Disjunctive Scheduling

Peter Stuckey

Scheduling Concepts (so far)

- Tasks
  - start time, duration, and end time
  - other attributes

```
array[TASK] of var int: s;
array[TASK] of var int: d;
array[TASK] of var int: e;
forall(t in TASK)(e[t] = s[t] + d[t]);
```

  - may omit end times, particular when d is fixed
- Precedences
  - one task can only start after another finishes
  - task t1 precedes t2

```
e[t1] <= s[t2]     (s[t1] + d[t1] <= s[t2])
```

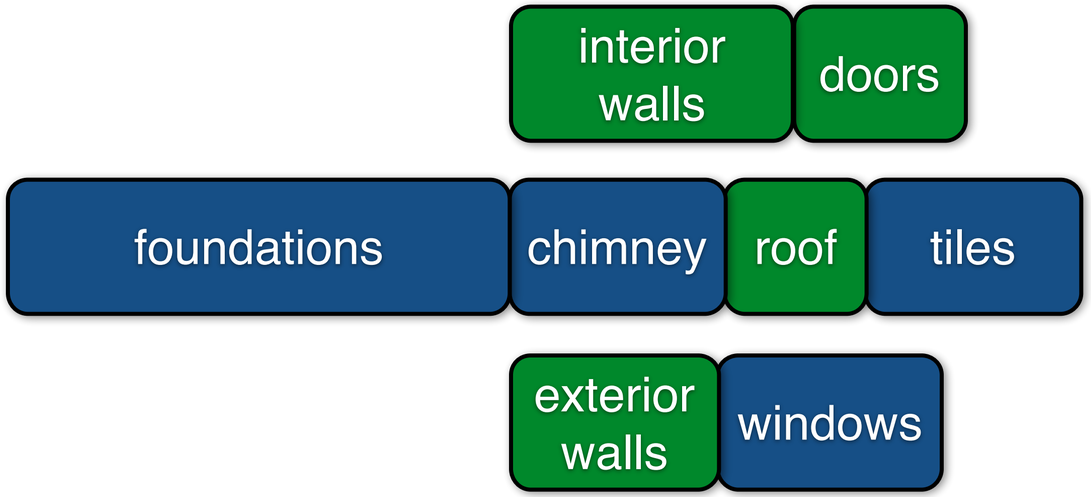
Nonoverlap

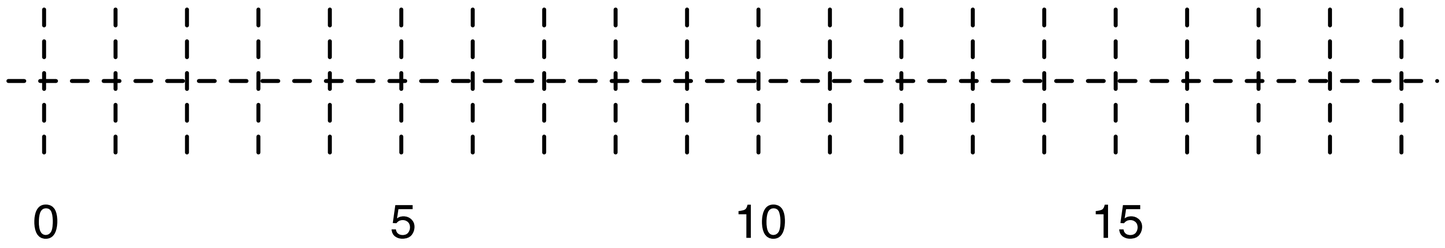
- Consider the ProjectScheduling problem where we only have one carpenter who can undertake the walls and roof work
  - these tasks cannot overlap in duration

```
predicate nonoverlap(var int:s1, var int:d1,
                     var int:s2, var int:d2)=
          s1 + d1 <= s2 \/ s2 + d2 <= s1;

set of TASK: CARPENTRY = { iw, ew, r, d };
forall(t1, t2 in CARPENTRY where t1 < t2)
      (nonoverlap(start[t1],duration[t1],
                  start[t2],duration[t2]));
```

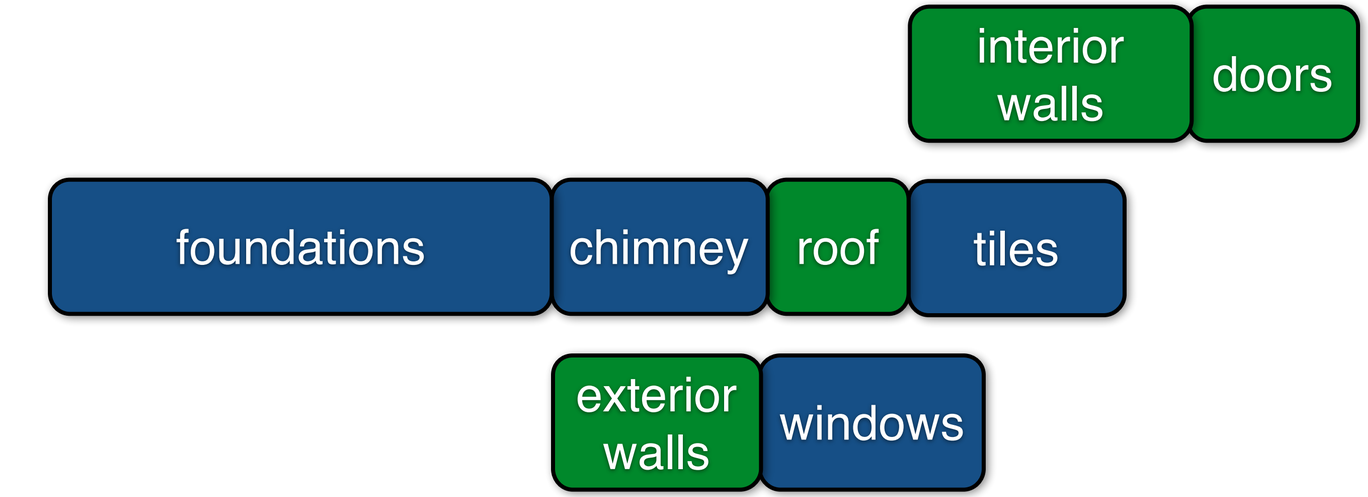
ProjectScheduling with Carpentry

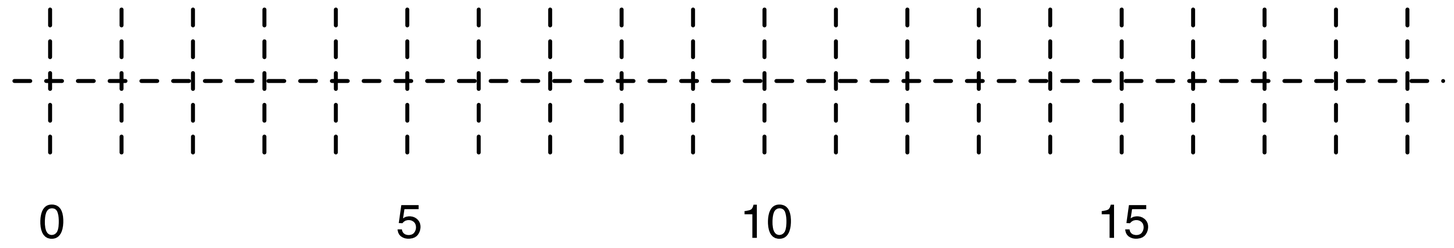





```
makespan     f   iw ew   c    r    d    t    w
15         = [0,  7, 7, 7, 10, 11, 12, 10]
```

ProjectScheduling with Carpentry






```
makespan     f   iw ew   c    r    d    t    w
18         = [0, 12, 7, 7, 10, 16, 12, 10]
```

Resources

- Critical to most scheduling problems are limited resources
  - unary resource (at most one task at a time)
  - cumulative resource (a limit on the amount of resource used at any time)

Unary Resources

- The ProjectScheduling problem with non overlap involved a unary resource
  - number of tasks executing at one time

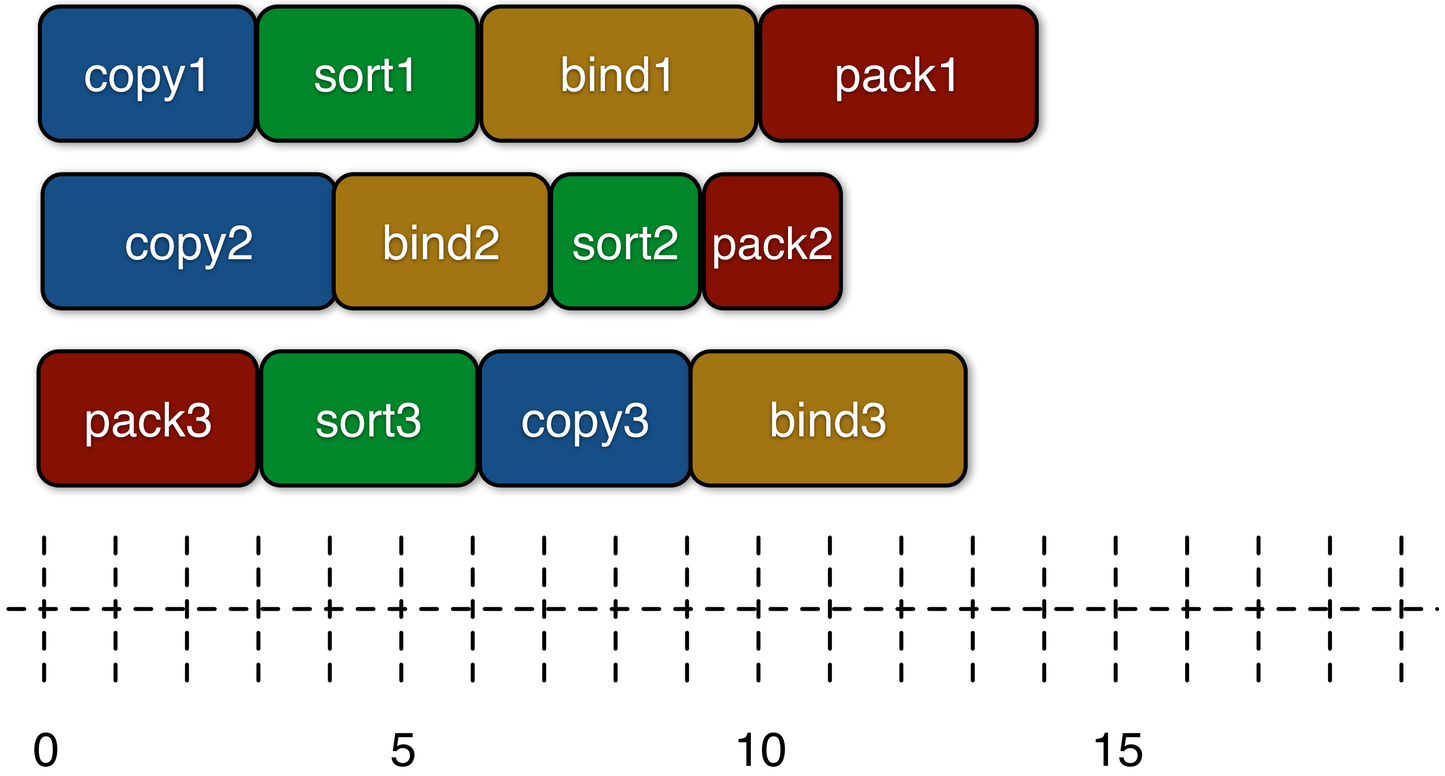
- Unary resources are common
  - machine
  - nurse, doctor, worker in a roster
  - track segment (one train at a time)
  - …

JobShop Scheduling

- JobShop: Given *n* jobs each made up of a sequence of *m* tasks, one each on each of *m* machines. Schedule the tasks to finish as early as possible where each machine can only run one task at a time
- Data

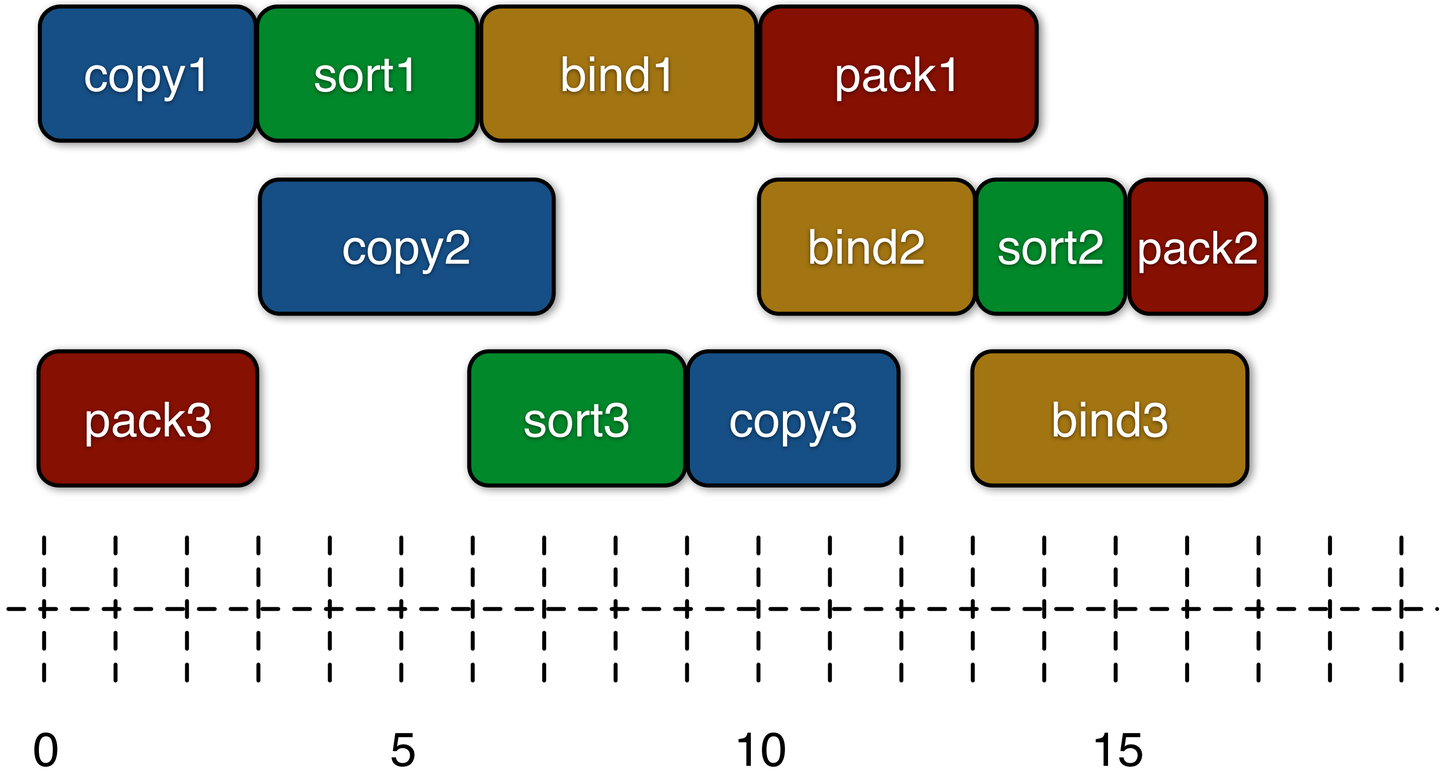
```
int: n;
set of int: JOB = 1..n;
int: m;
set of int: MACH = 1..m;
set of int: TASK = 1..m;
array[JOB,TASK] of int:  du; % length of task
array[JOB,TASK] of MACH: mc; % which machine
```

JobShop Example



- Rows indicate tasks in a Job
- Colors indicate different machine

JobShop Solution



- Tasks pushed later so no two of the same color are simultaneous

JobShop Variables + Constraints

- Variables

```
int: maxt = sum(j in JOB, t in TASK)(d[j,t]);
array[JOB,TASK] of var 0..maxt: s;
```

- Precedence Constraints

```
forall(j in JOB, t in 1..m-1)
       (s[j,t] + d[j,t] <= s[j,t+1]);
```

- Machine Constraints

```
forall(j1, j2 in JOB, t1, t2 in TASK where
       j1 < j2 /\ mc[j1,t1] = mc[j2,t2])
       (nonoverlap(s[j1,t1],d[j1,t1],
                   s[j2,t2],d[j2,t2]));
```

JobShop Objective

- Minimize the makespan (when the last job finishes)

```
var 0..maxt: makespan;
forall(j in JOB)
      (s[j,m] + d[j,m] <= makespan);
solve minimize makespan;
```

disjunctive

- Nonoverlap only considers two tasks at a time
  - a unary resource requires non overlap for all pairs of tasks that use it
- Disjunctive constraint
  - disjunctive(<start time array>,<duration array>)
    - ensure no two tasks in the array overlap in execution

```
predicate disjunctive(array[int] of var int:s,
                      array[int] of var int:d)=
   forall(i1,i2 in index_set(s) where i1 < i2)
         (nonoverlap(s[i1,d[i1],s[i2],d[i2]));
```

JobShop Revisited

- Replace nonoverlap with disjunctive
- We need to build the start times and durations for all jobs on a machine
  - perfect for a local variable

```
include "disjunctive.mzn";
forall(ma in MACH)
      ( let { array[int] of var int: ss =
              [ s[j,t] | j in JOB, t in TASK
                       where mc[j,t] = ma ];
            array[int] of int:      dd =
            [ d[j,t] | j in JOB, t in TASK
                  where mc[j,t] = ma ]; } in
        disjunctive(ss,dd));
```

JobShop Scheduling

- Is remarkably hard
- For some 10x10 instances from 1963
  - we did not know the optimal solution until 1989!
- There are a lot of approximation algorithms
- The online version is also heavily studied
  - where we have to schedule a job, given an existing schedule, then schedule the next job

A note about disjunctive

- In the current MiniZinc library
  - disjunctive is not included

- You can use cumulative to define it

```
include “cumulative.mzn";
predicate disjunctive(array[int] of var int:s,
                      array[int] of int: d) =
    cumulative(s,d,[1| i in index_set(s)], 1);
```

Overview

- Disjunctive scheduling
  - allows us to express that two tasks do not overlap in execution
  - without specifying the relative order

- disjunctive global constraint
  - capture a set of tasks on a unary resource

- Many classic scheduling problems
  - job shop scheduling
  - open shop scheduling

EOF