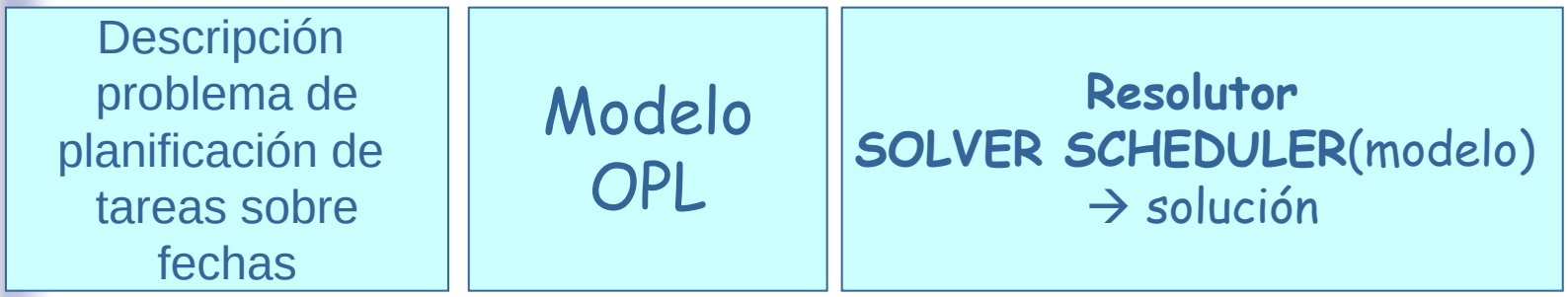
ILOG SOLVER SCHEDULER

Problemas de planificación de proyectos sobre calendarios.

Ciclo de vida



¿Qué nos ofrece ILOG?

- Sintaxis específica para modelarlos.
- Algoritmos de propósito específico para resolverlos.

Índice

- Adaptarme a la sintaxis básica específica de Scheduler.
  - Conceptos necesarios.
- Un problema sobre el que probar todos los recursos.
  - Problema de la construcción.

Diez ideas para empezar

- Puntos clave

1. Fechas sobre las que planificar el proyecto.
2. Actividades a realizar.
3. Recursos de que disponemos.

Diez ideas para empezar

- Fechas

[scheduleOrigin, scheduleHorizon)

- ✓ scheduleOrigin = 0;
- ✓ scheduleHorizon = 100;
- ✓ scheduleHorizon = sum(i in Tareas)
  duracion[Tareas];

Diez ideas para empezar

- Actividades

Activity mi_actividad;

mi_actividad.start
mi_actividad.end
mi_actividad.duration

¡Son las variables de decisión!

Diez ideas para empezar

- Una actividad → Restricción implícita

  a.end = a.start + a.duration

- ✓ Activity carpinteria;
- ✓ Activity carpinteria(10);
- ✓ Activity tareas[t in 1..10];
- ✓ Activity tareas[t in 1..10] (duracion[t]);

Diez ideas para empezar

- Actividades → Periodos de descanso

- ✓ Activity carpinteria breakable;
- ✓ Activity tareas[t in 1..10] (duracion[t])
  breakable if t in *conjunto*;

Diez ideas para empezar

- Recursos

  - *Unitarios*

```
UnaryResource excavadora;
UnaryResource maquinas[1..10];
```

  - *Discretos*

```
DiscreteResource presupuesto(30000);
DiscreteResource herramientas[t in 1..10] (cuantos[t]);
```

Diez ideas para empezar

- Restricciones entre actividades.

Caso normal:

b.start > a.end; → a precedes b;

Otros casos:

b.Start >= a.end + 3;

Influyen directamente sobre variables de decisión.

Diez ideas para empezar

- Restricciones actividad-recurso unitario.

✓ excavacion requires excavadora;
✓ excavacion consumes excavadora;

Influyen indirectamente en las variables de decisión.

Diez ideas para empezar

- Restricción de descanso.

  - Periódico
    - ✓ periodicBreak(excavadora,5,2,7);

  - Puntual
    - ✓ break(excavadora,10,12);

Diez ideas para empezar

- Restricción actividad-recurso discreto.

- ✓ excavacion requires(2) martillos;
- ✓ excavacion consumes(200) presupuesto;

Influyen indirectamente en las variables de decisión.

Problema de la construcción

- Queremos construir una casa.
- Veremos siete aproximaciones diferentes.
  Cada nuevo modelo utiliza todos los conceptos vistos en el anterior y añade algunos nuevos.
- El ritmo de trabajo será:
  i. Añadir nuevos conceptos teóricos de Scheduler.
  ii. Ponerlos en práctica en el siguiente modelo Casa*i*.mod

Casa1.mod

Con lo visto hasta ahora…

- Fechas → En principio no hay límite establecido.
- Actividades → Realizaremos una serie de tareas, cada una con cierta duración.
- Recursos →
  - ✓Unitarios: Tres trabajadores
  - ✓Discretos: Tres martillos

Casa1.mod

- Restricciones →
  - ✓Entre actividades:
    - Orden de las tareas
  - ✓Actividades-recurso unitario:
    - Cada tarea requiere un trabajador
  - ✓Actividades-recurso discreto:
    - Cada tarea requiere un martillo.

Casa1.mod

```
enum Tareas { albanileria, carpinteria, tuberias,
              techos, tejado, pintura, linux,
              fachada, jardines, mudanza };
int duracion[Tareas] = [7,3,8,3,1,2,1,2,1,1];
enum Trabajadores { Thomas, Maite, Antoine };
```

Fechas

```
scheduleHorizon = sum(t in Tareas) duracion[t];
```

Casa1.mod

Actividades

Activity a[t in Tareas](duracion[t]);

Recursos

Unitarios →

UnaryResource worker[t in Trabajadores];

Discretos →

DiscreteResource martillo(3);

DiscreteResource presupuesto(29000);

Casa1.mod

Variable de apoyo:

Muestra para cada tarea t si el trabajador w la realiza.

var int usa[Tareas,Trabajadores] in 0..1;

Objetivo

minimize

a[mudanza].end

Casa1.mod

Restricciones entre actividades.

a[albanileria] precedes a[carpinteria];

a[albanileria] precedes a[tuberias];

a[albanileria] precedes a[techos];

a[carpinteria] precedes a[tejado];

a[techos] precedes a[pintura];

a[tejado] precedes a[linux];

a[tejado] precedes a[fachada];

...

Casa1.mod

Restricciones actividad-recurso unitario.

```
forall(t in Tareas, w in Trabajadores)
    a[t] requires(usa[t,w]) worker[w];
forall(t in Tareas)
    sum(w in Trabajadores) usa[t,w] =1;
```

Casa1.mod

Restricciones actividad-recurso discreto.

```
forall(t in Tareas)
    a[t] requires(1) martillo;
```

Resultados OPL

Optimal Solution with Objective Value: 18

Actividades

a[albanileria] = [0 -- 7 --> 7]
a[carpinteria] = [7 -- 3 --> 10]
a[tuberias] = [7 -- 8 --> 15]
a[techos] = [10 -- 3 --> 13]
a[tejado] = [13 -- 1 --> 14]
a[pintura] = [14 -- 2 --> 16]
a[linux] = [16 -- 1 --> 17]
a[fachada] = [15 -- 2 --> 17]
a[jardines] = [15 -- 1 --> 16]
a[mudanza] = [17 -- 1 --> 18]

Resultados OPL

Recursos unitarios

worker[Antoine] = Unary Resource
  required by a[mudanza] over [17,18] in capacity 1
  …
worker[Thomas] = Unary Resource
  required by a[jardines] over [15,16] in capacity 1

worker[Maite] = Unary Resource
  required by a[fachada] over [15,17] in capacity 1
  …

Resultados OPL

Recursos discretos

martillo = Discrete Resource

- required by a[mudanza] over [17,18] in capacity 1
- required by a[jardines] over [15,16] in capacity 1
- required by a[fachada] over [15,17] in capacity 1
- required by a[linux] over [16,17] in capacity 1
- required by a[pintura] over [14,16] in capacity 1
- required by a[tejado] over [13,14] in capacity 1
- required by a[techos] over [10,13] in capacity 1

Resultados OPL

Log

Running

Solver + Scheduler

displaying solution ...

SOLVER

Variables: 60

Restricciones: 64

Fallos: 98

Choice Points: 109

Solver memory: 260708

Solving time: 0.25

Casa2.mod

Recursos alternativos

Antes veíamos que cada tarea requería un trabajador.

No podíamos hacer activity requires(1) worker, ya que el sistema entendería que durante la tarea podría cambiar de trabajador.

Alternatives resources modela un array de recursos unitarios, donde te da igual que casilla coger (son todos iguales) pero no lo puedes cambiar durante la tarea (son no intercambiables).

Casa2.mod

Recursos alternativos

```
UnaryResource worker[t in Trabajadores];
AlternativeResources workerAlternativos(worker);
```

Ahora si podemos hacer…

```
Activity a[t in Tareas] (duracion[t]);
forall(t in Tareas)
    a[t] requires(1) workerAlternativos;
```

Casa3.mod

Recursos discretos variables en el tiempo.

Se puede modelar no solo la cantidad que tendremos de cada recurso discreto, sino la cantidad máxima con que contaremos para cada intervalo de tiempo.

```
DiscreteResource presupuesto(29000);
capacityMax(presupuesto,0,15,20000);
```

Tendremos 20000 € desde el inicio al día 15, y 29000€ a partir del día 16.

Casa3.mod

Si las tareas consumen el presupuesto y este varía en el tiempo, es muy probable que haya que hacer cambios en las fechas en las que se planifica cada tarea.

```
forall(t in Tareas)
        a[t] consumes(1000*duracion[t]) presupuesto;
```

Casa4.mod

Proyecto de la casa con recursos que piden descanso y actividades que los conceden o no.

```
{Tareas} breakableSet = { albanileria, carpinteria,
                          tuberias, techos,
                          tejado, pintura, linux,
                        fachada, jardines, mudanza };

Activity a[t in Tareas](duracion[t]) breakable if t in
                                              breakableSet;
```

Casa4.mod

```
forall(t in Trabajadores)
  periodicBreak(worker[t],5,2,7);
```

Todas las tareas cambian su planificación para no ser realizadas en fines de semana.

Si alguna tarea con duración mayor que 5 no permite ser breakable entonces el sistema no encuentra solución.

Casa5.mod

Reservoirs

Es posible que haya recursos discretos que sean requeridos/consumidos por ciertas actividades y al mismo tiempo proporcionados/producidos por otras actividades.

Se les llamará Reservoirs.

- ✓ Reservoir cemento(1000);

Casa5.mod

Reservoirs

Activity a[1..4];

a1 requires(300) cemento;

a2 consumes(100) cemento;

a3 produces(150) cemento;

a4 provides(200) cemento;

Casa5.mod

Reservoirs

enum Tareas { albanileria, carpinteria, ..., hormigonar };

a[albanileria] requires(300) cemento;

/*1*/

a[hormigonar] produces(300) cemento;

Albañilería se planifica despues de hormigonar.

/*2*/

a[hormigonar] provides(300) cemento;

Albañilería se planifica al mismo tiempo que hormigonar

Casa6.mod

State resources

Recursos que puedan estar en un conjunto de estados.

En cualquier punto de la planificación el recurso estará en uno de los estados.

```
enum Estados { calido, lluvioso };
StateResource clima(Estados);
```

Casa6.mod

State resource

Las actividades pueden exigir a un State resource estar en un determinado estado durante su ejecución.

{States} noFrio = {caliente, templado};

Activity a;

a requiresState(hot) horno;

a requiresAnyState(noFrio) horno;

Casa6.mod

a[techos] requiresState(calido) clima;

a[tuberias] requiresState(lluvioso) clima;

Estas dos actividades compartían fechas en la planificación.

Ahora deberán ser separadas.

Casa7.mod

Ahora querremos planificar diez casas, de tres tipos diferentes.

Las tareas seguirán siendo las mismas, y con idénticas precedencias.

Las tareas tendrán diferente duración según el tipo de casa.

Cada tarea requerirá un trabajador.

Lo único que nos interesa es comprobar que en el calendario solución hay días en los que se planifican más de tres tareas.

Casa7.mod

```
range TipoCasa 1..3;
int duracion[TipoCasa,Tareas] = [ [7,3,8,3,1,2,1,2,1,1] ,
                                  [12,5,10,5,2,5,2,3,2,1] ,
                                  [15,3,10,6,2,3,2,3,2,1] ];
range rangoCasas 1..10;

Activity a[rangoCasas,Tareas];
```

Casa7.mod

Insertamos las precedencias en un array de datos.

Luego usaremos este array para declarar la restricción de precedencia de un modo más elegante.

```
{Tareas} precedencia[Tareas] = #[
        albanileria: {carpinteria,tuberias,techos},
        carpinteria: {tejado},
        tuberias: {fachada,jardines},
        techos: {pintura},
        tejado: {linux,fachada,jardines},
        pintura: {mudanza},
        linux: {mudanza},
        fachada: {mudanza},
        jardines: {mudanza},
        mudanza: {} ]#;
```

Casa7.mod

Cada casa es de uno de los tres tipos.

Cada casa debe ser planificada en un intervalo propio [scheduleOrigin, scheduleHorizon).

struct Casa { int TipoCasa; int startMin; int endMax; };

Casa casas[rangoCasas] = [<1,0,200>, <1,20,200>, <1,0,150>, <1,100,200>, <2,0,150>, <2,10,150>, <2,100,200>, <2,0,150>, <3,30,100>, <3,90,200>];

Casa7.mod

Discrete energy resource

Para utilizar distinta precisión según intervalos de tiempo durante la planificación.

- Si yo tengo 3 trabajadores…
  - Para una planificación diaria tendré como recurso 3 jornadas de trabajo.
  - Para una planificación semanal tendré 15 jornadas de trabajo. Por ejemplo, podré poner 7 jornadas en la planificación de un día y 8 en la de otro.
  - Para una planificación mensual tendré como recurso 90 jornadas de trabajo.

Casa7.mod

Sirven para planificar un proyecto con distinta precisión para diferentes etapas.

Necesitan ir acompañados de las diferentes etapas a las que se aplicarán diferentes precisiones.

```
int energia[1..4] = [1,2,5,10];
int intervalos[1..5] = [scheduleOrigin, 20, 40, 90,
                                              scheduleHorizon];
DiscreteEnergy trabajadores(2, energia, intervalos);
```

Puedes poner límites en el máximo y mínimo para un cierto intervalo.

Casa7.mod

El objetivo a minimizar será la fecha de la última mudanza.

Creamos una última tarea “ficticia”, de duración 0 y que deba ser la última en realizarse.

Activity inauguracion(0);

Casa7.mod

Las precedencias entre las tareas deberán ser respetadas:

```
forall(r in rangoCasas, t in Tareas, o in precedencia[t])
        a[r,t] precedes a[r,o];
```

Casa7.mod

Cada tarea debe tener asignada su duración y su requisito de un trabajador.

```
forall(r in rangoCasas, t in Tareas){
  a[r,t].duration = duracion[casas[r].TipoCasa, t];
  a[r,t] requires trabajadores;   };
```

Casa7.mod

La última tarea deberá ser la inauguración.

Todas las casas deben ser planificadas dentro de su intervalo correspondiente.

```
forall (r in rangoCasas) {
   a[r,mudanza] precedes inauguracion;
   a[r,albanileria].start >= casas[r].startMin;
   a[r,mudanza].end <= casas[r].endMax;    };
```

Fin