About the previous talk joke ;-)

Who is the dark-side: Gurobi or Cplex?

- MIP
- CP
- Rebellion
- Dark-Side
The good new is

• and this is why CPAIOR is such an interesting conference ;-)
A Minimalistic Educational Solver

Laurent Michel, Pierre Schaus, Pascal Van Hentenryck
@ldmbouge    @pschaus    @PVanHentenryck

code:  https://bitbucket.org/pschaus/minicp
slides  http://tinyurl.com/y8n4knhx
Hummingbirds are **small, beautiful, efficient**

- the smallest birds
- rapid wing-flapping rates
  - typically around 50 times per second,
  - allowing them also to fly at speeds 54 km/h
- plumage with bright, varied coloration
• Many students in CS graduate without having ever heard about CP

http://www.a4cp.org/cparchive/countries_by_year
• Donald Knuth latest volume:

<table>
<thead>
<tr>
<th>Volume 4B, Combinatorial Algorithms: Part 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mathematical Preliminaries Redux</td>
</tr>
<tr>
<td>7.2.2. Backtrack Programming</td>
</tr>
<tr>
<td>7.2.2.1. Dancing Links</td>
</tr>
<tr>
<td>7.2.2.2. Satisfiability</td>
</tr>
</tbody>
</table>

http://www.cs.utsa.edu/~wagner/knuth/
As a community, what can we do to improve and increase the teaching of constraint programming?

Unanimous answer/observations

- communicate better and make teaching material more broadly available
- most CP teachers build their own teaching material without necessarily sharing it

The ACP decided to promote the sharing of teaching material such that any university or professor who wants to propose a CP course can do it with a modest effort.
Our hope:

- With MiniCP any professor having a basic background in algorithmic can easily propose a CP course at his institution.

MiniCP (will) provides teaching material, exercises, unit tests, and development projects.
Target audience

• **CS students** with
  ‣ background in data-structures and algos.

• Students/Instructors interested into teaching CP modeling language should consider
  ‣ MOOC on Minizinc by Peter Stuckey
  ‣ Tutorial on XCSP3 format
  ‣ User-manual of OPL, AIMMS, etc
Why not use an existing Solver?

- Existing solvers often try to balance three conflicting objectives
  - Efficiency
  - Flexibility
  - Simplicity

  solvers participating to competitions

  solvers focussed on real-life appli (hybridization, etc)

  most important criteria in the design of MiniCP
Design of MiniCP

- Influenced by cc(fd), Comet, Objective-CP and OscaR
- Similar design in other solvers (OR-Tools, Choco, etc)
- Implemented in Java8
- MiniCP is
  - trailed-based
  - propagator centered
  - adopt the mantra

\[ \text{CP = Modeling + Search} \]
MiniCP is small

- Code-base of +/- 1500 lines of Java code

<table>
<thead>
<tr>
<th>package</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine</td>
<td>867</td>
</tr>
<tr>
<td>reversible</td>
<td>301</td>
</tr>
<tr>
<td>cp</td>
<td>154</td>
</tr>
<tr>
<td>search</td>
<td>148</td>
</tr>
<tr>
<td>examples</td>
<td>288</td>
</tr>
<tr>
<td>tests</td>
<td>1316</td>
</tr>
</tbody>
</table>
Hello World = n-queens

- No two queens on the same line of diagonal

\[ \forall i, j \in 0..n - 1 \land i < j : q_i \neq q_j \]
\[ \forall i, j \in 0..n - 1 \land i < j : q_i \neq q_j + i - j \]
\[ \forall i, j \in 0..n - 1 \land i < j : q_i \neq q_j + j - i \]

- \( q[0] = 3 \)
- \( q[1] = 6 \)
- \( q[2] = 2 \)
- …
```java
int n = 8;
Solver cp = makeSolver();
IntVar[] q = makeIntVarArray(cp, n, n);
for(int i=0; i < n; i++)
    for(int j=i+1; j < n; j++) {
        cp.post(notEqual(q[i], q[j]));
        cp.post(notEqual(q[i], q[j], j-i));
        cp.post(notEqual(q[i], q[j], i-j));
    }
SearchStatistics stats = makeDfs(cp, selectMin(q, 
    qi -> qi.getSize() > 1, 
    qi -> qi.getSize(), 
    qi -> {
        int v = qi.getMin();
        return branch(() -> equal(qi, v), 
            () -> notEqual(qi, v));
    }));
).onSolution(() -> System.out.println("solution:" + Arrays.toString(q)));
).start();
```
Depth-First Search exploration letting the constraints prune the search tree
A domain is a finite set of discrete values $D \subseteq \mathbb{Z}$

A decision variable $x \in X$ has a domain $D$, denoted $D(x)$
- is instantiated (bound) when $|D(x)| = 1$,
- inconsistent when $D(x) = \emptyset$ and free when $|D(x)| \geq 2$.

A constraint $c \in C$ is a relation defined over a subset of $k$ variables $\{x_1, \cdots, x_k\} = \text{vars}(c) \subseteq X$.

Given a set of decision variables $X$, a solution $\sigma$ is a domain $D$, such that $\forall x \in X : |\sigma(x)| = 1$
Given decision variables $X$, and a constraint set $C$, a **feasible solution** $\sigma$ is a domain $D$, such that

$$(\forall x \in X : |\sigma(x)| = 1) \land \bigwedge_{c \in C} c(\sigma)$$

Given a CSP $\langle X, D, C \rangle$, the **solution set** $S(\langle X, D, C \rangle)$ is the set of all feasible solutions to $\langle X, D, C \rangle$.

A **filtering algorithm** $F$ for a constraint $c \in C$

- removes inconsistent values from the domain (contracting)
- consistent if it does not remove feasible solutions

$$S(\langle X, D, C \rangle) = S(\langle X, F_c(D), C \rangle)$$

- monotonic if $D_1 \subseteq D_2 \Rightarrow F_c(D_1) \subseteq F_c(D_2)$

Some formalism 2/2
Example of filtering rules

- $x = y + 1$

Whenever $D(y)$ loses some value $v$ from its domain, $v + 1$ is removed from $D(x)$

$v \notin D(y) \implies v + 1 \notin D(x)$
$v \notin D(x) \implies v - 1 \notin D(y)$

$|D(y)| = 1 \implies D(x) = \{\min(D(y)) + 1\}$
$|D(x)| = 1 \implies D(y) = \{\min(D(x)) - 1\}$
• Is the domain $D$ solution to the fix-point equation

$$D = \bigcap_{c \in C} \mathcal{F}_c(D)$$

• In practice it is computed as an iterative procedure

---

**Algorithm 1: Fixpoint algorithm**

**Data:** $D$, $C$

**Result:** $D$ the solution to the fixpoint equation (2)

1. $fix \leftarrow false$
2. while $\neg fix$ do
   3. $fix \leftarrow true$
   4. foreach $c \in C$ do
      5. $D' \leftarrow \mathcal{F}_c(D)$
      6. if $D' \neq D$ then
         7. $D \leftarrow D'$
         8. $fix \leftarrow false$
Fix-point outcome

• Computation of the fix-point with constraints \( C \) on a domain \( D_0 \)

\[ D_1 = \mathcal{F}_C(D_0) \]

• Possible outcomes

1. \( \text{failure}(D_1) \Rightarrow \text{no solution} \)

2. \( \text{success}(D_1) \Rightarrow D_1 \text{ can be reported as a solution} \)

3. \( \text{not success}(D_1) \) and \( \text{not failure}(D_1) \Rightarrow D_1 \text{ may contain a solution further splitting is necessary (divide and conquer)} \)
Generic Search

Algorithm 2: Generic Search in MiniCP

Data: $X, D, C$
Result: $S\langle X, D, C \rangle$

1. $S \leftarrow \emptyset$
2. if $\text{success}(D)$ then
   3. return $\{D\}$
4. $Q \leftarrow \{\langle X, D, C \rangle\}$
5. while $Q \neq \emptyset$ do
6.   $\langle X_0, D_0, C_0 \rangle \leftarrow \text{deQueue}(Q)$
7.   $(c_1, \ldots, c_k) \leftarrow \text{branching}(X_0, D_0)$
8.   foreach $i \in 1..k$ do
9.     $D_i \leftarrow FC_{\land c_i}(D_0)$
10.    if $\text{success}(D_i)$ then
11.       $S \leftarrow S \cup \{D_i\}$
12.      else if $\text{failure}(D_i)$ then
13.         continue;
14.    else
15.      enQueue($Q, \langle X_0, D_i, C_0 \land c_i \rangle$)
16. return $S$

Splitting of the search space
Example: $x=2, x \neq 2$
compute the fix-point with $c_i$
CP mainly uses DFS so $Q$ is generally a stack
Domain Implementation

- Sparse-Set = data-structure for set implementation
  - O(1) value removal
  - O(1) remove all except one given value
  - O(1) testing if a value is present
  - Iteration in O(k), k = number of values in the set

- Sparse-Sets are convenient for domain implementation
  - easy to implement and explain
Initialization for \{0,1,2,3,4,5,6,7,8\}

\[
\text{values}[\text{indexes}[v]] = v, \; \forall v \in \{0..n - 1\}
\]
Removal operation

- Remove 4

Values:

```
0 1 2 3 4 5 6 7 8
```

Indexes:

```
0 1 2 3 4 5 6 7 8
```

Size:

```
0 1 2 3 4 5 6 7 8
```

In the set:

```
0 1 2 3 4 5 6 7 8
```
Removal operation

- Remove 4
Removal operation

• Remove 4
Removal operation

- Remove 6

```
0 1 2 3 8 5 6 7 4
```

Values: 0 1 2 3 8 5 6 7 4
Indexes: 0 1 2 3 8 5 6 7 4

Size: 4

In the set: 0 1 2 3 8 5 6 7
Removed: 4
Removal operation

- Remove 6
Removal operation

- Assignment operation: only keep 3

<table>
<thead>
<tr>
<th>values</th>
<th>indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 8 5 7 6 4</td>
<td>0 1 2 3 8 5 7 6 4</td>
</tr>
</tbody>
</table>

- in the set
- removed

- size
Removal operation

- Assignment operation: only keep 3 in the set
public class SparseSet {
    private int[] values;
    private int[] indexes;
    private int size;
    private int n;

    public boolean remove(int val) {...}

    public void removeAllBut(int v) {...}

    public boolean contains(int val) {
        return indexes[val] < size;
    }
}

Sparsset-Set API
public interface DomainListener {
    void bind();
    void change(int domainSize);
    void removeBelow(int domainSize);
    void removeAbove(int domainSize);
}
public abstract class IntDomain {

    public abstract int getMin();

    public abstract int getMax();

    public abstract int getSize();

    public abstract boolean contains(int v);

    public abstract boolean isBound();

    public abstract void remove(int v, DomainListener x)
        throws InconsistencyException;

    public abstract void removeAllBut(int v, DomainListener x)
        throws InconsistencyException;

    public abstract int removeBelow(int value, DomainListener x)
        throws InconsistencyException;

    public abstract int removeAbove(int value, DomainListener x)
        throws InconsistencyException;

}
public class SparseSetDomain extends IntDomain {

    private SparseSet domain;
    private int offset;

    public SparseSetDomain(int min, int max) {
        offset = min;
        domain = new SparseSet(max-min+1);
    }

    public int getMin() {
        return domain.getMin() + offset;
    }

    public void remove(int v, DomainListener x) throws InconsistencyException {
        if (domain.contains(v - offset)) {
            boolean maxChanged = getMax() == v;
            boolean minChanged = getMin() == v;
            domain.remove(v - offset);
            if (domain.getSize() == 0) throw INCONSISTENCY;
            x.change(domain.getSize());
            if (maxChanged) x.removeAbove(domain.getSize());
            if (minChanged) x.removeBelow(domain.getSize());
            if (domain.getSize() == 1) x.bind();
        }
    }

    ...
}
public class IntVarImpl implements IntVar {

  private Solver cp;
  private IntDomain domain;
  private Stack<Constraint> onDomain;
  private Stack<Constraint> onBind;

  public IntVarImpl(Solver cp, int min, int max) {
    this.cp = cp;
    domain = new SparseSetDomain(min, max);
    onDomain = new Stack<>();
    onBind = new Stack<>();
  }

  public void propagateOnDomainChange(Constraint c) {
    onDomain.push(c);
  }

  public void propagateOnBind(Constraint c) {
    onBind.push(c);
  }
}

constraints interested to be called whenever the domain changes or if it bind

used by the constraint to register themselves to the changes of the domains
public class IntVarImpl implements IntVar {

    private DomainListener domListener = new DomainListener() {
        public void bind() {
            scheduleAll(onBind);
        }
        public void change(int domainSize) {
            scheduleAll(onDomain);
        }
    };

    private void scheduleAll(Stack<Constraint> constraints) {
        for (int i = 0; i < constraints.size(); i++)
            cp.schedule(constraints.get(i));
    }

    public void remove(int v) throws InconsistencyException {
        domain.remove(v, domListener);
    }

    public void assign(int v) throws InconsistencyException {
        domain.removeAllBut(v, domListener);
    }
}

Schedule the constraints for the fix-point computation
public abstract class Constraint {

    protected final Solver cp;
    protected boolean scheduled = false;

    public Constraint(Solver cp) {
        this.cp = cp;
    }

    public abstract void post() throws InconsistencyException;

    public void propagate() throws InconsistencyException {}
}

setup the constraint:
• first check of consistency
• register propagation events
• often terminate by a call to propagate

the filtering

state flag to avoid scheduling twice the constraint in the fix-point
public class NotEqual extends Constraint {

    private IntVar x, y;
    private int c;

    public NotEqual(IntVar x, IntVar y, int c) {
        \[|D(y)| = 1 \Rightarrow \min(D(y)) + c \notin D(x)\]
        \[|D(x)| = 1 \Rightarrow \min(D(x)) - c \notin D(y)\]
    }

    @Override
    public void post() throws InconsistencyException {
        if (y.isBound())
            x.remove(y.getMin() + c);
        else if (x.isBound())
            y.remove(x.getMin() - c);
        else {
            x.propagateOnBind(this);
            y.propagateOnBind(this);
        }
    }

    @Override
    public void propagate() throws InconsistencyException {
        if (y.isBound()) x.remove(y.getMin() + c);
        else y.remove(x.getMin() - c);
        this.deactivate();
    }
}
public class Solver {

    private Stack<Constraint> propagationQueue = new Stack<>();

    public void schedule(Constraint c) {
        if (!c.scheduled && c.isActive()) {
            c.scheduled = true;
            propagationQueue.add(c);
        }
    }

    public void fixPoint() throws InconsistencyException {
        boolean failed = false;
        while (!propagationQueue.isEmpty()) {
            Constraint c = propagationQueue.pop();
            c.scheduled = false;
            if (!failed) {
                try { c.propagate(); } catch (InconsistencyException e) {
                    failed = true;
                }
            }
        }
        if (failed) throw new InconsistencyException();
    }

    public void post(Constraint c, boolean enforceFixPoint) throws InconsistencyException {
        c.post();
        if (enforceFixPoint) fixPoint();
    }
}
So far so good

- **IntVarImpl**
  - DomainListener
  - SparseDomain
  - Stacks of <constraints>

- **Solver**
  - fix-point()

- **Constraint**
  - x.propagateOnDomainChange(this)

Constraints register to the domain changes (added to the stacks)

Constraints are schedule by the domain listener whenever something changes in the domain.
@FunctionalInterface
class DFS {
    private Choice branching;
    public DFS(Choice b) { branching = b; }
    public void dfs() {
        Alternative[] alternatives = branching.call();
        if (alternatives.length == 0)
            notifySolution();
        else
            for (a : alternatives) {
                a.call();
                dfs();
            }
    }
}

API for creation of child nodes.

Generate the child nodes

call the closure before recursion
Not enough: we need state restoration

- because what we typically want to do is with branch is:

```java
int n = 8;
Solver cp = makeSolver();
IntVar[] q = makeIntVarArray(cp, n, n);

SearchStatistics stats = makeDfs(cp,
    selectMin(q,
        qi -> qi.getSize() >
        qi -> qi.getSize(),
        qi -> {
            int v = qi.getMin();
            return branch(() -> equal(qi,v),
                () -> notEqual(qi,v));
        }
    ).onSolution(() ->
        System.out.println("solution:" + Arrays.toString(q))
    ).start();
```
The answer = The Trail

• The trail = a mechanism for doing and undoing

  Why does life not have a Ctrl/Z option...

• before doing => trail.push()

• undoing    => trail.pop()
State restoration with Trail

Trail trail = new Trail();

ReversibleInt a = new ReversibleInt(trail, 7);
ReversibleInt b = new ReversibleInt(trail, 13);

trail.push(); // record a=7, b=13
a.setValue(11);
b.setValue(14);
trail.push(); // record a=11, b=14
a.setValue(4);
b.setValue(9);
trail.pop(); // restore a=11, b=14
trail.pop(); // restore a=7, b=13

like an integer except that we can undo the changes with the push()/pop() of the trail
This is exactly what we need for the search

- Assume all our objects are reversibles (domains, variables, etc)

- The search would do
How is this trail implemented??

```
public interface TrailEntry {
    public void restore();
}
```

```
Trail trail = new Trail();
ReversibleInt a = new ReversibleInt(trail, 7);
ReversibleInt b = new ReversibleInt(trail, 13);
trail.push();
a.setValue(11);
b.setValue(14);
trail.push();
a.setValue(4);
b.setValue(9);
trail.pop();
trail.pop();
```

![Diagram of trail stack with values](image)
public class Trail {
    public long magic = 0;
    private Stack<TrailEntry> trail = new Stack<TrailEntry>();
    private Stack<Integer> trailLimit = new Stack<Integer>();

    public void pushOnTrail(TrailEntry entry) {
        trail.push(entry);
    }

    public void push(){
        magic ++;
        trailLimit.push(trail.size());
    }

    public void pop() {
        int n = trail.size() - trailLimit.pop();
        for (int i = 0; i < n; i++) trail.pop().restore();
        magic ++;
    }
}

public interface TrailEntry {
    public void restore();
}

Trail Implementation
public class ReversibleInt implements RevInt {

    class TrailEntryInt implements TrailEntry {
        private final int v;
        public TrailEntryInt(int v) {
            this.v = v;
        }
        public void restore() { ReversibleInt.this.v = v; }
    }

    private Trail trail;
    private int v;

    public ReversibleInt(Trail trail, int initial) {

    }

    public int setValue(int v) {
        if (v != this.v) {
            trail.pushOnTrail(new TrailEntryInt(v));
            this.v = v;
        }
        return this.v;
    }
}
Implementation trick

```javascript
b.setValue(6);
trail.push();
a.setValue(11);
b.setValue(14);
b.setValue(1);
b.setValue(4);
trail.push();
a.setValue(4);
b.setValue(9);
trail.pop();
trail.pop();
```

3 trail entries are stacked on the trail, are they really necessary?

only the one created at that time is useful. The value b=14 and b=1 will never be restored.

![Diagram of trail entries]

```
trail
```

- TrailEntry b=1
- TrailEntry b=14
- TrailEntry b=6
public class ReversibleInt implements RevInt {

    private Trail trail;
    private int v;
    private Long lastMagic = -1L;

    private void trail() {
        long trailMagic = trail.magic;
        if (lastMagic != trailMagic) {
            lastMagic = trailMagic;
            trail.pushOnTrail(new TrailEntryInt(v));
        }
    }

    public int setValue(int v) {
        if (v != this.v) {
            trail();
            this.v = v;
        }
        return this.v;
    }
}
We need:

- Reversible Domains => Reversible Sparse-Sets

- Reversible addition of constraints (they must withdraw upon backtrack)

- Reverse all the state that you can possibly put inside the constraints
  - Constraint implementors should only focus on incremental aspects down in the search tree
Trail trail = new Trail();
ReversibleSparseSet set = new ReversibleSparseSet(trail, 9);
trail.push();
set.remove(4);

All we need to change is size is now a ReversibleInt

in the set

values

0 1 2 3 4 5 6 7 8

indexes

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
Trail trail = new Trail();
ReversibleSparseSet set = new ReversibleSparseSet(trail, 9);
trail.push();
set.remove(4);
set.remove(6);
Trail trail = new Trail();
ReversibleSparseSet set = new ReversibleSparseSet(trail, 9);
trail.push();
set.remove(4);
set.remove(6);
train.push();
set.assign(3);
Removal operation

Trail trail = new Trail();
ReversibleSparseSet set = new ReversibleSparseSet(trail, 9);
trail.push();
set.remove(4);
set.remove(6);
trail.push();
set.assign(3);
trail.pop();
Trail trail = new Trail();
ReversibleSparseSet set = new Reversible SparseSet(trail, 9);
trail.push();
set.remove(4);
set.remove(6);
trail.push();
set.assign(3);
trail.pop(); // {0,1,2,3,5,7,8}
Trail trail = new Trail();
ReversibleSparseSet set = new ReversibleSparseSet(trail, 9);
trail.push();
set.remove(4);
set.remove(6);
train.push()
set.assign(3);
trail.pop(); // {0,1,2,3,5,7,8}
trail.pop(); // {0..9}
Adding a constraint = reversible operation

can do on a branch
 cp.post(a <= 4)
this must be a reversible operation
public class IntVarImpl implements IntVar {

    private Solver cp;
    private IntDomain domain;
    private ReversibleStack<Constraint> onDomain;
    private ReversibleStack<Constraint> onBind;

    private DomainListener domListener = new DomainListener() {
        public void bind() { scheduleAll(onBind); }
        public void change(int domainSize){
            scheduleAll(onDomain);
        }
    };

    public IntVarImpl(Solver cp, int min, int max) {
        this.cp = cp;
        cp.registerVar(this);
        domain = new SparseSetDomain(cp.getTrail(),min,max);
        onDomain = new ReversibleStack<>(cp.getTrail());
        onBind = new ReversibleStack<>(cp.getTrail());
    }
}

IntVarImpl: Making it reversible

encapsulates a ReversibleSparseSet
public class ReversibleStack<E> {

    ReversibleInt size;
    ArrayList<E> stack;

    public ReversibleStack(Trail rc) {
        size = new ReversibleInt(rc, 0);
        stack = new ArrayList<E>();
    }

    public void push(E elem) {
        stack.add(size.getValue(), elem);
        size.increment();
    }

    public int size() { return size.getValue(); }

    public E get(int index) {
        return stack.get(index);
    }
}

All we need to change is:

size = ReversibleInt
public class DFS {
    private Trail trail;
    private Choice branching;

    public DFS(Trail t, Choice b) {
        ...
    }

    public void dfs() {
        Alternative[] alternatives = branching.call();
        if (alternatives.length == 0)
            notifySolution();
        else
            for (a : alternatives) {
                trail.push();
                a.call();
                dfs();
                trail.pop();
            }
    }
}
Locations: 2, 3, 4, 5

Facilities: 1, 2, 3, 4, 5

Input: Distance between any to location $D_{2,3}$

Input: Weight between any two facilities (e.g. amount of traffic) $W_{1,3}$

Decision: Where to place each warehouse?
Problem:
Assign one facility to each location minimizing
\[ \sum_{i,j} D_{x_i,x_j} \cdot W_{i,j} \]

2D element constraint = 2D array indexed by two variables
Quadratic Assignment Model

Solver cp = makeSolver();
IntVar[] x = makeIntVarArray(cp, n, n);

    cp.post(allDifferent(x));

    // build the objective function
IntVar[] weightedDist = new IntVar[n*n];
    int ind = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            weightedDist[ind] = mul(element(d,x[i],x[j]),w[i][j]);
            ind++;
        }
    }
    IntVar objective = sum(weightedDist);

DFSearch dfs = makeDfs(cp,firstFail(x));
    cp.post(minimize(objective,dfs));
Element2D(int[][] T, IntVar x, IntVar y, IntVar z)

- $T[x][y] = z$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

- How to create an efficient propagator for Element2D?
- Don’t want to create holes in $D(z)$ but well in $D(x)$ and $D(y)$
\[ T[x][y] = z \]

- \( D(x) = \{0,1,2,3\} \)
- \( D(y) = \{0,1,2,3\} \)
- \( D(z) = [1..9] \) (interval domain)

\[
\begin{array}{cccc|c}
\hline
&0&1&2&3&\text{rSup} \\
\hline
0&1&8&9&6&4 \\
1&1&9&2&4&4 \\
2&9&8&9&8&4 \\
3&1&9&2&5&4 \\
c\text{Sup}&4&4&4&4&4 \\
\hline
\end{array}
\]

\[
\begin{array}{ccc}
\text{z}&\text{x}&\text{y} \\
\hline
1&0&0 \\
1&1&0 \\
1&3&0 \\
2&1&2 \\
2&3&2 \\
4&1&3 \\
5&3&3 \\
6&0&3 \\
8&0&1 \\
8&2&1 \\
8&2&3 \\
9&0&2 \\
9&1&1 \\
9&2&0 \\
9&2&2 \\
9&3&1 \\
\hline
\end{array}
\]

\text{sorted}

\text{low} \rightarrow \text{up}
\(T[x][y] = z\)

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>rSup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>z</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- \(D(x) = \{0,1,2,3\}\)
- \(D(y) = \{0,1,2,3\}\)
- \(D(z) = [1..7]\) (interval domain)
$T[x][y] = z$

- $D(x) = \{0,1,2,3\}$
- $D(y) = \{0,1,2,3\}$
- $D(z) = [1..7]$ (interval domain)

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>rSup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>cSup</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>z</th>
<th>x</th>
<th>y</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
\[ T[x][y] = z \]

- \( D(x) = \{0,1,2,3\} \)
- \( D(y) = \{0,1,2,3\} \)
- \( D(z) = [1..7] \) (interval domain)

**Table:**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>( rSup )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>cSup</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Sorted:**

<table>
<thead>
<tr>
<th></th>
<th>z</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Diagram:**

The diagram shows a sorted list of values for \( z, x, \) and \( y \) with corresponding indices for each value. The values are sorted in a specific order, indicated by the arrows labeled 'low' and 'up'.
D(x) = \{0,1,2,3\}
D(y) = \{0,1,2,3\}
D(z) = [1..7] (interval domain)

T[x][y] = z

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 8 9 6</td>
</tr>
<tr>
<td>1</td>
<td>1 9 2 4</td>
</tr>
<tr>
<td>2</td>
<td>9 8 9 8</td>
</tr>
<tr>
<td>3</td>
<td>1 9 2 5</td>
</tr>
<tr>
<td>cSup</td>
<td>3 3 3 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>z</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
\[ T[x][y] = z \]

- \( D(x) = \{0,1,2,3\} \)
- \( D(y) = \{0,1,2,3\} \)
- \( D(z) = [1..7] \) (interval domain)
\[ T[x][y] = z \]

- \( D(x) = \{0,1,2,3\} \)
- \( D(y) = \{0,1,2,3\} \)
- \( D(z) = [1..7] \) (interval domain)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>rSup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>cSup</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

sorted

sorted

\( z \) \( x \) \( y \)
\( 8 \) \( 2 \) \( 3 \)
\( 9 \) \( 0 \) \( 2 \)
\( 9 \) \( 1 \) \( 1 \)
\( 9 \) \( 2 \) \( 0 \)
\( 9 \) \( 2 \) \( 2 \)
\( 9 \) \( 3 \) \( 1 \)
\[ T[x][y] = z \]

- \( D(x) = \{0,1,2,3\} \)
- \( D(y) = \{0,1,2,3\} \)
- \( D(z) = [1..7] \) (interval domain)
\[ T[x][y] = z \]

- \( D(x) = \{0, 1, 2, 3\} \)
- \( D(y) = \{0, 1, 2, 3\} \)
- \( D(z) = [1..7] \) (interval domain)
\[ T[x][y] = z \]

- \( D(x) = \{0,1,2,3\} \)
- \( D(y) = \{0,1,2,3\} \)
- \( D(z) = [1..6,7] \) (interval domain)
\( T[x][y] = z \)

- \( D(x) = \{0, 1, 3\} \)
- \( D(y) = \{0, 2, 3\} \)
- \( D(z) = [2..6] \) (interval domain)
\[ T[x][y] = z \]

- \( D(x) = \{0,1,3\} \)
- \( D(y) = \{0,2,3\} \)
- \( D(z) = [2..6] \) (interval domain)
\[ T[x][y] = z \]

- \( D(x) = \{0, 1, 3\} \)
- \( D(y) = \{2, 3\} \)
- \( D(z) = [2..6] \) (interval domain)
\[ T[x][y] = z \]

- \(D(x) = \{0,1,3\}\)
- \(D(y) = \{2,3\}\)
- \(D(z) = [2..6] \text{ (interval domain)}\)

<table>
<thead>
<tr>
<th>(\text{cSup})</th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{rSup})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(x)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(y)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(z)</th>
<th>(x)</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

The table is sorted in ascending order (low) and then in descending order (up) for each column.
\[ T[x][y] = z \]

- **D(x) = \{0,1,3\}**
- **D(y) = \{2,3\}**
- **D(z) = [2,3,4..6] (interval domain)**

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>rSup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>cSup</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4..6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>cSup</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**sorted**

<table>
<thead>
<tr>
<th>z</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**low** → **up**
What do we need to restore these values on backtrack?
public class Element2D extends Constraint {

private final int[][] T;
private final IntVar x, y, z;

private final ReversibleInt[] nRowsSup, nColsSup;
private final ReversibleInt low, up;
private final ArrayList<Tripple> xyz;

public void post() throws InconsistencyException {
    . . . // initialize counters nRowsSup, nColsSup
    x.propagateOnDomainChange(this);
    y.propagateOnDomainChange(this);
    z.propagateOnBoundChange(this);
    propagate();
}
}
public class Element2D extends Constraint {

    private void updateSupports(int lostPos) throws InconsistencyException {
        if (nColsSup[xyz.get(lostPos).x].decrement() == 0) {
            x.remove(xyz.get(lostPos).x);
        }
        if (nRowsSup[xyz.get(lostPos).y].decrement() == 0) {
            y.remove(xyz.get(lostPos).y);
        }
    }

    public void propagate() throws InconsistencyException {
        int l = low.getValue();
        int u = up.getValue();
        int zMin = z.getMin();
        while (xyz.get(l).z < zMin || !x.contains(xyz.get(l).x) || !y.contains(xyz.get(l).y)) {
            updateSupports(l);
            l++;
            if (l > u) throw new InconsistencyException();
        }
        z.removeBelow(xyz.get(l).z);
        low.setValue(l);
        // do something similar for updating u
    }
}
Solver cp = makeSolver();
IntVar[] x = makeIntVarArray(cp, n, n);

cp.post(allDifferent(x));

// build the objective function
IntVar[] weightedDist = new IntVar[n*n];
int ind = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        weightedDist[ind] = mul(element(d,x[i],x[j]),w[i][j]);
        ind++;
    }
}
IntVar objective

DFSearch dfs = makeDfs(cp, firstFail(x));

Any idea how CP is able to minimize?
Minimization with CP with a special constraint

```java
public class Minimize extends Constraint {

    public int bound = Integer.MAX_VALUE;
    public final IntVar x;
    public final DFSearch dfs;

    public Minimize(IntVar x, DFSearch dfs) {
        // ...
    }

    protected void tighten() {
        if (!x.isBound())
            throw new RuntimeException("objective not bound");
        this.bound = x.getMax() - 1;
    }

    public void post() throws InconsistencyException {
        x.whenBoundsChange(() -> x.removeAbove(bound));
        // Ensure that the constraint is scheduled on backtrack
        dfs.onSolution(() -> {
            tighten();
            cp.schedule(this);
        });
        dfs.onFail(() -> cp.schedule(this));
    }
}
```

- B&B Constraint
- Tighten objective on each solution found (this is why we need dfs)
- Don’t forget to schedule it in the propagation queue on backtrack
The weakness of CP

- Huge search tree
- Very poor exploration of the search space
How to fix this?

- When you get stuck for too-long not improving, restart at another place
- Intensify the search where it looks promising
Large Neighborhood Search (LNS)

- LNS = Fix + Relax + Restart

1. Find a first initial solution, $S^*$
2. Randomly relax $S^*$ and re-optimize with search limit
   - Relax = fix some variables to their values in $S^*$
3. Replace $S^*$ by the best solution found

It can be more general than that, for instance in scheduling relax = keep some of the precedences from best solution
Advantages over pure LS

- The neighborhood is large
  - no need for meta-heuristic to avoid local minima

- Modeling power of CP (declarative),
  - no need for designing complex neighborhood
  - ease of implementation

- Scalability of LS
  - very good « any-time » behavior
LNS on top of our QAP model

```java
// Current best solution
int[] xBest = new int[n];
for (int i = 0; i < n; i++) {
    xBest[i] = i;
}

dfs.onSolution(() -> {
    // Update the current best solution
    for (int i = 0; i < n; i++) {
        xBest[i] = x[i].getMin();
    }
});

int nRestarts = 1000;
int failureLimit = 50;
Random rand = new java.util.Random(0);
for (int i = 0; i < nRestarts; i++) {
    // Record the state such that the fragment constraints can be cancelled
    cp.push();
    // Assign the fragment 50% of the variables randomly chosen
    for (int j = 0; j < n; j++) {
        if (rand.nextInt(100) < 50) {
            equal(x[j], xBest[j]);
        }
    }
    dfs.start(statistics -> statistics.nFailures >= failureLimit);
    // cancel all the fragment constraints
    cp.pop();
}
```

- simple initial assignment (could be random)
- update current best solution whenever one is found
- fix randomly 50% of the variables to their value in the current best solution
- start a DFS search but give it a maximum number of failure credit (not too long)
Mini-Solver Tracks

less than 8,000 lines discarding code for parsing XCSP3, comments and code of standard libraries).

Feel free to use MiniCP as a starting point
Want to know more about MiniCP

- Ready for the **ACP summer school 2017** using MiniCP for teaching CP to the students come in MiniCP:
  - Coding exercises and their unit-tests
  - Technical documentation with theoretical foundations

Joint ACP and GdR RO Summer School 2017

**CP and RO**
September 18-22, 2017
Poquerolles, France

http://school.a4cp.org/summer2017/
Take away message

• Want to improve your CP knowledge
  ‣ implement your own solver (MiniCP is a good starting point but don’t hesitate to change or adopt a different design, domain implem, etc)
  ‣ implement a few constraints (table, sum, element, etc)
    ✴ the largest and most difficult code-base in a solver are the constraints!
    ✴ try to design incremental filtering (you can already do a lot with reversible integers)
• Implement a few black-box and LNS searches
• Solve and model many problems
A Minimalistic Educational Solver

Laurent Michel, Pierre Schaus, Pascal Van Hentenryck

https://bitbucket.org/pschaus/minicp