THE UNIVERSITY OF MELBOURNE 香港中文大學 The Chinese University of Hong Kong

Cumulative Scheduling

Jimmy Lee & Peter Stuckey

香港中文大學 The Chinese University of Hong Kong THE UNIVERSITY OF MELBOURNE

Need for More Manpower Resources

2

Quality Assurance

3

Combat Expertise

4

Warfare Expertise

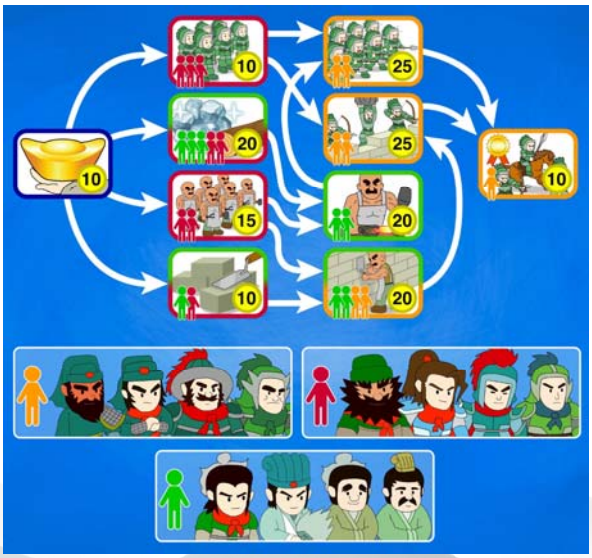
5

Resource Constrained Project Scheduling (RCPSP)

- Given tasks $t \in TASK$
- Given precedences p ∈ PREC
  - pre[p,1] precedes pre[p,2]
- Assume resources $r \in RESOURCES$
- Each task *t* needs *res*[*r*,*t*] resources during its execution
- We have a limit *L*[*r*] for each resource
- Find the shortest schedule to run every task!
- Possibly the most studied scheduling problem

6

RCPSP City Reinforcement



7

RCPSP City Data (cumul_sched.mzn)

```
include "globals.mzn";

enum TASK;

array[TASK] of int: duration;
int: p; % number of precedences
set of int: PREC = 1..p;
array[PREC,1..2] of TASK: pre;

int: t = sum(duration);
array[TASK] of var 0..t: start;

enum RESOURCE;
array[RESOURCE] of int: L; % resource limit
array[RESOURCE,TASK] of int: res;
```

8

RCPSP City Data File (cumul_sched.dzn)

```
TASK = {FUNDS, SOLDIERS, DEFENSE, WEAPONRY,
ELITEARMY, RAW_MATERIALS, CRAFTSMEN, CASTING,
BRICKS, WALL};

duration = [10,10,25,25,10,20,15,20,10,20];
p = 14; % number of precedences
pre =
  [| FUNDS, SOLDIERS          | FUNDS, RAW_MATERIALS
   | FUNDS, CRAFTSMEN         | FUNDS, BRICKS
   | SOLDIERS, WEAPONRY       | SOLDIERS, DEFENSE
   | WEAPONRY, ELITEARMY      | DEFENSE, ELITEARMY
   | RAW_MATERIALS, CASTING   | CRAFTSMEN, CASTING
   | CRAFTSMEN, WALL          | BRICKS, WALL
   | CASTING, WEAPONRY        | WALL, DEFENSE |];
```

9

RCPSP City Data File (cumul_sched.dzn)

```
RESOURCE = {QUALITY, MILITARY, COMBAT};
L = [4, 4, 4];
res = [| 0, 0, 0, 0, 0, 3, 0, 2, 1, 2
       | 0, 3, 0, 0, 0, 2, 2, 0, 1, 0
       | 0, 0, 2, 2, 1, 0, 0, 0, 0, 2 |];
```

10

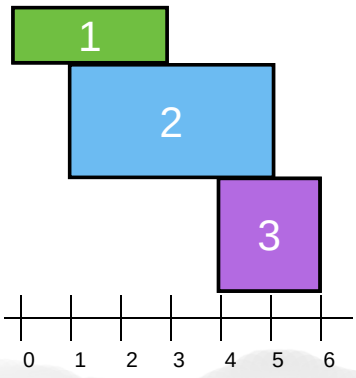
Resources

- Unary resources are unique
- Often we have multiple identical copies of a resource
  - bulldozers
  - workers (of equal capability)
  - operating theaters
  - airplane gates
- How do we model multiple identical resources?
  - assume task `t` uses `res[t]`
  - assume a limit `L` of resource at all times

11

Visualizing Resource Requirements

- A task t is a box of length duration[t] and height res[t] starting at time start[t]



12

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

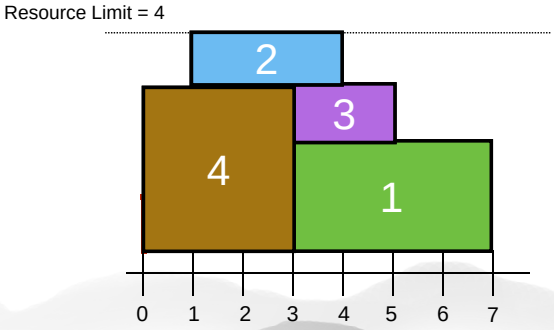
- Note the expression
  - `start[t]<=i /\ start[t]+duration[t]>i`
  - represents whether task `t` runs at time `i`

13

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

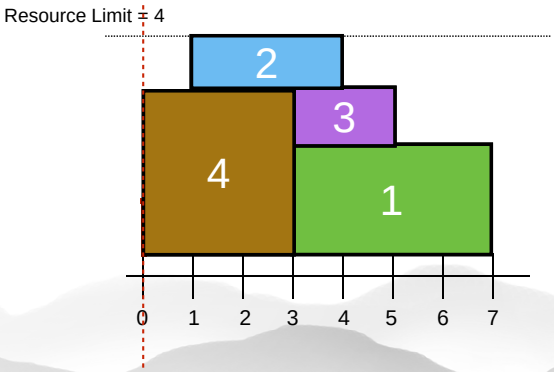


14

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

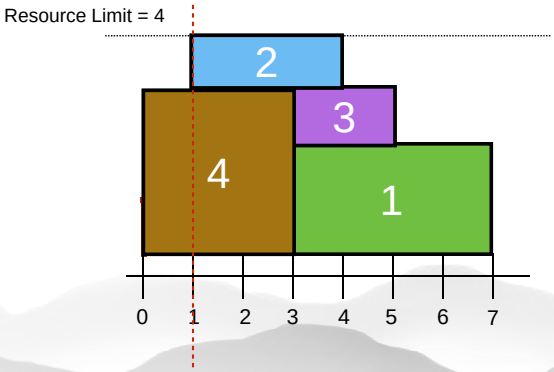


15

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

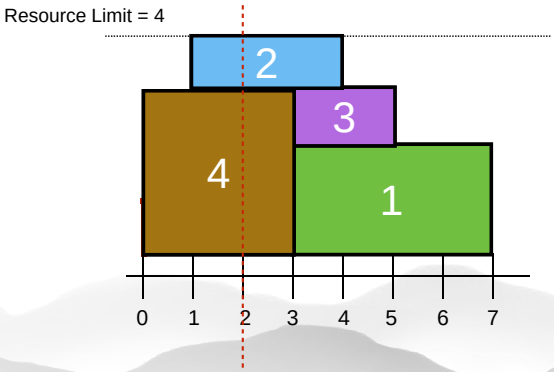


16

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

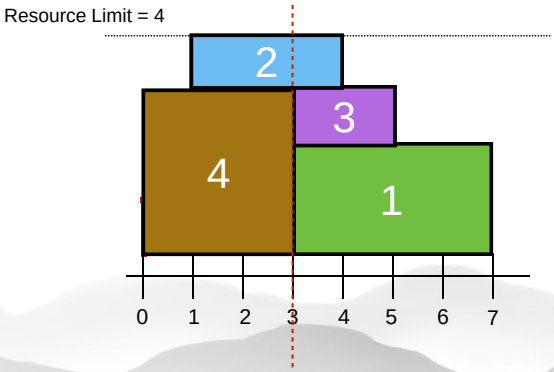


17

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

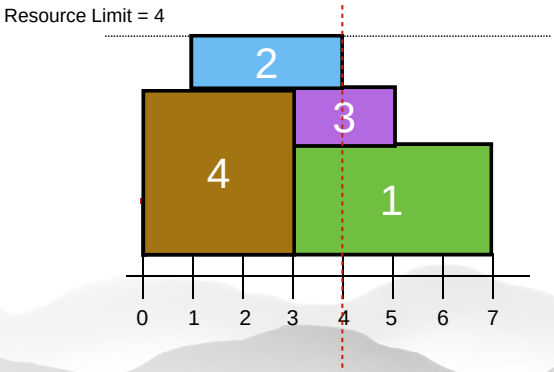


18

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

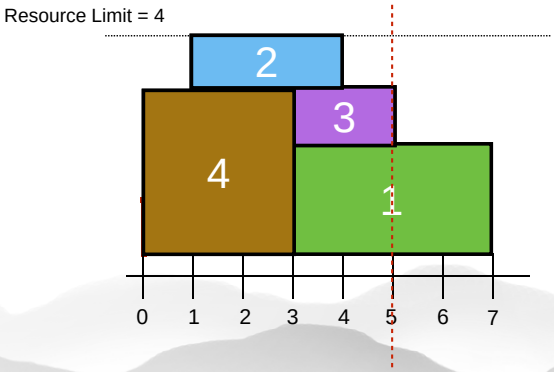


19

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

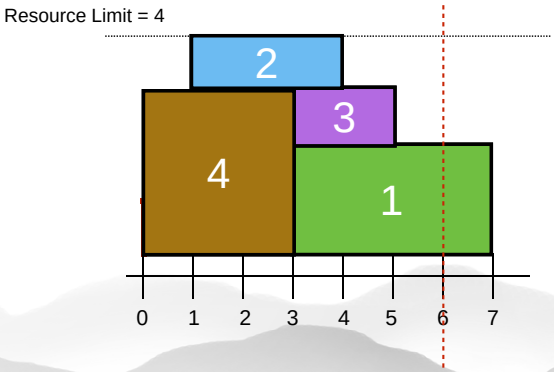


20

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

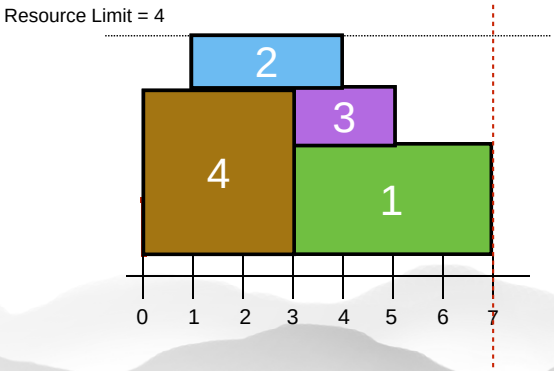


21

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```



22

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit L

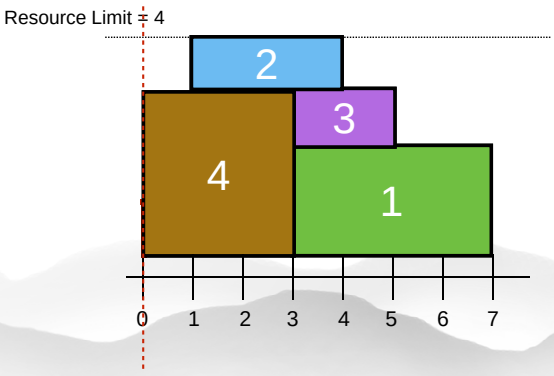
```
forall(i in TIME)(sum(t in TASK)
   ((start[t]<=i /\ start[t]+duration[t]>i)
      * res[t]) <= L);
```

- Note the expression
  - `start[t]<=i /\ start[t]+duration[t]>i`
  - represents whether task `t` runs at time `i`
- Problem: size is card(TASK)*card(TIME)
  - many time periods in TIME

Modeling Resources: Task Decomposition

- Note we can overload a resource only when a task starts (otherwise no increase)
- Alternate model: only check start times

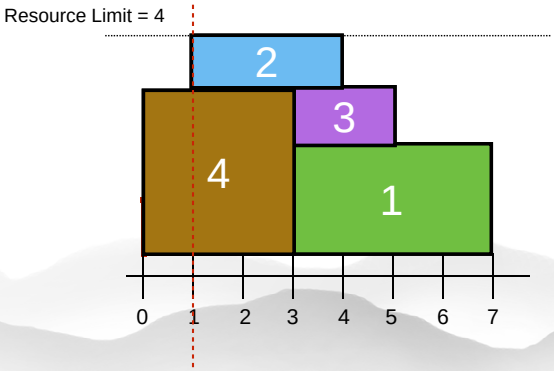
```
forall(t2 in TASK)(sum(t in TASK)
   ((start[t] <= start[t2] /\ start[t] + duration[t]
      > start[t2]) * res[t]) <= L);
```



Modeling Resources: Task Decomposition

- Note we can only overload a resource when a task starts (otherwise no increase)
- Alternate model: only check start times

```
forall(t2 in TASK)(sum(t in TASK)
   ((start[t] <= start[t2] /\ start[t] + duration[t]
      > start[t2]) * res[t]) <= L);
```

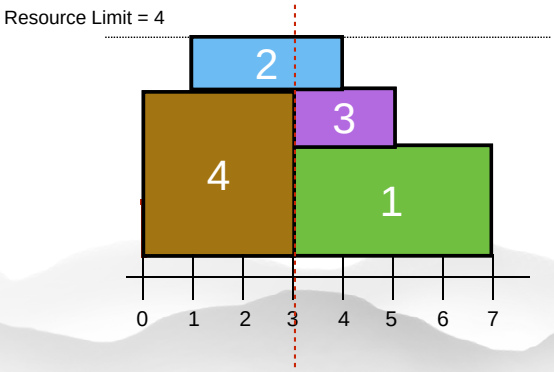


25

Modeling Resources: Task Decomposition

- Note we can only overload a resource when a task starts (otherwise no increase)
- Alternate model: only check start times

```
forall(t2 in TASK)(sum(t in TASK)
   ((start[t] <= start[t2] /\ start[t] + duration[t]
      > start[t2]) * res[t]) <= L);
```



26

Modeling Resources: Task Decomposition

- A more explicit formulation

```
forall(t2 in TASK)(sum(t in TASK where t != t2)
   ((start[t] <= start[t2] /\ start[t] + duration[t]
      > start[t2]) * res[t]) + res[t2] <= L);
```

- Comparison with time decomposition
  - Advantage: much smaller than time decomposition with card(TASK)^2
  - Problem: not as much information (fewer constraints) to the solver

27

Cumulative

- The cumulative global constraint captures exactly a resource constraint

```
cumulative(<start time array>, <duration array>,
           <resource usage array>, <limit>)
```

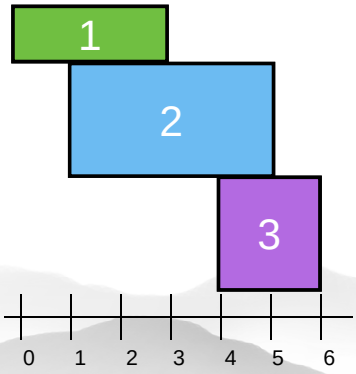
  - ensure no more than the limit of the resource is used at any time during the execution of tasks

```
predicate cumulative(array[int] of var int: s,
                     array[int] of var int: d,
                     array[int] of var int: r,
                     var int: L);
```

28

Visualizing Cumulative

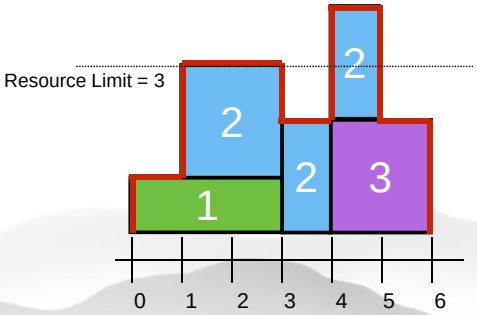
- Does the constraint below hold?
  - e.g. cumulative([0,1,4],[3,4,2],[1,2,2],3)



29

Visualizing Cumulative

- A task t is a box of length d[t] and height r[t] starting at time s[t]
  - e.g. cumulative([0,1,4],[3,4,2],[1,2,2],3)
- They are not really boxes
- Timetable (red skyline) shows the usage



30

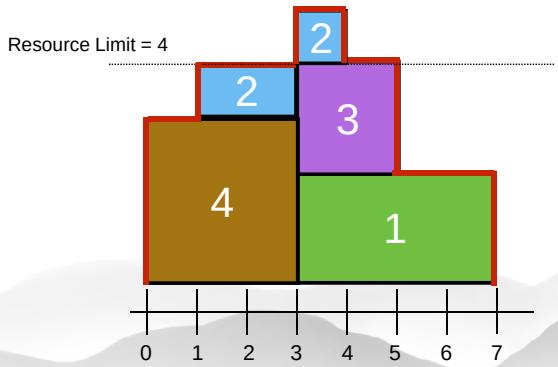
More Cumulative Examples

- Does the constraint below hold?
  - cumulative([3,1,3,0],[4,3,2,3],[2,1,2,3],4)
- Given the cumulative constraint below, does it have a solution?
  - start time possibilities are given as ranges
  - cumulative([0..3,0..3,2..3,0..4],[4,3,2,3],[2,1,2,3],4)

31

Visualizing Cumulative

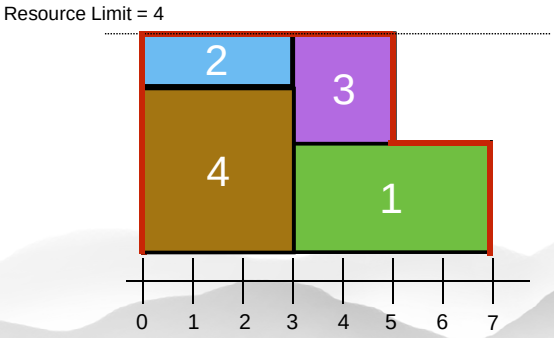
- Does the constraint below hold?
  - cumulative([3,1,3,0],[4,3,2,3],[2,1,2,3],4)



32

Visualizing Cumulative

- Given the cumulative constraint below, does it have a solution?
  - start time possibilities are given as ranges
  - cumulative([0..3,0..3,2..3,0..4],[4,3,2,3],[2,1,2,3],4)



33

RCPSP City Model (cumul_sched.mzn)

- Decisions

```
array[TASK] of var TIME: start;
```

- Constraints

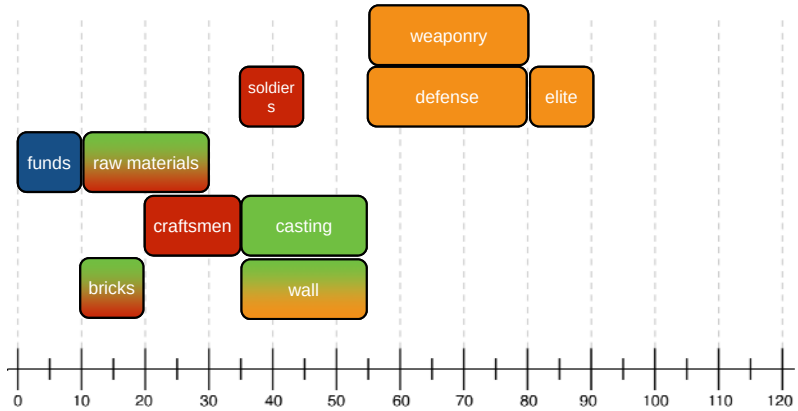
```
forall(p in PREC)
      (start[pre[i,1]] + duration[pre[i,1]]
         <= start[pre[i,2]]);
forall(r in RESOURCE)
      (cumulative(start, duration,
         [res[r,t] | t in TASK], L[r]));
```

- Objective

```
solve minimize max(t in TASK)
   (start[t] + duration[t]);
```

34

RCPSP City Solution



4 3 4 0 3 0
3 4 3 0 0 0
0 0 2 4 2 2

```
makespan    F   S   D  We   E   R  Cr  Ca   B  Wa
90        = [0, 35, 55, 55, 80, 10, 20, 35, 10, 35]
```

35

Summary

- There is a lot of research in how to propagate cumulative constraints
  - timetable propagation
    - equivalent to the time decomposition
    - but faster than the task decomposition
  - edge finding
    - reasoning about time intervals rather than single times
  - energy based reasoning
    - more inference than edge finding, but slower
  - TTEF time table edge finding
    - a combination of timetable with some energy based reasoning
    - state of the art

36

Summary

- Renewable capacitated resources
  - a resource capacity available over the schedule
- Time decomposition:
  - check resource usage at each time
- Task decomposition
  - check resource usage as each task starts
- `cumulative` global constraint

- RCPSP: a core scheduling problem

37

Image Credits

All graphics by Marti Wong, ©The Chinese University of Hong Kong and the University of Melbourne 2016

38