

Constraint problems

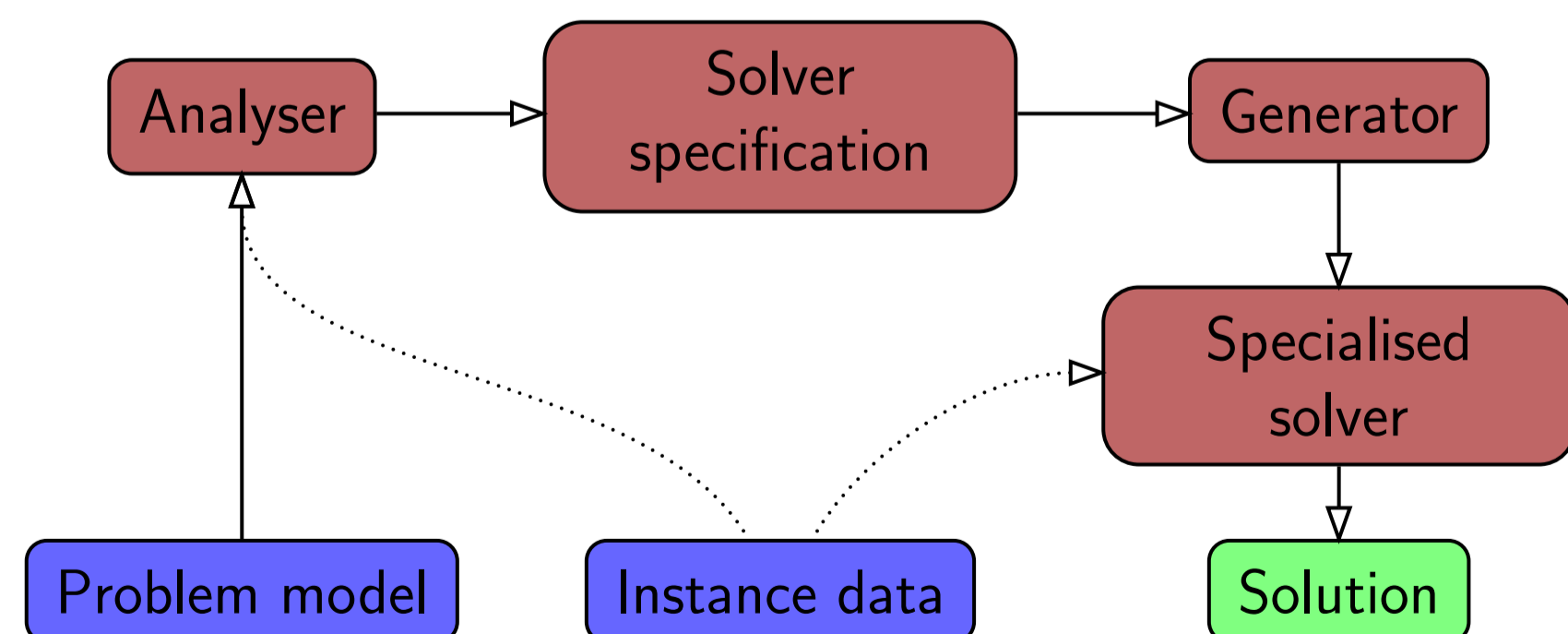
- ▷ **combinatorial** problems which take **exponential** search to solve
- ▷ only solvable by computers by throwing **Artificial Intelligence** at the problem
- ▷ for example the **8-queens puzzle** – how to place 8 queens on a chess board such that no queen is attacking another queen
- ▷ **instances** of problems (e.g. 8-queens) and **classes** of problems with parameters (e.g. n-queens)

Idea – constraint solver synthesis

- ▷ constraint problems are solved by **monolithic software systems** which are able to handle many different problem classes
- ▷ by synthesising solvers, we can **vary design decisions** which affect performance significantly[3] depending on the problem
- ▷ this can make the difference between being able to solve the problem and not being able to do so

Dominion architecture

- ▷ two main parts – **Analyser** and **Generator**
- ▷ the two parts are linked by the **Solver specification**, which contains the result of the analysis and the information the generator needs to synthesise a solver
- ▷ solvers can be synthesised for **problem classes** as well as **problem instances** with a higher level of specialisation

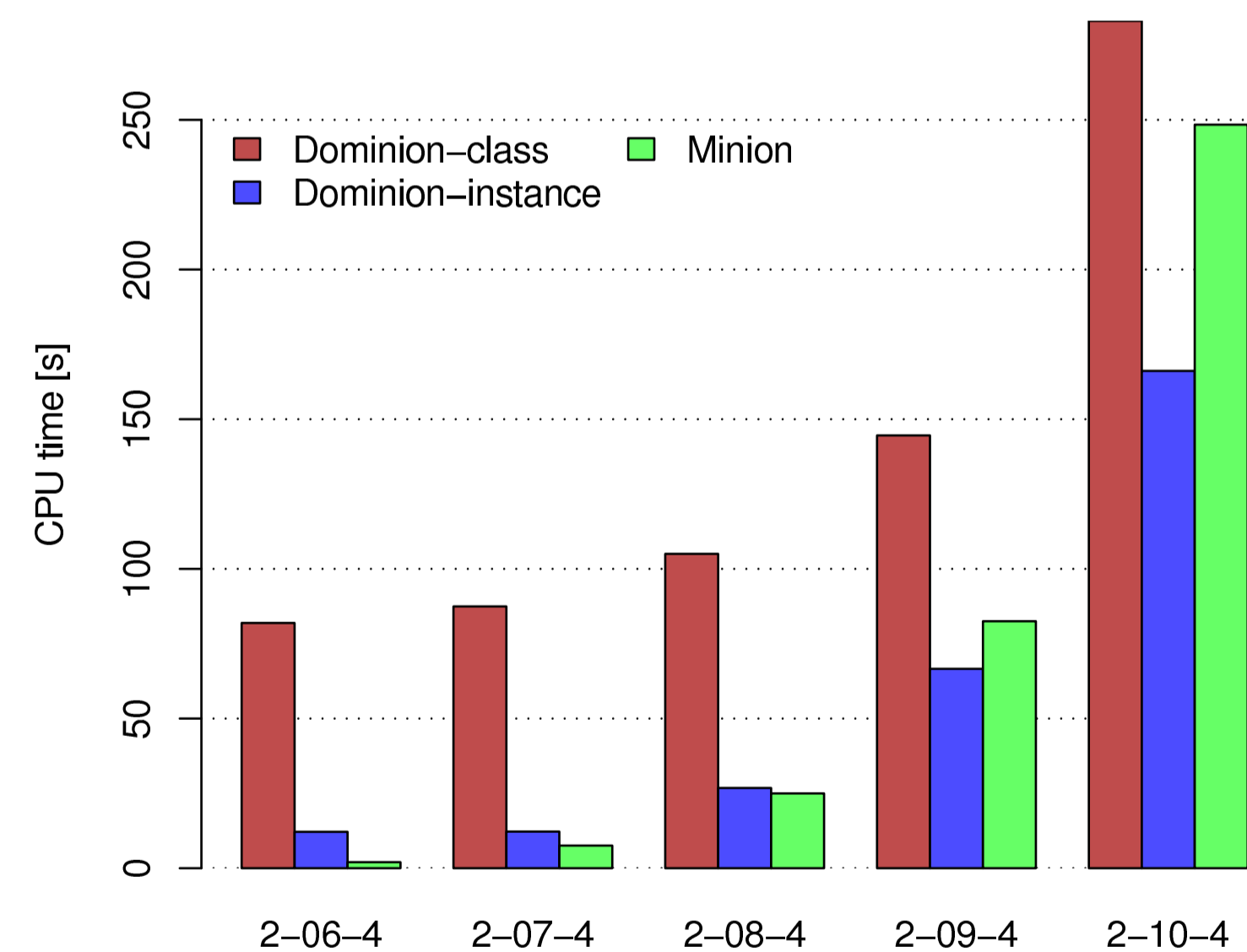


- ▷ the instance data is required by the Analyser or the specialised solver depending on whether an instance solver or a class solver is generated

Prototype

- ▷ implementation based on the **Minion**[1] constraint solver
- ▷ the **Analyser** determines the constraints and variable types the problem model requires
- ▷ the **Solver specification** encodes this knowledge and the problem model
- ▷ the **Generator** modifies the Minion source code –
 - ▷ removes modules and infrastructure code which is not required,
 - ▷ generates new variable types for every domain size,
 - ▷ encodes the problem model in file which can be compiled into the solver
- ▷ the modified source can be compiled into
 - ▷ a **class solver**, which is specialised for the analysed problem model and similar ones and retains the input file parser
 - ▷ or an **instance solver**, which, when run, solves the analysed problem
- ▷ the modified solver still finds the correct solutions and compiles several orders of magnitude faster than standard Minion

Experimental results



- ▷ graph shows results for instances of the Social golfers problem
- ▷ numbers include generation time and solve time
- ▷ results for other problem classes are similar
- ▷ specialised solver is **significantly faster** than standard Minion
- ▷ for large problems, the prototype **wins** even taking the time it takes to generate the specialised solver into account

Towards a general implementation

- ▷ need an **input language** that is able to specify both problem **classes** and **instances** to be analysed
- ▷ current **specification** version 0.2
- ▷ implemented **parser**, **type checker**, and **converter** which allows Minion to solve problems specified in the new language

```

language Dominion 0.2
given n: int {1..}
letting sum = n*(n*n+1)/2
dim matrix[n,n]: int
find matrix[...]: int {1..n*n}
such that
rows [ sum(matrix[i,0..n-1], sum) | i in {0..n-1} ]
  
```

Problem analysis

- ▷ currently investigating using **Machine Learning** for problem analysis
- ▷ attributes of problem instances used to determine whether or not to use **conflict learning**
- ▷ learning comes with a **high overhead** but can give a **significant speedup**

Future work

- ▷ have the Analyser add **annotations** to the problem specification to drive the Generator – the **Solver specification**
- ▷ **architecture** of the specialised solver
- ▷ **componentisation** of a constraint solver and **code generation** for the Generator

References

- [1] Ian P. Gent, Chris Jefferson, and Ian Miguel, **MINION: A fast scalable constraint solver**, ECAI 2006, pp. 98–102.
- [2] Lars Kotthoff, **Dominion – a constraint solver generator**, Doctoral Program of CP 2009.
- [3] Lars Kotthoff, **Constraint solvers: An empirical evaluation of design decisions**, CIRCA preprint, 2009, <http://www-circa.mcs.st-and.ac.uk/Preprints/solver-design.pdf>.
- [4] Steven Minton, **Automatically configuring constraint satisfaction programs: A case study**, Constraints 1 (1996), 7–43.