Variable & Value Ordering Heuristics

Heuristics for backtracking algorithms

- Variable ordering
  - what variable to branch on next
- Value ordering
  - given a choice of variable, what order to try values
- Constraint ordering
  - what order to propagate constraints
  - most likely to fail or cheapest propagated first

Variable ordering

- Domain dependent heuristics
- Domain independent heuristics
- Static variable ordering
  - fixed before search starts
- Dynamic variable ordering
  - chosen during search

Basic idea

- Assign a heuristic value to a variable that estimates how difficult/easy it is to find a satisfying value for that variable

SVO

Static variable orderings

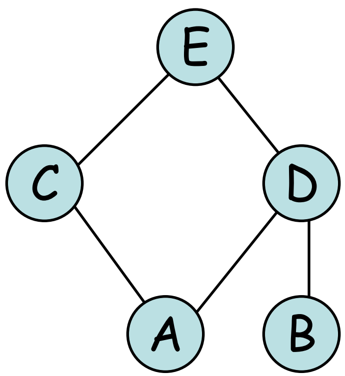
- based on constraint graph topology
  - minimum width
  - minimum induced width
  - max degree ordering
  - minimum bandwidth ordering
- based on something else

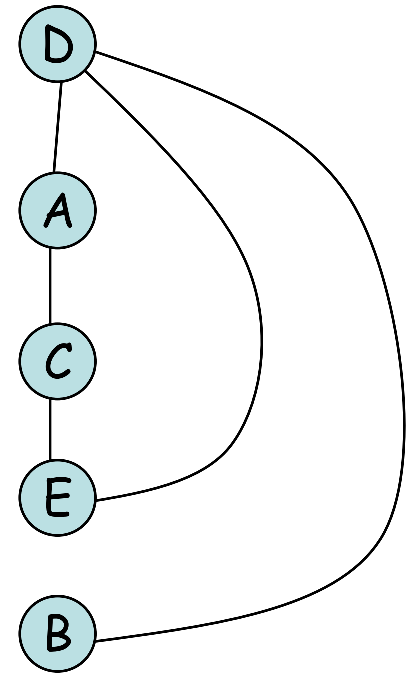
Usually for backward checking algorithms

- why?

Static variable orderings

"order" the constraint graph in a certain way





Minimum width ordering
- width of a node is number of adjacent predecessors
- width of an ordering is maximum width of the nodes
- width of a graph is minimal width of all orderings

Max degree ordering (shown)
- in non-decreasing degree sequence

Why should this work?
Is there anything bad bout it?

Minimum width aka degeneracy ordering

Minimum width aka degeneracy ordering

1. Select vertex v of maximum degree
2. Remove v from graph
    - reduce degree of vertices adjacent to v
3. If vertices remain, go to 1

Minimum Bandwidth Ordering (MBO)

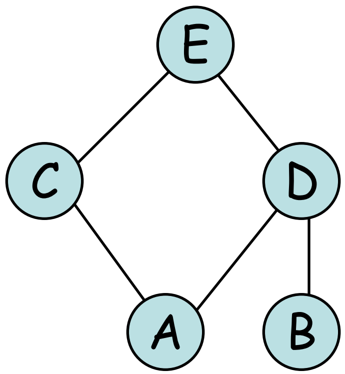
What is that?

What's its complexity?

Do we need it if we can jump?

- Bandwidth of a variable is the "distance" between variables in the ordered constraint graph
- Bandwidth of ordering is max bandwidth of varaibles/vertices

Minimum Bandwidth Ordering (MBO)



Measuring backwards

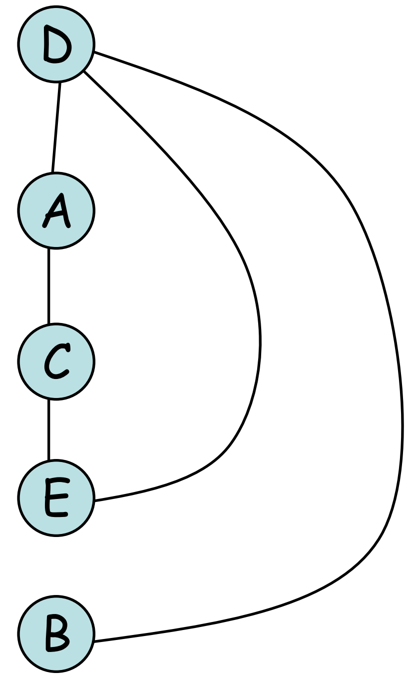
bw(D) = 0

bw(A) = 1

bw(C) = 1

bw(E) = 3

bw(B) = 4



Bandwidth of ordering is 4

MBO is minimum of all orderings
NP-hard to find ☹

Bandwidth is the "distance" between variables in the ordered constraint graph

DVO

Dynamic variable ordering (dvo)

Domain Indepentent

- Mainly based on the FF principle
- Mainly used by MAC and FC (why?)
  - smallest domain first
  - brelaz
  - dom/deg

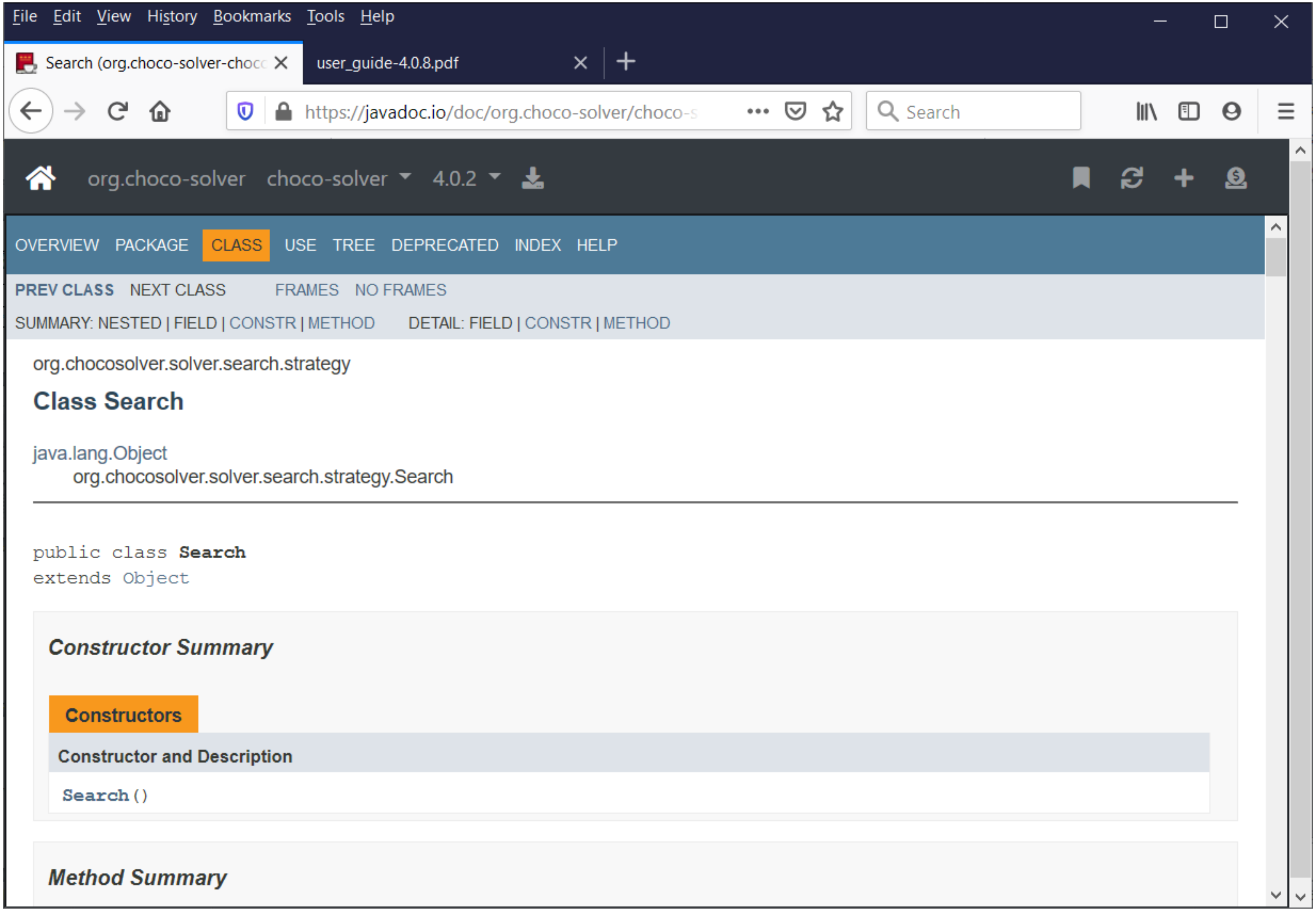
Regret
For each variable measure it's regret as (best value – next best value)
Chose variable with maximum regret

Fail First Principle: "*To succeed, try first where you are most likely to fail*" Haralick & Elliott 1980

3.2 Search Strategies

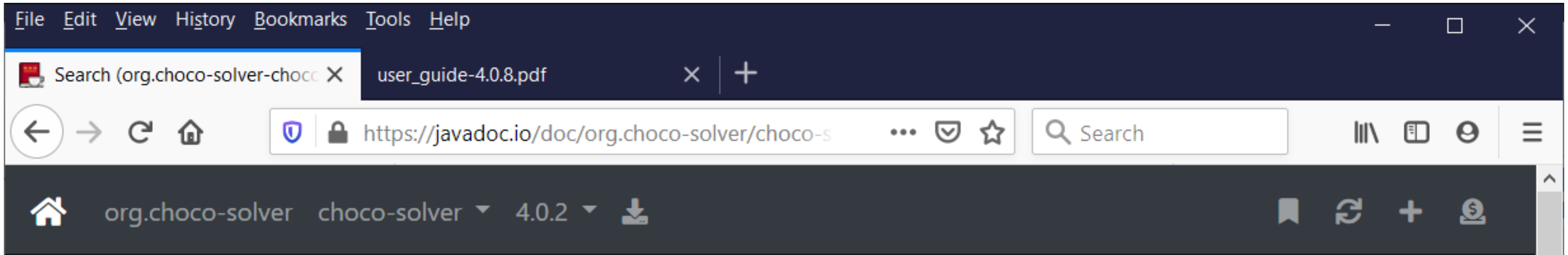
The search space induced by variable domains is equal to $S = |d_1| * |d_2| * ... * |d_n|$ where d_i is the domain of the i^{th} variable. Most of the time (not to say always), constraint propagation is not sufficient to build a solution, that

is, to remove all values but one from variable domains. Thus, the search space needs to be explored using one or more *search strategies*. A search strategy defines how to explore the search space by computing *decisions*. A decision involves a variables, a value and an operator, e.g. $x = 5$, and triggers new constraint propagation. Decisions are computed and applied until all the variables are instantiated, that is, a solution has been found, or a failure has been detected (backtrack occurs). Choco 4.0.8 builds a binary search tree: each decision can be refuted (if $x = 5$ leads to no solution, then $x! = 5$ is applied). The classical search is based on Depth First Search.



File Edit View History Bookmarks Tools Help
Search (org.choco-solver-choc
user_guide-4.0.8.pdf
https://javadoc.io/doc/org.choco-solver/choco-s
Search
org.choco-solver
choco-solver
4.0.2
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
org.chocosolver.solver.search.strategy
Class Search
java.lang.Object
org.chocosolver.solver.search.strategy.Search
public class Search
extends Object
Constructor Summary
Constructors
Constructor and Description
Search()
Method Summary

For example ...



inputOrderUBSearch

```
public static IntStrategy inputOrderUBSearch(IntVar... vars)
```

Assigns the first non-instantiated variable to its upper bound.

Parameters:

`vars - list of variables`

Returns:

`assignment strategy`

minDomLBSearch

```
public static IntStrategy minDomLBSearch(IntVar... vars)
```

Assigns the non-instantiated variable of smallest domain size to its lower bound.

Parameters:

`vars - list of variables`

Returns:

`assignment strategy`

minDomUBSearch

```
solver.setSearch(Search.minDomLBSearch(q)); // fail-first
```

Dom over weighted degree (example)

When propagation of a constraint results in a dwo (domain wipe out)
Increment the weight of that constraint

For a variable v, sum up the weight of the constraints it is involved in

$h(v) = card(dom(v))/weightedDegree(v)$

Select variable with minimum h(v)

Some more recent dvo's

- Conflict ordering search [cp2015]
- Reasoning from last conflict(s) [AIJ 173, 2009]
- Boosting systematic search by weighting constraints [ECAI2004]

Cutset decomposition

If constraint graph is a tree then *AC* is a decision procedure
(result due to E.C. Freuder (Gene))

Select a variable that cuts the constraint graph

Value Ordering

Value ordering

- All solutions
  - value ordering not important
  - why?
- One solution
  - if a solution exists, there exists a *perfect* value ordering
- Insoluble instance
  - like all solutions
  - why?

Value ordering: Intuition (*promise*)

- Goal: minimize size of search space explored
- Principle:
  - given that we have already chosen the next variable to instantiate, choose first the values that are most likely to succeed
  - The most ***promising*** value

Promise

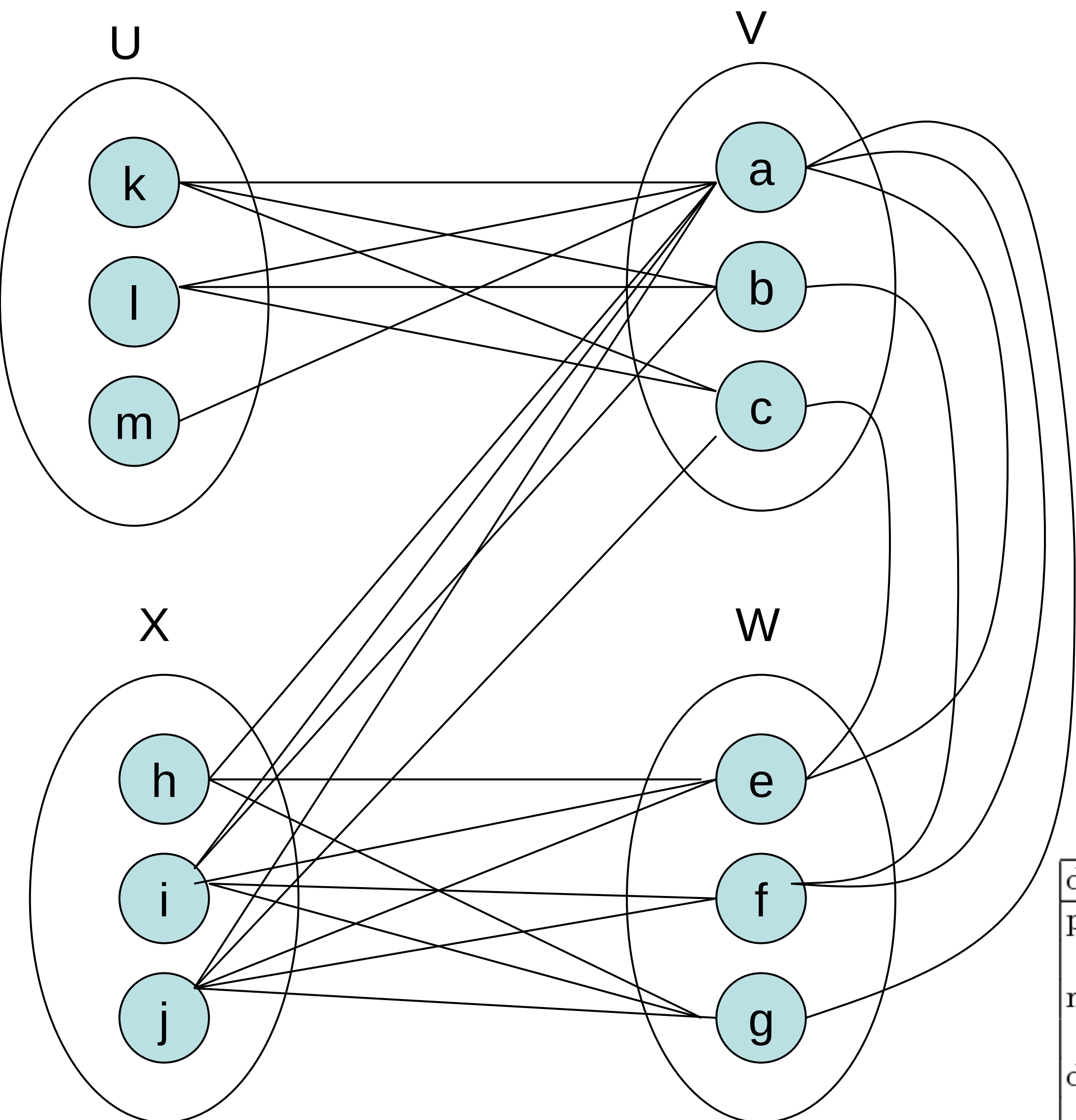
Domain Indepentent

Measure promise of a value as follows

- count the number of supports in adjacent domain
- take the product of this value
- choose the value with the highest amount
- the most promising

A dual viewpoint (Geelen)
Choose the least promising variable
Assign it the most promising value

Microstructure & promise



domain values	a	b	c	e	f	g	h	i	j	k	l	m
promise	27	2	2	6	4	3	2	6	6	3	3	1
normalised	1	$\frac{2}{27}$	$\frac{2}{27}$	1	$\frac{2}{3}$	$\frac{1}{2}$	$\frac{1}{3}$	1	1	1	1	$\frac{1}{3}$
discrepancy	0	$\frac{25}{27}$	$\frac{25}{27}$	0	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	0	0	0	0	$\frac{2}{3}$

Table 1. Promise of domain values, giving discrepancy values

Can FF show Promise?

Might FF actually be promising?

If FF is on path to a solution we would prefer promise to failure
But does FF actually do this?

Experiments using probing suggest FF shows promise

Domain Specific Heuristics

- Golomb ruler
  - index order (!)
- Stable marriage (maybe not a heuristic)
  - value ordering!
- Jobshop/Factory scheduling
  - texture based heuristics
  - slack based heuristics
- Car Sequencing Problem
  - various (see literature)
- Bin packing
  - first-fit decreasing
- … the quest goes on

But remember, heuristic can play havoc with symmetry breaking

Domain Specific Heuristics

- Consider HC
    - different models
    - different heuristics?

AR33: section 5 (pages 27-29) and section 8 (pages 47-49)

Big question: why do heuristics work?

Is a heuristic similar to an umbrella lent to you by the bank?