```
using CP;
// Load task set
{string} Tasks = ...;
// Load duration for each task
int durations[Tasks] = ...;
// Load start time for each task
int start    [Tasks] = ...;

// Load groups
{string} Groups = ...;
//Load max unused workers for each group
int maxUnusedWorkers[Groups] = ...;
// Load group set which can perform each task
{string} mayperform[Tasks] = ...;

// Inferred data, tuple of task and group which can perform this task
tuple OptTask {
  string task;
  string group;
}
{OptTask} optTasks = { <t,g> | t in Tasks, g in mayperform[t] };
// Load workers
{string} Workers = ...;
// Load workers for each group
{string} workers[Groups] = ...;
// Decide task interval
dvar interval tasks[t in Tasks] size durations[t];
// Each task is performed by one of the groups allowed
dvar interval opttasks[optTasks] optional;
// Deciee worker interval, each worker can only perform one job
dvar interval worker[Workers];
// FOr each group, get unused workers?
// When we have worker interval present, we add 1, this gets total number of workers on duty at
any time
// When a task is performed by a group, we remove a resource, therefore we get unused workers
at any time
// The worker interval variable will make sure we have proper period to cover each job and
constraints
cumulFunction group[g in Groups] =
  sum (w in workers[g]) pulse(worker[w], 1)
  - sum (<t,g> in optTasks) pulse(opttasks[<t,g>], 1);
```

```
execute {
                        cp.param.FailLimit = 5000;
}
// Minimise max interval for each worker
minimize max(w in Workers) lengthOf(worker[w]);

subject to {
  // Constrain that each task start time
  forall(t in Tasks) /* starts of Tasks */
    startOf(tasks[t]) == start[t];
  // Constrain that task is done by one of the options
  forall(t in Tasks)
    alternative(tasks[t], all(<t,g> in optTasks) opttasks[<t,g>]);
  // Make sure unused resources is >=0, and <= max unused.
  forall(g in Groups) {
    0 <= group[g];
    group[g] <= maxUnusedWorkers[g];
  }
};

execute {
  for (var w in Workers)
    writeln(w + " present from " + worker[w].start + " to " + worker[w].end);
}
```