Modeling Languages

Yves Deville\textsuperscript{1}, Christine Solnon\textsuperscript{2}

(1) UCLouvain, Belgium
(2) University of Lyon, France

SLS 2009

In collaboration with Vianney le Cl\'ement\textsuperscript{1}, Jean-No\'el Monette\textsuperscript{1}, Pascal Van Hentenryck (Brown Univ.)
The context of SLS 2009

This tutorial has close relationships with the first tutorial *Computer-assisted design of high-performance algorithms* by Holger Hoos

**Similar objectives**

- Help the user to design an efficient algorithm
- The user focuses on higher level design issues

**The main differences**

- Offering a *modeling language* to the user to design problems
- Dedicated to an application domain
- Focus on *Constraint-Based approaches*
- Algorithm synthesis based on the *structure of the problem*
The context of SLS 2009

This tutorial has close relationships with the first tutorial *Computer-assisted design of high-performance algorithms* by Holger Hoos

**Similar objectives**
- Help the user to design an efficient algorithm
- The user focusses on higher level design issues

**The main differences**
- Offering a *modeling language* to the user to design problems
- Dedicated to an application domain
- Focus on *Constraint-Based approaches*
- Algorithm synthesis based on the *structure of the problem*
Overview

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Constraint Satisfaction Problems

CSP

- A set of variables, defined over domains
- A set of constraints over the variables
- A solution is an assignment of values to the variables which satisfies all the constraints
- Possibility to add an objective function
Example: N-Queens

The problem

How to place 8 queens on a 8x8 board such that they do not attack each other

Variables

\[ X_i \in \{1, \ldots, 8\} : \text{position (column) of the queen on row } i \]

Constraints

- Two queens cannot be on the same column: \( X_i \neq X_j \)
- Two queens cannot be on the same diagonal:
  \[ |X_i - X_j| \neq |i - j| \]
Example: Bin Packing

The problem

Given a list of objects of size $w_1, \ldots, w_n$, $m$ bins of capacity $W$
Assign each object to a bin, such that
- the capacity of the bins are not exceeded
- the number of used bins is minimal

\[ \text{Example: Bin Packing} \]
Solving problems modeled with constraints

Program = Model + Search

**Modeling**
- Model the problem as a CSP
  - Define the constrained variables
  - Specify constraints expressing relations between objects
  - Specify the objective function (optimization problem)

**Search**
- Design the search component
- Depend on the chosen approach
  - Local Search (incomplete, perturbative)
  - ACO (incomplete, constructive)
  - Constraint Programming (complete, constructive)
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Comet: a language for Constraint-Based Programming

The Comet paradigm

Comet program = Model + Search
Models are expressed by means of constraints

Comet approaches

Comet includes

- Constraint-Based Local Search
- Constraint Programming
- MIP

The Comet language is freely available for academics.

Comet is distributed by Dynadec.
Constraint-Based Approaches

Objectives
Modeling Language for Graph Matching
Modeling Language for Scheduling
Conclusion

CP versus CBLS with Comet

\[ x < y \]

**CP search**

- **Constraints → propagators**
- **Complete**

- \( x = \{1, 2, 3\} \)
- \( y = \{1, 2, 3\} \)
- \( x = 3 \)
- \( y = \{1, 2, 3\} \)

- \( x = 3 \)
- \( y = \{1, 2, 3\} \)

- **failure**

- \( x = \{1, 2\} \)
- \( y = \{1, 2, 3\} \)

- \( x = 3 \)
- \( y = \{1, 2, 3\} \)

- **...**

**Local search (LS)**

- **Constraints → violations**
- **Quickly find good solution**

- \( x = 7 \)
- \( y = 3 \)

- assign 5 to \( x \)
- \( x = 5 \)
- \( y = 3 \)

- swap \( x \) and \( y \)
- \( x = 3 \)
- \( y = 5 \)

- \( x = 3 \) viol. = 3

- \( x = 3 \) viol. = 0
## Constraint Programming with Comet

### The CP search

**Branch & Propagate**

### Branching

- Decompose into subproblems (e.g., giving a value for a variable)
- Automatic support for backtracking

### Propagation

- Reduction of the search space
- Find an equivalent CSP with *smaller domains*
- Based on consistency techniques
CBLS with Comet

Constraints
- Measure of their violation
- Differentiable objects: show how much a local move affects the violations

Objective function
- Measure of its value
- Differentiable object: show how much a local move affects the objective

Invariant
Maintain an expression incrementally
N-Queens : Comet / CP

```
import cotfd;
Solver<CP> cp();
int n = 8;
range S = 1..n;
var<CP>{int} q[i in S](cp,S);
solve<cp> {
    cp.post(alldifferent(q));
    cp.post(alldifferent(all(i in S) q[i] + i));
    cp.post(alldifferent(all(i in S) q[i] - i));
} using {
    forall(i in S : !q[i].bound())
        by (q[i].getSize()) {
            tryall<cp>(v in S : q[i].memberOf(v))
            label(q[i],v);
        }
    }
cout << q << endl;
```
N-Queens: Comet / CP
N-Queens: Comet / CBLS

```c
import cotls;
Solver<LS> ls();
int n = 8;
range S = 1..n;
var{int} q[i in S](ls,S) := i;
ConstraintSystem<LS> CS(ls);
    CS.post(allDifferent(q));
    CS.post(allDifferent(all(i in S) q[i] + i));
    CS.post(allDifferent(all(i in S) q[i] - i));
l.s.close();
int it = 0;
while (CS.violations() > 0 && it < 50*n) {
    selectMax(i in S)(CS.violations(q[i]))
    selectMin(v in S)(CS.getAssignDelta(q[i],v))
        q[i] := v;
    it++;
}
cout << q << endl;
```
N-Queens : Comet / CBLS
Bin-packing data in **Comet**

```
1  int n = ...; // number of bins
2  int m = ...; // number of objects
3  int W = ...; // capacity
4  range Rbin = 1..n; // range of the bins
5  range Robj = 1..m; // range of the objects
6  range RW = 0..W; // range of the capacity
7  int w[Robj] = ...; // size of each object
```
Constraint-Based Approaches

Objectives

Modeling Language for Graph Matching

Modeling Language for Scheduling

Conclusion

CP model in **Comet**

```cpp
1 import cotfd;
2
3 Solver<CP> cp();
4 var<CP>{int} b[Obj](cp,Rbin); // bin assigned to each object
5 var<CP>{int} load[Rbin](cp,RW); // load of the bins
6 cp.close();

7 minimize<cp>
8 max(obj in Obj) b[obj]
9 subject to {
10   forall(j in Rbin)
11     cp.post( load[j] == sum(i in Obj) (b[i] == j)*w[i] );
12     cp.post( sum(j in Rbin) load[j] == sum(i in Obj) w[i] );
13   }
14 using {
15   label(b);
16   }
```
import cotls;

Solver<LS> ls();

var{int} b[Robj](ls,Rbin);

var{int} load[bin in Rbin](ls,RW)

    <- sum(obj in Robj) (b[obj] == bin) * w[obj];

Function<LS> objective = MinBinObjective(ls, b, load, w, W);

ls.close();

// Initialization

int curBin = Rbin.getLow();

forall (obj in Robj) {
    if (curBin < Rbin.getUp() && load[curBin] + w[obj] > W)
        curBin++;
    b[obj] := curBin;
}

// Search
int iter = 0;
while (iter < 1000) {
    iter++;
    selectMin(obj in Robj, bin in Rbin : load[bin] + w[obj] <= W)
        (objective.getAssignDelta(b[obj], bin)) {
            b[obj] := bin;
        }
}
Outline

1 Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2 Objectives

3 Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4 Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5 Conclusion
The holy grail of constraint programming...

The user states the problem by means of constraints
The computer solves it thanks to embedded solvers

...faced to the reality of NP-hardness

The user often has to help the computer:

- Choose the most appropriate search paradigm
  - CP when constraints are tight enough to prune efficiently
  - CBLS for looser constraints and/or optimization

- Design the “right” model that leads to an efficient search
  - CP: add redundant constraints
  - CBLS: choose appropriate invariants

- “Program” the search
  - CP: design ordering heuristics
  - CBLS: neighborhoods and strategies for escaping local optima
The holy grail of constraint programming...

The user states the problem by means of constraints
The computer solves it thanks to embedded solvers

...faced to the reality of NP-hardness

The user often has to help the computer:

- Choose the most appropriate search paradigm
  - CP when constraints are tight enough to prune efficiently
  - CBLS for looser constraints and/or optimization
- Design the “right” model that leads to an efficient search
  - CP: add redundant constraints
  - CBLS: choose appropriate invariants
- “Program” the search
  - CP: design ordering heuristics
  - CBLS: neighborhoods and strategies for escaping local optima
Generic *versus* dedicated approaches

**Dedicated approaches (operation research)**

Design a customized algorithm to solve a problem: very efficient... but cannot be used to solve a slightly different problem.

**Constraint-based approaches**

Design a solver to solve all CSPs: very generic... but not always efficient (unless the user helps the solver !)
## Generic *versus* dedicated approaches

### Dedicated approaches (operation research)
Design a customized algorithm to solve a problem: very efficient... but cannot be used to solve a slightly different problem.

### Modeling languages
Focus on an application domain and design:
- a high level modeling language for this domain
- a synthesizer that generates an appropriate solver from the model

### Constraint-based approaches
Design a solver to solve all CSPs: very generic... but not always efficient (unless the user helps the solver !)
Our goal

Bridge the gap between high-level modeling and efficient solving:

**High-level modeling**

- High level objects and constraints
- Related to an application domain
  (graph matching, scheduling, routing, line balancing, ...)

⇒ declarative modeling of problems within this domain

**Efficient solving**

Synthesize the appropriate search strategy:

- analyze the structure of the model
- automatically generate a customized solver
  ⇒ reuse state-of-the-art approaches, combine them, ...
Characteristics of our approach

Written in Comet

- Supports both CP, CBLS, and MIP
- Object-Oriented

Easy to use as a black-box

- Easy modeling of classical problems
- May be used to model new problems
  \(\Rightarrow\) Handling specificities through additional constraints

The box may be opened and is easily extensible

- Add new constraints
- Add new solving algorithms, heuristics
  \(\Rightarrow\) state-of-the-art

\(\Rightarrow\) extend the synthesizer
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Graph matching problems

Why matching graphs?

- Many applications require to measure object similarity
  - Classification, Search by example, Case-based Reasoning, ...
- Graphs are often used to model objects
  - Images, Molecules, Documents, Design objects, ...
- Graph similarity is measured by matching their vertices

What is a matching?

A matching of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a relation $m \subseteq V_1 \times V_2$
\[ (u_1, u_2) \in m \Rightarrow \text{vertex } u_1 \text{ is matched to vertex } u_2 \]
Graph matching problems

Why matching graphs?

- Many applications require to measure object similarity
  - Classification, Search by example, Case-based Reasoning, ...
- Graphs are often used to model objects
  - Images, Molecules, Documents, Design objects, ...
- Graph similarity is measured by matching their vertices

What is a matching?

A matching of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a relation $m \subseteq V_1 \times V_2$

- $(u_1, u_2) \in m \Rightarrow$ vertex $u_1$ is matched to vertex $u_2$
Well known examples of graph matching problems

- Graph Isomorphism $\sim$ decide equivalence
- Subgraph Isomorphism $\sim$ decide inclusion
- Maximum common subgraph $\sim$ Intersection
- Graph Edit Distance $\sim$ Best univalent matching
- Extended Graph Edit Distance $\sim$ Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism \( \leadsto \) decide equivalence
- Subgraph Isomorphism \( \leadsto \) decide inclusion
- Maximum common subgraph \( \leadsto \) Intersection
- Graph Edit Distance \( \leadsto \) Best univalent matching
- Extended Graph Edit Distance \( \leadsto \) Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism $\leadsto$ decide equivalence
- Subgraph Isomorphism $\leadsto$ decide inclusion
- Maximum common subgraph $\leadsto$ Intersection
- Graph Edit Distance $\leadsto$ Best univalent matching
- Extended Graph Edit Distance $\leadsto$ Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism \(\sim\) decide equivalence
- Subgraph Isomorphism \(\sim\) decide inclusion
- Maximum common subgraph \(\sim\) Intersection
- Graph Edit Distance \(\sim\) Best univalent matching
- Extended Graph Edit Distance \(\sim\) Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism $\leadsto$ decide equivalence
- Subgraph Isomorphism $\leadsto$ decide inclusion
- Maximum common subgraph $\leadsto$ Intersection
- Graph Edit Distance $\leadsto$ Best univalent matching
- Extended Graph Edit Distance $\leadsto$ Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism $\sim$ decide equivalence
- Subgraph Isomorphism $\sim$ decide inclusion
- Maximum common subgraph $\sim$ Intersection
- Graph Edit Distance $\sim$ Best univalent matching
- Extended Graph Edit Distance $\sim$ Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism $\sim$ decide equivalence
- Subgraph Isomorphism $\leadsto$ decide inclusion
- Maximum common subgraph $\leadsto$ Intersection
- Graph Edit Distance $\leadsto$ Best univalent matching
- Extended Graph Edit Distance $\sim$ Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism $\sim$ decide equivalence
- Subgraph Isomorphism $\sim$ decide inclusion
- Maximum common subgraph $\sim$ Intersection
- Graph Edit Distance $\sim$ Best univalent matching
- Extended Graph Edit Distance $\sim$ Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism $\leadsto$ decide equivalence
- Subgraph Isomorphism $\leadsto$ decide inclusion
- Maximum common subgraph $\leadsto$ Intersection
- Graph Edit Distance $\leadsto$ Best univalent matching
- Extended Graph Edit Distance $\leadsto$ Best multivalent matching
Well known examples of graph matching problems

- Graph Isomorphism $\sim$ decide equivalence
- Subgraph Isomorphism $\sim$ decide inclusion
- Maximum common subgraph $\sim$ Intersection
- Graph Edit Distance $\sim$ Best univalent matching
- Extended Graph Edit Distance $\sim$ Best multivalent matching
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Modeling graph matching by means of constraints

**Constraints on the cardinality of the matching**
- bijective (1,1), injective (1,0..1), univalent (0..1,0..1), or multivalent (0..n,0..n)
  - hard constraints: exact matchings
  - soft constraints: error-tolerant matchings

**Constraints on edges**
- hard constraints: edges must be matched
- soft constraints: maximize the number of matched edges

**Constraints on labels (in case of labeled graphs))**
- hard constraints: matched components must have identical labels
- soft constraints: maximize the similarity of matched component labels
Example 1: Graph isomorphism

- Declare 2 graph objects \(g_1\) and \(g_2\) and a matching \(m\)
  
  ```
  bool[,] adj1 = ...
  bool[,] adj2 = ...
  SimpleGraph<Mod> g1(adj1);
  SimpleGraph<Mod> g2(adj2);
  Matching<Mod> m(g1,g2);
  ```

- Post cardinality constraints on \(m \rightarrow\) bijective matching \((1,1)\)
  
  ```
  m.post(cardMatch(g1.getAllNodes(), 1, 1));
  m.post(cardMatch(g2.getAllNodes(), 1, 1));
  ```

- Post constraints to ensure edge matching
  
  ```
  m.post(matchedToSomeEdges(g1.getAllEdges()));
  m.post(matchedToSomeEdges(g2.getAllEdges()));
  ```

- Ask the synthesizer to create the solver... and search a solution
  
  ```
  m.close();
  DefaultGMSynthesizer synth();
  GMSolution<Mod> sol = synth.solveMatching(m);
  ```
Example 1: Graph isomorphism

- Declare 2 graph objects $g_1$ and $g_2$ and a matching $m$
  
  ```
  bool[,] adj1 = ...  
  bool[,] adj2 = ...  
  SimpleGraph<Mod> g1(adj1);  
  SimpleGraph<Mod> g2(adj2);  
  Matching<Mod> m(g1,g2);
  ```

- Post cardinality constraints on $m \rightsimeq$ bijective matching $(1,1)$
  
  ```
  m.post(cardMatch(g1.getAllNodes(), 1, 1));  
  m.post(cardMatch(g2.getAllNodes(), 1, 1));
  ```

- Post constraints to ensure edge matching
  
  ```
  m.post(matchedToSomeEdges(g1.getAllEdges()));  
  m.post(matchedToSomeEdges(g2.getAllEdges()));
  ```

- Ask the synthesizer to create the solver... and search a solution
  
  ```
  m.close();  
  DefaultGMSynthesizer synth();  
  GMSolution<Mod> sol = synth.solveMatching(m);
  ```
Example 1: Graph isomorphism

- Declare 2 graph objects $g_1$ and $g_2$ and a matching $m$
  ```
  bool[,] adj1 = ...
  bool[,] adj2 = ...
  SimpleGraph<Mod> g1(adj1);
  SimpleGraph<Mod> g2(adj2);
  Matching<Mod> m(g1,g2);
  ```

- Post cardinality constraints on $m \rightsquigarrow$ bijective matching (1,1)
  ```
  m.post(cardMatch(g1.getAllNodes(), 1, 1));
  m.post(cardMatch(g2.getAllNodes(), 1, 1));
  ```

- Post constraints to ensure edge matching
  ```
  m.post(matchedToSomeEdges(g1.getAllEdges()));
  m.post(matchedToSomeEdges(g2.getAllEdges()));
  ```

- Ask the synthesizer to create the solver... and search a solution
  ```
  m.close();
  DefaultGMSynthesizer synth();
  GMSolution<Mod> sol = synth.solveMatching(m);
  ```
Example 1: Graph isomorphism

- Declare 2 graph objects \( g_1 \) and \( g_2 \) and a matching \( m \)
  ```
  bool[,] adj1 = ... 
  bool[,] adj2 = ... 
  SimpleGraph<Mod> g1(adj1); 
  SimpleGraph<Mod> g2(adj2); 
  Matching<Mod> m(g1,g2); 
  ```

- Post cardinality constraints on \( m \) \( \leadsto \) bijective matching \((1,1)\)
  ```
  m.post(cardMatch(g1.getAllNodes(), 1, 1)); 
  m.post(cardMatch(g2.getAllNodes(), 1, 1)); 
  ```

- Post constraints to ensure edge matching
  ```
  m.post(matchedToSomeEdges(g1.getAllEdges())); 
  m.post(matchedToSomeEdges(g2.getAllEdges())); 
  ```

- Ask the synthesizer to create the solver... and search a solution
  ```
  m.close(); 
  DefaultGMSynthesizer synth(); 
  GMSSolution<Mod> sol = synth.solveMatching(m); 
  ```
Example 2: Induced Subgraph Isomorphism

- Declare 2 graph objects $g_1$ and $g_2$ and a matching $m$
  ```cpp
  bool[,] adj1 = ... 
  bool[,] adj2 = ... 
  SimpleGraph<Mod> g1(adj1); 
  SimpleGraph<Mod> g2(adj2); 
  Matching<Mod> m(g1,g2); 
  ```

- Post cardinality constraints on $m \leadsto$ injective matching $(1,0..1)$
  ```cpp
  m.post(cardMatch(g1.getAllNodes(), 1, 1)); 
  m.post(cardMatch(g2.getAllNodes(), 0, 1)); 
  ```

- Post constraints to ensure edges of $G_1$ to be matched
  ```cpp
  m.post(matchedToSomeEdges(g1.getAllEdges())); 
  ```

- Ask the synthesizer to create the solver... and search a solution
  ```cpp
  m.close(); 
  DefaultGMSynthesizer synth(); 
  GMSolution<Mod> sol = synth.solveMatching(m); 
  ```
Example 2: Induced Subgraph Isomorphism

- Declare 2 graph objects \( g_1 \) and \( g_2 \) and a matching \( m \)
  ```
  bool[,] adj1 = ...
  bool[,] adj2 = ...
  SimpleGraph<Mod> g1(adj1);
  SimpleGraph<Mod> g2(adj2);
  Matching<Mod> m(g1,g2);
  ```

- Post cardinality constraints on \( m \) \( \rightsquigarrow \) injective matching \((1,0..1)\)
  ```
  m.post(cardMatch(g1.getAllNodes(), 1, 1));
  m.post(cardMatch(g2.getAllNodes(), 0, 1));
  ```

- Post constraints to ensure edges of \( G_1 \) to be matched
  ```
  m.post(matchedToSomeEdges(g1.getAllEdges()));
  ```

- Ask the synthesizer to create the solver... and search a solution
  ```
  m.close();
  DefaultGMSynthesizer synth();
  GMSolution<Mod> sol = synth.solveMatching(m);
  ```
Example 2: Induced Subgraph Isomorphism

- Declare 2 graph objects $g_1$ and $g_2$ and a matching $m$
  ```
  bool[,] adj1 = ... 
  bool[,] adj2 = ... 
  SimpleGraph<Mod> g1(adj1); 
  SimpleGraph<Mod> g2(adj2); 
  Matching<Mod> m(g1,g2);
  ```

- Post cardinality constraints on $m \sim \rightarrow$ injective matching $(1,0..1)$
  ```
  m.post(cardMatch(g1.getAllNodes(), 1, 1)); 
  m.post(cardMatch(g2.getAllNodes(), 0, 1));
  ```

- Post constraints to ensure edges of $G_1$ to be matched
  ```
  m.post(matchedToSomeEdges(g1.getAllEdges()));
  ```

- Ask the synthesizer to create the solver... and search a solution
  ```
  DefaultGMSynthesizer synth(); 
  GMSolution<Mod> sol = synth.solveMatching(m);
  ```
Example 2: Induced Subgraph Isomorphism

- Declare 2 graph objects $g_1$ and $g_2$ and a matching $m$
  
  ```
  bool[,] adj1 = ... 
  bool[,] adj2 = ... 
  SimpleGraph<Mod> g1(adj1); 
  SimpleGraph<Mod> g2(adj2); 
  Matching<Mod> m(g1,g2); 
  ```

- Post cardinality constraints on $m \leadsto$ injective matching $(1,0..1)$
  
  ```
  m.post(cardMatch(g1.getAllNodes(), 1, 1)); 
  m.post(cardMatch(g2.getAllNodes(), 0, 1)); 
  ```

- Post constraints to ensure edges of $G_1$ to be matched
  
  ```
  m.post(matchedToSomeEdges(g1.getAllEdges())); 
  ```

- Ask the synthesizer to create the solver... and search a solution
  
  ```
  m.close(); 
  DefaultGMSynthesizer synth(); 
  GMSolution<Mod> sol = synth.solveMatching(m); 
  ```
Example 3: Largest Common Induced Subgraph

- Declare 2 graph objects g1 and g2 and a matching m
  ```
  bool[,] adj1 = ... 
  bool[,] adj2 = ... 
  SimpleGraph<Mod> g1(adj1); 
  SimpleGraph<Mod> g2(adj2); 
  Matching<Mod> m(g1,g2); 
  ```
- Post cardinality constraints on m ~ univalent matching (0..1, 0..1)
  ```
  m.post(cardMatch(g1.getAllNodes(), 0, 1)); 
  m.post(cardMatch(g2.getAllNodes(), 0, 1)); 
  ```
- Post a soft constraint to maximize the number of matched vertices
  ```
  m.softpost(minMatch(g1.getAllNodes(), 1), 1) 
  ```
- Post constraints to ensure edge matching
  ```
  m.post(matchedToAllEdges(g1.getAllEdges())); 
  m.post(matchedToAllEdges(g2.getAllEdges())); 
  ```
- Ask the synthesizer to create the solver... and search a solution
  ```
  m.close(); DefaultGMSynthesizer synth(); 
  GMSolution<Mod> sol = synth.solveMatching(m); 
  ```
Example 3: Largest Common Induced Subgraph

- Declare 2 graph objects $g_1$ and $g_2$ and a matching $m$
  
  ```
  bool[,] adj1 = ...
  bool[,] adj2 = ...
  SimpleGraph<Mod> g1(adj1);
  SimpleGraph<Mod> g2(adj2);
  Matching<Mod> m(g1,g2);
  ```

- Post cardinality constraints on $m \rightsquigarrow$ univalent matching (0..1, 0..1)
  
  ```
  m.post(cardMatch(g1.getAllNodes(), 0, 1));
  m.post(cardMatch(g2.getAllNodes(), 0, 1));
  ```

- Post a soft constraint to maximize the number of matched vertices
  
  ```
  m.softpost(minMatch(g1.getAllNodes(), 1), 1)
  ```

- Post constraints to ensure edge matching
  
  ```
  m.post(matchedToAllEdges(g1.getAllEdges()));
  m.post(matchedToAllEdges(g2.getAllEdges()));
  ```

- Ask the synthesizer to create the solver... and search a solution
  
  ```
  m.close(); DefaultGMSynthesizer synth();
  GMSolution<Mod> sol = synth.solveMatching(m);
  ```
Example 3: Largest Common Induced Subgraph

- Declare 2 graph objects \( g1 \) and \( g2 \) and a matching \( m \)
  
  ```
  bool[,] adj1 = ... 
  bool[,] adj2 = ... 
  SimpleGraph<Mod> g1(adj1); 
  SimpleGraph<Mod> g2(adj2); 
  Matching<Mod> m(g1,g2); 
  ```

- Post cardinality constraints on \( m \) \( \hookrightarrow \) univalent matching \( (0..1,0..1) \)
  
  ```
  m.post(cardMatch(g1.getAllNodes(), 0, 1)); 
  m.post(cardMatch(g2.getAllNodes(), 0, 1)); 
  ```

- Post a soft constraint to maximize the number of matched vertices
  
  ```
  m.softpost(minMatch(g1.getAllNodes(), 1), 1) 
  ```

- Post constraints to ensure edge matching
  
  ```
  m.post(matchedToAllEdges(g1.getAllEdges())); 
  m.post(matchedToAllEdges(g2.getAllEdges())); 
  ```

- Ask the synthesizer to create the solver... and search a solution
  
  ```
  m.close(); DefaultGMSynthesizer synth(); 
  GMSolution<Mod> sol = synth.solveMatching(m); 
  ```
Example 3: Largest Common Induced Subgraph

- Declare 2 graph objects g1 and g2 and a matching m
  
  ```
  bool[,] adj1 = ...;  
  bool[,] adj2 = ...;  
  SimpleGraph<Mod> g1(adj1);  
  SimpleGraph<Mod> g2(adj2);  
  Matching<Mod> m(g1,g2);  
  ```

- Post cardinality constraints on m \( \mapsto \) univalent matching (0..1,0..1)
  
  ```
  m.post(cardMatch(g1.getAllNodes(), 0, 1));  
  m.post(cardMatch(g2.getAllNodes(), 0, 1));  
  ```

- Post a soft constraint to maximize the number of matched vertices
  
  ```
  m.softpost(minMatch(g1.getAllNodes(), 1), 1)  
  ```

- Post constraints to ensure edge matching
  
  ```
  m.post(matchedToAllEdges(g1.getAllEdges()));  
  m.post(matchedToAllEdges(g2.getAllEdges()));  
  ```

- Ask the synthesizer to create the solver... and search a solution
  
  ```
  m.close(); DefaultGMSynthesizer synth();  
  GMSolution<Mod> sol = synth.solveMatching(m);  
  ```
Example 3: Largest Common Induced Subgraph

- Declare 2 graph objects \( g_1 \) and \( g_2 \) and a matching \( m \)

```cpp
bool[,] adj1 = ...  
bool[,] adj2 = ...  
SimpleGraph<Mod> g1(adj1);  
SimpleGraph<Mod> g2(adj2);  
Matching<Mod> m(g1,g2);
```

- Post cardinality constraints on \( m \) \( \leadsto \) univalent matching \((0..1,0..1)\)

```cpp
m.post(cardMatch(g1.getAllNodes(), 0, 1));  
m.post(cardMatch(g2.getAllNodes(), 0, 1));
```

- Post a soft constraint to maximize the number of matched vertices

```cpp
m.softpost(minMatch(g1.getAllNodes(), 1), 1);
```

- Post constraints to ensure edge matching

```cpp
m.post(matchedToAllEdges(g1.getAllEdges()));  
m.post(matchedToAllEdges(g2.getAllEdges()));
```

- Ask the synthesizer to create the solver... and search a solution

```cpp
m.close(); DefaultGMSynthesizer synth();  
GMSolution<Mod> sol = synth.solveMatching(m);
```
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
### Synthesizing a solver for graph matching problems (1/3)

**Warning:** Ongoing research with a very first prototype

⇝ many improvements are still to be done!

## Canonical form of modeling constraints

Aggregate all modeling constraints of a same type

- **Cardinality** (MinMatch, MaxMatch, CardMatch, ...)
- **Edge matching** (MatchedToSomeEdges, MatchedToAllEdges, ...)
- **Label matching** (MatchAllNodeLabels, MatchAllEdgeLabels, ...)

⇝ Derive characteristics from the canonical model

## Choose a search approach

- **CP** if no soft constraints and \( \text{MaxCard} \leq 1 \) for all vertices of a graph
  
  ⇝ Maintaining Arc Consistency

- **CBLS** otherwise

  ⇝ Tabu search
Synthesizing a solver for graph matching problems (2/3)

Creation of low level variables

Associate a variable with every vertex of both graphs

- Domains are defined wrt cardinality constraints

<table>
<thead>
<tr>
<th>MinMatch</th>
<th>MaxMatch</th>
<th>Type</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>int</td>
<td>$N$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>int</td>
<td>$N \cup {\bot}$</td>
</tr>
<tr>
<td>Otherwise</td>
<td></td>
<td>set</td>
<td>$2^N$</td>
</tr>
</tbody>
</table>

- Ensure symmetry ($X_u$ matched to $v$ $\Rightarrow$ $X_v$ matched to $u$):
  - CP $\leadsto$ Channeling constraints
  - CBLS $\leadsto$ invariants
Synthesizing a solver for graph matching problems (3/3)

Post the canonical constraints

- CP (hard constraints only)
  - Cardinality constraints
    - $\rightsquigarrow$ Partly handled by variable domains
    - $\rightsquigarrow$ Global allDiff for injective and bijective matchings
  - Edge constraints $\rightsquigarrow$ binary constraints
  - Label constraints on nodes $\rightsquigarrow$ variable domains
  - Label constraints on edges $\rightsquigarrow$ binary constraints

- CBLS (hard and soft constraints)
  - Cardinality $\rightsquigarrow$ neighborhood if hard; invariants if soft
  - Edge $\rightsquigarrow$ invariants
  - Node labels $\rightsquigarrow$ neighborhood if hard; invariants if soft
  - Edge labels $\rightsquigarrow$ invariants
Outline

1 Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2 Objectives

3 Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4 Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5 Conclusion
### (Preliminary) Experimental Results (1/2)

\[ SI \rightarrow \text{Subgraph Isomorphism} \]

<table>
<thead>
<tr>
<th>#Nodes</th>
<th>( \text{Synthesizer/CP} ) &amp;</th>
<th>( \text{vf2 [Cordella et al. 99]} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5%</td>
<td>10%</td>
</tr>
<tr>
<td>100</td>
<td>0.8</td>
<td>0.5</td>
</tr>
<tr>
<td>500</td>
<td>19.3</td>
<td>4.7</td>
</tr>
<tr>
<td>1000</td>
<td>30.6</td>
<td>595.8</td>
</tr>
</tbody>
</table>

- Vf2 better for small instances
- Synthesizer outperforms vf2 for larger instances
- Additional constraint improves the search process
(Preliminary) Experimental Results (1/2)

\[
SI \sim \text{Subgraph Isomorphism}
\]
\[
SI^+ \sim \text{Subgraph Isomorphism} + \text{additional distance constraint}
\]

<table>
<thead>
<tr>
<th>#Nodes</th>
<th>( \frac{\text{Synthesizer/CP}}{\text{vf2 [Cordella et al. 99]}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5% 10% 20% 33% 50%</td>
</tr>
<tr>
<td>SI</td>
<td>100 0.8 0.5 0.7 0.1 0.2 0.0 0.0 0.0 2.0 0.0</td>
</tr>
<tr>
<td></td>
<td>500 19.3 4.7 10.5 15.8 30.7 0.1 0.1 246.7 192.3 –</td>
</tr>
<tr>
<td></td>
<td>1000 30.6 595.8 119.0 152.3 – 86.7 – – – –</td>
</tr>
<tr>
<td>(S)</td>
<td>100 0.3 0.1 0.1 0.1 0.2</td>
</tr>
<tr>
<td></td>
<td>500 3.0 4.4 9.5 16.9 28.9</td>
</tr>
<tr>
<td></td>
<td>1000 16.1 47.8 82.5 148.0 –</td>
</tr>
</tbody>
</table>

- Vf2 better for small instances
- Synthesizer outperforms vf2 for larger instances
- Additional constraint improves the search process
(Preliminary) Experimental Results (2/2)

Maximum common subgraph $\leadsto$ CBLS

<table>
<thead>
<tr>
<th>#nodes</th>
<th>time</th>
<th>iterations</th>
<th>edges%</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>8.5 (2.5)</td>
<td>7768.1 (2301.3)</td>
<td>48.3 (1.1)</td>
</tr>
<tr>
<td>50</td>
<td>33.9 (10.7)</td>
<td>8023.8 (2543.3)</td>
<td>40.2 (0.5)</td>
</tr>
<tr>
<td>100</td>
<td>141.5 (46.4)</td>
<td>8398.4 (2755.0)</td>
<td>34.5 (0.2)</td>
</tr>
</tbody>
</table>

- First results to assess feasibility
- Complete approaches cannot handle these instances
- We haven’t (yet) compared these results with other approaches
Further works on modeling for graph matching

- Improve the analysis of the matching characteristics
  - identify sub-problems that are “easy” to solve
- Integrate dedicated filtering algorithms $\leadsto$ CP
  - Iterative partitionning for graph isomorphism (Nauty)
  - Iterative labeling for subgraph iso. (Zampelli et al 2009)
- Integrate reactive search and other meta-heuristics for CBLS
  - Parameter tuning... !
- Combine CP and CBLS
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Scheduling

The goal is to allocate scarce resources to a set of activities over time.

Scheduling is everywhere

- Products Manufacturing
- Construction Planning
- Code Optimization in Compilers
- Project Management (Pharmaceutic Industry, for instance)
- Trains and Airplanes Scheduling
- Closely Related to Timetabling, Vehicle Routing, Planning...
Construction Scheduling
Airport Scheduling
Project Scheduling
Scheduling

There exists a lot of variations

- Models for activities
  - Preemption, Jobs, ...
- Models for resources
  - Cumulative, Machines, Reservoirs, ...
- Constraints
  - Precedences, Max-Slack, ...
- Objective functions
  - Makespan, Sum of Tardiness, ...
Examples of Scheduling Problems

- Job-Shop (Makespan, Tardiness, Earliness-Tardiness)
- Open-Shop
- Cumulative Job-Shop
- RCPSP, RCPSP/\text{max}
- MMRCPSP, MMRCPSP/\text{max}
- Trolley Problem
- MascLib (NCOS and NCGS classes)
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Job-Shop Scheduling Problem
Job-Shop Scheduling Problem

Job-Shop Problem: a Solution
Job-Shop: Objectives

Makespan

Time

0
Job-Shop: Objectives

Sum of Tardiness

Due-date

Time
Job-Shop : Objectives

Sum of Tardiness and Earliness

Due-date

Time
Modeling : Job-Shop

```plaintext
range jobs = 1..nbjobs;
range machines = 0..nbmachines-1;
range tasks = 1..nbjobs*nbmachines;
int proc[ tasks ];
int mach[ tasks ];
int job[ jobs, machines ];

Schedule< Mod > s();
Job< Mod > J[ i in jobs ]( s, "J"+i );
Machine< Mod > M[ i in machines ]( s, "M"+i );
Activity< Mod > A[ i in tasks ]( s, proc[ i ], "A"+i );
forall( i in tasks )
   A[ i ].requires( M[ mach[ i ] ] );
forall( i in jobs )
   J[ i ].containsInSequence(
      all( j in machines ) A[ job[ i, j ] ] );
s.minimizeObj( makespanOf( s ) );
```
Solving

```java
1 GreedyTabuSynthesizer synth();
2 // CPSynthesizer synth();
3 Solution<Mod> sol = synth.solve(s);
4 sol.printSolution();
```

Classifier Set-Up

```java
1 Classifier Set-Up
2 Models[JobShopWithMakespan, CumulativeJobShopWithMakespan, CumulativeJobShop
3 ...
4 ...
5
6 930.000000
7 Time = 12568
```
Modeling: RCPSP

```c
range tasks;
range resources;
int proc[ tasks]; //Processing Times
int cap[ resources]; //Capacities
int succ[ ][ tasks]; //Successors
int req[ tasks, resources]; //Requirements

Schedule< Mod> s();
Activity< Mod> A[ i in tasks](s, proc[ i], "J");
Resource< Mod> R[ i in resources](s, cap[ i], "R");

forall( i in tasks){
  forall( j in succ[ i].getRange())
    A[ i].precedes(A[ succ[ i][ j]]);
  forall( j in resources)
    if(req[ i, j]!=0)A[ i].requires(R[ j], req[ i, j]);
}
s.minimizeObj(Tardiness< Mod>(s, A[ sink], due date)*tardCost);
```
## Available Abstractions

<table>
<thead>
<tr>
<th>Description</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Schedule</td>
</tr>
<tr>
<td>Activities</td>
<td>Activity</td>
</tr>
<tr>
<td></td>
<td>MultiModeActivity</td>
</tr>
<tr>
<td>Jobs</td>
<td>Job</td>
</tr>
<tr>
<td>Resources</td>
<td>Resource</td>
</tr>
<tr>
<td></td>
<td>Machine</td>
</tr>
<tr>
<td></td>
<td>Reservoir</td>
</tr>
<tr>
<td></td>
<td>StateResource</td>
</tr>
<tr>
<td>Objectives</td>
<td>ScheduleObjective</td>
</tr>
<tr>
<td></td>
<td>CompletionTime, PiecewiseLinearFunction</td>
</tr>
<tr>
<td></td>
<td>Tardiness, Earliness, Lateness, UnitCost</td>
</tr>
<tr>
<td></td>
<td>AbsenceCost, AlternativeCost</td>
</tr>
<tr>
<td></td>
<td>MultObjective, ShiftObjective</td>
</tr>
<tr>
<td></td>
<td>SumObjective, MaxObjective</td>
</tr>
</tbody>
</table>
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Context

- **Scheduling** is a large domain of research and application for optimization techniques.
- Among the techniques: Constraint Programming, Local Search, Integer Programming, Genetic Algorithms, Greedy Algorithms
- Most algorithms are specific to a restricted class of problems. A lot of parameters must be tuned.
- It may be hard to recognize problems, find the most appropriate algorithm and code it.
### Available Synthesizers

<table>
<thead>
<tr>
<th>Description</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesizers</td>
<td>ScheduleSynthesizer</td>
</tr>
<tr>
<td></td>
<td>CPSynthesizer</td>
</tr>
<tr>
<td></td>
<td>TSSynthesizer</td>
</tr>
<tr>
<td></td>
<td>SASynthesizer</td>
</tr>
<tr>
<td></td>
<td>GreedySynthesizer</td>
</tr>
<tr>
<td></td>
<td>SequenceSynthesizer</td>
</tr>
<tr>
<td></td>
<td>ScheduleAnimator</td>
</tr>
<tr>
<td>Solutions</td>
<td>Solution</td>
</tr>
</tbody>
</table>
Represented problems

- Job-Shop (Makespan, Tardiness, Earliness-Tardiness)
- Open-Shop
- Cumulative Job-Shop
- RCPSP, RCPSP/max
- MMRCSP, MMRCSP/max
- Trolley Problem
- MascLib (NCOS and NCGS classes)
Internal Representation

- **Canonical**: Allow different models of the same problem to be classified in the same way
- **Homogeneous**: Ease the analysis and the information retrieval
- **Structured**: To keep the structure of the problem also helps in the analysis
Simplification of the precedences

Schedule S

Activity A1
"d1"

Activity A2
"d2"

Job J1

Schedule S

Activity A1
"d1"

Activity A2
"d2"

Job J1
Classification of problems

- Goal: Classify the model in one of the classes of problems.
- Based on characteristics.
- A simple “constraint” imposes a value for the characteristic. Its value for a model can be true or false.
- More complex constraints are build as boolean formulas of constraints (using negation, disjunction and conjunction).
- A class of problem is represented by a boolean formula.
## Model Example: Job-Shop & RCPSP

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Type</th>
<th>JSP</th>
<th>RCPSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Processing Time</td>
<td>boolean</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Fixed Processing Time</td>
<td>boolean</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Preemption Allowed</td>
<td>enum</td>
<td>never</td>
<td>never</td>
</tr>
<tr>
<td>Common Release Dates</td>
<td>boolean</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Common Deadlines</td>
<td>boolean</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Deadlines Exist</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Form of the Precedence Graph</td>
<td>enum</td>
<td>chains</td>
<td>DAG</td>
</tr>
<tr>
<td>Delay between Activities</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>No wait between Activities</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Jobs inside Jobs</td>
<td>boolean</td>
<td>false</td>
<td>–</td>
</tr>
<tr>
<td>Number Of State Resources</td>
<td>integer</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Capacity</td>
<td>integer</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td>All Capacities are Equal</td>
<td>boolean</td>
<td>true</td>
<td>–</td>
</tr>
</tbody>
</table>
## Model Example: Job-Shop & RCPSP

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Type</th>
<th>JSP</th>
<th>RCPSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservoir Consumption</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Reservoir Production</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Setup Times</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Disjunctive Requirements</td>
<td>boolean</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>All Activities in Jobs</td>
<td>boolean</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Nb of Multi-Mode Activities</td>
<td>integer</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sum Of Requirements</td>
<td>integer</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td>Objective Type</td>
<td>enum</td>
<td>minimize</td>
<td>minimize</td>
</tr>
<tr>
<td>Objective Form</td>
<td>enum</td>
<td>maximum</td>
<td>total</td>
</tr>
<tr>
<td>Objective Components</td>
<td>enum</td>
<td>completion time</td>
<td>lateness</td>
</tr>
<tr>
<td>Objective Scope</td>
<td>enum</td>
<td>all activities</td>
<td>one activity</td>
</tr>
<tr>
<td>All Due-Dates are equal</td>
<td>enum</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Synthesis

- In input, we have a model and its classification
- The user (optionally) specifies a technology by choosing a synthesizer
- Each synthesizer associates a solving strategy to problem classes
- The synthesizer instantiates the strategy
Search

- Greedy Search
- Local Search
- Constraint Programming
- Linear Programming
- Large Neighborhood Search
- Hybrids: sequence, parallelization, master-slave combinations...
Aeon is an open system

Extension Mechanisms

- Modelling Abstractions: Requires a lot of work
- Problem Characteristics: Requires to modify several classes
- Problem Classes: Write a XML file
- Synthesizers: Write a subclass of ScheduleSynthesizer
- Strategy: Write a subclass of Strategy
Outline

1. Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2. Objectives

3. Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4. Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5. Conclusion
Experiments

- **Goal**: assess the practibility of the approach
- **Settings**: 3 benchmarks:
  - Job-Shop Problem with makespan minimization (JSP)
  - Open-Shop Problem with makespan minimization (OSP)
  - Job-Shop with weighted tardiness minimization (JSPWT)
- **Compare three synthesizers together with a specific algorithm**:
  - LS (Tabu Search or Simulated Annealing)
  - CP
  - Sequence of LS and CP
  - Reference, a specific algorithm: state of the art algorithm coded in Comet
- **Evaluation**:
  - MRE (Mean Relative Error) = \(100 \times \frac{(UB - Opt)}{Opt}\)
  - Time to best found solution
## Experiments: Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>#Inst.</th>
<th>Average MRE</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ref.</td>
<td>LS</td>
</tr>
<tr>
<td>JSP</td>
<td>78</td>
<td>2.08</td>
<td>2.09</td>
<td>54.40</td>
</tr>
<tr>
<td>OSP</td>
<td>80</td>
<td>1.68</td>
<td>1.70</td>
<td>1.58/0.01</td>
</tr>
<tr>
<td>JSPTW</td>
<td>22</td>
<td>4.28</td>
<td>3.87</td>
<td>97.88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>#Inst.</th>
<th>Average running time to best solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSP</td>
<td>78</td>
<td>2.6</td>
</tr>
<tr>
<td>OSP</td>
<td>80</td>
<td>24.1</td>
</tr>
<tr>
<td>JSPTW</td>
<td>22</td>
<td>24.4</td>
</tr>
</tbody>
</table>
Experiments: Overhead of **AEON**
Outline

1 Constraint-Based Approaches
   - Modeling and Solving Problems with Constraints
   - The Comet Constraint Programming Language

2 Objectives

3 Modeling Language for Graph Matching
   - Graph Matching
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

4 Modeling Language for Scheduling
   - Scheduling
   - Modeling Language
   - Synthesis of Comet Programs
   - Experimental Results

5 Conclusion
Related Work (1/2)

Constraint-Based modeling systems for specific domains

e.g. scheduling **opl**, **ilog Scheduler**, **Comet**

- Feature high-level abstractions
- Map them directly to the structure used in the search
- Still necessary to write one’s own search algorithm

Synthesizing Algorithms from High-Level Models

[Van Hentenryck and Michel, 2007]

- Not specific to a domain application
- Limited to LS
- Analysis of the structure to create neighborhoods and searches
## Related Work (2/2)

### Adaptive Searches
- Self-adapting Large Neighborhood Search [Laborie and Godard, 2007]
- Impact-based search strategies [Refalo, 2004]
- Reactive search, ...

⇝ should be integrated in our systems

### Engineering SLS
- Parameter optimization
- Instance-based algorithm selection
- Algorithm portfolios, ...

⇝ could be integrated in our systems
Further Work (1/2)

From prototype to robust systems

- Better detection of problem properties
- Hybrid approaches; combination of solvers
- Validation through real world applications

Extensions of the systems

- Adaptive search techniques
- Computer-assisted design techniques
Further Work (2/2)

New application domains
- Routing problems
- Graph partitioning
- Line balancing
- ...

Integration of other search paradigms
- Ant Colony Optimization
- Genetic algorithm
- ...

Modeling Languages

Yves Deville\textsuperscript{1}, Christine Solnon\textsuperscript{2}

\begin{itemize}
\item (1) UCLouvain, Belgium
\item (2) University of Lyon, France
\end{itemize}

SLS 2009

In collaboration with Vianney le Clément\textsuperscript{1}, Jean-Noël Monette\textsuperscript{1}, Pascal Van Hentenryck (Brown Univ.)