Session 6
Search Strategies

Daniel Diaz
Salvador Abreu

Search

When all constraints are posted, domains are reduced but generally no solution is yet available.

The Search phase discovers which values in the (reduced) domains form an actual solution.

For each variable **X**, all possible values **v** are tried. This is done by posting a constraint **X = v** which, by propagation, reduces the domain of other variables. Then another variables **X’** is selected together with another value **v’**. The constraint **X’=v’** is posted, etc...

When a **failure occurs**, the **last assigned variable is reconsidered** (its assignment is undone by **backtracking**) and a new value **v”** is tried.

Search

The Search phase explores in a **depth-first** manner a search tree whose structure depends on the order the variables and values are considered. These 2 orders are important for performance:

- The order in which **variables** are considered (the first variable enumerated being at the root of the search tree)
- For a variable, the order in which its candidate **values** are tried (each value gives rise to a child in the search-tree).

Making a bad choice of value for the root variable will be reconsidered only when the subsequent sub-tree is totally explored, thereby costing a lot of useless runtime.

Variable selection

The simplest heuristics use a static ordering (e.g. chronological order or input order) while the more complex use a dynamic ordering (the “best” variable being recomputed at each step, after the propagations of each assignment of a value to a variable).

Examples of heuristics:

Input order: no heuristic is used, variables are considered in the order in which they were given (chronological order).

Fail-first: the variable with smallest domain size is enumerated first. In case of ties, other sub-strategies are possible.

Search in Choco

By default Choco uses a strategy called **`domOverWDegSearch`**. It is a first-fail also taking into account the number of attached constraints (degree) and the number of times they have failed (weight).

It is possible to override the default strategy with new strategies:

```
    solver.setSearch(AbstractStrategy... strategies)
```

Several strategies can be passed (they are <u>executed sequentially</u>).

NB: **`setSearch()`** only defines the strategy to use. The search itself only occurs when the problem is solved (calling **`solver.solve()`** or **`solver.findSolution()`**).

Example of use:

```
    solver.setSearch(Search.inputOrderLBSearch(vars));
```

Choco: predefined all-in-one strategies

Several (combined) strategies are available as static (class) methods:

```
Search.inputOrderLBSearch(IntVar... vars)
Search.inputOrderUBSearch(IntVar... vars)
```

Select variables in input order. Select values in ascending order (from the Lower Bound) or descending (from Upper Bound).

```
Search.domOverWDegSearch(IntVar... vars)
```

Select a variable according to `DomOverWDeg`, assign it to its LB.

```
Search.minDomLBSearch(IntVar... vars)
Search.minDomUBSearch(IntVar... vars)
```

Select the variable of smallest domain size, assign it to its lower/upper bound.

```
Search.randomSearch(IntVar[] vars, long seed)
```

Select a variable randomly, assign it a randomly chosen value.

Choco: defining separate strategies

With this method it is possible to customize both the variable and the value selection strategy:

```
Search.intVarSearch(VariableSelector<IntVar> varSelector,
      IntValueSelector valSelector, IntVar... vars)
```

Example of use:

```
solver.setSearch(
   Search.intVarSearch(
      new Smallest(),      // select the variable with lowest LB
      new IntDomainMin(),  // select values from LB to UB
      myVars));
```

Choco: defining compound strategies

setSearch may take more than one strategy

Example of use:

```
solver.setSearch(
    Search.intVarSearch(
        new Smallest(),      // select the variable with lowest LB
        new IntDomainMin(),  // select values from LB to UB
        vars1),
    Search.intVarSearch(
        new Occurrence(),        // variable with more links
        new IntDomainMedian(),   // values from median value <>
        vars2)
);
```

Predefined variable selection strategies

`InputOrder<>(Model model)`
  Select the variables in the order they are given

`FirstFail(Model model)` / `AntiFirstFail(Model model)`
  select the variable with the smallest / largest domain size

`Smallest()` / `Largest()`
  select the variable with the smallest LB / largest UB

`MaxRegret()`
  select the variable with the largest difference between the two smallest values in its domain

`Random(long seed)`
  select a variable randomly

`Occurrence()`
  select the variable with the largest number of constraints

`Cyclic()`
  select the next variable circularly (i = (i+ 1) % vars.length)

Predefined value selection strategies

`IntDomainMin()` / `IntDomainMax()`
  Select the LB / UB value (i.e. ascending / descending order)

`IntDomainMiddle(boolean roundingPolicy)`
  Select the value closest to the average of its current bounds according to `roundingPolicy` (`true` = floor, `false` = ceil).

`IntDomainMedian()`
  Selects the median value in the domain

`IntDomainRandom(long seed)`
  Selects randomly a value in the domain

`IntDomainRandomBound(long seed)`
  Selects randomly between the LB and the UB of the variable

Exercises

- Encode the N-queens problem.
  - Use the 1-variable-per-queen encoding
  - Vary the search strategy
  - Compare to your previous (naïve Choco) implementation
- Encode the Magic Squares (N) problem
  - Vary the search strategy, try, for instance (check others too!):
    - (default)
    - `Search.domOverWDegSearch()`, `Search.inputOrderLBSearch()`
    - search over diagonals (i.e. initiate on a subset of variables), e.g.
      ```
      solver.setSearch(Search.intVarSearch(new InputOrder<>(model),
                                           new IntDomainRandom(1), d1),
                       Search.intVarSearch(new FirstFail(model),
                                           new IntDomainRandom(1), all));
      ```
  - What is the runtime for each value of N (4, 5, 6, ...)?
  - What is the current limit under 5 minutes runtime?
  - Compare to your previous (naïve Choco) implementation