Constraint Systems

Reified Constraints

An Example Problem

We have to select warehouses for serving customers

An Example Problem

We have to select warehouses for serving customers

- There are N warehouses and M customers
- Each customer i should be served by a single warehouse
- Each customer i has a demand d_i
- Each warehouse j has a limited capacity c_i
- There is a travel cost $t_{i,j}$ for serving customer i from warehouse j

Goal: find the assignment that minimizes the cost

An Example Problem

How do we model the problem?

An Example Problem

How do we model the problem?

A first alternative

- $x_i \in \{0..n-1\}, \quad \forall i = 0..m-1$
- Index = customer
- Value = warehouse

How does it fare with the constraints?

An Example Problem

"Each customer i should be served by a single warehouse"

An Example Problem

"Each customer i should be served by a single warehouse"

- Trivial

"Each warehouse j has a limited capacity c_i"

An Example Problem

"Each customer i should be served by a single warehouse"

- Trivial

"Each warehouse j has a limited capacity c_i"

- Assigning customer i to warehouse j consumes some capacity
- Not easy to model with out current tools!

An Example Problem

We can tackle the problem by changing the representation...

Second alternative: "inverted" approach

An Example Problem

We can tackle the problem by changing the representation...

Second alternative: "inverted" approach

- Index = warehouse, value = customer
- Not so easy!
- One warehouse can serve multiple customers

An Example Problem

We can tackle the problem by changing the representation...

Third alternative: binary model

- One variable for each possible assignment
- $x_{i,j} \in \{0, 1\}, \quad \forall i = 0..m-1, j = 0..n-1$

The typical modeling approach in Integer Linear Programming

How does it fare with the constraints?

An Example Problem

"Each customer i should be served by a single warehouse"

$$\sum_{j=0..n-1} x_{i,j} = 1, \quad \forall i = 0..m-1$$

"Each warehouse j has a limited capacity c_i"

$$\sum_{i=0..m-1} d_i \, x_{i,j} \leq c_j, \quad \forall j = 0..n-1$$

There is a travel cost $t_{i,j}$ for serving customer i from warehouse j

$$\min \; f(x) = \sum_{j=0..n-1} \sum_{i=0..m-1} t_{i,j} \, x_{i,j}$$

Binary Model: PROs and CONs

We managed to model the problem! But:

Binary Model: PROs and CONs

We managed to model the problem! But:

- It's not compact
- Constraint propagation may be weak

Is there another alternative?

Binary Model: PROs and CONs

We manage to model the problem! But:

- It's not compact
- Constraint propagation may be weak

Is there another alternative?

- Yes, we can extend our modeling tools
- HP: let's use our classical $x_i \in \{0..n-1\}$ variables
- We need the ability to:
  - Take into account a demand d_i for warehouse j if $x_i = j$
  - Take into account a cost $t_{i,j}$ for warehouse j if $x_i = j$

Constraints as Expressions

We could achieve both goals by treating constraints as expressions:

$$z = (x_i = j)$$

Intuitively:

- $z = 1$ iff the constraint $x_i = j$ is satisfied
- $z = 0$ iff the constraint is not satisfied

This is actually possible in most constraint solvers

Reified Constraints

A reified constraint is an expression that corresponds to the feasibility state of a constraint

Our notation:

- A constraint that appears as a term in an expression is reified
- A constraint (c) between brackets is reified

A meta-constraint is a constraint over reified constraints

Reified Constraints

A **reified constraint** is an expression that corresponds to the feasibility state of a constraint

Example: warehouse capacity as a meta-constraint:

$$\sum_{i=0..m-1} d_i\,(x_i = j) \leq c_j, \quad \forall j = 0..n-1$$

Reified Constraints

A reified constraint is an expression that corresponds to the feasibility state of a constraint

Example: assignment costs using reified constraints:

$$\min f(x) = \sum_{j=0..n-1} \sum_{i=0..m-1} t_{i,j} \, (x_i = j)$$

Essentially:

- We gain the power to "materialize" binary variables...
- ...whenever they are needed

Reified Constraints

A reified constraint is an expression that corresponds to the feasibility state of a constraint

How does it work in practice?

- We need to define a notion of consistency
- We need a filtering algorithm

Consistency for Reified Constraints

GAC for reified constraints:

The (original) domain $D(c)$ of a reified constraint is always $\{0, 1\}$

- value 1 in $D(c)$ has a support iff c can be feasible
- value 0 in $D(c)$ has a support iff c can be infeasible

Consistency for Reified Constraints

GAC for reified constraints:

The (original) domain $D(c)$ of a reified constraint is always $\{0, 1\}$

- value 1 in $D(c)$ has a support iff c can be feasible
- value 0 in $D(c)$ has a support iff c can be infeasible

Examples:

Consider the constraint $x \leq y$:

- $x \in \{0, 1\}, y \in \{0, 1\} \longrightarrow D(x \leq y) = \{0, 1\}$
- $x \in \{1\}, y \in \{0\} \longrightarrow D(x \leq y) = \{0\}$
- $x \in \{0, 1\}, y \in \{1\} \longrightarrow D(x \leq y) = \{1\}$

Filtering for Reified Constraints

Filtering Rules

Let (c) be the reification of constraint c. Start by filtering c. Then:

- If we have a domain wipeout $\longrightarrow 1 \notin D(c)$
- If c is resolved $\longrightarrow 0 \notin D(c)$

Resolved constraint: A constraint is resolved iff

$$c_j = \prod_{x_i \in X(c_j)} D(x_i)$$

i.e. if all the possible assignments are feasible.

Filtering for Reified Constraints

Filtering Rules

Let (c) be the reification of constraint c. Start by filtering c. Then:

- If we have a domain wipeout $\longrightarrow 1 \notin D(c)$
- If c is resolved $\longrightarrow 0 \notin D(c)$

Some comments:

The first rule is simple to implement

Filtering for Reified Constraints

Filtering Rules

Let (c) be the reification of constraint c. Start by filtering c. Then:

- If we have a domain wipeout $\longrightarrow 1 \notin D(c)$
- If c is resolved $\longrightarrow 0 \notin D(c)$

Some comments:

For the second, we need to check whether c is resolved:

- If we can, then GAC is enforced on (c)
- Otherwise, we have a weaker form of consistency
- Worst case: check feasibility c once all variables are bound

In practice, the approach is typically used for $=, \neq, <, \leq, >, \geq$

A Final Word on Meta-constraints

Meta-constraints are **extremely powerful modeling tools**

However, **they are not always the best choice:**

- They may lead to complicated models
- They may lead to larger models (hence, more filtering time)
- They may lead to weak filtering (same as binary vars.)

And the last point deserves some discussion…

A Final Word on Meta-constraints

Consider the following expression:

$$2\,(x = 0) + 3\,(x = 1)$$

With $x \in \{0, 1\}$

A Final Word on Meta-constraints

Consider the following expression:

$$2\,(x = 0) + 3\,(x = 1)$$

With $x \in \{0, 1\}$

- We have: $D(x = 0) = D(x = 1) = \{0, 1\}$
- Via propagation, we deduce that:

$$lb = 2 \times 0 + 3 \times 0 = 0$$
$$ub = 2 \times 1 + 3 \times 1 = 5$$

- i.e. the bounds are $\mathbf{0}$ and $\mathbf{5}$

But the true bounds are $\mathbf{2}$ and $\mathbf{3}$

In a few chapters we will see how to address this

Constraint Systems

Logical Constraints

An Example Problem *(courtesy of Ines Lynce)*

We need to schedule a meeting:

- John is available on Monday, Wednesday, or Thursday
- Catherine is not available on Wednesday
- Anne is not available on Friday
- Peter is not available neither on Tuesday nor on Thursday

When can the meeting take place?

We can use a logic-based approach...

An Example Problem (courtesy of Ines Lynce)

We need to schedule a meeting:

- John is available on Monday, Wednesday, or Thursday
- Catherine is not available on Wednesday
- Anne is not available on Friday
- Peter is not available neither on Tuesday nor on Thursday

When can the meeting take place?

- Binary variables $M, Tu, W, Th, Fr \in \{0, 1\}$
- A single constraint:

$$(M \vee W \vee Th) \wedge (\neg W) \wedge (\neg F) \wedge (\neg Tu \wedge \neg Th) = 1$$

An Example Problem (courtesy of Ines Lynce)

We need to schedule a meeting:

- John is available on Monday, Wednesday, or Thursday
- Catherine is not available on Wednesday
- Anne is not available on Friday
- Peter is not available neither on Tuesday nor on Thursday

When can the meeting take place?

The only solution is $\mathbf{M = 1, Tu, W, Th, F = 0}$

Boolean Satisfiability Problem (SAT)

Our problem is an instance of the:

Boolean Satisfiability Problem:
determine if a boolean clause is satisfiable

It's a special type of CSP:

- Only logical variables (i.e. $\in \{0, 1\}$)
- A single constraint, in the form "$\textbf{logical expression} = 1$"

Boolean Satisfiability Problem (SAT)

- The SAT problem is simple, but very important
- Many practical applications (mostly in HW/SW verification)
- Dedicated, very efficient solvers

But we can solve a SAT instance in CP, too

- Do we have all tools we need?

Boolean Satisfiability Problem (SAT)

- The SAT problem is simple, but very important
- Many practical applications (mostly in HW/SW verification)
- Dedicated, very efficient solvers

But we can solve a SAT instance in CP, too

- Do we have all tools we need?

Remember a SAT instance is in the form:

$$\text{logical expression} = 1$$

We need support for **logical expressions/constraints**

Logical Expressions/Constraints

All logical expressions can be obtained starting from three operators:

$$\wedge, \vee, \neg$$

Other operators are not strictly necessary, but useful in practice:

$$\Rightarrow, \Leftrightarrow, \oplus (xor)$$

So, **those are the constraints that we should add**

Logical Expressions/Constraints

All logical expressions can be obtained starting from three operators:

$$\wedge, \vee, \neg$$

Other operators are not strictly necessary, but useful in practice:

$$\Rightarrow, \Leftrightarrow, \oplus (xor)$$

So, **those are the constraints that we should add**

- But **we won't!**
- Instead, we will "cheat"...

"Not" Expression/Constraint

Let's consider a "Not" constraint:

$$z = \neg x$$

"Not" Expression/Constraint

Let's consider a "Not" constraint:

$$z = \neg x$$

And the following expression/constraint over binary variables:

$$z = (1 - x)$$

"Not" Expression/Constraint

Let's consider a "Not" constraint:

$$z = \neg x$$

And the following expression/constraint over binary variables:

$$z = (1 - x)$$

They have the same semantic

- $z = 1$ iff $x = 0$
- $z = 0$ iff $x = 1$

"Not" Expression/Constraint

Let's consider a "Not" constraint:

$$z = \neg x$$

And the following expression/constraint over binary variables:

$$z = (1 - x)$$

And we get GAC filtering!

- $0 \in D(z)$ has a support iff $1 \in D(x)$
- $1 \in D(z)$ has a support iff $0 \in D(x)$

The filtering rules for x are analogous

Arithmetic Equivalent Expressions

Take-home message:

- The constraint $z = \neg x$ is not necessary
- We can post instead $z = (1 - x)$

The two are totally equivalent (for binary variables)

- This is what I meant for "cheating"...
- ...Using arithmetic expressions to mimic logical expressions

Let's see if it works with other logical expressions...

"And" Constraints

Let's consider an "and" constraint:

$$z = x \wedge y$$

"And" Constraints

Let's consider an "and" constraint:

$$z = x \wedge y$$

And the arithmetic expression (over binary variables):

$$z = x\,y$$

- Same semantic? Yep
- GAC filtering? Yep

So, we can use the product to model the "and" operator

Alternative: use $z = \min(x, y)$

"Or" Constraints

Let's consider an "or" constraint:

$$z = x \vee y$$

"Or" Constraints

Let's consider an "or" constraint:

$$z = x \lor y$$

And the arithmetic expression (over binary variables):

$$z = \max(x, y)$$

- Same semantic? Yep
- GAC filtering? Yep

So, we can use max to model the "or" operator

Arithmetic Equivalent Expressions

So, we have **an arithmetic equivalent for all basic operators**

We can get the other logical operators by combining the basic ones:

- $x \Rightarrow y$ is equivalent to $\neg x \vee y$
- $x \Leftrightarrow y$ is equivalent to $(x \wedge y) \vee (\neg x \wedge \neg y)$
- $x \oplus y$ is equivalent to $(\neg x \wedge y) \vee (x \wedge \neg y)$

It's bit verbose, though. E.g.:

- $x \Leftrightarrow y$ becomes $\max(x\,y, (1 - x)\,(1 - y))$

Can we find a more compact formulation?

"Implication" Expression/Constraint

Let's consider an "implication" constraint:

$$z = (x \Rightarrow y)$$

"Implication" Expression/Constraint

Let's consider an "implication" constraint:

$$z = (x \Rightarrow y)$$

And the expression (over binary variables):

$$z = (x \leq y)$$

It's a meta constraint!

- Same semantic? Yep
- GAC filtering? Yep

So, we can use the reified "≤" constraint to model the "⇒" operator

Reified Constraints in Action: Equivalence

Let's consider the equivalence constraint:

$$z = (x \Leftrightarrow y)$$

Reified Constraints in Action: Equivalence

Let's consider the equivalence constraint:

$$z = (x \Leftrightarrow y)$$

And consider the meta-constraint (over binary variables):

$$z = (x = y)$$

- Same semantic? Yep
- GAC filtering? Yep

So, we can use the reified "=" constraint to model the "⇔" operator

Reified Constraints in Action: Xor

Let's consider the exclusive-or constraint:

$$z = (x \oplus y)$$

Reified Constraints in Action: Xor

Let's consider the exclusive-or constraint:

$$z = (x \oplus y)$$

And consider the meta-constraint (over binary variables):

$$z = (x \neq y)$$

- Same semantic? Yep
- GAC filtering? Yep

So, we can use the reified "$\neq$" constraint to model the "$\oplus$" operator

Meta-constraints and Logical Constraints

- We have used reified constraints to <u>encode</u> $\Rightarrow$, $\Leftrightarrow$, $\oplus$
- But we can also <u>combine</u> reified constraints with $\Rightarrow$, $\Leftrightarrow$, $\oplus$!

Reified constraints shine when used in logical expressions

Meta-constraints and Logical Constraints

Some examples:

"If $x = 1$, then y must be positive"

$$(x = 1) \leq (y > 0)$$

"x and y are either both non-negative or both negative"

$$(x \geq 0) = (y \geq 0)$$

"Variable x is either less then -1 or greater than 1"

$$(x < -1) \neq (x > 1)$$

Meta-constraints and Logical Constraints

By combining reified and logical constraints we can model literally every combinatorial relation

Although this is not necessarily a good idea:

- The usual caveats: complicated and large models
- And weak filtering, of course

And the last point deserves (again) some discussion...

A Final Word on Meta-Constraints

A meta-constraint is in fact a network of constraints:

- **The meta-constraint can be non-GAC**
- **Even if all the individual constraints are GAC**

A Final Word on Meta-Constraints

A meta-constraint is in fact a network of constraints:

- **The meta-constraint can be non-GAC**
- **Even if all the individual constraints are GAC**

Classical example:

$$(x = 1) \neq (x = 2), \quad \text{with } x \in \{0, 1, 2\}$$

- $x = 1$, $x = 2$ and $(x = 1) \neq (x = 2)$ are GAC with domain $\{0, 1\}$
- But $x = 0$ is not feasible

Constraint Systems

A Modeling Exercise

A Production Scheduling Problem

Let's consider a simple production scheduling problem:

- We have a single production line that must process a set O of orders
- Processing an order takes one unit of time (e.g. one day)
- Processing an order consume all our resources for that time unit
- There are precedence constraints $i \prec j$ between some orders
  - The precedences are stored as pairs in a set P
- Each order i has a deadline d_i
- We get a revenue r_i if an order is processed by the deadline
  - For the remaining orders, we do not get anything

Our goal is maximize the revenue

A Production Scheduling Problem

Which variables? Start times!

$$s_i \in \{0..eoh\}$$

With $eoh = |O|$

How do we model the resources?

$$s_i \neq s_j \quad \forall i, j \in O, i < j$$

How do we model the precedences?

$$s_i < s_j \quad \forall (i, j) \in P$$

A Production Scheduling Problem

And what about the revenues?

They define our cost function:

$$\max z = \sum_{i \in O} r_i \,(s_i \leq d_i)$$

- But should consider only the orders processed by the deadline!

A Production Scheduling Problem

And what about the revenues?

They define our cost function:

$$\max z = \textstyle\sum_{i \in O} r_i \, (s_i \leq d_i)$$

- But should consider only the orders processed by the deadline!

So, our full model is:

$$
\begin{aligned}
\max z = & \sum_{i \in O} r_i (s_i \leq d_i) & \\
\text{subject to: } & s_i \neq s_j & \forall i, j \in O, i < j \\
& s_i < s_j & \forall (i, j) \in P \\
& s_i \in \{0..eoh\} & \forall i \in O
\end{aligned}
$$

Constraint Systems

A Modeling Exercise

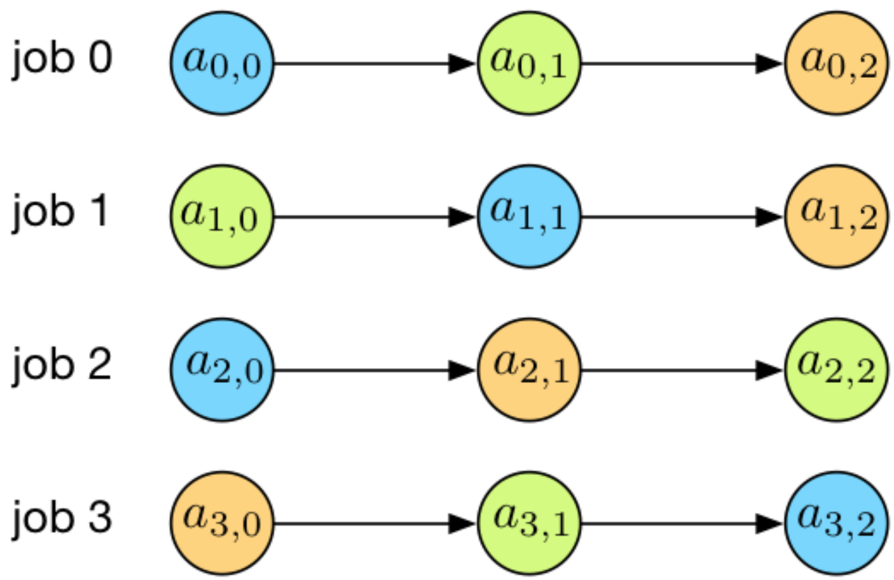
Job Shop Scheduling Problem

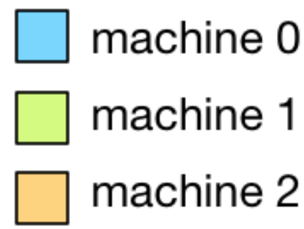
We need to schedule activities in an industrial workshop.

- Activities are organized in jobs
- A job is a set of m activities, to be performed in sequence
- There are n jobs to be scheduled
- The j-th activity in the i-job is called $a_{i,j}$
- Activity $a_{i,j}$ has non-negative duration $d_{i,j}$
- The workshop has m machines
- Each of the m activities requires a different machine
- In particular, $a_{i,j}$ requires machine $m(a_{i,j})$

Objective: complete all jobs as soon as possible

Job Shop Scheduling Problem





$d_{0,0} = 2 \quad d_{0,1} = 4 \quad d_{0,2} = 5$

$d_{1,0} = 3 \quad d_{1,1} = 1 \quad d_{2,2} = 4$

$d_{0,0} = 2 \quad d_{1,1} = 3 \quad d_{2,2} = 2$

$d_{0,0} = 5 \quad d_{1,1} = 2 \quad d_{2,2} = 3$

How do we model the problem?

Job Shop Scheduling Problem

Which variables? Start times!

Job Shop Scheduling Problem

Which variables? Start times!

$$s_{i,j} \in \{0..eoh\}, \quad \forall i = 0..n-1, j = 0..m-1$$

With $eoh = \sum_{\substack{i=0..n-1 \\ j=0..m-1}} d_{i,j}$

Ordering constraints for each job:

$$s_{i,j} + d_{i,j} \leq s_{i,j+1} \quad \forall i = 0..n-1, j = 0..m-2$$

Cost function: minimize the makespan (maximum end time)

$$\min z = \max_{i=0..n-1} \left(s_{i,m-1} + d_{i,m-1} \right)$$

Job Shop Scheduling Problem

The tricky part is handling the resources...

- Activities on the same machine cannot run in parallel
- In our old scheduling problem: we used $\neq$ constraints

Let's parse "cannot run in parallel" for $a_{i,j}$ and $a_{h,k}$

Job Shop Scheduling Problem

The tricky part is handling the resources...

- Activities on the same machine cannot run in parallel
- Old scheduling problem: we used $\neq$ constraints

Let's parse "cannot run in parallel" for $a_{i,j}$ and $a_{h,k}$

- Either $a_{i,j}$ ends before $a_{h,k}$ starts
- or $a_{h,k}$ ends before $a_{i,j}$ starts

This can be stated using meta constraints:

$$(s_{i,j} + d_{i,j} \leq s_{h,k}) \vee (s_{h,k} + d_{h,k} \leq s_{i,j})$$

For all i, j, h, k such that $m(i, j) = m(h, k)$

Job Shop Scheduling Problem

So we get a first model for the job-shop scheduling problem:

$$\min z = \max_{i=0..n-1} \left(s_{i,m-1} + d_{i,m-1}\right)$$

$$\text{subject to: } s_{i,j} + d_{i,j} \leq s_{s,j+1} \qquad \forall i = 0..n-1,\ j = 0..m-2$$

$$(s_{i,j} + d_{i,j} \leq s_{h,k}) \vee (s_{h,k} + d_{h,k} \leq s_{i,j}) \qquad \begin{array}{c} \forall i, j, h, k : i < h \\ m(i,j) = m(h,k) \end{array}$$

$$s_{i,j} \in \{0..eoh\} \qquad \forall i = 0..n-1,\ j = 0..m-1$$

Where the $\vee$ expression will be modeled using a $\mathbf{max}$.

We will return to the JSSP again in the course