

```
Θ := ∅;
for j ∈ T in non-decreasing order of lct_j do begin
    Θ := Θ ∪ {j};
    if ect_Θ > lct_j then
        fail; {No solution exists}
end;
```

Overload Checking



- Using the idea of a Θ-Tree, Vilím presented the following algorithm for the overload check.

```
Θ := ∅;
for j ∈ T in non-decreasing order of lct_j do begin
   Θ := Θ ∪ {j};
   if ect_Θ > lct_j then
      fail; {No solution exists}
end;
```

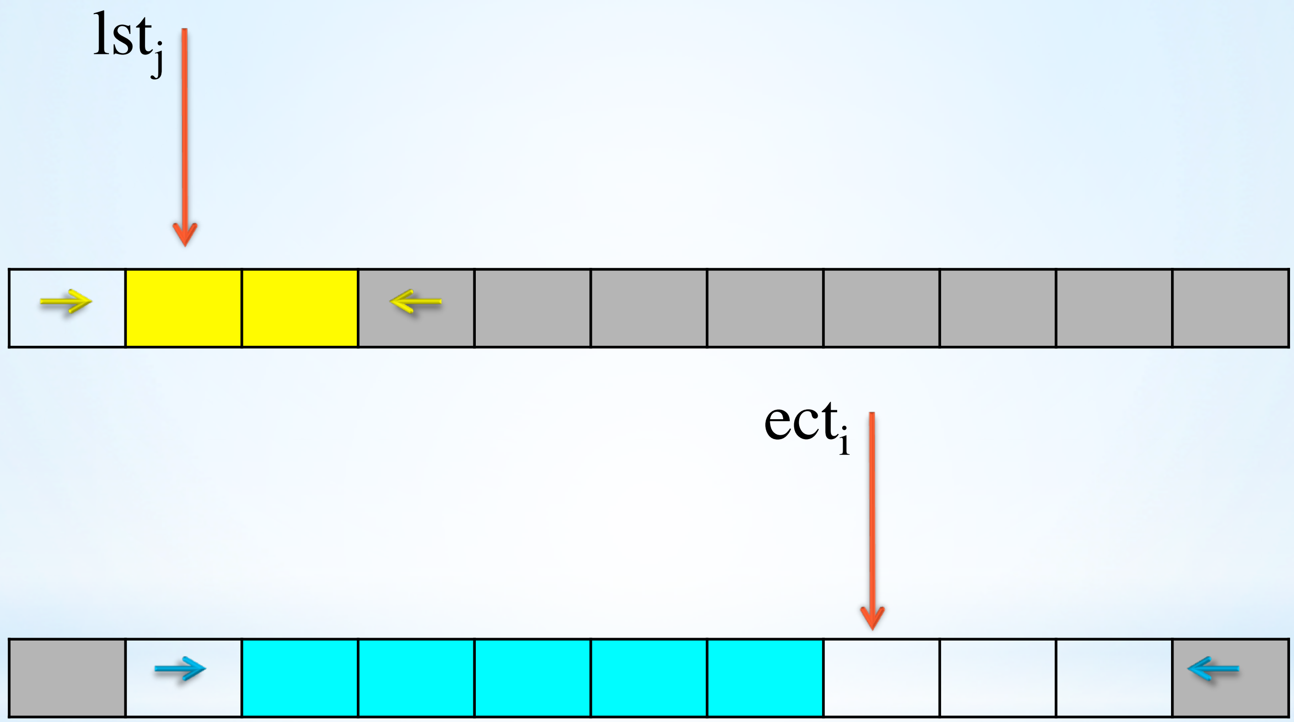
- We keep the same algorithm and only replace the Θ-Tree with time line to achieve a linear time algorithm.

Detectable Precedences

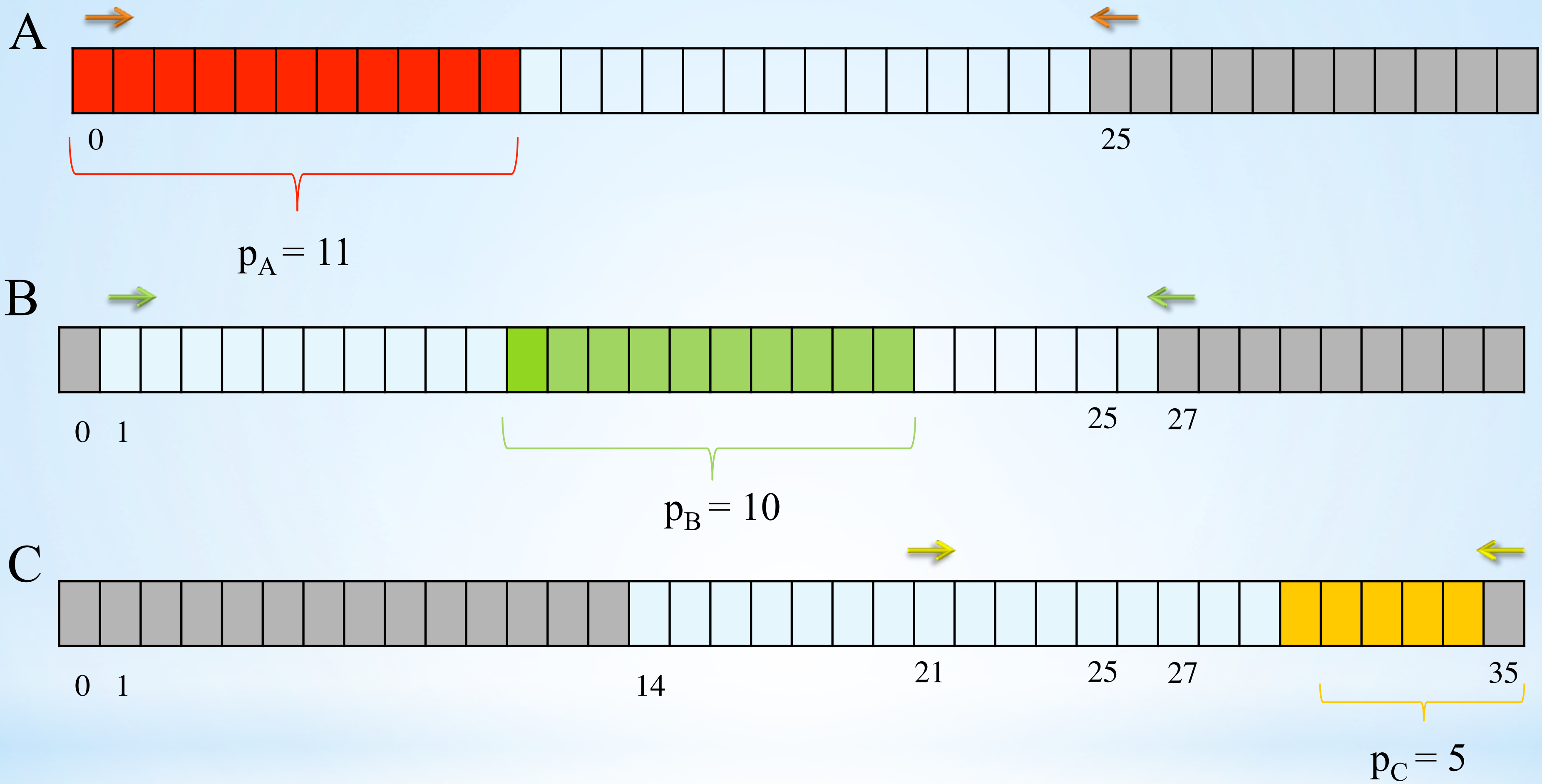
- Let A_i and A_j be two tasks. If $ect_i > lst_j$, the precedence $A_j \ll A_i$ is called *detectable*.

Detectable Precedences

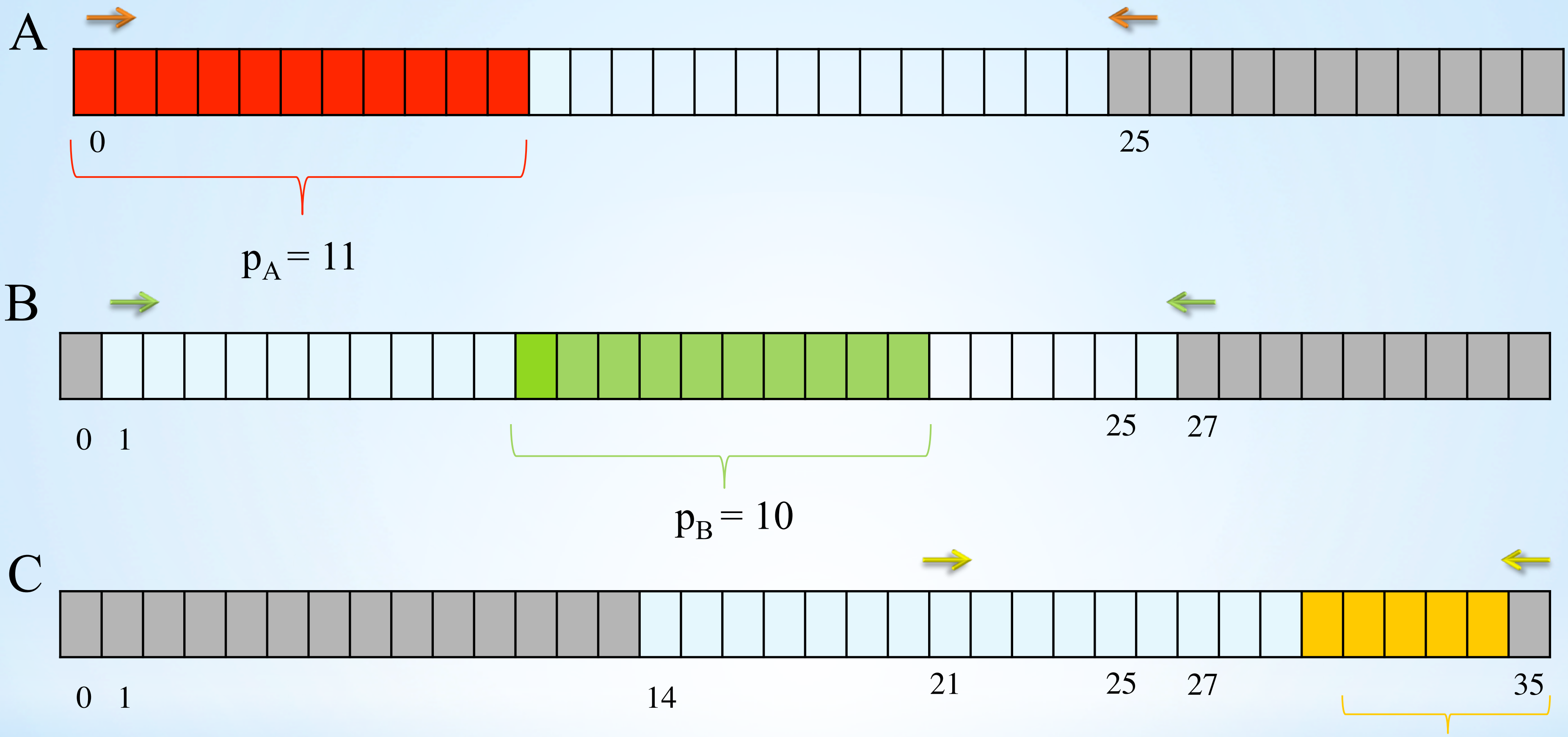
- Let A_i and A_j be two tasks. If $ect_i > lst_j$, the precedence $A_j << A_i$ is called *detectable*.



Example

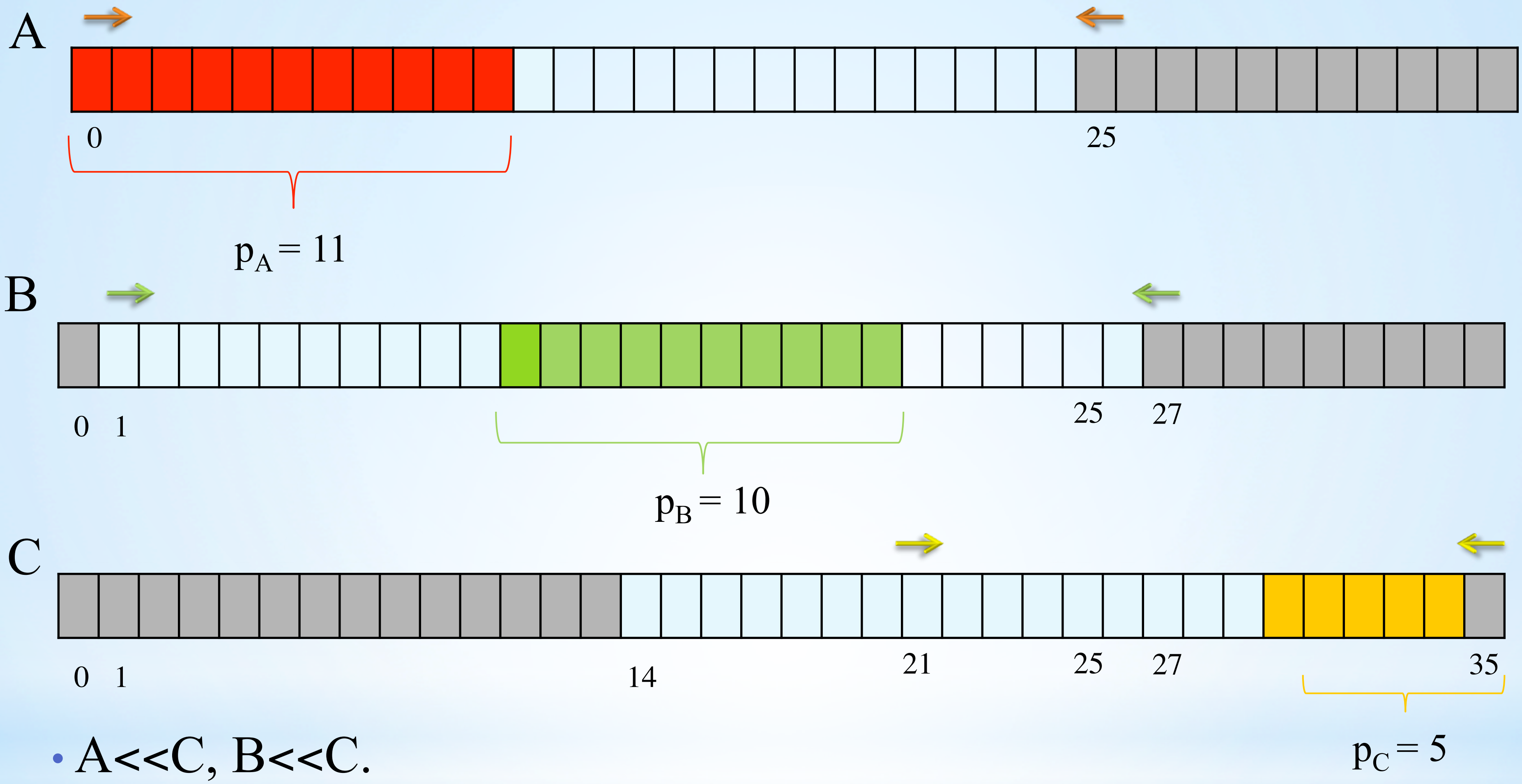


Example



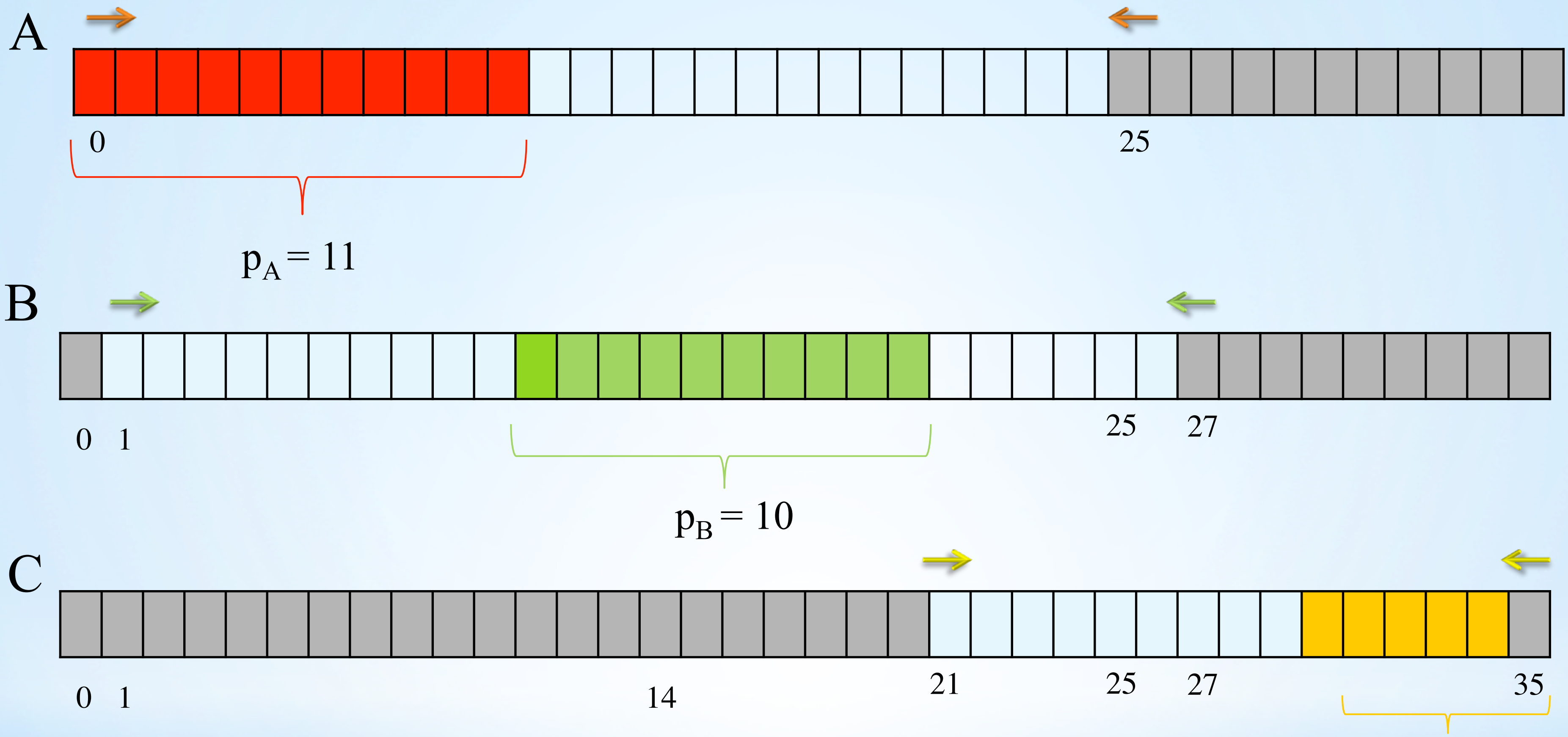
- A<<C, B<<C.

Example



- A<<C, B<<C.
- Since {A , B } << C, the domain of C will be filtered to $est_C \geq est_A + p_A + p_B = 21$.

Example



- A<<C, B<<C.
- Since {A , B } << C, the domain of C will be filtered to $est_C \geq est_A + p_A + p_B = 21$.
- The domain of C after filtering.

Detectable Precedences

- Vilím introduced the idea of detectable precedences and presented an algorithm in O($n$log(n)).

Detectable Precedences

- Vilím introduced the idea of detectable precedences and presented an algorithm in O($n$log(n)).

- This algorithm temporarily removes a task from the schedule, computes the earliest completion time of the set, and reinserts the task to the schedule.

Detectable Precedences

- Vilím introduced the idea of detectable precedences and presented an algorithm in $O(n\log(n))$.

- This algorithm temporarily removes a task from the schedule, computes the earliest completion time of the set, and reinserts the task to the schedule.

- The time line does not allow the removal of a task.

Detectable Precedences

- Vilím introduced the idea of detectable precedences and presented an algorithm in O($n$log(n)).

- This algorithm temporarily removes a task from the schedule, computes the earliest completion time of the set, and reinserts the task to the schedule.

- The time line does not allow the removal of a task.

- We modified the algorithm so that no removal of a task is required.

Experiments

- In order to show the advantage of the state of the art algorithms, we ran the experiments on job-shop and open-shop scheduling problems.
- After 10 minutes of computations, the program halts
- The problems are not solved to optimality.
- The number of backtracks that occur will be counted.
- We compare two algorithms which explore the same tree in the same order.
- A larger portion of the search tree will be traversed within 10 minutes with the faster algorithm.

Tables of results

$n \times m$	OC	DP	TT
4×4	0.96	1.00	1.00
5×5	1.03	1.12	1.75
7×7	1.02	1.16	2.09
10×10	1.06	1.33	2.14
15×15	1.03	1.39	2.15
20×20	1.06	1.56	2.17
p-value	0.25	8.28E-14	5.95E-14

$n \times m$	OC	DP	TT
10×5	1.07	1.27	2.11
15×5	1.02	1.35	2.27
20×5	1.00	1.55	2.12
10×10	1.01	1.25	2.18
15×10	1.26	1.42	1.97
20×10	1.00	1.47	2.14
30×10	1.08	1.56	2.36
50×10	1.05	1.48	3.18
15×15	0.95	1.48	2.16
20×15	1.04	1.61	2.13
20×20	1.09	1.46	1.71
p-value	0.17	1.41E-12	3.38E-20

- The results of three methods on open-shop and job-shop benchmark problems with *n* jobs and *m* tasks per job. The numbers indicate the ratio of the cumulative number of backtracks between all instances of size *nm* after 10 minutes of computations.

Conclusion

- Thanks to the constant time operation of the Union-Find data structure, we designed a new data structure, called time line, to speed up filtering algorithms for the Disjunctive constraint.
- We came up with three faster algorithms to filter the disjunctive constraint.

Algorithm	**Previous complexity**	**Now complexity**
Time-Tabling	O($n$log(n)) (Ouellet & Quimper)	O(n) (Fahimi & Quimper)
Overload check	O($n$log(n)) Vilím	O(n) (Fahimi & Quimper)
Detectable precedences	O($n$log(n)) Vilím	O(n) (Fahimi & Quimper)


Thank
you!