CONSTRAINT PROGRAMMING

Introduction

- Disadvantages of SAT solvers:
  - The range of problems that can be solved is limited
    - integer variables can not be represented easily and efficiently
    - not every constraint can easily and efficiently be rewritten in CNF:
      - numerical constraints $x_1 + x_2 + \cdots + x_n \geq 4$
      - graph constraints
        (“from node *x* node *y* can be reached”, “the shortest path from node *x* to node *y* may not be longer than *a*”)
    - dealing with optimization problems is not straightforward
  - The specification language is not very simple to use

Constraint Programming

- **Constraint programming:** a programming paradigm in which a problem is specified declaratively in terms of high-level constraints, and solvers find solutions

  "Constraint programming =

  Model (by user)

  +

  Search (by solver)"

Non-boolean Variables & High-level Constraints

- variables

  $E_{11} \ldots E_{99}$

- variables have domains

  $E_{xy} = \{1 \ldots 9\}$

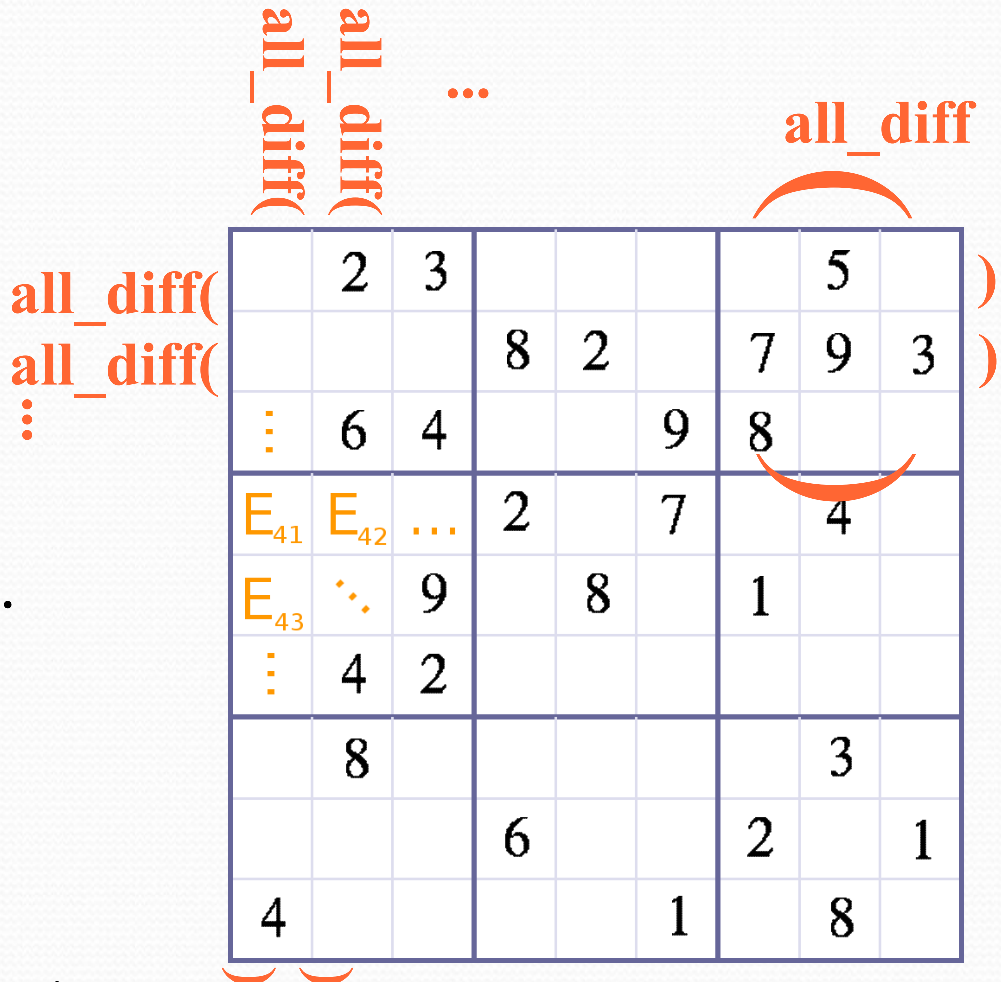
- Constraints

  all_different([E_{1x}]), ...

  all_different([E_{x1}]),

  all_different([E_{11}...E_{33}]), ...

High-level all difference constraint



Solving

- Two approaches:
  - automatically translate high-level constraints into a low-level representation (like a CNF formula)
    - MiniZinc (specialized language) + G12 (solvers)
    - NumberJack (Python library)
  - run a solver which directly supports high-level constraints

Domains must be finite

Common in constraint programming are finite domain solvers based on exhaustive search & propagation

Propagation

- Each (high-level) constraint is implemented in a **propagator**, which **only** operates on the variables listed in the constraint

- For each variable we store the **domain** of values the variable can still take, which may be
  - the complete domain (i.e., all values – clearly only works for problems with finite domains)

$D(x) = \{ 2 \}, D(y) = \{ 2, 3 \}$

  - lower and upper bounds, i.e. the minimum and maximal value the variable can still take

Propagation

- The task of the propagator is to maintain **domain consistency**, i.e. to **shrink** the domains of variables to values that they can still take

if domain $D(x) = \{ 2 \}$, $D(y) = \{ 2, 3 \}$ and constraint $x \neq y$ apply, then we can deduce that $D(y) = \{3\}$.

Bounds

if domain $D(x) = \{ 1, \ldots, 5 \}$, $D(y) = \{ 1, 2 \}$ and constraint $x + y < 5$ apply then we deduce that $D(x) = \{ 1, \ldots, 3 \}$

CP Search

no domain to change any more

Search (***Variables*** **):**
propagate all constraints till fix point
if contradiction found **then return**
if at least one variable is not fixed yet **then**
pick one variable ***V*** **not fixed**
for each possible ***value*** **of** ***V*** **do**
let *V=value* in this iteration
Search (*Variables*)
od
else
print solution in *Variables*

CP Search

all rows: all_different(row)
all columns: all_different(col)
all squares: all_different(square)

	2					6	5	4
				2		7	9	3
						8	1	2
						1		
								1

CP: Branch & Propagate

- propagate 2 (row)
- branch 4
- propagate 6 (square)

Propagation

- Propagators may implement special algorithms and data structures

all-different constraint:

all variables in a list must have a different value

algorithm 1: use inequality constraints independently

$D(x_1) = \{ 1, 2 \}$

$D(x_2) = \{ 1, 3 \}$

$D(x_3) = \{ 1, 3 \}$

$x_1 \neq x_2, x_1 \neq x_3, x_2 \neq x_3$

Propagation for inequality:
if one variable is fixed,
remove the corresponding
value from the domain
of the other variable
→ nothing happens in example

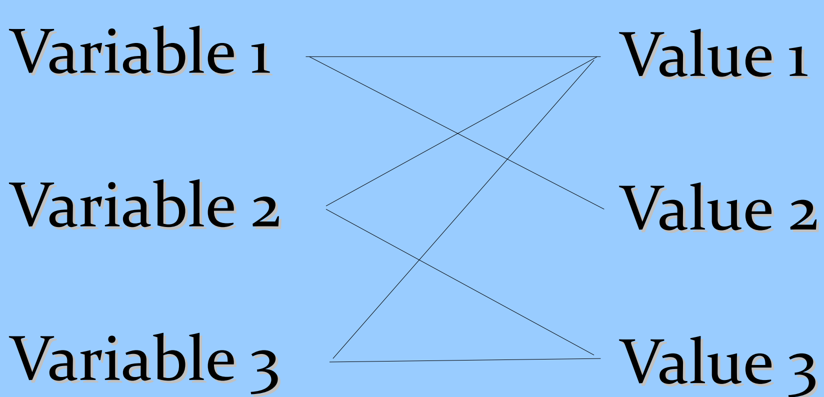
Propagation

- Propagators may implement special algorithms and data structures

<u>all-different constraint:</u>

all variables in a list must have a different value

<u>algorithm 2:</u> graph-based; bipartite matching



Variable 1 is fixed to value 2

Comparison to SAT solvers

- CP solvers support larger numbers of constraints & optimization
- When applied to CNF formulas, they search less efficiently as:
  - there is no clause learning
  - there is no propagation for pure symbols

These weaknesses led to the development of SMT SAT solvers (SAT-Modulo-Theories), which combine ideas of constraint programming and SAT solvers

Robert Nieuwenhuis, 2006.

Implementation issues

- When to run a propagator?
  - when a variable changes? (In any way)
  - when one particular bound changes?

for domain $D(x) = \{ 1, 2, 3 \}$, $D(y) = \{ 1, 2, 3 \}$ and constraint $x + y < 5$; should we propagate when we remove value 1 from $D(y)$? When we remove value 3?

In the CP literature, many different such strategies have been explored, called AC1, AC2, <u>AC3</u>, … AC5

Implementation issues

- Should we store simplified constraints during the search?

$D(x)=\{1,2,3\}$, $D(y) = \{ 4 \}$, $D(z) = \{ 1, 2\}$,
$x + y + z < 10 \rightarrow x + z < 6$

- Which order to select variables?
- Which order to select values?

Implementation issues

- How to branch over variables?

$D(x)=\{1,\ldots,10\}$, $D(z) = \{ 1,\ldots,10\}$, $x + y < 20$

Branch with *D(x)={c} for all c in 1…10?*

Branch with *D(x)={1…,5}* and *D(x)={6,..10}*?