Talent scheduling problem.

Problem description:
A set of actors need to perform some scenes in a movie, we try to sort the scenes in order so that overall wait time for actors is minimised. A wait time means the actor is between his first slot and last slot, but he is not at a scene, so he needs to wait for that scene to finish. Search for talent in the project loader. We add comments here to explain the model

```
// --------------------------------------------------------------------------
// Licensed Materials - Property of IBM
//
// 5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55
// Copyright IBM Corporation 1998, 2020. All Rights Reserved.
//
// Note to U.S. Government Users Restricted Rights:
// Use, duplication or disclosure restricted by GSA ADP Schedule
// Contract with IBM Corp.
// --------------------------------------------------------------------------

//
// This example is inspired from the talent hold cost scheduling problem
// described in:
//
// T.C.E Cheng, J. Diamond, B.M.T. Lin.    Optimal scheduling in film
// production to minimize talent holding cost.    Journal of Optimization
// Theory and Applications, 79:197-206, 1993.
//
// of which the 'Rehearsal problem' is a specific case:
//
// Barbara M. Smith.    Constraint Programming In Practice: Scheduling
//                          a Rehearsal.    Report APES-67-2003, September 2003.
//
// See: http://www.csplib.org/Problems/prob039/
//



using CP;

execute{
```

```
    }
//number of actors
int numActors = ...;
range Actors = 1..numActors;
// salary of each actor
int actorPay[Actors] = ...;
// number of sicnes
int numScenes = ...;
range Scenes = 1..numScenes;
// duration of each movie scene
int sceneDuration[Scenes] = ...;
// binary matrix, 1 means this actor is in this scene
int actorInScene[Actors][Scenes]   = ...;
// In each slot, the scene index
dvar int scene[Scenes] in Scenes;
// define the slot for each scene
dvar int slot[Scenes] in Scenes;


// First and last slots where each actor plays
// define an expression: the first slot
dexpr int firstSlot[a in Actors] = min(s in Scenes:actorInScene[a][s] == 1) slot[s];
// define the last slot
dexpr int lastSlot[a in Actors] = max(s in Scenes:actorInScene[a][s] == 1) slot[s];

// Expression for the waiting time for each actor
// if a scene's slot is between the first slot of the actor and the last slot, but the actor is
not in that scene
// add up the scene duration as wait time
dexpr int actorWait[a in Actors] = sum(s in Scenes: actorInScene[a][s] == 0)
    (sceneDuration[s] * (firstSlot[a] <= slot[s] && slot[s] <= lastSlot[a]));

// Expression representing the global cost
dexpr int idleCost = sum(a in Actors) actorPay[a] * actorWait[a];

minimize idleCost;
subject to {
    // use the slot-based secondary model
    // this is to ensure each scene is allocated to one slot and each slot corresponds to
one scene
```

```
    // inverse ==> slot[scene[i]] = i, the ith scene's slot corresponds to the ith scene'
    inverse(scene, slot);
}

tuple slotSolutionT{
      int Scenes;
      int value;
};
{slotSolutionT} slotSolution = {<i0,slot[i0]> | i0 in Scenes};
tuple sceneSolutionT{
      int Scenes;
      int value;
};
{sceneSolutionT} sceneSolution = {<i0,scene[i0]> | i0 in Scenes};
```