Visualizations to Explain Algorithm Performance

Carston Wiebe, **Andrew Kula**, Simon Schoenbeck, and Berthe Y. Choueiry
Constraint Systems Laboratory • School of Computing

Context and Contributions

Context: Constraint Programming (CP) is powerful paradigm for modeling and solving complex decision problems in management and engineering, including resource allocation, scheduling, product configuration, and design.

These problems are represented as Constraint Satisfaction Problems (CSPs), which are NP-complete, making their solving cost difficult to predict.

Contributions: We design visualizations that summarize the behavior of the CP solver to

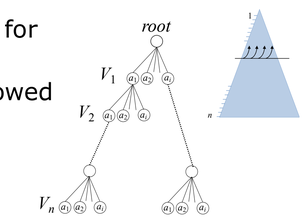
1. Explain the cost of the solver
2. Allow the users to design new algorithms and heuristics

Solving CSPs with Search

Constraint Satisfaction Problem: Given

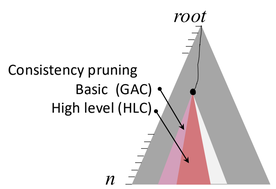
- A set of *decisions* (variables), *options* for each decision (values), and
- A set of *constraints* restricting the allowed combinations of options to decisions (combinations of values to variables),

Determine if the problem has a solution that satisfies all constraints
Example: a Sudoku puzzle



Algorithms

1. *Backtrack search*: Enumerates all combinations of values for variables
2. *Ordering heuristics*: Choose which variable to instantiate first
3. *Consistency algorithms*: Prune inconsistent combinations



Parameters and Visualizations

Cost of search

1. Number of variable instantiations: #NV
2. Number of backtracks: #BT
3. Number of calls to a consistency algorithm
4. CPU time

Visualizations

1. BpD: Number of backtracks per depth
2. VIpD: Number of variable instantiations per depth
3. CpD: Number of calls to a consistency algorithm per depth
   - #Wipeouts: Consistency prunes the entire subtree (super effective)
   - #Filter: Consistency reduces size of subtree (beneficial)
   - #No-filter: Consistency does not affect subtree (wasted efforts)

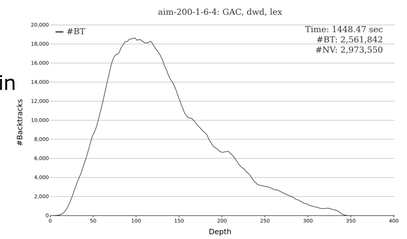
High-Level Consistency

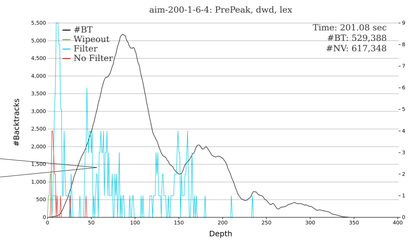
Solving aim-200-1-6-4

- GAC, the basic consistency algorithm, takes more than 24 min
- POAC, a high-level consistency applied reactively because it is expensive, takes less than 4 min

Consistency	Basic GAC	HLC, reactively POAC
CPU time	1,448.5 sec	201.08 sec
#BT	2,561,842	529,.388
#NV	2,973,550	617,348
max #BT	18,632	5,185

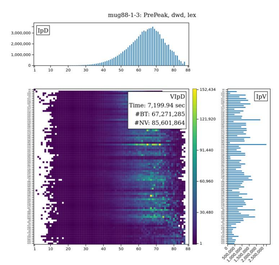
#Calls to (expensive) HLC
- Wipeouts: super effective
- Filter: beneficial
- No filter: wasted efforts





Ordering Heuristics

mug88-1-3 (UNSAT)

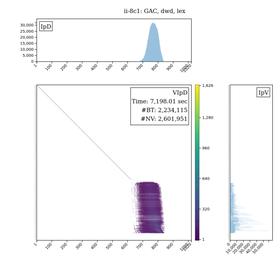


`dwd`, currently the leading dynamic variable ord. heuristic, is sometime erratic

`dd`, an older dynamic variable ord. heuristic, is sometimes more stable than `dwd`

`MaxClique`, our static variable ord. heuristic, exploits the problem's structure and performs best

ii-8c1 (SAT)



`dwd`, again, is erratic: search does not terminate within two hours

Reactively applying `PVal0` [Geelen 92],a costly value ord. heuristic, terminates in less than 10 seconds

`MaxClique`, our static variable ord. heuristic, exploits the structure of the CSP and solves this instance *backtrack-free*!

Future Work

- Visualize the constraint graph
- Coordinate various visualizations
- Generate animations of significant search regimes

Acknowledgments: Supported by UCARE; experiments run on HCC

April 7, 2026