## Motivation

- **Challenge**: Solving complex reasoning problems on large data.
- Declarative approaches make succinct specifications possible but runtime is still an issue. What to do?
- Instances in the real world usually exhibit structure.
- Graphs occurring in practice are normally not totally random.
- Many problems are easy on trees.
- Can this help us for “tree-like” graphs?

## Background

**Answer Set Programming (ASP)** is a logic programming language that allows for succinct specifications via “Guess & Check.”

Example encoding for 3-COLORABILITY:

```prolog
color(red;green;blue).
1 (map(V,C) : color(C) | 1 ← vertex(V).
   ← edge(X,Y), map(X,Y,C).
```

**Treewidth** is a parameter that measures “tree-likeness” of graphs.

**Tree decompositions** break graphs down into pieces. In many cases, this makes it possible to solve a problem by combining partial solutions to subproblems.

**Dynamic programming** can often be applied to tree decompositions to compute and combine partial solutions – the smaller the treewidth the better.

- Runtime of such algorithms usually exponential in general
- ... but for bounded treewidth often linear in the input size

**Figure**: An example instance of **Minimum Vertex Cover** (left), one of its tree decompositions (middle) and the computed tables (right)

## Shortcomings of the State of the Art

- Frameworks exist
  - allowing to solve problems on tree decompositions
  - but not in a declarative way.
- Declarative programming paradigms exist
  - making succinct specifications of hard problems possible
  - but offer no easy way to exploit decomposition.
- Systems that evaluate logical formulas exist
  - exploiting decomposition
  - allowing a declarative problem specification by means of logic
  - but they offer only a very restricted language.
- None of them combine the benefits of decomposition and ASP.

## Main Contributions

- We propose “Decompose, Guess & Check”
  - Approach that combines decomposition with ASP
  - Idea: Call ASP solver in each tree decomposition node
  - We address the questions...
    - How to give child tables to ASP?
    - How to obtain the new table from the answer sets?
- We show it to be applicable to a wide range of problems.
  - Currently supported: decision, counting, enumeration and optimization
  - Can solve any problem definable in monadic second-order logic
  - Not just applicable when instances are graphs but, more generally, when they can be represented as graphs
- We present an implementation called D-FLAT.

## The D-FLAT System

- A framework implementing “Decompose, Guess & Check”
  - Takes care of tasks unrelated to the actual problem
  - Lets the user declare the problem-specific algorithm with ASP
  - Communication with the framework via special predicates
- Calls ASP solver in each tree decomposition node as follows:

```
instance ← TreeDecomposition.
Answer Sets ← Decompose(instance).
```

**Encoding for 3-COLORABILITY** (input / output predicates highlighted):

```prolog
color(red;green;blue).
1 (map(V,C) : color(C) | 1 ← vertex(V).
   ← edge(X,Y), map(X,Y,C).
```

## Future Work

- “Lazy evaluation” of tables
  - Sometimes not all tables must be computed in their entirety.
  - Might also allow to eliminate solutions known to be suboptimal
- Integrate features for debugging and testing
- Exploit multi-core architectures by parallelization

## References

- Bernhard Bliem, Michael Morak, and Stefan Woltran.
  D-FLAT: Declarative problem solving using tree decompositions and answer-set programming.
- Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski.
  Answer set programming at a glance.
- Rolf Niedermeier.
  *Invitation to Fixed-Parameter Algorithms.*