DPML Software Report

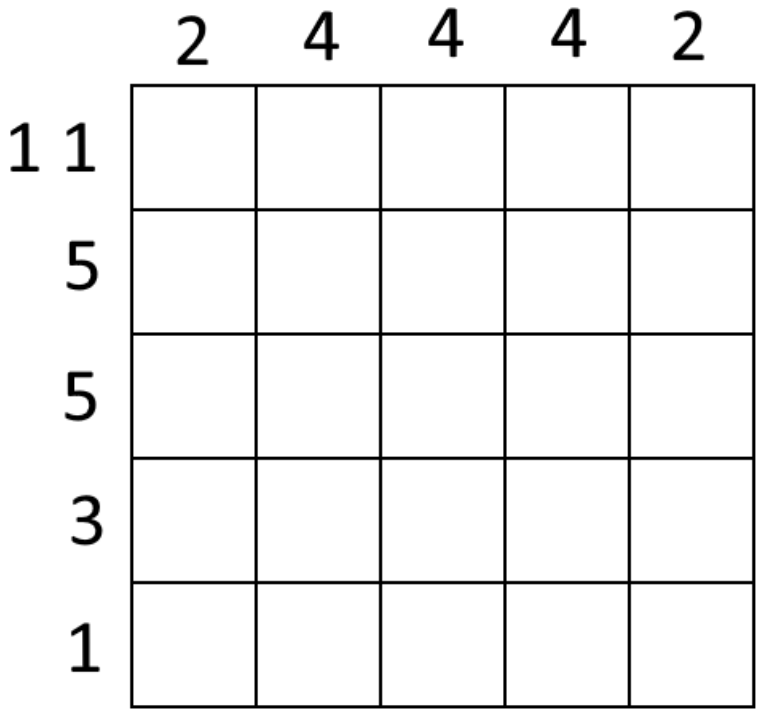
Analyzing the impact of reduce and search algorithm on different CSP problems

Liviu-Ștefan Neacșu-Miclea
ICA 256/2

Agenda

- Introducing the problems
- CSP system design overview
- Reducing algorithms
- Search algortihms (Solvers)
- Evaluation method
- Experiments
- Discussion
- Conclusion

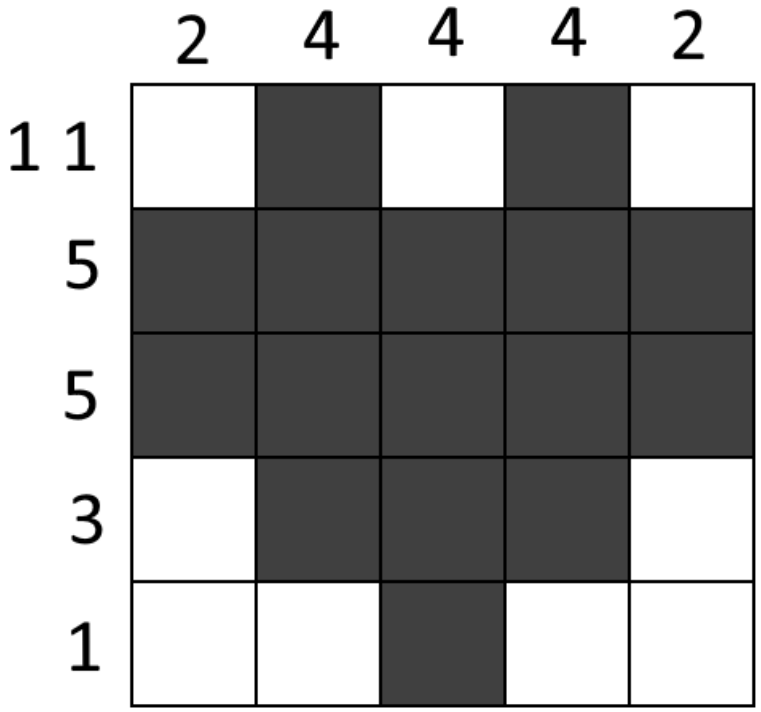
Introducing the problems (1) Picross/Nonogram puzzle



- $M \times N$ grid fill-in puzzle
- Clues on each row/column
  - Fill in with black cells based on a list of number associated to each row/column that describe the lengths of continuous black stripes separated by white spaces what exist on that row/column

- the set of variables $Z = \{z_{ij} | 1 \le i \le N, 1 \le j \le M\}$
- the domain mapping $D(z) = \{0,1\}\ \forall\ z \in Z$
- the set of constraints C, defined as follows:
  - $\forall\ (s_1, \dots, s_k) \in R_i, c_{R_i} \in C$, where
    - c_{R_i} constraints the variables $z_{i1}, \dots, z_{iM}$
    - to fulfill $\forall p \in \{1, \dots, k\}\ \exists a_p, b_p \in \{1, \dots M\}, a_p + s_p = b_p, \sum_{j=a_p}^{b_p - 1} z_{ij} = s_p,$ $(p > 1 \wedge a_p > b_{p-1}),$
  - $\forall\ (s_1, \dots, s_k) \in C_j, c_{C_j} \in C$, where
    - c_{C_j} constraints the variables $z_{1j}, \dots, z_{Nj}$
    - to fulfill $\forall p \in \{1, \dots, k\}\ \exists a_p, b_p \in \{1, \dots N\}, a_p + s_p = b_p, \sum_{i=a_p}^{b_p - 1} z_{ij} = s_p,$ $(p > 1 \wedge a_p > b_{p-1}),$
  - no other constraints belong to C.

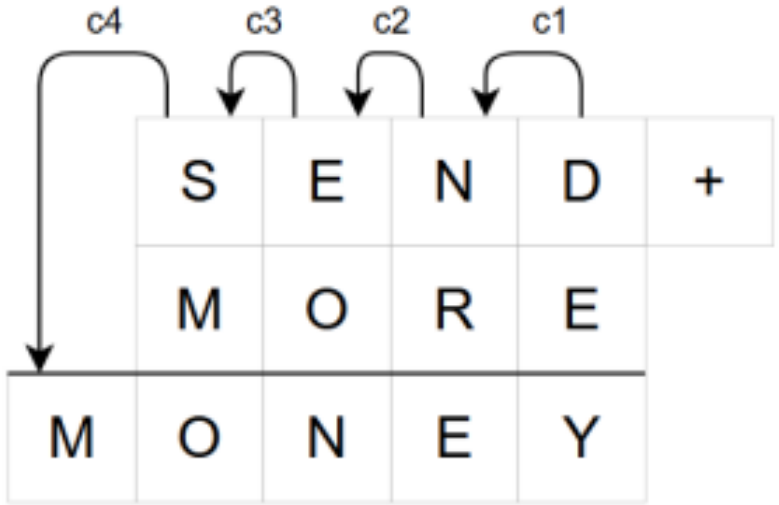
Introducing the problems (1) Picross/Nonogram puzzle



- $M \times N$ grid fill-in puzzle
- Clues on each row/column
  - Fill in with black cells based on a list of number associated to each row/column that describe the lengths of continuous black stripes separated by white spaces what exist on that row/column

- the set of variables $Z = \{z_{ij} | 1 \leq i \leq N, 1 \leq j \leq M\}$
- the domain mapping $D(z) = \{0,1\}\ \forall\ z \in Z$
- the set of constraints C, defined as follows:
  - $\forall\ (s_1, \dots, s_k) \in R_i, c_{R_i} \in C$, where
    - c_{R_i} constraints the variables $z_{i1}, \dots, z_{iM}$
    - to fulfill $\forall p \in \{1, \dots, k\}\ \exists a_p, b_p \in \{1, \dots M\}, a_p + s_p = b_p, \sum_{j=a_p}^{b_p - 1} z_{ij} = s_p,$ $(p > 1 \wedge a_p > b_{p-1})$,
  - $\forall\ (s_1, \dots, s_k) \in C_j, c_{C_j} \in C$, where
    - c_{C_j} constraints the variables $z_{1j}, \dots, z_{Nj}$
    - to fulfill $\forall p \in \{1, \dots, k\}\ \exists a_p, b_p \in \{1, \dots N\}, a_p + s_p = b_p, \sum_{i=a_p}^{b_p - 1} z_{ij} = s_p,$ $(p > 1 \wedge a_p > b_{p-1})$,
  - no other constraints belong to C.

Introducing the problems (2) Paleoarithmetics puzzle



- Replace each of the letters S, E, N, D, M, O, R, Y with a different digit such that the equality SEND+MORE=MONEY holds, and the numbers are not starting with trailing zeros

- the set of variables $Z = \{S, E, N, D, M, O, R, Y, c_1, c_2, c_3, c_4\}$
- the domain mapping
  - $D(z) = \{1, \dots, 9\}, z \in \{S, M\}$
  - $D(z) = \{0, \dots, 9\}, z \in \{E, N, D, O, R, Y\}$
  - $D(z) = \{0,1\}, z \in \{c_1, \dots, c_4\}$
- the set of constraints $C = \{C_1, \dots, C_6\}$
  - $C_1: D + E = Y + 10 \cdot c_1$
  - $C_2: N + R + c1 = E + 10 \cdot c_2$
  - $C_3: E + O + c_2 = N + 10 \cdot c_3$
  - $C_4: S + M + c_3 = O + 10 \cdot c_4$
  - $C_5: M = c4$
  - $C_6: \forall x, y \in \{S, E, N, D, M, O, R, Y\}, x \neq y$

Introducing the problems (3)
Dinosaurs Mystery – bonus problem

- S1: Skeletons of Eucentrosaurus, Hadrosaurus, Herrerasaurus, Megasaurus, and Nuoerosaurus have been found in Argentina, Canada, China, England, and the U.S.
- S2: None of these dinosaurs has been found in more than one of these countries.
- S3: The dinosaur from China is the largest of the bunch.
- S4: Herrerasaurus was the smallest.
- S5: Megasaurus was larger than Hadrosaurus or Eucentrosaurus.
- S6: The dinosaurs from China and North America were planteaters.
- S7: Megasaurus ate meat.
- S8: Eucentrosaurus lived in North America.
- S9: The dinosaurs from North America and England lived later than Herrerasaurus.
- S10: Hadrosaurus lived in Argentina or the U.S.

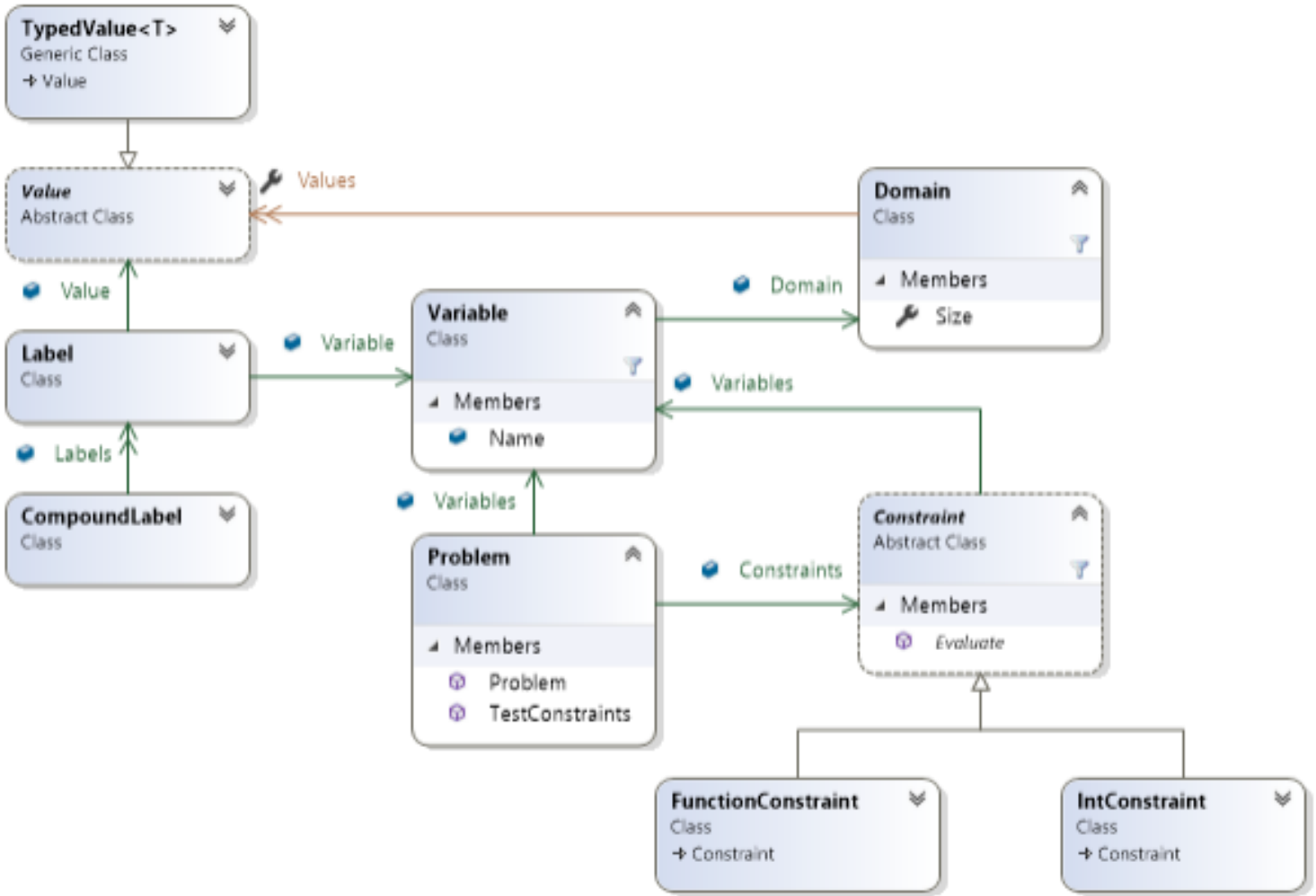
- The set of variables $Z = \{c_{Euc}, c_{Had}, c_{Her}, c_{Meg}, c_{Nuo}, s_{Euc}, s_{Had}, s_{Her}, s_{Meg}, s_{Nuo}\}$
- The domain mappings D, such as
  - $D(c) = D_c = \{Argentina, Canada, China, England, US\}, \forall c \in Z_c = \{c_{Euc}, \dots c_{Nuo}\}$
  - $D(s) = D_s = \{1,2,3,4,5\} \forall s \in Z_s = \{s_{Euc}, \dots, s_{Nuo}\}$

- The sets of constraints $C = \{C_1, \dots, C_{10}\}$:
  - $C1: \forall c_1, c_2 \in Z_c, c_1 \neq c_2$
    (S2: no two dinosaurs were found in the same country)
  - $C2: \forall s_1, s_2 \in Z_s, s_1 \neq s_2$
    (assume unique sizes)
  - $C3: c_i = China \rightarrow s_i > s_j, \forall j \neq i$
    (S3: China dinosaur is the largest)
  - $C4: s_{Her} = 1$
    (S4: Herrerasaurus was the smallest)
  - $C5: s_{Meg} > s_{Had}$
  - $C6: s_{Meg} > s_{Euc}$
    (S5 splitted in 2 binary constraints)
  - $C7: c_{Meg} \notin \{China, Canada, US\}$
    (S6 & S7: Megasaurus ate meat, therefore was not a plant eater, so it can't be from China and North America = CA+US)
  - $C8: c_{Euc} \in \{Canada, US\}$
    (S8)
  - $C9: c_{Her} \notin \{Canada, US, England\}$
    (S9 "The dinosaurs from North America and England lived later than Herrerasaurus" is not about time at all, it actually suggests that Herrerasaurus didn't live in NA or En)
  - $C10: c_{Had} \in \{Argentina, US\}$
    (S10)

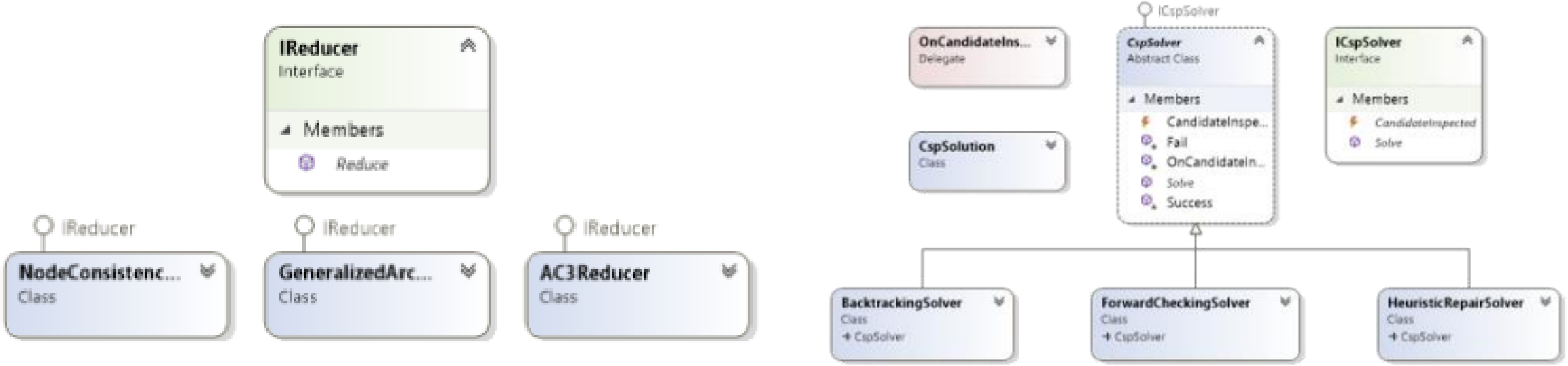
Introducing the problems
Motivation of choice

	Picross	**Arithmetics**	**Dinosaurs**
Origin	Personal idea	Lecture notes	Lecture notes
Difficulty	⭐ ⭐	⭐ ⭐ ⭐	⭐
Unary constraints			✓
Binary constraints		✓	✓

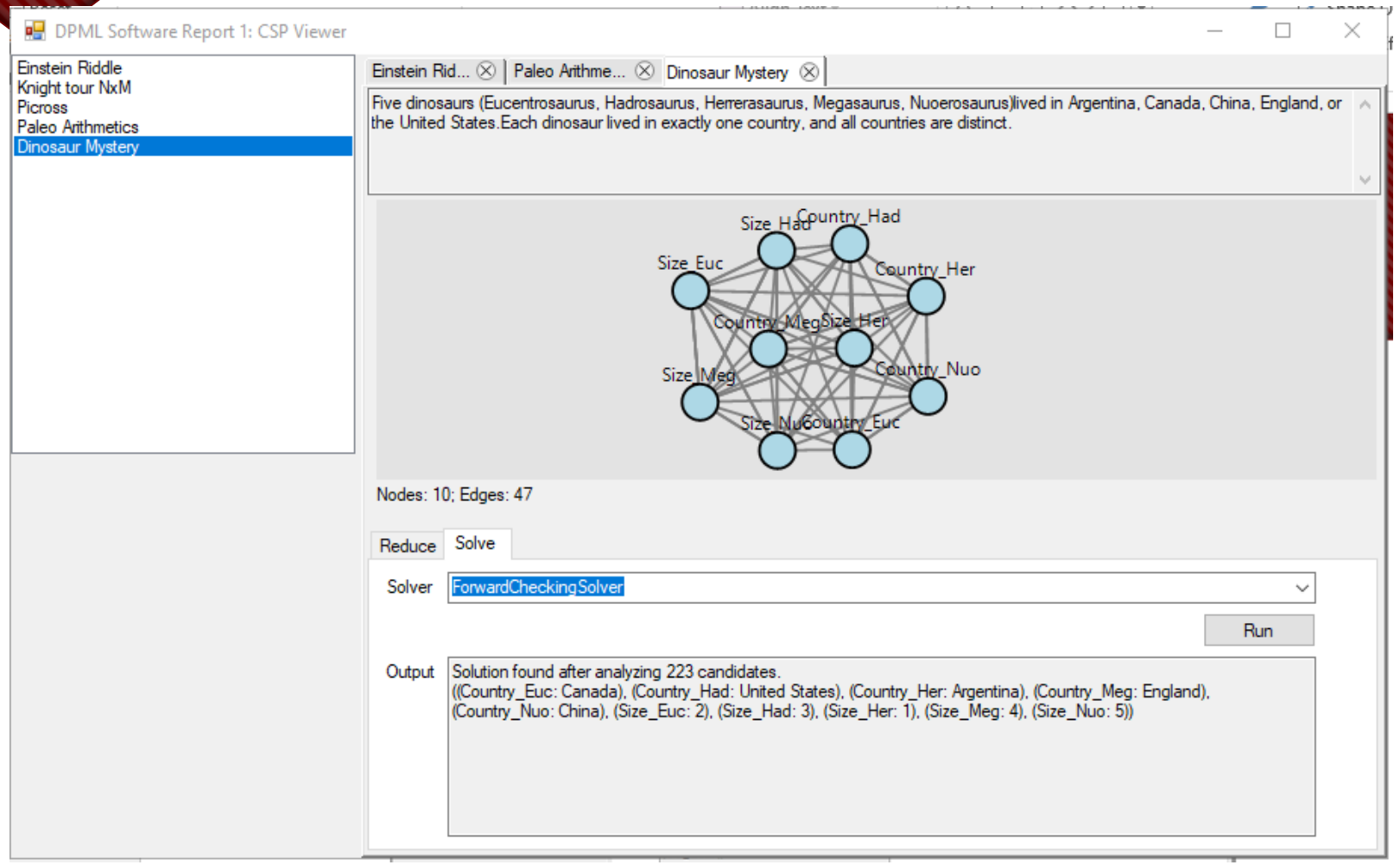
CSP System Design overview (1)



CSP System Design overview (2)



CSP System Design overview (3)



CSP System Design overview (4)

```
// very simple example on formal declaration of a problem
var problem = new ProblemBuilder()
    .Domain(Domain.IntRange(1, 5), out var dom)
    .Variable("x", dom, out var x)
    .Variable("y", dom, out var y)
    .Variable("z", dom, out var z)
    .Constraint(Constraint.Of(x, y).IntegerRelation((xVal, yVal) => xVal + yVal == 6))
    .Constraint(Constraint.Of(x, y).IntegerRelation((xVal, yVal) => xVal < yVal))
    .Constraint(Constraint.Of(z).IntegerRelation((zVal) => zVal > 3))
    .Build();

// Instantiate the Zebra problem for test purposes
problem = ProblemGeneratorTemplates.Riddle3H3N.GenerateProblem();

// example on applying reducers and solve the problem
var solution = problem
    .Reduce<NodeConsistencyReducer>()
    .Reduce<AC3Reducer>()
    .Reduce<GeneralizedArcConsistencyReducer>()
    .Solve<ForwardCheckingSolver>(onCandidateInspected: (label) => { Debug.WriteLine(label); })
    ;
```

Reducing algorithms

$$Problem \xrightarrow{Reducer} Problem'$$

All implemented algorithms focus on local consistency only.

NC	AC-3	GAC
Enforces consistency on **unary** constraints (domain of each variable individually)	Enforces consistency on **binary** constraints between pairs of variables	Extends arc consistency to **n-ary** constraints. Every value in a variable's domain must have some support across all other variables appearing in that constraint

Search algorithms

$$Problem \xrightarrow{Solver} Solution$$

Backtracking	**Forward Checking (FC-1)**	**Heuristic Repair**
(baseline method)	*Basic search method*	*Stochastic search method*
Depth-first search that assigns variables one by one and backtracks whenever a constraint is violated; no look-ahead.	After each assignment, eliminates inconsistent values from the domains of future variables; detects dead ends earlier than simple backtracking.	Starts with a random full assignment and iteratively fixes conflicts by locally adjusting variable values; does not systematically explore the search tree.

Evaluation method

- For each problem and solver, solve:
  - The unreduced problem
  - Problem reduced by NC
  - Problem reduced by AC-3
  - Problem reduced by GAC
  - Problem reduced by a relevant chain of algorithms
- Problem-specific metrics
  - Constraint graph size (nodes & edges)
- Reducers performance
  - Average domain size: $\frac{1}{|Z|}\sum_{z\in Z}|D(z)|$
  - Maximum domain size: $\max_{z\in Z}|D(z)|$
  - Search space size: $\prod_{z\in Z}|D(z)|$
- Solvers performance
  - Number of evaluated candidates
  - Failure rate of stochastic method

Experiments (1)

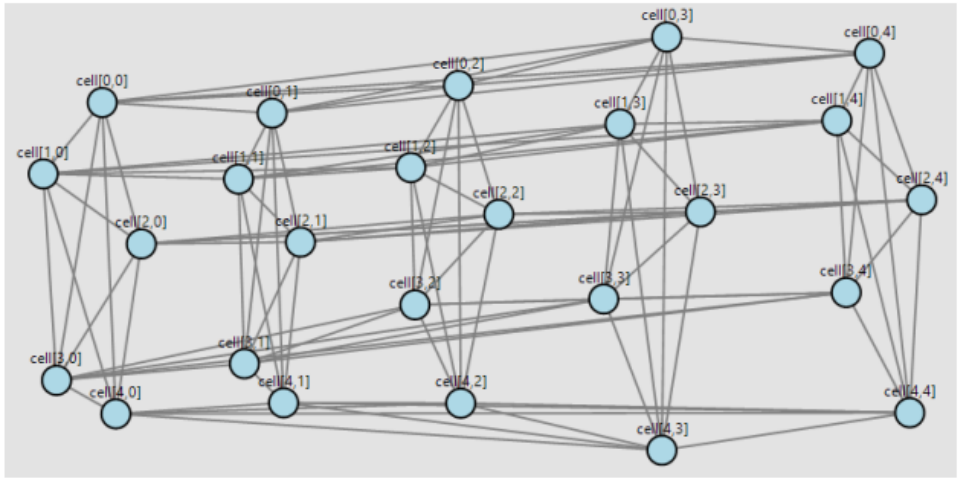


Fig.8. Picross constraint graph

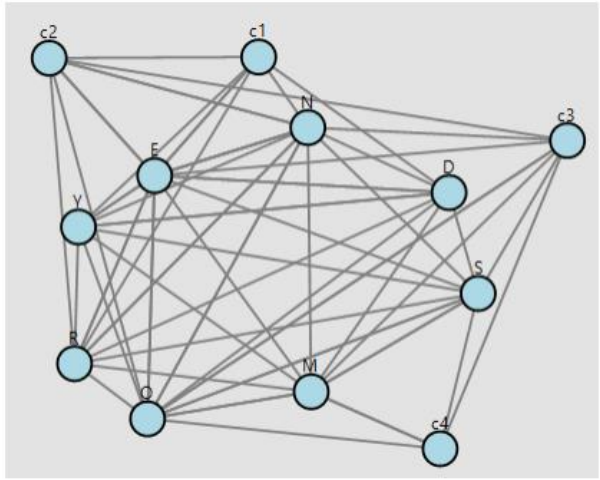


Fig.9. Paleo-arithmetics constraint graph

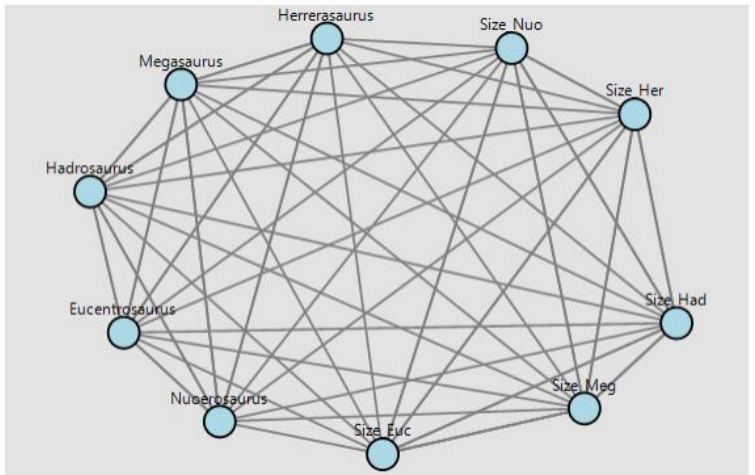


Fig.10. Dinosaurs problem constraint graph

Experiments (2)

Constraint graph size (Picross 5x5)					
Nodes	25		**Edges**	100	
Algorithms performance					
Reducer	**None**	**NC**	**AC3**	**GAC**	**GAC+GAC**
Domain Size Avg	2	2	2	1.24	1
Domain Size Max	2	2	2	2	1
Search Space	33554432	33554432	33554432	64	1
Backtracking steps	4685803	4685803	4685803	22	1
FC steps	232	232	232	33	25
HR min steps (10 runs)	125	16	67	3	1
HR avg steps	172	117	188	13	1
HR failures (10 runs)	5	5	7	7	0

Table.1. Picross metrics

Constraint graph size (Dinosaurs problem)							
Nodes	10			**Edges**	47		
Algorithms performance							
Reducer	**None**	**NC**	**AC3**	**GAC**	**NC+AC3**	**NC+GAC**	**GAC+ GAC+ GAC**
Domain Size Avg	5	3.4	4.7	3.1	3.1	2.2	1.8
Domain Size Max	5	5	5	5	5	4	3
Search Space	9765625	50000	5000000	25600	25600	864	81
Backtracking steps	9007897	46603	4520397	23563	23563	735	65
FC steps	223	120	217	114	114	20	15
HR min steps	16	26	53	38	14	2	2
HR avg steps	62	61	79	44	80	36	27
HR failures (10 runs)	6	6	6	8	7	0	0

Table.3. Dinosaur problem metrics

Constraint graph size (SEND+MORE=MONEY)						
Nodes	12			**Edges**	55	
Algorithms performance						
Reducer	**None**	**NC**	**AC3**	**GAC***	**AC3+ GAC**	**GAC*+ GAC**
Domain Size Avg	7.16	7.16	6.41	5.83	4.25	4.25
Domain Size Max	10	10	10	9	8	8
Search Space	1296000000	1296000000	72000000	34012224	524288	524288
Backtracking steps	913686894	913686894	29526894	13598680	203464	203464
FC steps	1197261	1197261	1128725	672929	12648	12648
HR min steps	1010	-	-	-	11	151
HR avg steps	1010	-	-	-	89	212
HR failures (10 runs)	9	10	10	10	7	8

Table.2. Paleo-arithmetics metrics (*=takes more time to compute)

Discussion (1)

- **Large search spaces** heavily disadvantage naive backtracking and stochastic search, especially without variable-ordering heuristics.
- **Reducers vary in impact**
  - NC & AC only help when unary or binary structure allows pruning; otherwise no effect.
  - Structure-dependent: binary expansion of an **all-different constraint** prevents AC from pruning
    - E.g. $X, Y \in \{1,2,3\}$, $CXY: X \neq Y$ ⇒ for each value of X there always exists a value of Y that ensures consistency ⇒ AC fails to reduce any domains
  - **GAC provides the strongest pruning**: down to 0.0001–2% of original domains in some puzzles, but may become expensive when constraints span many variables (can emulate full backtracking).
- **Combining reducers** improves results: NC → AC → GAC is generally effective.
  - AC before GAC often lowers GAC's cost.
  - Repeated GAC calls can drastically shrink domains (e.g., dinosaurs down to 3 candidates).
  - Best reducer order **depends on the problem's structure**.

Discussion (2)

- Backtracking struggles on unreduced problems, often exploring >70% of the search space
- FC-1 provides major improvements thanks to look-ahead: steps reduced to 0.0006–0.13% of backtracking in tests.
- Heuristic repair (local search) is fast but prone to failure:
  - High failure rates on puzzles with unique solutions (often 7–10 failures / 10 runs).
  - Performance depends on solution density and constraint flexibility
- Stochastic methods excel in domains where a "good enough" solution is acceptable (e.g., scheduling with preferences, old graphics memory allocation).

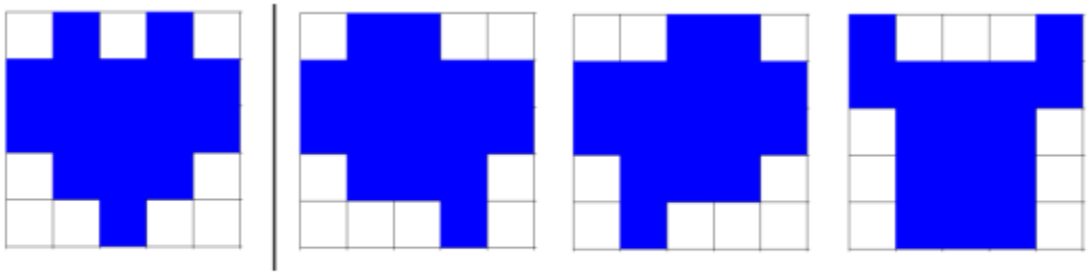


Fig.11. Picross puzzle: the correct solution vs. possible local optimum failure points reached by HR

- Overall: deterministic search benefits most from strong pruning; stochastic solvers benefit from flexible problem spaces rather than strict CSP structure.

Conclusion

- Domain-consistency reduction is key for efficient CSP solving; GAC delivers the strongest pruning, often shrinking huge search spaces to manageable size.
- Combining reducers yields strong synergy, especially when applied by increasing constraint arity.
- HR is fast but requires flexible or redundant problems
- FC offers speed with reliability
- Reduction strategy needs tweaking based on the specific problem

- Future work
  - Improve the framework to support path consistency
  - Include variable ordering strategies
  - Introduce additional evaluation metrics: GAC support tests, memory usage, search-tree depth/branching, constraint violations in stochastic methods
  - Adapt solvers to target completeness, not only soundness

Thank you for your attention!

References

- Roucairol, M., & Cazenave, T. (2024, November). Generating Difficult and Fun Nonograms. In International Conference on Computers and Games (pp. 119-129). Cham: Springer Nature Switzerland.
- Pop, H.F., (2025). 02b problems-1 [Lecture notes]. Declarative programming in Machine Learning, Babeș-Bolyai University of Cluj-Napoca
- Pop, H.F., (2025). 02b problems-2 [Lecture notes]. Declarative programming in Machine Learning, Babeș-Bolyai University of Cluj-Napoca
- Kondrak, G., & Van Beek, P. (1997). A theoretical evaluation of selected backtracking algorithms. Artificial Intelligence, 89(1-2), 365-387.
- Tsang, E. (1993). Foundations of Constraint Satisfaction. Academic Press.