

DM841
Discrete Optimization

Part I

Lecture 16

Propagation Events and Implementations

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

1. Generic Rules Iteration

2. Systems

1. Generic Rules Iteration

2. Systems

Algorithms for constraint propagation:

- ▶ scheduling steps of atomic reduction
- ▶ termination criterion: local consistency

- ▶ How to schedule the application of reduction rules to guarantee termination?
- ▶ How to avoid (at low cost) the application of redundant rules?
- ▶ Have all derivations the same result?
- ▶ How can we characterize it?

Propagators

- ▶ Given \mathcal{P} a **reduction rule** is a function f from $\mathcal{S}_{\mathcal{P}}$ to $\mathcal{S}_{\mathcal{P}}$ for all $\mathcal{P}' \in \mathcal{S}_{\mathcal{P}}$, $f(\mathcal{P}') \in \mathcal{S}_{\mathcal{P}}$.
(most cases take care of one a single variable and a single constraints):
- ▶ Given C in \mathcal{P} a **propagator** f for C is a reduction rule from $\mathcal{S}_{\mathcal{P}}$ to $\mathcal{S}_{\mathcal{P}}$ that tightens only domains independently of the constraints other than C .
- ▶ A propagator f can be seen as a function: $f : \mathcal{S}_{ND} \rightarrow \mathcal{S}_{ND}$
- ▶ A propagator f is **correct** for C iff it does not remove any assignment for C : $\{a \in D\} \cap C = \{a \in f(D)\} \cap C$

Systems consider set of propagators to implement a constraint
(However global constraints have a single propagator.)

Example

$$C \equiv x_1 \leq x_2 + 1$$

$$f(D, x_1) = p(D)(x_1) = \{n \in D(X_1) \mid n \leq \max_D\{x_2\} + 1\}$$

$$\text{input}(p) = x_2, \text{output}(p) = x_1$$

Propagators

Properties of propagators:

- ▶ A propagator f is:
 - ▶ **contracting** (or decreasing): $f(\mathcal{P}) \leq \mathcal{P}$
 - ▶ **monotonic** if $\mathcal{P}_1 \leq \mathcal{P}_2 \Rightarrow f(\mathcal{P}_1) \leq f(\mathcal{P}_2)$

 - ▶ A propagator f can be:
 - ▶ **idempotent** if $f(f(\mathcal{P})) = f(\mathcal{P})$ (strong if for all $\mathcal{P} \in \mathcal{S}_{\mathcal{P}}$, weak if for some $\mathcal{P} \in \mathcal{S}_{\mathcal{P}}$)
 - ▶ **commuting** if $fg(\mathcal{P}) = gf(\mathcal{P})$
 - ▶ **subsumed** (or entailed) by \mathcal{P} iff $\forall \mathcal{P}_1 \leq \mathcal{P} : f(\mathcal{P}_1) = \mathcal{P}_1$
- Eg:

$$p(D)(x) = D(x) \cap \{1, 2, 3\}$$

implementing the domain constraint $x \in \{1, 2, 3\}$. After p has been executed once, there is no point to execute p again as for all D'

$$D' \leq p(D) \implies p(D') = D'$$

(particular case when all variables are instantiated)

- ▶ **Iteration:** Let $\mathcal{P} = \langle X, \mathcal{D}, \mathcal{C} \rangle$ and $F = \{f_1, \dots, f_k\}$ a finite set of propagators on $\mathcal{S}_{\mathcal{P}}$. An iteration of F on \mathcal{P} is a sequence $\langle \mathcal{P}_0, \mathcal{P}_1, \dots \rangle$ of elements of $\mathcal{S}_{\mathcal{P}}$ defined by

$$\begin{aligned} \mathcal{P}_0 &= \mathcal{P} \\ &\vdots \\ \mathcal{P}_j &= f_{n_j}(\mathcal{P}_{j-1}) \end{aligned}$$

where $j > 0$ and $n_j \in [1, \dots, k]$.

- ▶ \mathcal{P} is **stable** for F iff $\forall f \in F, f(\mathcal{P}) = \mathcal{P}$
- ▶ There may be several stable \mathcal{P} but if F are monotonic then **unique**
- ▶ Let $\mathcal{P} = \langle X, \mathcal{D}, \mathcal{C} \rangle$ and $F = \{f_1, \dots, f_k\}$. If $\langle \mathcal{P}_0, \mathcal{P}_1, \dots \rangle$ is infinite iteration of F where each $f \in F$ is activated infinitely often then there exists $j \geq 0$ such that \mathcal{P}_j is stable for F ($\equiv j$ is finite!)
- ▶ If \mathcal{P} is stable for F then it is its **weakest simultaneous fixed point** (greatest mutual fixed point of all propagators).
A strongest simultaneous fixed point would be a solution (hence not unique) which would not violate solution preservation

Iteration of Reduction Rules

```
procedure Generic-Iteration( $N, F$ );  
   $G \leftarrow F$ ;  
  while  $G \neq \emptyset$  do  
    select and remove  $g$  from  $G$ ;  
    if  $N \neq g(N)$  then  
      update( $G$ );  
       $N \leftarrow g(N)$ ;  
  
  /* update( $G$ ) adds to  $G$  at least all functions  $f$  in  $F \setminus G$  for which  
   $g(N) \neq f(g(N))$  */
```

If the propagator is contracting then Generic-Iteration terminates.
If propagator is monotonic then the final result does not change with the order in which propagators are applied.

If propagators in addition to monotonic are also idempotent and commutative then:

```
procedure Direct-Iteration( $N, F$ );  
   $G \leftarrow F$ ;  
  while  $G \neq \emptyset$  do  
    select and remove  $g$  from  $G$ ;  
     $N \leftarrow g(N)$ ;
```

Iteration of Reduction Rules

Example

Recall for arc consistency:

Arc Consistency rule 1 (propagator):

$$\frac{\langle C; x \in D(x), y \in D(y) \rangle}{\langle C; x \in D'(x), y \in D(y) \rangle}$$

where $D'(x) := \{a \in D(x) \mid \exists b \in D(y), (a, b) \in C\}$

This can also be written as (\bowtie represents the join):

$$D(x) \leftarrow D(x) \cap \pi_{\{x\}}(\bowtie(C, D(y)))$$

$\forall N_1 = (X, D_1, C) \in \mathcal{P}_{ND}, \forall x_i \in X, \forall c_j \in C, f_{i,j}(N_1) = (X, D'_1, C)$ with

$$D'_1(x_i) = \pi_{\{x_i\}}(c_j \cap \pi_{X(c_j)}(D_1)) \text{ and } D'_1(x_k) = D_1(x_k), \forall k \neq i.$$

Set of propagators $F_{AC} = \{f_{ij} \mid x_i \in X, c_j \in C\}$ all monotonic. \Rightarrow terminates in arc consistency closure, which is fixed point for F_{AC} .

Improvements

Generic iteration is an example of **propagator engine**

What makes it naive?

- ▶ Termination relies on strict contraction
- ▶ We always have to check all propagators for one that can strictly contract

Ideas:

- ▶ Maintain propagators which are known to be at fixpoint
- ▶ Look at the variables that propagators actually compute with
Dependency-directed propagation

Fixpoint knowledge avoids useless execution (idempotence, subsumption)
knowledge provided by propagator

Generic iteration is an example of **propagator engine**

```
propagate( $P_f, P_n, D$ )
1:  $N \leftarrow P_n$ 
2:  $P \leftarrow P_f \cup P_n$ 
3: while  $N \neq \emptyset$  do
4:    $p \leftarrow \text{select}(N)$ 
5:    $N \leftarrow N - \{p\}$ 
6:    $D' \leftarrow p(D)$ 
7:    $M \leftarrow \{x \in \mathcal{V} \mid D(x) \neq D'(x)\}$ 
8:    $N \leftarrow N \cup \{p' \in P \mid \text{input}(p') \cap M \neq \emptyset\}$ 
11:   $D \leftarrow D'$ 
12: return  $D$ 
```

P_f is set of propagators at fixed point (idempotent or subsumed)

Scheduling p : adding a propagator to the set N (not known to be at fixed point). Yet undefined how a propagator is chosen from N

Note: search can be seen as doing incremental propagation

Improvements: Events

Most solvers implement arithmetic-oriented propagators

↪ a reduction of a domain of a variable has different implications depending on the type of reduction

Four types of **Events**:

- ▶ Any or RemValue: when a value v is removed from $D(x_i)$
- ▶ Min or IncMin: when the minimum value of $D(x_i)$ increases
- ▶ Max or DecMax: when the maximum value of $D(x_i)$ decreases
- ▶ Fix or Instantiate: when $D(x_i)$ becomes a singleton

Modified AC3 to handle parameter Mtype (modification type)

```

function Constraint-Propag(in  $X$ : set): Boolean ;
  begin
1    foreach  $c \in C$  do perform init-propag on  $c$  and update  $Q$  with relevant
    events;
2    while  $Q \neq \emptyset$  do
3      select and remove  $(x_i, c, x_j, Mtype)$  from  $Q$ ;
4      if  $Revise(x_i, c, (x_j, Mtype), Changes)$  then
5        if  $D(x_i) = \emptyset$  then return false ;
6        foreach  $c' \in \Gamma^C(x_i), Mtype \in Changes$  do
7          foreach  $x_j \in X(c'), j \neq i$  do  $Q \leftarrow Q \cup \{(x_j, c', x_i, Mtype)\}$ ;
8    return true ;
  end
  /*  $\Gamma^C(x_i)$  is the set of constraints with  $x_i$  in their scheme */

```

The presence of $(x_j, c, x_i, Mtype)$ in Q means that x_j should be revised on c because of an $Mtype$ change in $D(x_i)$.

Process constraint propagation differently according to the type of event

```

function revise(inout  $x_i$ ; in  $c \equiv x_{k_1} \leq x_{k_2}$ ; in  $(x_j, Mtype)$ ; out  $Changes$ ):
  Boolean ;
   $Changes \leftarrow \emptyset$ ;
  switch  $Mtype$  do
    case  $RemValue$ 
      nothing;
    case  $IncMin$ 
      if  $j = k_1$  then remove all  $v < min_D(x_j)$  from  $D(x_i)$ ;
    case  $DecMax$ 
      if  $j = k_2$  then remove all  $v > max_D(x_j)$  from  $D(x_i)$ ;
    case  $Instantiate$ 
      if  $j = k_1$  then remove all  $v < min_D(x_j)$  from  $D(x_i)$ ;
      else remove all  $v > max_D(x_j)$  from  $D(x_i)$ ;
   $Changes \leftarrow$  the types of changes performed on  $D(x_i)$ ;
  
```

Also: for a certain constraint it can be that a given event cannot alter the other variables of the constraint. Hence it makes sense to:

6: **foreach** $c' \in \Gamma_{Mtype}^c(x_j), Mtype \in Changes$ **do** ...

Example. Let $c \equiv x_1 \leq x_2$. The only events that require propagation are IncMin and Instantiate on x_1 , and DecMax and Instantiate on x_2 .

```

3   select and remove  $(x_i, c, x_j, Mtype, \Delta_j)$  from  $Q$ ;
4   if  $\text{Revise}(x_i, c, (x_j, Mtype, \Delta_j), \text{Changes}, \Delta_i)$  then
5       if  $D(x_i) = \emptyset$  then return false;
6       foreach  $c' \in \Gamma_{Mtype}^C(x_i), Mtype \in \text{Changes}$  do
7           foreach  $x_j \in X(c'), j \neq i$  do  $Q \leftarrow Q \cup \{(x_j, c', x_i, Mtype, \Delta_i)\}$ 

```

```

function revise(inout  $x_i$ ; in  $c \equiv x_{k_1} = x_{k_2} + m$ ; in  $(x_j, Mtype, \Delta_j)$ ;
                out  $\text{Changes}$ ; out  $\Delta_i$ ): Boolean ;

```

$\text{Changes} \leftarrow \emptyset$;

switch $Mtype$ do

 case *RemValue*

 if $j = k_1$ then foreach $v \in \Delta_j$ do remove $(v - m)$ from $D(x_i)$;

 else foreach $v \in \Delta_j$ do remove $(v + m)$ from $D(x_i)$;

 case *IncMin*

 if $j = k_1$ then remove all $v < \min_D(x_j) - m$ from $D(x_i)$;

 else remove all $v < \min_D(x_j) + m$ from $D(x_i)$;

 case *DecMax*

 if $j = k_1$ then remove all $v > \max_D(x_j) - m$ from $D(x_i)$;

 else remove all $v > \max_D(x_j) + m$ from $D(x_i)$;

 case *Instantiate*

 if $j = k_1$ then assign $\min_D(x_j) - m$ to x_i ;

 else assign $\min_D(x_j) + m$ to x_i ;

$\text{Changes} \leftarrow$ the types of changes performed;

$\Delta_i \leftarrow$ all values removed from $D(x_i)$;

More Optimization

Priorities

Choose propagator

- ▶ according to cost: cheapest first
- ▶ according to expected impact
- ▶ general (queue): last-in last-out (starvation avoided), first-in first-out

Propagator Rewriting

Another observation:
propagator for

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

and values for x are smaller than for y
Replace by propagator for $y = z$

1. Generic Rules Iteration

2. Systems

- ▶ Detecting failure and entailment
- ▶ Domains: single data structure continuously updated.
constraint store \equiv domain extension \mathcal{D}
- ▶ State restoration
- ▶ Finding dependent propagators (compute events and find propagators)
- ▶ Variables for propagators

- ▶ Events
- ▶ Selecting next propagator

Variable Domains

- ▶ Domain representation
range sequence: $s = \{[n_1, m_1], \dots, [n_k, m_k]\}$ (singly/doubly linked lists)
bit vector
- ▶ Value operations
`x.getmin()`, `x.getmax()`, `x.hasval()`, `x.adjmin(n)`,
`x.adjmax(n)`, `x.excval(n)`

- ▶ Iterators:

```
for (IntVarValues i(x); i(); ++i)
    std::cout << i.val() << ' ';

for (IntVarRanges i(x); i(); ++i)
    std::cout << i.min() << ".." << i.max() << ' ';
```

- ▶ Domain operations
- ▶ Subscriptions (p is executed whenever the domains of one of its variables changes according to an event). Options:
 - ▶ list E_i, p_i pair event propagator that requires execution
 - ▶ a list for each event and one for each propagator
 - ▶ array of propagators partitioned by events

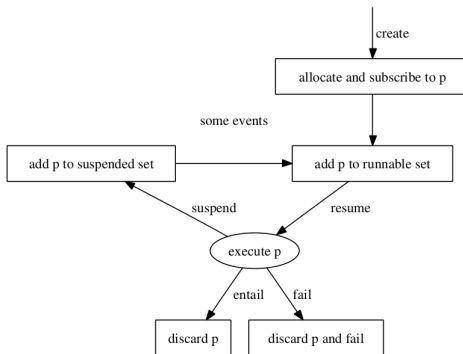
Operations	Range sequence	Bitvector
$x.getmin()$	$O(1)$	$O(1)$
$x.getmax()$	$O(1)$	$O(1)$
$x.hasval(n)$	$O(r)$	$O(1)$
$x.adjmin(n)$	$O(r)$	$O(1)$
$x.adjmax(n)$	$O(r)$	$O(1)$
$x.excv(n)$	$O(r)$	$O(v)$
$i.done()$	$O(1)$	$O(v)$
$i.value()$	$O(1)$	$O(1)$
$i.next()$	$O(1)$	$O(v)$

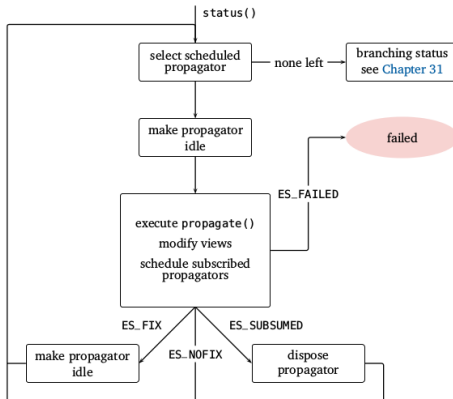
Propagators

Piece of software with some private state that implements a constraint C over some variables or *parameters*

The algorithm implemented is called **filtering algorithm**. It uses value and domain operations and raises events that cause scheduling of other propagators

Life cycle





- ▶ Idempotency: it may be costly and difficult to guarantee. Some propagators return a state:
 - ▶ fixpoint (weak idempotent, ie, with respect to \times rather than for all),
 - ▶ no fixpoint (we do not know),
 - ▶ subsumed (entailed),
 - ▶ failure.

- Apt K.R. (2003). **Principles of Constraint Programming**. Cambridge University Press.
- Barták R. (2001). **Theory and practice of constraint propagation**. In *Proceedings of CPDC2001 Workshop*, pp. 7–14. Gliwice.
- Bessiere C. (2006). **Constraint propagation**. In *Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh, chap. 3. Elsevier. Also as Technical Report LIRMM 06020, March 2006.
- Schulte C. (2011). **Course notes, constraint programming (id2204), vt 2012**. Unpublished.
- Schulte C. and Carlsson M. (2006). **Finite domain constraint programming systems**. In *Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh. Elsevier.