Constraint Programming con ILOG Solver

Modellare e risolvere problemi con CP

Michele Lombardi <michele.lombardi2@unibo.it>

Cosa ci aspetta

Di cosa parleremo...

- Modellazione e soluzione di problemi con CP
  - Strumenti per modellare (variabili, vincoli)
  - Problematiche di modellazione
  - Ottimizzazione del modello e del metodo di soluzione
- Risolutore: **ILOG solver**

...e come ne parleremo

- Considereremo un unico esempio di riferiento...
- ...e TANTE sue varianti ;-)
- Ad ogni passo sarà introdotto qualcosa di nuovo...
- ...e vi sarà proposto qualche esercizio

Modellare un problema

Modellare un problema con CP

Introduzione a ILOG Concert & Solver

Il nostro esempio

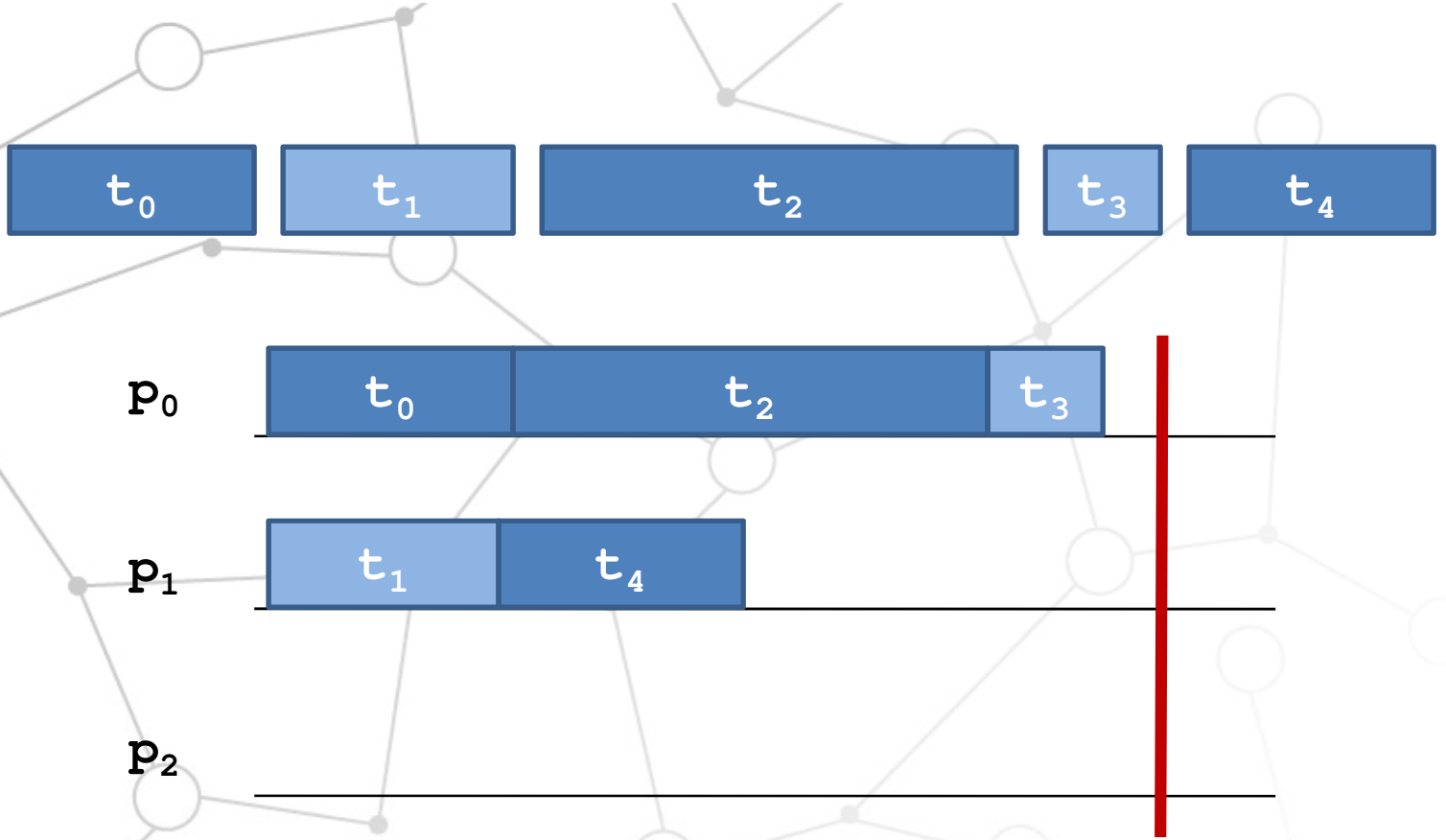
Supponiamo di dover ottimizzare l'esecuzione di tasks su un sistema multiprocessore

Siano:

- $T = \{t_0, t_1, \ldots, t_{n-1}\}$ = insieme degli **n** tasks
- $P = \{p_0, p_1, \ldots, p_{p-1}\}$ = insieme dei **p** processori
- $dur(t_i)$ = durata del task i-mo

> Ogni task esegue su un solo processore
> Su ogni processore i task eseguono in sequenza
> Il tempo totale di esecuzione non deve superare una **deadline**

Obiettivo: usare il minimo numero di processori



Semplice, no?

- Come lo affrontiamo?
- **Vediamo che strumenti abbiamo a disposizione**

Introduzione ad ILOG CP

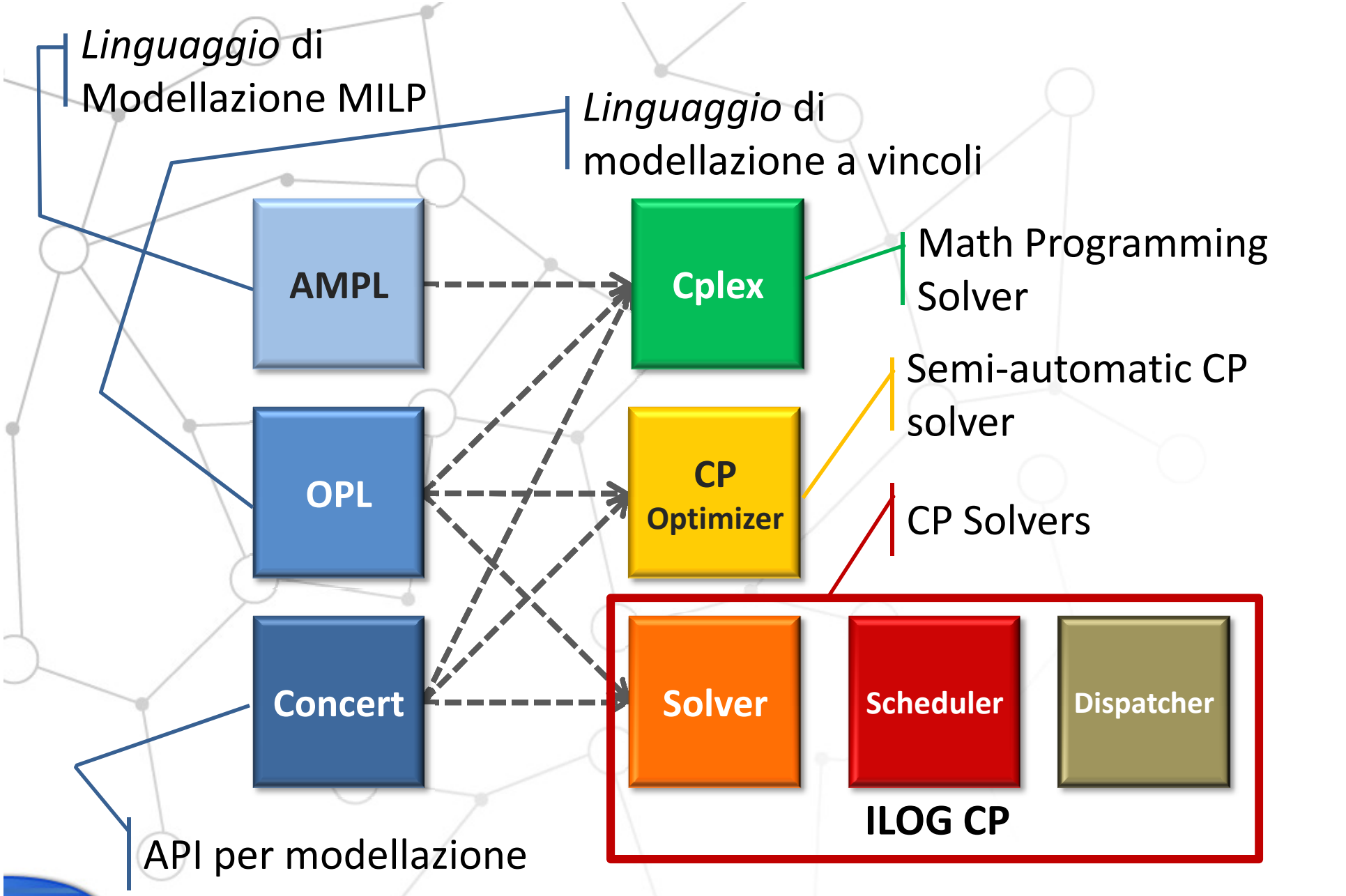
Che cosa è ILOG?

- ILOG è una azienda francese (recentemente acquistata da IBM)
- Produce strumenti per la gestione efficiente di processi di manageriali e per la soluzione di problemi di ottimizzazione

A noi interessano questi

In particolare:

- Strumenti per modellare problemi
- Strumenti per risolvere problemi


Linguaggio di
Modellazione MILP
Linguaggio di
modellazione a vincoli
AMPL
Cplex
Math Programming
Solver
Semi-automatic CP
solver
OPL
CP
Optimizer
CP Solvers
Concert
Solver
Scheduler
Dispatcher
ILOG CP
API per modellazione

In pratica...

- AMPL e OPL sono linguaggi per descrivere modelli
- **Concert è una libreria in C++**
- Fornisce classi e funzioni per costruire modelli

Per esempio:

- `class IloModel` `modello`
- `class IloIntVar` `variabile intera`
- `class IloNumVar` `variabile reale`
- `class IloBoolVar` `variabile logica`
- `class IloConstraint` `vincolo`
- `...`

- **Solver è un risolutore (sempre in C++)** che dato un modello (ex. costruito in concert) trova una soluzione

Allora da dove partiamo?

Da carta e penna

Un primo modello

One dimensional bin packing:

- **Variabili:**

$$y_i = \begin{cases} 1 & se\ p_i\ è\ usato \\ 0 & altrimenti \end{cases} \qquad x_{ij} = \begin{cases} 1 & se\ t_i\ è\ su\ p_i \\ 0 & altrimenti \end{cases}$$

- **Vincoli:**

> Ogni task esegue su un solo processore

$$\forall j \quad \sum_{i=0}^{p-1} x_{ij} = 1$$

> Il tempo totale di esecuzione non deve superare una **deadline**

$$\forall i \quad \sum_{j=0}^{n-1} dur(t_j) x_{ij} \leq deadline \cdot y_i$$

Nel complesso:

obiettivo: $\min z = \sum_{i=0}^{p-1} y_i$

soggetto a: $\forall j \quad \sum_{i=0}^{p-1} x_{ij} = 1$

$\forall i \quad \sum_{j=0}^{n-1} dur(t_j) x_{ij} \leq deadline \cdot y_i$

$y_i, x_{ij} \in \{0,1\}$

Formulato il modello su carta, possiamo “costruirlo” usando Concert

Qualche richiamo di C++

Dal sito del corso, scaricate i files:

- base.cpp
- makefile

Sequenza di build in C++

1. Pre-processing:
   - Sostituzione di testo (“MACRO” – direttiva #define)
   - Inclusione di files (direttiva #include)
2. Compilazione
   - Un file alla volta
   - Produce dei file oggetto (“.o”)
3. Linking

Qualche richiamo di C++

```
#include <ilconcert/ilomodel.h>

ILOSTLBEGIN

int main(int argc, char** argv){
   IloEnv env;

   IloModel model(env);

   cout << "...." << endl;

}
```

Include il file specificato

MACRO: testo sostituito dal pre-processore

Istanza di una classe (allocata nel record di attivazione)

Istanza di una classe, ottenuta mediante un costruttore specifico

Variabile (istanza di un **output stream**)

Operatore di output

Variabile: carattere di fine linea

Struttura di un programma ILOG

```
#include <ilconcert/ilomodel.h>
#include <ilsolver/ilosolver.h>

ILOSTLBEGIN

int main(int argc, char** argv){
   IloEnv env;
   IloModel model(env);
   <inizializzazione modello>
   <invocazione del solver>
   <output>
}
```

Per usare concert e solver

MACRO: definisce il namespace std

IloEnv: gestore della memoria, fa da contenitore per il modello

Classe che rappresenta un modello

Variabili e vincoli vengono "aggiunti" al modello

Costruzione del modello

- Per inizializzare un modello innanzitutto vanno definite le sue **variabili**:

```
IloIntVarArray Y(env, nproc, 0, 1);

IloArray<IloIntVarArray> X(env, nproc);
for(int i = 0; i < nproc; i++){
  X[i] = IloIntVarArray(env, ntask, 0, 1);
}
```

Array di arrays

Array di variabili intere

Parametri di `IloIntVarArray`:

- `IloEnv`: gestore della memoria
- `IloInt`: dimensione
- `IloInt`: lower bound del dominio
- `IloInt`: upper bound del dominio

Costruzione del modello

- Poi vanno specificati i **vincoli**

$$\forall j \quad \sum_{i=0}^{p-1} x_{ij} = 1$$

```
for(int j = 0; j < ntask; j++){
  IloIntExpr sum(env);
  for(int i = 0; i < nproc; i++){
    sum += X[i][j];
  }
  model.add(sum == 1);
}
```

Costruisce una **espressione** vuota...

...che può essere estesa con i normali operatori aritmetici!

I vincoli vanno aggiunti al modello (vengono aggiunte anche le variabili su cui sono definiti)

L'operatore di confronto ("==") tra espressioni restituisce un **vincolo**

Costruzione del modello

$$\forall i \quad \sum_{j=0}^{n-1} dur(t_j)x_{ij} \leq deadline \cdot y_i$$

```
for(int i = 0; i < nproc; i++){
   IloIntExpr sum(env);
   for(int j = 0; j < ntask; j++){
      sum += durations[j]*X[i][j];
   }
   model.add(sum <= deadline*Y[i]);
}
```

Costruzione del modello

- In qualunque punto (dopo la definizione delle variabili) si può specificare una **funzione obiettivo**

$$\min z = \sum_{i=0}^{p-1} y_i$$

```
model.add(IloMinimize(env, IloSum(Y)));
```

Indica che la soluzione deve minimizzare l'espressione specificata

Un modo compatto per costruire una espressione di somma (`Y` è un array)

Utilizzo del solver

```
<inizializzazione del modello>
IloSolver solver(env);
solver.extract(model);
solver.solve();
<output>
```

Costruisce un solver (classe IloSolver)

Trova la soluzione ottima (se esiste)

"estrae" il modello

- **Prima di iniziare la ricerca di una soluzione il modello deve essere convertito nel formato interno del solver**
- Questa operazione si chiama "estrazione"
- Classi concert: `Ilo...` (ex `IloIntVar`)
- Classi solver: `Ilc...` (ex. `IlcIntVar`)

Input & Output

```
int ntask = 10;
int nproc = 6;
int durations[] = {3, 4, 7, 2, 2, 8, 7, 10, 8, 9};
int deadline = 16;
```

```
Number of fails                 : 9282
Number of choice points         : 9312
...
Running time since creation     : 0.2

 Y[0]:0
 Y[1]:0
 Y[2]:1
 Y[3]:1
 Y[4]:1
 Y[5]:1
```

Stili di modellazione

Modelli alternativi

Un altro modello

- CP ha un linguaggio di modellazione molto “ricco” (molti tipi di vincoli)
- Questo permette di migliorare le performance adottando **stili di modellazione differenti**

Esempio:
Cambiamo le variabili!

$x_j = i \quad se\ t_i\ è\ su\ p_i$

Solo una variabile per task

- In questo modo ogni task esegue per forza su un solo processore

Esercizio

Provate a formulare un modello alternativo

Variabili:

$$x_j \in \{0 \ldots n_p\}$$

Vincoli:

- La deadline deve essere rispettata:

- Obiettivo: minimizzare il numero di processori utilizzati