DM826 – Spring 2014

Modeling and Solving Constrained Optimization Problems

Lecture 12

Global Variables

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Resume

- Modelling in IP and CP
- Global constraints
- Local consistency notions
- Filtering algorithms for global constraints
  Scheduling
- Search
- Set variables
- Symmetries

Global Variables

Global variables: complex variable types representing combinatorial structures in which problems find their most natural formulation

Eg:
sets, multisets, strings, functions, graphs
bin packing, set partitioning, mapping problems

We will see:

- Set variables
- Graph variables

Outline

1. Set Variables
2. Graph Variables
3. Float Variables

Finite-Set Variables

- A finite-domain integer variable takes values from a finite set of integers.
- A finite-domain set variable takes values from the power set of a finite set of integers.
  Eg.:
  domain of x is the set of subsets of $\{1, 2, 3\}$:

$$\{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

Finite-Set Variables

Recall the shift-assignment problem

We have a lower and an upper bound on the number of shifts that each worker is to staff (symmetric cardinality constraint)

- one variable for each worker that takes as value the set of shifts covererd by the worker. $\leadsto$ exponential number of values

- set variables with domain $D(x) = [lb(x), ub(x)]$
  $D(x)$ consists of only two sets:
  - $lb(x)$ mandatory elements
  - $ub(x) \setminus lb(x)$ of possible elements

  The value assigned to x should be a set $s(x)$ such that
  $lb \subseteq s(x) \subseteq ub(x)$

In practice good to keep dual views with channelling

Finite-Set Variables

Example:

domain of x is the set of subsets of $\{1, 2, 3\}$:

$$\{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

can be represented in space-efficient way by:

$$[\{\}..\{1, 2, 3\}]$$

The representation is however an approximation!

Example:

domain of x is the set of subsets of $\{1, 2, 3\}$:

$$\{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$$

cannot be captured exactly by an interval. The closest interval would be still:

$$[\{\}..\{1, 2, 3\}]$$

$\leadsto$ we store additionally cardinality bounds: $\#[i..j]$

Set Variables

Definition

set variable is a variable with domain $D(x) = [lb(x), ub(x)]$
$D(x)$ consists of only two sets:

- $lb(x)$ mandatory elements (intersection of all subsets)
- $ub(x) \setminus lb(x)$ of possible elements (union of all subsets)

The value assigned to x must be a set $s(x)$ such that $lb \subseteq s(x) \subseteq ub(x)$

We are not interested in domain consistency but in bound consistency:

Enforcing bound consistency

A bound consistency for a constraint C defined on a set variable x requires that we:

- Remove a value v from $ub(x)$ if there is no solution to C in which $v \in s(x)$.
- Include a value $v \in ub(x)$ in $lb(x)$ if in all solutions to C, $v \in s(x)$.

In Gecode

```
#include <gecode/set.hh>
SetVar(Space home, int glbMin, int glbMax, int lubMin, int lubMax, int cardMin=MIN,
       int cardMax=MAX);
```

```
SetVar A(home, 0, 1, 0, 5, 3, 3);
cout << A: {0,1}..{0..5}#(3) // prints a set variable
```

```
A.glbSize(); 2 // num. of elements in the greatest lower bound
A.glbMin(); 0 // minimum element of greatest lower bound
A.glbMax(); 1 // maximum of greatest lower bound
for (SetVarGlbValues i(x); i(); ++i) cout << i.val() << ' ';
for (SetVarGlbRanges i(x); i(); ++i) cout << i.min() << ".." << i.max();

A.lubSize(): 6 // num. of elements in the least upper bound
A.lubMin(): 0 // minimum element of least upper bound
A.lubMax(): 5 // maximum element of least upper bound
for (SetVarLubValues i(x); i(); ++i) cout << i.val() << ' ';
for (SetVarLubRanges i(x); i(); ++i) cout << i.min() << ".." << i.max();

A.unknownSize(): 4 // num. of unknown elements (elements in lub but not in glb)
for (SetVarUnknownValues i(x); i(); ++i) cout << i.val() << ' ';
for (SetVarUnknownRanges i(x); i(); ++i) cout << i.min() << ".." <<i.max();

A.cardMin(): 3 // cardinality minimum
A.cardMax(): 3 // cardinality maximum
```

In Gecode

```
SetVar(home, IntSet glb, int lubMin, int lubMax, int cardMin=MIN, int cardMax=MAX)
```

```
SetVar A(home, intset(), 0, 5, 0, 4)
```

```
cout << A;
A.glbSize(): 0 // num. of elements in the greatest lower bound
A.glbMin(): 1073741823 // minimum element of greatest lower bound
A.glbMax(): −1073741823 // maximum of greatest lower bound

A.lubSize(): 6 // num. of elements in the least upper bound
A.lubMin(): 0 // minimum element of least upper bound
A.lubMax(): 5 // maximum element of least upper bound

A.unknownSize)(): 6 // num. of unknown elements (elements in lub but not in glb)

A.cardMin(): 0 // cardinality minimum
A.cardMax(): 4 // cardinality maximum
```

In Gecode

```
SetVar(home, int glbMin, int glbMax, IntSet lub, int cardMin=MIN, int cardMax=MAX)
```

```
A.SetVar(1, 3, IntSet({ {1,4}, {8,12} }), 2, 4)
```

```
cout << A;
A.glbSize(A): 3 // num. of elements in the greatest lower bound
A.glbMin(A): 1 // minimum element of greatest lower bound
A.glbMax(A): 3 // maximum of greatest lower bound

A.lubSize(A): 9 // nuA. of elements in the least upper bound
A.lubMin(A): 1 // minimum element of least upper bound
A.lubMax(A): 12 // maximum element of least upper bound


// A.unknownValues(A): [4, 8, 9, 10, 11, 12]
A.unknownSize)(A): 6 // num. of unknown elements (elements in lub but not in glb)
// A.unknownRanges(A): [(4, 4), (8, 12)]

A.cardMin(A): 3 // cardinality minimum
A.cardMax(A): 4 // cardinality maximum
```

Social Golfers Problem

Find a schedule for a golf tournament:

- $g \cdot s$ golfers
- who want to play a tournament in g groups of s golfers each over w weeks
- such that no two golfers play against each other more than once during the tournament.

A solution for the instance $w = 4, g = 3, s = 3$
(players are numbered from 0 to 8)

	Group 0			*Group 1*			*Group 2*		
Week 0	0	1	2	3	4	5	6	7	8
Week 1	0	3	6	1	4	7	2	5	8
Week 2	0	4	8	1	5	6	2	3	7
Week 3	0	5	7	1	3	8	2	4	6

Model

```
w = 4;
g = 3;
s = 3;

golfers = g * s;
Golfer = range(golfers)

m=space()

assign = m.intvars(len(Golfer)*w, intset(range(g)))
assignM = Matrix(len(Golfer), w, assign)

# C1: Each group has exactly groupSize players
for gr in range(g):
  for wk in range(w):
    tmp=m.boolvars(golfers)
    for gl in Golfer:
      m.rel(assignM[gl,wk], IRT_EQ, gr, tmp[gl])
    m.linear(tmp, IRT_EQ, s)

c=[]
for i in range(g):
  c.append(intset(s,s))

for wk in range(w):
  m.count(assignM.col(wk), c, ICL_DOM)


# C2: Each pair of players only meets once
for g1,g2 in combinations(Golfer, 2):
  a=m.boolvars(w)
  for wk1 in range(w):
    m.rel(assignM[g1,wk1],IRT_EQ,assignM[g2,wk1],a[wk1])
  m.linear(a,IRT_EQ,1)

m.branch(assign,INT_VAR_SIZE_MIN,INT_VAL_MIN)
```

In Gecode

Array of set variables:

```
SetVarArray(home, int N, ...)
SetVarArray groups(g∗w, IntSet(), 0, g∗s−1, s, s)
```

size $g \cdot w$, where each group can contain the players $[0..g \cdot s - 1]$ and has cardinality s

```
int w = 4;
int g = 3;
int s = 3;

int golfers = g ∗ s;

SetVarArray groups(g∗w, IntSet(), 0, g∗s−1, s, s)
```

Constraints on FS variables

Domain constraints


```
dom(x, SRT_SUB, 1, 10);
dom(x, SRT_SUP, 1, 3);
dom(y, SRT_DISJ, IntSet(4, 6));
```

```
cardinality(x, 3, 5);
```

Constraints on FS variables
Relation constraints


```
Space.rel(x, SRT_SUB, y)
```

```
Space.rel( x, IRT_GR, y)
```

Constraints on FS variables

Set operations

```
Space.rel(x, SOT_UNION, y, SRT_EQ, z)
```

```
Space.rel(SOT_UNION, x, y)
```

Constraints on FS variables

Element

element(x, y, z)

for an array of set variables or constants *x*,
an integer variable *y*,
and a set variable *z*.

It constrains *z* to be the element of array *x* at index *y* (where the index starts at 0).

Example

```
element([{{1,2,3},{2,3},{3,4}},{{2,3},{2}},{{1,4},{3,4},{3}}], 3,
```

Constraints on FS variables

Set Global Cardinality

bounds the minimum and maximum number of occurrences of an element in an array of set variables:

$$\forall v \in U : l_v \leq |\mathcal{S}_v| \leq u_v$$

where $\mathcal{S}_v$ is the set of set variables that contain the element v, i.e., $\mathcal{S}_v = \{s \in S : v \in s\}$

(not present in gecode)

Constraints on FS variables

Set Global Cardinality

Bessiere et al. [2004]

Table 1. Intersection × Cardinality.

	$\forall i < j \ldots$			
$\forall k \ldots$	$\lvert X_i \cap X_j\rvert = 0$	$\lvert X_i \cap X_j\rvert \leq k$	$\lvert X_i \cap X_j\rvert \geq k$	$\lvert X_i \cap X_j\rvert = k$
-	Disjoint polynomial *decomposable*	$\mathtt{Intersect}_{\leq}$ polynomial *decomposable*	$\mathtt{Intersect}_{\geq}$ polynomial *decomposable*	$\mathtt{Intersect}_{=}$ NP-hard *not decomposable*
$\lvert X_k\rvert > 0$	NEDisjoint polynomial *not decomposable*	$\mathtt{NEIntersect}_{\leq}$ polynomial *decomposable*	$\mathtt{NEIntersect}_{\geq}$ polynomial *decomposable*	$\mathtt{FCIntersect}_{=}$ NP-hard *not decomposable*
$\lvert X_k\rvert = m_k$	FCDisjoint poly on sets, NP-hard on multisets *not decomposable*	$\mathtt{FCIntersect}_{\leq}$ NP-hard *not decomposable*	$\mathtt{FCIntersect}_{\geq}$ NP-hard *not decomposable*	$\mathtt{NEIntersect}_{=}$ NP-hard *not decomposable*

Table 2. Partition + Intersection × Cardinality.

	$\bigcup_i X_i = X \ \wedge \ \forall i < j \ldots$			
$\forall k \ldots$	$\lvert X_i \cap X_j\rvert = 0$	$\lvert X_i \cap X_j\rvert \leq k$	$\lvert X_i \cap X_j\rvert \geq k$	$\lvert X_i \cap X_j\rvert = k$
-	Partition: polynomial *decomposable*	?	?	?
$\lvert X_k\rvert > 0$	NEPartition: polynomial *not decomposable*	?	?	?
$\lvert X_k\rvert = m_k$	FCPartition polynomial on sets, NP-hard on multisets *not decomposable*	?	?	?

Constraints on FS variables
Constraints connecting set and integer variables

the integer variable y is equal to the cardinality of the set variable x.

```
Space.cardinality(x, y);
```

Minimal and maximal elements of a set:

```
Space.min(x, y);
```

Weighted sets: assigns a weight to each possible element of a set variable x, and then constrains an integer variable y to be the sum of the weights of the elements of x

```
e = [6, 1, 3, 4, 5, 7, 9]
w = [6, −1, 4, 1, 1, 3, 3]
Space.weights(e, w, x, y)
```

enforces that x is a subset of $\{1, 3, 4, 5, 7, 9\}$ (the set of elements), and that y is the sum of the weights of the elements in x, where the weight of the element 1 would be -1, the weight of 3 would be 4 and so on.
Eg. Assigning x to the set $\{3, 7, 9\}$ would therefore result in y be set to $4 + 3 + 3 = 10$

Constraints on FS variables

Channeling constraints

X an array of integer variables,
SA an array of set variables

```
channel(home, X, SA)
```

$$X_i = j \iff i \in SA_j \quad 0 \leq i, j < |X|$$

$$SA_i = s \iff \forall j \in s : X_j = i$$

Example

```
SA = [{1,2},{3}]
X = [1,1,2]
```

Constraints on FS variables

Channeling constraints

an array of Boolean variables X
set variable S

```
channel(home, X, S)
```

$$X_i = 1 \iff i \in S \quad 0 \leq i < |X|$$

Example

```
S = {1,2}
X = [1,1,0]
```

Constraints on FS variables
Channeling constraints

An array of integer variables $\vec{x}$
a set variable S:

```
rel(home, SOT_UNION, x, S)
```

constrains S to be the set $\{x_0, \ldots, x_{|x|-1}\}$

```
channelSorted(home, x, S);
```

constrains S to be the set $\{x_0, \ldots, x_{|x|-1}\}$, and the integer variables in $\vec{x}$ to be sorted in increasing order ($x_i < x_{i+1}$ for $0 \leq i < |x|$)

Example

```
rel(home, SOT_UNION, [3,6,2,1], {1,2,3,6})
channelSorted(home, [1,2,3,6], {1,2,3,6})
```

Constraints on FS variables
Channeling constraints

SA_1 and SA_2 two arrays of set variables

```
channel(home, SA1, SA2)
```

$$SA_1[i] = s \iff \forall j \in s : i \in SA_2[j]$$

$$SA_1[i] = \{j \mid SA_2[j] \text{ contains } i\}$$
$$SA_2[j] = \{i \mid SA_1[i] \text{ contains } j\}$$

Example

```
SA1 = [{1,2},{3},{1,2}]
SA2 = [{1,3},{1,3},{2}]
```

Constraints on FS variables
Convexity

set variable S:

```
convex(home, S)
```

The convex hull of a set S is the smallest convex set containing S

```
convex(home, S1, S2)
```

enforces that the set variable $S2$ is the convex hull of the set variable $S1$.

Example

```
S={{1,2,5,6,7},{2,3,4},{3,5}}  convex(S)={2,3,4}
convex({1,2,5,6,7},{1,2,3,4,5,6,7})
```

Constraints on FS variables
Sequence constraints

enforce an order among an array of set variables x

```
Space.sequence(x)
```

sets x being pairwise disjoint, and furthermore $\max(x_i) < \min(x_{i+1})$ for all $0 \leq i < |x| - 1$

```
Space.sequence(x, y)
```

additionally constrains the set variable y to be the union of the x.

Constraints on FS variables
Value precedence constraints

enforce that a value precedes another value in an array of set variables.
x is an array of set variables and both s and t are integers,

```
Space.precede(x, s, t)
```

if there exists j ($0 \leq j < |x|$) such that $s \in x_j$ and $t \in x_j$, then there must exist i with $i < j$ such that $s \in x_i$ and $t \in x_i$

Social golfers

Model with set variables


```
w = 4;
g = 3;
s = 3;

golfers = g * s;
Golfer = range(golfers)

m=space()

groups = m.setvars(g*w, intset(), 0, g*s-1, s, s)
schedule = Matrix(w, g, groups) # is the set of group i in week j

# For each week, the union of all groups must be disjoint and contain all players
allPlayers = m.setvar(0, g*s-1, 0, g*s-1)
for wk in range(w):
  m.rel(SOT_DUNION, schedule.row(wk), allPlayers)


# intersection between groups is at most 1
z=m.setvars(g*w*(g*w-1)/2, intset(), 0, g*s-1, 0, s)
l=0
for i,j in combinations(range(g*w),2):
  m.rel(groups[i], SOT_INTER, groups[j], SRT_EQ, z[l]);
  m.cardinality(z[l], 0, 1)
  l+=1


m.dom(groups[0],SRT_EQ,intset(0,2))

m.branch(groups, SET_VAR_MIN_MIN, SET_VAL_MIN_INC);
```

Set Domain representation

- A finite integer set V can be represented by its characteristic function χ_V:

$$\chi_V : \mathbb{Z} \mapsto \{0, 1\} \text{ where } \chi_v(i) = 1 \text{ iff } i \in V$$

  hence we can use a set of Boolean variables v_i to represent the set V, which correspond to the propositions $v_i \iff i \in V$

  Set bounds propagation is equivalent to performing domain propagation in a naive way on this Boolean representation

- Sets of sets: disjunction of characteristic functions

$$\chi_{\mathcal{V}}(i) \iff \bigvee_{V \in \mathcal{V}} \chi_V(i)$$

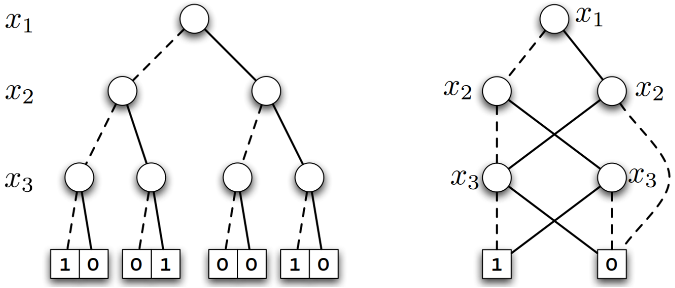
- Consider the domain $\{\{\}, \{1,2\}, \{2,3\}\}$
- Introduce propositional variables x_1, x_2, x_3
- Represent single variable domain as

$$(\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3))$$

- Represent all variable domains as conjunction
- Efficient datastructure: ROBDDs

ROBDD

A Reduced Ordered Binary Decision Diagram (ROBDD) is a compact data structure:
a canonical function representation up to reordering, which permits an efficient implementation of many Boolean function operations.



Implementation in Gecode

- *Set variables in Gecode do not use Reduced Ordered Binary Decision Diagrams (ROBDDs).*
- *A prototype alternative implementation using ROBDDs proved to be a lot slower in many cases (and quite painful to maintain because of additional dependencies).*
- *The current implementation uses range lists (i.e. linked lists of contiguous, sorted, non-overlapping ranges) to store a lower and an upper bound, together with a lower and upper bound on the cardinality.*

Guido Tack

Outline

1. Set Variables
2. Graph Variables
3. Float Variables

Graph Variables

Definition

A graph variable is simply two set variables V and E, with an inherent constraint $E \subseteq V \times V$.

Hence, the domain $D(G) = [lb(G), ub(G)]$ of a graph variable G consists of:

- mandatory vertices and edges $lb(G)$ (the lower bound graph) and
- possible vertices and edges $ub(G) \setminus lb(G)$ (the upper bound graph).

The value assigned to the variable G must be a subgraph of $ub(G)$ and a super graph of the $lb(G)$.

Bound consistency on Graph Variables

Graph variables are convinient for possiblity of efficient filtering algorithms

Example:

`Subgraph(G,S)`

specifies that S is a subgraph of G. Computing bound consistency for the `subgraph` constraint means the following:

1. If $lb(S)$ is not a subgraph of $ub(G)$, the constraint has no solution (consistency check).
2. For each $e \in ub(G) \cap lb(S)$, include e in $lb(G)$.
3. For each $e \in ub(S) \setminus ub(G)$, remove e from $ub(S)$.

Constraint on Graph Variables

- Tree constraint: enforces the partitioning of a digraph into a set of vertex-disjoint anti-arborescences. (see, [Beldiceanu2005])
- Weghted Spanning Tree constraint: given a weighted undirected graph $G = (V, E)$ and a weight K, the constraint enforces that T is a spanning tree of cost at most K (see, [Regin2008,2010] and its application to the TSP [Rousseau2010]).
- Shorter Path constraint: given a weighted directed graph $G = (N, A)$ and a weight K, the constraint specifies that P is a subset of G, corresponding to a path of cost at most K. (see, [Sellmann2003, Gellermann2005])
- (Weighted) Clique Constraint, (see, [Regin2003]).

Outline

1. Set Variables

2. Graph Variables

3. Float Variables

Float Variables

- Floating point values represented as a closed interval of two floating point numbers (short, float number):
  closed interval $[a..b]$ to represent all real numbers n such that $a \leq n \leq b$.
- correct computations: no possible real number is ever excluded due to rounding $\leadsto$ Interval arithmetic
- The float number type `FloatNum` defined as double
- `FloatVar x; x.min(); x.max(); x.tight()` ($a = b$ assigned)
- predefined values `pi_half(), pi(), pi_twice()`
- `x<y` $\leadsto$ `x.max()<y.min()`

function	meaning	default
`max(x,y)`	maximum max(x, y)	✓
`min(x,y)`	minimum max(x, y)	✓
`abs(x)`	absolute value $\|x\|$	✓
`sqrt(x)`	square root $\sqrt{x}$	✓
`sqr(x)`	square x^2	✓
`pow(x,n)`	n-th power x^n	✓
`nroot(x,n)`	n-th root $\sqrt[n]{x}$	✓
`fmod(x,y)`	remainder of x/y	
`exp(x)`	exponential exp(x)	
`log(x)`	natural logarithm log(x)	
`sin(x)`	sine sin(x)	
`cos(x)`	cosine cos(x)	
`tan(x)`	tangent tan(x)	
`asin(x)`	arcsine arcsin(x)	
`acos(x)`	arccosine arccos(x)	
`atan(x)`	arctangent arctan(x)	
`sinh(x)`	hyperbolic sine sinh(x)	
`cosh(x)`	hyperbolic cosine cosh(x)	
`tanh(x)`	hyperbolic tangent tanh(x)	
`asinh(x)`	hyperbolic arcsine arcsinh(x)	
`acosh(x)`	hyperbolic arccosine arccosh(x)	
`atanh(x)`	hyperbolic arctangent arctanh(x)	

Non default functions need recompilation

As for integer variables, the default and copy constructors do not create new variable implementations. Instead, the variable does not refer to any variable implementation (default constructor) or to the same variable implementation (copy constructor). For example in

```
FloatVar x(home, -1.0, 1.0);
FloatVar y(x);
FloatVar z;
z=y;
cout<<x;
```

the variables x, y, and z all refer to the same float variable implementation.

Constraints


```
dom(home, x, -2.0, 12.0);
dom(home, x, d);

rel(home, x, FRT_LE, y);
rel(home, x, FRT_LQ, 4.0);

rel(home, x, FRT_LQ, y);
rel(home, x, FRT_GR, 7.0);

min(home, x, y);

linear(home, a, x, FRT_EQ, c);
linear(home, x, FRT_GR, c);

channel(home, x, y);
```

Interval Analysis

Whereas classical arithmetic defines operations on individual numbers, interval arithmetic defines a set of operations on intervals:

$$T \cdot S = \{x |\text{ there is some } y \text{ in } T\text{, and some } z \text{ in } S\text{, such that } x = y \cdot z\}.$$

The basic operations of interval arithmetic are, for two intervals $[a, b]$ and $[c, d]$ that are subsets of the real line $(-\infty, infty)$:

- $[a, b] + [c, d] = [a + c, b + d]$,
- $[a, b] - [c, d] = [a - d, b - c]$,
- $[a, b] \times [c, d] = [\min(a \times c, a \times d, b \times c, b \times d), \max(a \times c, a \times d, b \times c, b \times d)]$,
- $[a, b]/[c, d] = [\min(a/c, a/d, b/c, b/d), \max(a/c, a/d, b/c, b/d)]$ when 0 is not in $[c, d]$.

Division by an interval containing zero is not defined under the basic interval arithmetic.

The addition and multiplication operations are commutative, associative and sub-distributive: the set $X(Y + Z)$ is a subset of $XY + XZ$.

References

Bessiere C., Hebrard E., Hnich B., and Walsh T. (2004). **Disjoint, partition and intersection constraints for set and multiset variables**. In *Principles and Practice of Constraint Programming – CP 2004*, edited by M. Wallace, vol. 3258 of **Lecture Notes in Computer Science**, pp. 138–152. Springer Berlin / Heidelberg.

Gervet C. (2006). **Constraints over structured domains**. In *Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh, chap. 17, pp. 329–376. Elsevier.

van Hoeve W. and Katriel I. (2006). **Global constraints**. In *Handbook of Constraint Programming*, chap. 6. Elsevier.