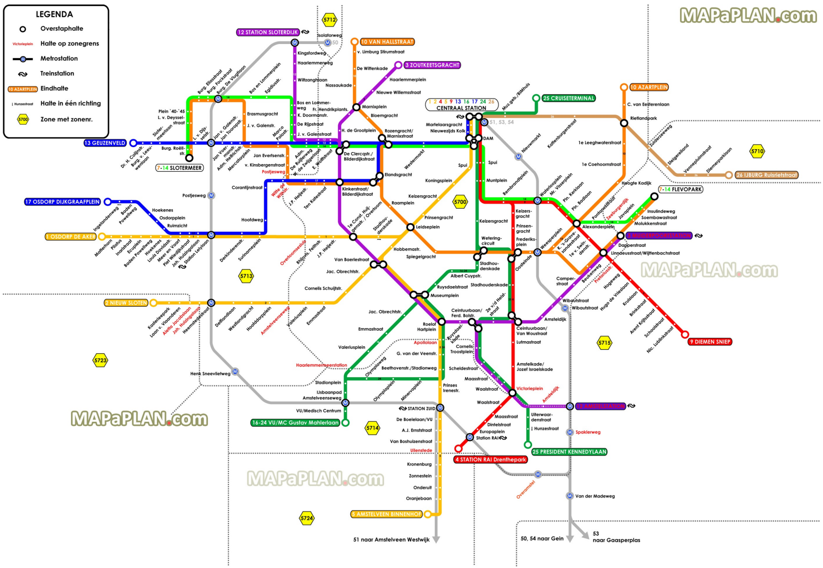
Dynamic Programming on Tree Decompositions in Practice

Stefan Woltran

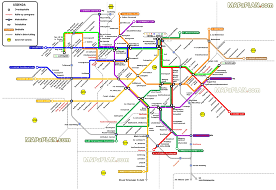
TU Wien (Vienna University of Technology)

August 30, 2016

Graphs are Everywhere ...

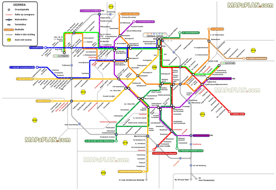


Let’s Decompose them ...

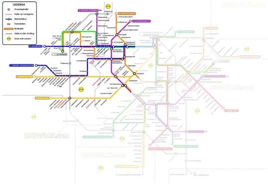


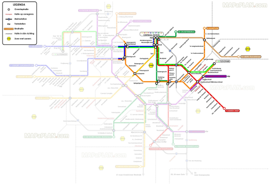
$$\Longrightarrow$$

Let’s Decompose them ...



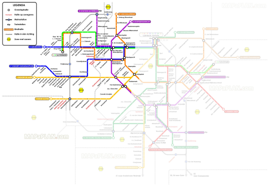
$$\Longrightarrow$$

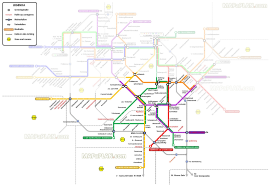


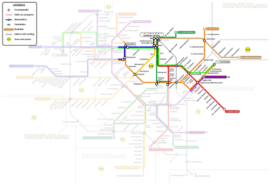


Let’s Decompose them ...

$\Longrightarrow$

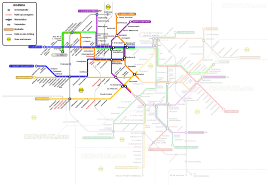


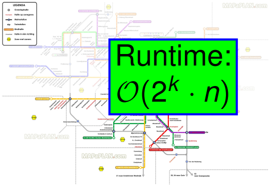


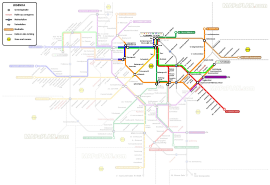


Let's Decompose them ...

$$\Longrightarrow$$

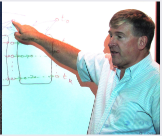






The Whole Story in 3 Minutes ...

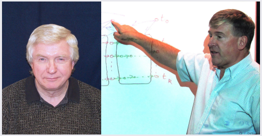
Tree Decomposition and Treewidth

By-product in the theory of graph minors due to Robertson and Seymour (1984); similar notions appeared even earlier (Bertelè and Brioschi, 1972; Halin, 1976).

The Whole Story in 3 Minutes ...

Tree Decomposition and Treewidth



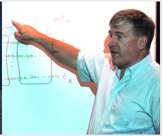
By-product in the theory of graph minors due to Robertson and Seymour (1984); similar notions appeared even earlier (Bertelè and Brioschi, 1972; Halin, 1976).

Courcelle's Theorem (1990)

Any property of finite structures which is definable in MSO can be decided in time $O(f(k) \cdot n)$ where n is the size of the structure and k is its treewidth.

The Whole Story in 3 Minutes ...

Tree Decomposition and Treewidth

By-product in the theory of graph minors due to Robertson and Seymour (1984); similar notions appeared even earlier (Bertelè and Brioschi, 1972; Halin, 1976).

Courcelle's Theorem (1990)

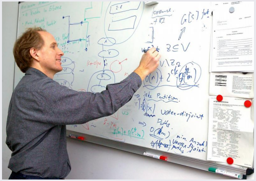
Any property of finite structures which is definable in MSO can be decided in time $O(f(k) \cdot n)$ where n is the size of the structure and k is its treewidth.

SEQUOIA (2011)

A system developed by Rossmanith's group at RWTH Aachen; SEQUOIA takes a graph and MSO description of problem and does decomposition and dynamic programming "inside".

The Whole Story in 3 Minutes ...

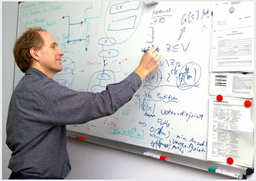
But ...



"*. . . rather than synthesizing methods indirectly from Courcelle's Theorem, one could attempt to develop practical direct methods.*" (Niedermeier, 2006)

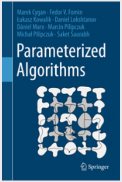
The Whole Story in 3 Minutes ...

But ...



"...*rather than synthesizing methods indirectly from Courcelle's Theorem, one could attempt to develop practical direct methods.*" (Niedermeier, 2006)

... and, more recently ...



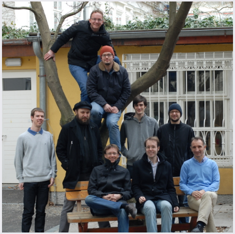
"*Courcelle's theorem [...] should be regarded primarily as classification tool, whereas designing efficient dynamic programming routines on tree decompositions requires 'getting your hands dirty' and constructing the algorithm explicitly.* " (Cygan et al., 2015)

The Whole Story in 3 Minutes ...

Our Vision

A system that

- supports declarative specifications of dynamic programming on tree decompositions
- performs reasonably efficient
- bothers the user only with the actual algorithm design



Quick thanks to all collaborators...

Michael Abseher, *Bernhard Bliem*, *Günther Charwat*, Frederico Dusberger, Johannes Fichte, *Markus Hecher*, *Marius Moldovan*, Michael Morak, Nysret Musliu and Reinhard Pichler.

Outline

Treewidth

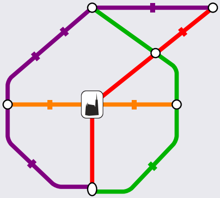
- Some graphs are more "tree-like" than others.
- Treewidth measures "tree-likeness".
  - Trees have treewidth 1.
  - The higher the treewidth, the more complex the graph.
- Often "easy on trees" implies "easy on tree-like graphs".
  - Many problems are fixed-parameter tractable w.r.t. treewidth k, i.e. can be decided in $O(2^k \cdot n)$.
  - That is, they become easy when putting a bound on the treewidth.

Treewidth

- Some graphs are more “tree-like” than others.
- Treewidth measures “tree-likeness”.
  - Trees have treewidth 1.
  - The higher the treewidth, the more complex the graph.
- Often “easy on trees” implies “easy on tree-like graphs”.
  - Many problems are fixed-parameter tractable w.r.t. treewidth k, i.e. can be decided in $O(2^k \cdot n)$.
  - That is, they become easy when putting a bound on the treewidth.
- It works for many hard problems.
- Real-world applications often have small treewidth.

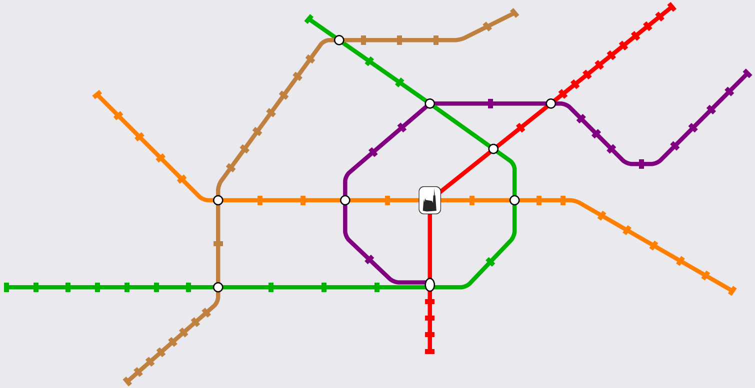
Treewidth (ctd.)

Example: Treewidth 3.



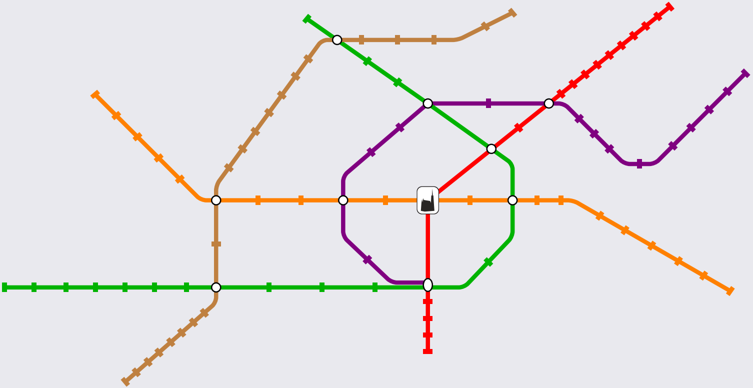
Treewidth (ctd.)

Example: Treewidth 3. Still.



Treewidth (ctd.)

Example: Treewidth 3. Still.



Treewidth is defined in terms of tree decompositions.

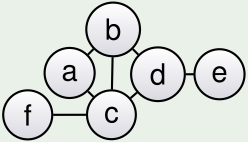
Tree Decompositions

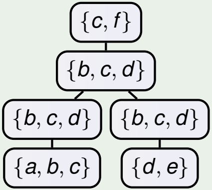
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. Each vertex must occur in some *bag*.
2. For each edge, there is a bag containing both endpoints.
3. If vertex v appears in bags of nodes n_0 and n_1, then v is also in the bag of each node on the path between n_0 and n_1.

Example





- *Decomposition width*: size of the largest bag (minus 1)
- Treewidth: minimum width over all possible tree decompositions

Tree Decompositions (ctd.)

Constructing a Tree Decomposition

- Any graph admits at least a trivial tree decomposition.
- But finding a *minimum-width* tree decomposition is difficult.
- However, there are good heuristics!

Tree Decompositions (ctd.)

Constructing a Tree Decomposition

- Any graph admits at least a trivial tree decomposition.
- But finding a *minimum-width* tree decomposition is difficult.
- However, there are good heuristics!

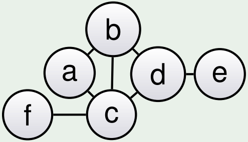
Dynamic Programming on Tree Decompositions

- Traverse tree decomposition from leaves to root and compute partial solutions in each node by
- suitably combining partial solutions of child nodes.
- Algorithms often exponential only in decomposition width but *linear* in the input size.

Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

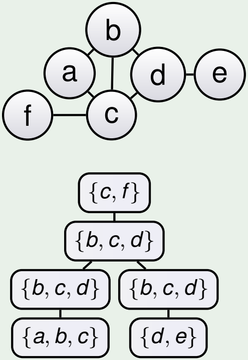


Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance

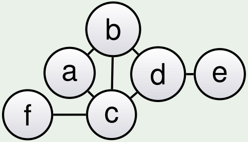


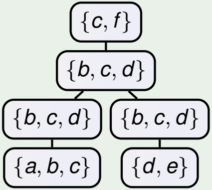
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

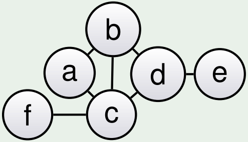
	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

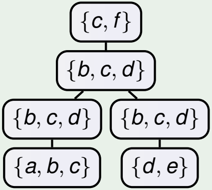
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems

	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

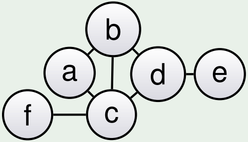
	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

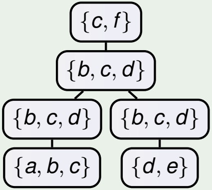
	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0



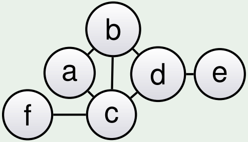


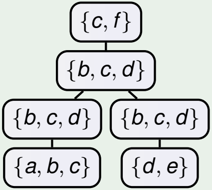
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

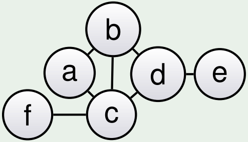
	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

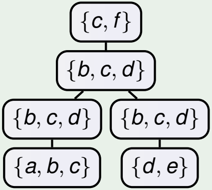
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

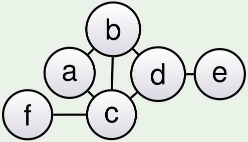
	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

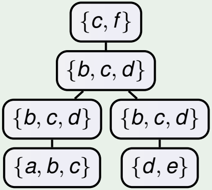
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

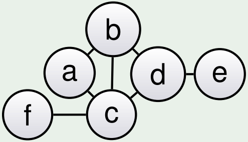
	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

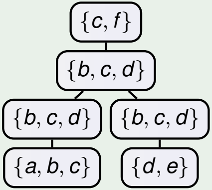
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

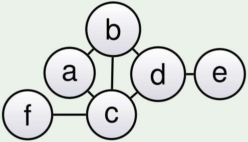
	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

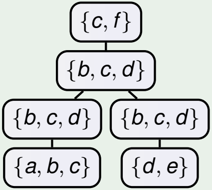
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

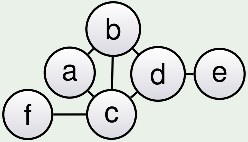
	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

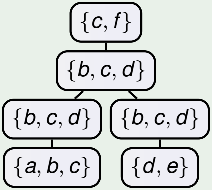
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	*f*	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	*c*	*d*	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	*c*	*d*	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	*c*	*d*	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	*b*	*c*	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

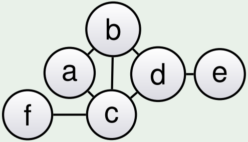
	d	*e*	cost
0	s	d	1
1	d	s	1
2	-	-	0

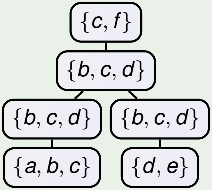
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	*f*	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	*c*	*d*	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	*c*	*d*	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	*c*	*d*	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	*b*	*c*	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

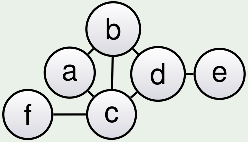
	d	*e*	cost
0	s	d	1
1	d	s	1
2	-	-	0

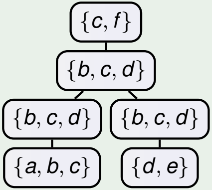
Dynamic Programming on Tree Decompositions

Example: MINIMUM INDEPENDENT DOMINATING SET

Methodology:

1. Decompose instance
2. Solve partial problems





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

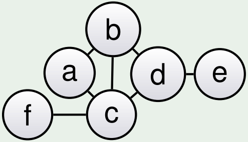
	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

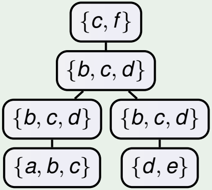
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems





	c	*f*	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	*c*	*d*	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	*c*	*d*	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	*c*	*d*	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	*b*	*c*	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

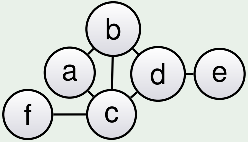
	d	*e*	cost
0	s	d	1
1	d	s	1
2	-	-	0

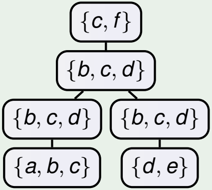
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems
3. Combine the solutions





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

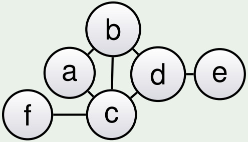
	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

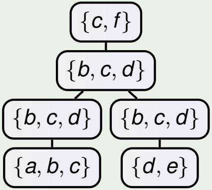
Dynamic Programming on Tree Decompositions

Example: Minimum Independent Dominating Set

Methodology:

1. Decompose instance
2. Solve partial problems
3. Combine the solutions





	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

Outline

D-FLAT

Dynamic Programming **F**ramework with **L**ocal Execution of **A**SP on **T**ree Decompositions

What does it do?

1. Constructs a tree decomposition of the input structure
2. In each node: Executes user-supplied logic program that describes the dynamic programming algorithm
3. Decides the problem (or materializes solutions)

Properties

- Relies on Answer-Set Programming (ASP) paradigm
- Users only need to write an ASP program
- Communication with the user's program via special predicates
- Uses external libraries for ASP solving, tree decomposition, etc.

Answer-Set Programming (ASP)

- Successful declarative programming paradigm in AI
- Has its roots in nonmonotonic reasoning and datalog
- Systems have been developed since the late 90s
- Applications in many diverse areas
  - Bio-Informatics
  - Diagnosis
  - Configuration
  - Linguistics
  - ...

Answer Set Programming (ctd.)

- ASP provides a convenient Guess & Check method
  1. Guess a candidate solution non-deterministically
  2. Check if the candidate is indeed a solution
- Any search problem in NP (even in Σ_2^P) can be solved with ASP

MINIMUM INDEPENDENT DOMINATING SET

Input:
Graph $G = (V, E)$ via predicates `vertex/1` and `edge/2`.

```
{ in(X) : vertex(X) }.
← in(X), in(Y), edge(X,Y).
dominated(X) ← in(Y), edge(Y,X).
← vertex(X), not in(X), not dominated(X).
#minimize{ 1,X : in(X) }.
```

Why ASP for Dynamic Programming?

- Compact declarative description of combinatorial problems
- ASP typically delivers *all* solutions
- Powerful systems available

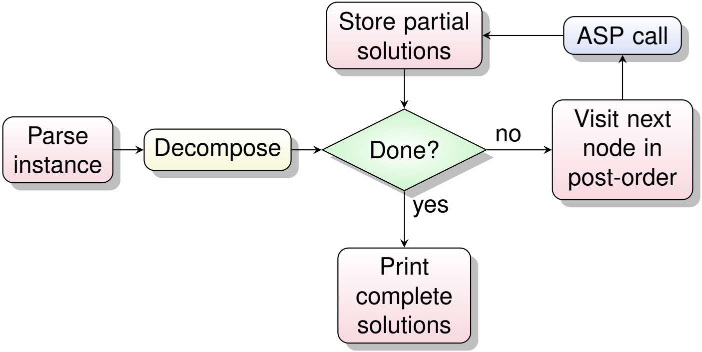
Practical Observation:

- If ASP is well suited for a problem, it is usually also well suited for the subproblems required in a decomposition

  $\Longrightarrow$ allows for rapid prototyping of dynamic programming on tree decompositions

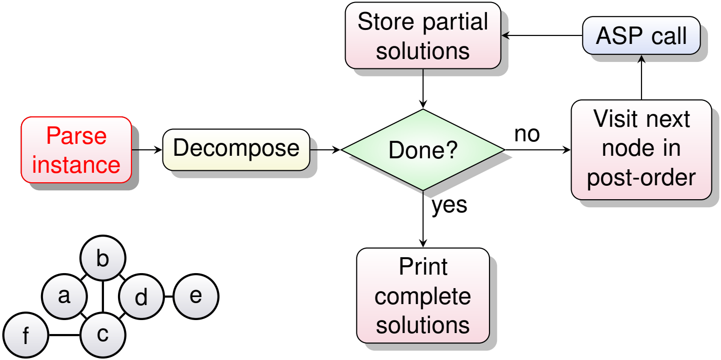
D-FLAT at Work

Illustrated by means of Independent Dominating Set



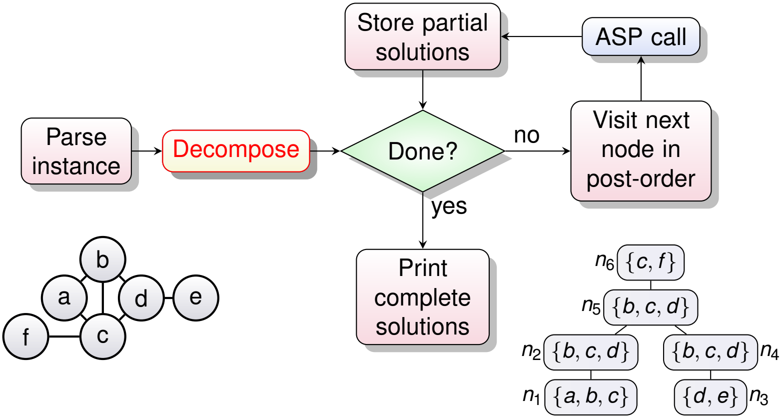
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



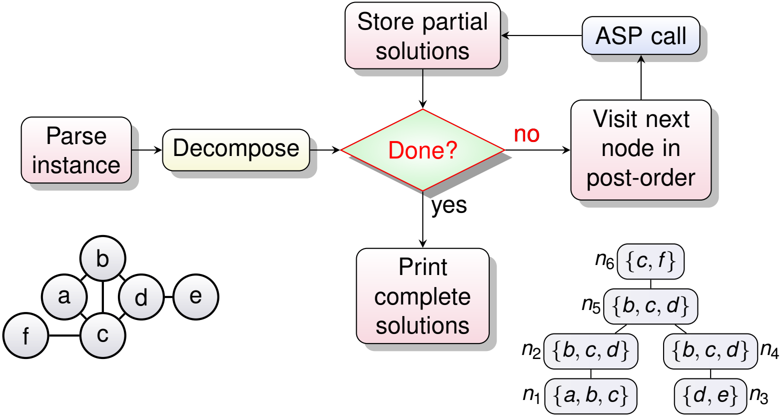
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



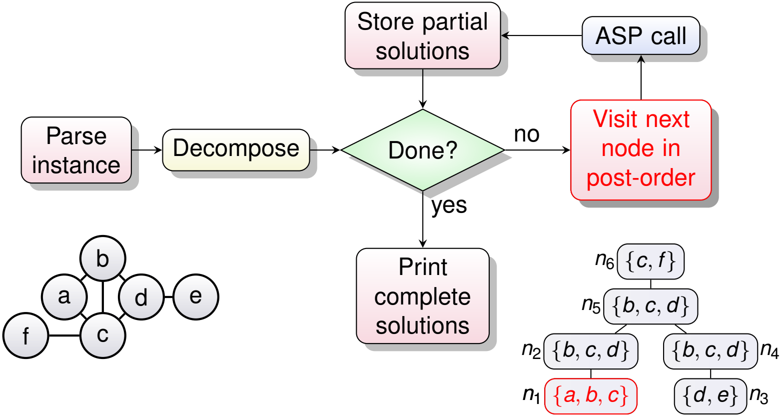
D-FLAT at Work

Illustrated by means of Independent Dominating Set



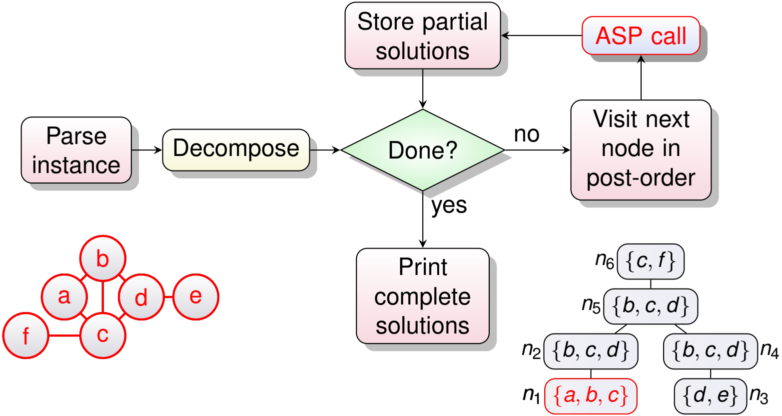
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



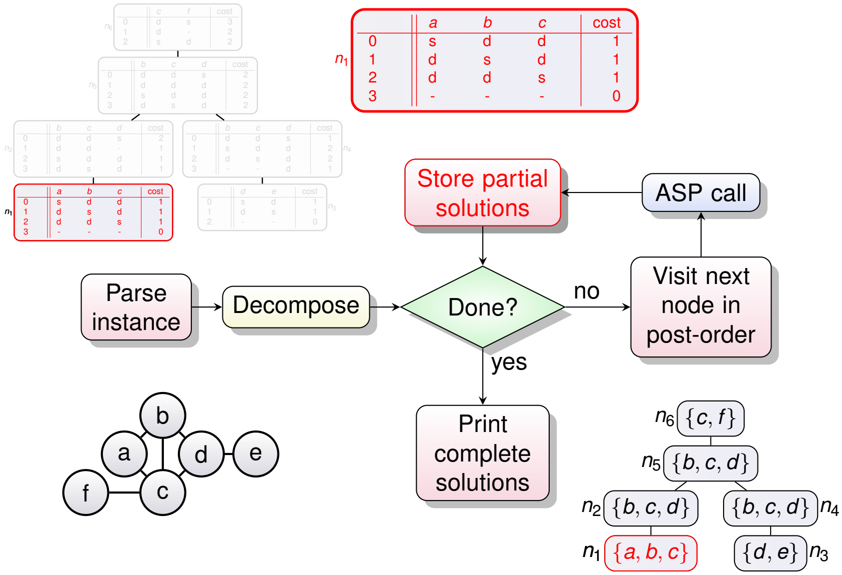
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



D-FLAT at Work

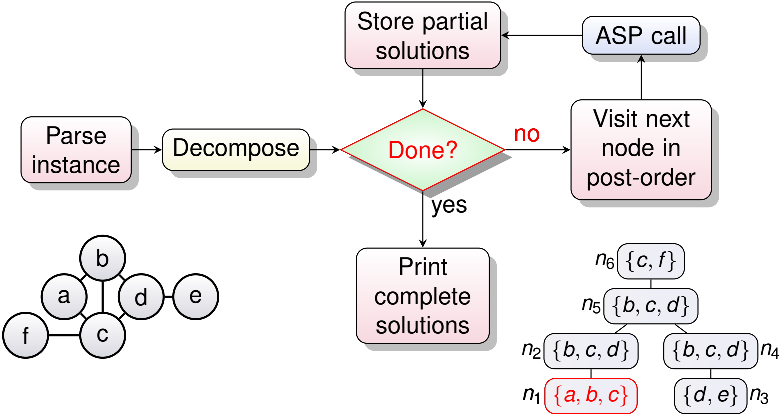
Illustrated by means of INDEPENDENT DOMINATING SET



	a	*b*	*c*	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

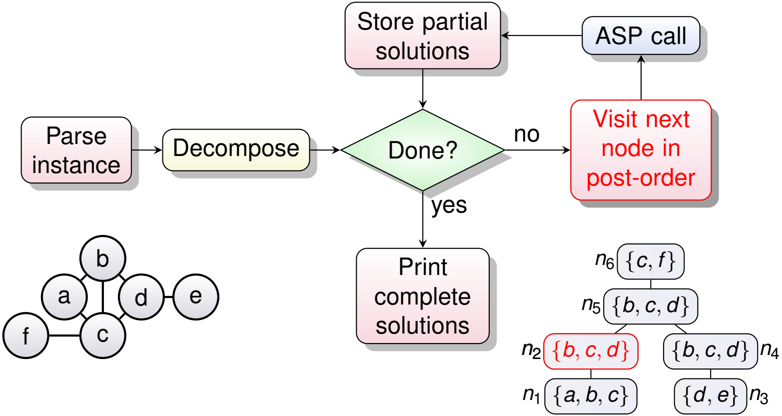
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



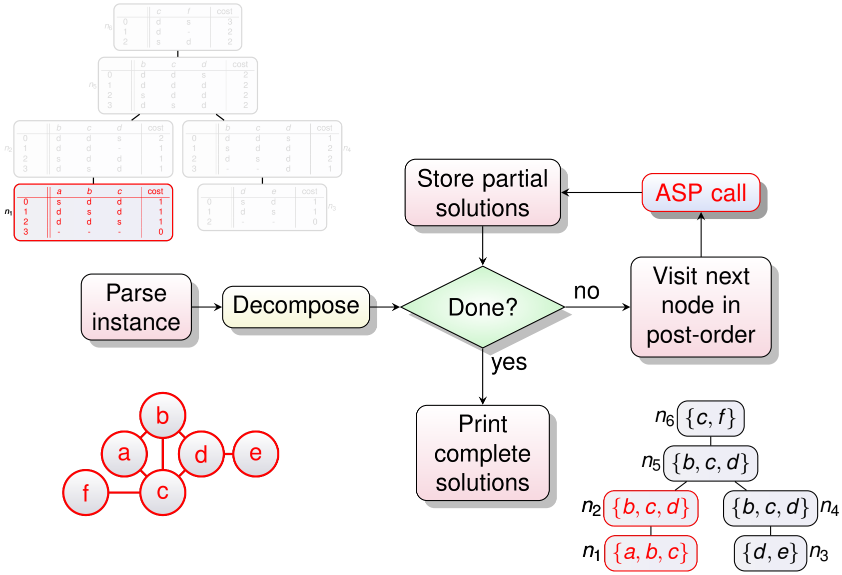
D-FLAT at Work

Illustrated by means of Independent Dominating Set



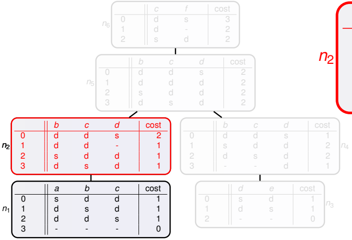
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



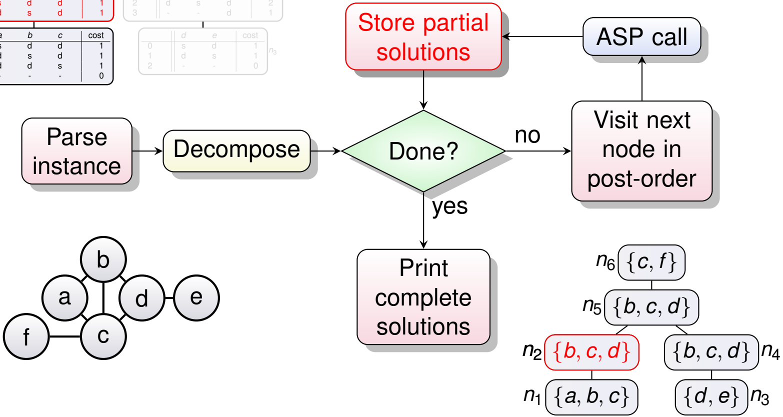
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



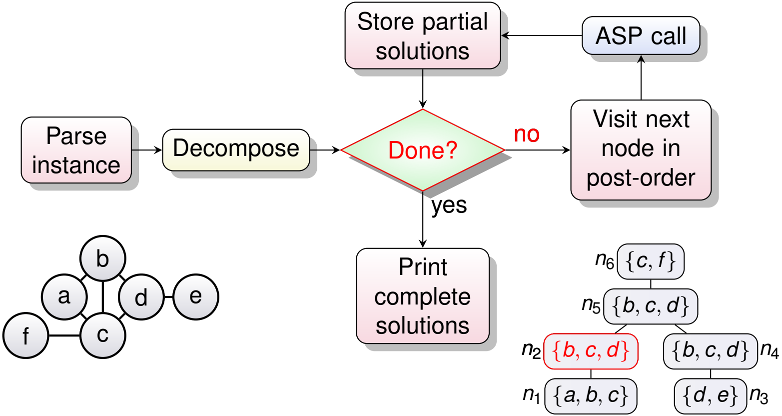
n_2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1



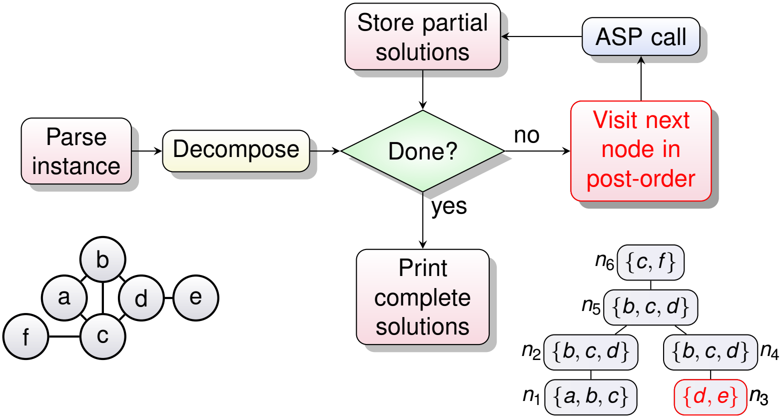
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



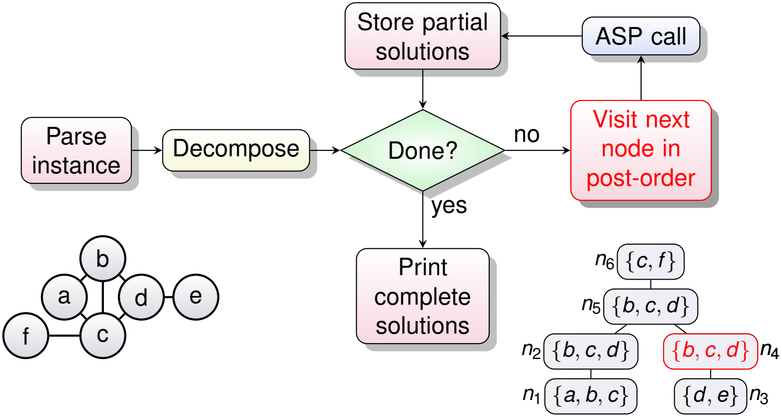
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



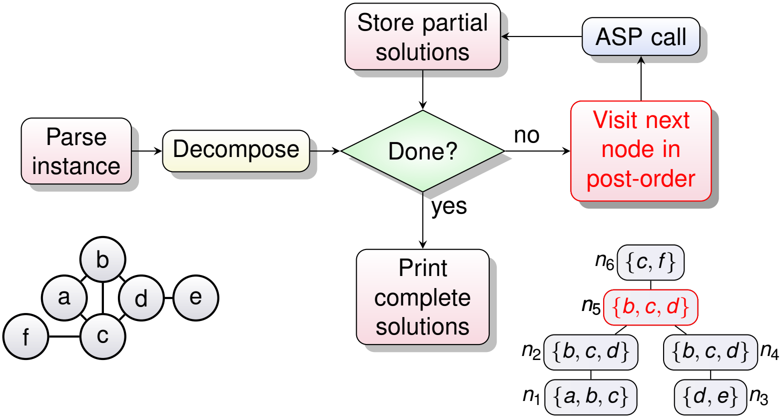
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



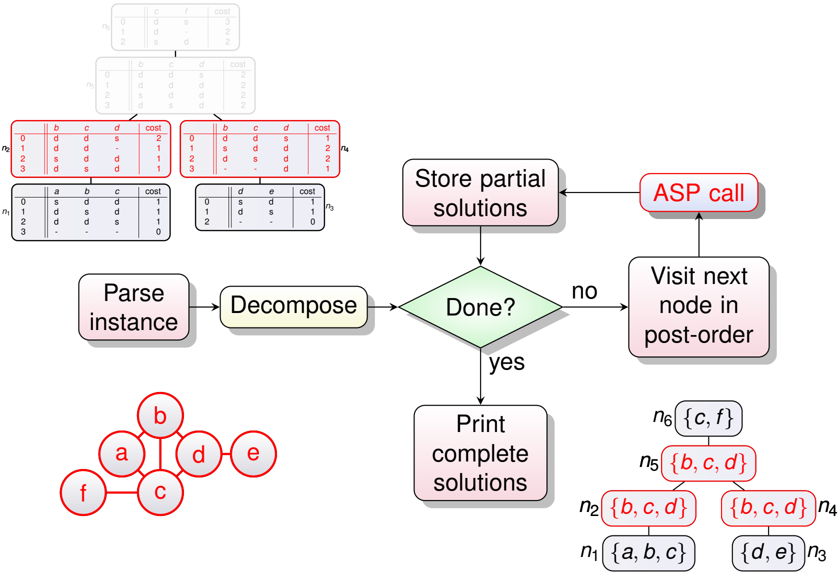
D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



n_6

	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

n_5

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

n_2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

n_4

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

n_1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

n_3

	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET

n_6

	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

n_5

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

n_2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

n_4

	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

n_1

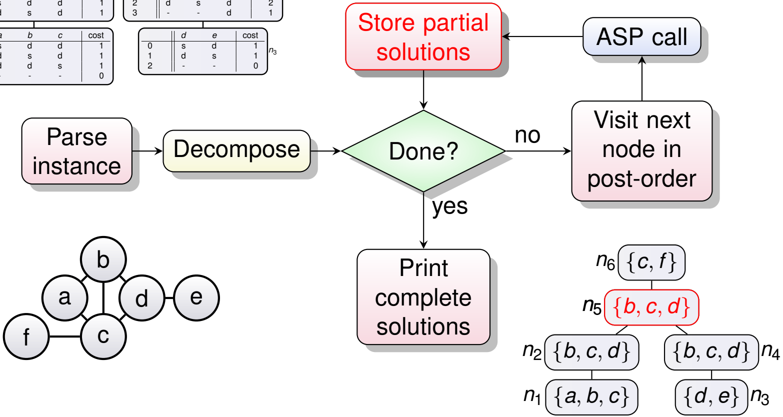
	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

n_3

	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0

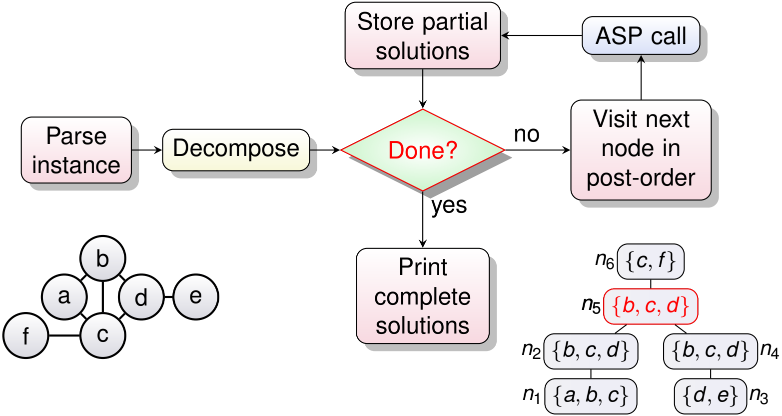
n_5

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2



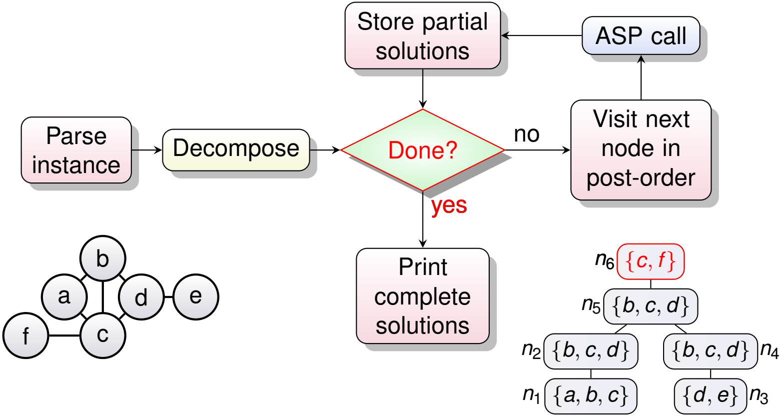
D-FLAT at Work

Illustrated by means of Independent Dominating Set



D-FLAT at Work

Illustrated by means of INDEPENDENT DOMINATING SET



D-FLAT at Work

Illustrated by means of Independent Dominating Set

n_6

	c	f	cost
0	d	s	3
1	d	-	2
2	s	d	2

n_5

	b	c	d	cost
0	d	d	s	2
1	d	d	d	2
2	s	d	d	2
3	d	s	d	2

n_2

	b	c	d	cost
0	d	d	s	2
1	d	d	-	1
2	s	d	d	1
3	d	s	d	1

n_4

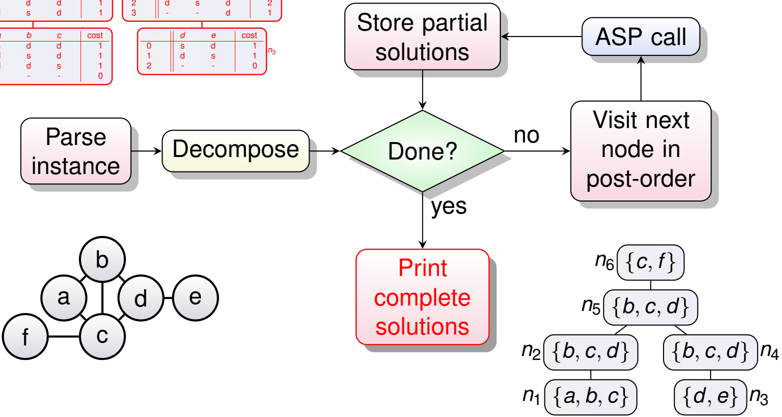
	b	c	d	cost
0	d	d	s	1
1	s	d	d	2
2	d	s	d	2
3	-	-	d	1

n_1

	a	b	c	cost
0	s	d	d	1
1	d	s	d	1
2	d	d	s	1
3	-	-	-	0

n_3

	d	e	cost
0	s	d	1
1	d	s	1
2	-	-	0



D-FLAT at Work (ctd.)

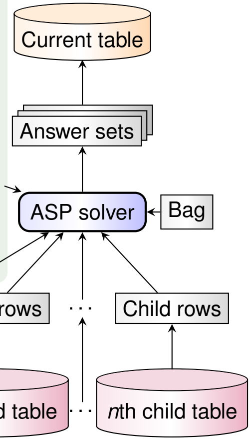
Illustrated by means of INDEPENDENT DOMINATING SET

User-supplied program

```
1 { extend(R) : childRow(R,N) } 1 ← childNode(N).
← extend(R1;R2), childItem(R1,in(X)),
               not childItem(R2,in(X)).
← removed(X), extend(R),
   not childItem(R,in(X)), not childItem(R,dom(X)).
item(in(X))    ← extend(R), childItem(R,in(X)),
                 current(X).
item(dom(X))   ← extend(R), childItem(R,dom(X)),
                 current(X).
{ item(in(X)) : introduced(X) }.
← edge(X,Y), item(in(X;Y)).
item(dom(X))   ← item(in(Y)), edge(Y,X),
                 current(X).
```

Instance

```
vertex(a;b;c;d;e).
edge(a,b). edge(a,c). edge(b,c).
edge(b,d). edge(c,d). edge(d,e).
```



Another Example: Boolean Satisfiability (SAT)

Although SAT is not a graph problem, we can still decompose it.

- Use the incidence graph of the formula:
- One vertex for each variable and each clause.
- Edge (v, c) if variable v occurs in clause c.

D-FLAT encoding

```
% Extend compatible rows from child nodes.
1 { extend(R) : childRow(R,N) } 1 ← childNode(N).
← extend(R;S), atom(A), childItem(R,A), not childItem(S,A).
% Retain extended assignment and guess on introduced atoms.
item(X) ← extend(R), childItem(R,X), current(X).
{ item(A) : atom(A), introduced(A) }.
% Additional clauses might have become satisfied.
item(C) ← current(C;A), pos(C,A), item(A).
item(C) ← current(C;A), neg(C,A), not item(A).
% Kill assignments that leave some clause unsatisfied.
← clause(C), removed(C), extend(R), not childItem(R,C).
```

What about Performance?

"About your cat, Mr. Schrödinger—I have good news and bad news."

What about Performance?

"About your cat, Mr. Schrödinger—I have good news and bad news."

Time for a Demo!

D-FLAT Features

- Special predicates in LP allow the user to delegate tasks to D-FLAT
- Additional D-FLAT features for arithmetics
- Different modes for decision, counting, optimization and enumeration problems
- Support of different normalizations of the decomposition
- Support of hypergraphs
- “Default Join”
- Two modes for storing and handling solutions of subproblems

D-FLAT Features (ctd.)

"Table-Mode" for Problems in NP

- We compute a table at each node
- We guess rows using ASP
- ... yields all accepting computation branches of an NTM

D-FLAT Features (ctd.)

“Table-Mode” for Problems in NP

- We compute a table at each node
- We guess rows using ASP
- … yields all accepting computation branches of an NTM
- D-FLATˆ2 frontend
  - designed for minimization problems on top of “table-mode”
  - DP is automatically obtained from simpler principles

D-FLAT Features (ctd.)

“Table-Mode” for Problems in NP

- We compute a table at each node
- We guess rows using ASP
- … yields all accepting computation branches of an NTM
- D-FLATˆ2 frontend
  - designed for minimization problems on top of “table-mode”
  - DP is automatically obtained from simpler principles

“Tree-Mode” for Problems in the Polynomial Hierarchy

- We compute a tree at each node
- We guess branches using ASP
- … yields all accepting computation branches of an ATM (D-FLAT appropriately handles the trees inside).

General Applicability

Recall Courcelle's theorem

Any problem definable in MSO can be solved in linear time on graphs of bounded treewidth.

It is such problems that decomposition is usually employed for.

General Applicability

Recall Courcelle's theorem

Any problem definable in MSO can be solved in linear time on graphs of bounded treewidth.

It is such problems that decomposition is usually employed for.

Good news

D-FLAT can be effectively used for all such problems

- It can evaluate MSO formulas in linear time if the treewidth is bounded
- Encoding for MSO is not overly complicated (approx. 30 lines of ASP code)
- However, expressing the problem at hand via MSO and then feed to D-FLAT is not recommended
  - instead, D-FLAT is designed for problem-specific dynamic programming solutions

A First Conclusion

Summary

- Hard problems often become tractable when instances exhibit certain properties.
- Especially bounded treewidth often leads to tractability (problems expressible in MSO).
- The “D-FLAT” method [TPLP 2012, JELIA 2014] allows to specify dynamic programming algorithms in a declarative way.
  - This works for all MSO-definable problems [JLC 2016]

Outline

Outline

Motivation

Lesson Learnt

- Generation of decompositions rather cheap (compared to the runtime of dynamic programming)
- Shape of decomposition crucial for performance (it's not the width only!)
- Better understanding needed how "good tree decompositions" look like

Motivation

Lesson Learnt

- Generation of decompositions rather cheap (compared to the runtime of dynamic programming)
- Shape of decomposition crucial for performance (it's not the width only!)
- Better understanding needed how "good tree decompositions" look like

Goal

- Identification of features for tree decompositions (rather than on the actual input instance)
- Development of system that allows to customize tree decompositions

Methodology

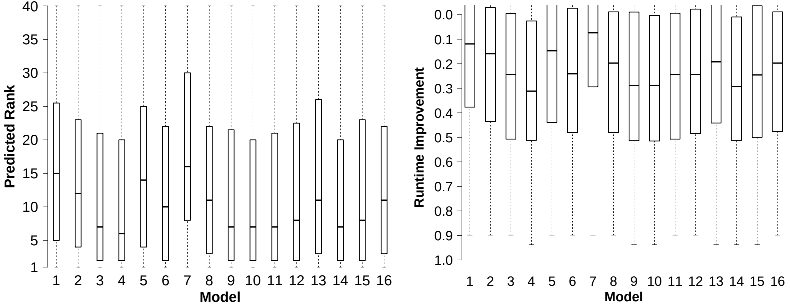
Given a specific problem

- Training data:
  - 70 random instances with rather low treewidth
  - 40 decompositions for each problem instance
- Obtain regression models (16 different methods) for ranking decompositions using specific tree decomposition features
- Apply model to real-world instances (treewidth up to 8)
  - Generate 50 tree decompositions per instance
  - Model selects the best-ranked decomposition

Experimental Results

- Accelerating Minimum Dominating Set using Machine Learning

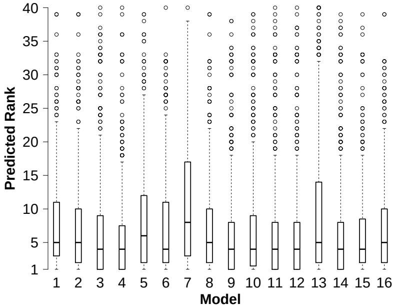
Minimum Dominating Set D-FLAT			
Minimum Improvement:	7.39 %	Average Improvement:	21.80 %
Maximum Improvement:	31.15 %	Median Improvement:	24.25 %
Statistical Significance:	$\geq$ 99.95 %		

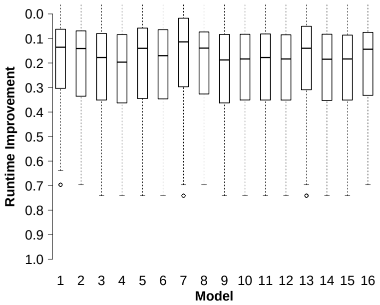


Experimental Results

- Accelerating Minimum Dominating Set using Machine Learning

Minimum Dominating Set SEQUOIA			
Minimum Improvement:	11.39 %	Average Improvement:	16.24 %
Maximum Improvement:	19.65 %	Median Improvement:	17.39 %
Statistical Significance:		$\geq$ 99.95 %	





Towards exploiting Decomposition Features

New decomposition library: htd

- htd provides efficient implementations of well-known algorithms
- htd allows to fully customize the tree decomposition via several strategies
- htd offers a wide range of convenience functions like the possibility to access the subgraph induced by each bag at almost no cost (Performance boost for large graphs!).
- 3rd place in recent tree-decomposition competition
  - `https://pacechallenge.wordpress.com/`
- Available at: `https://github.com/mabseher/htd`

Discussion

- We conducted huge test series [IJCAI 2015] for several problems and two state-of-the-art systems (D-FLAT and SEQUOIA)
- Feature-based ML successfully identified good decompositions
- However, crucial features are in general not problem independent
- New decomposition library allows the user to specify what kind of tree decomposition she prefers

Outline

Motivation

Lesson Learnt

- Drawback of classical DP on TDs: Always computes all solutions even if only one is required.
- Optimization problems: Sometimes table rows have higher costs than optimal solution.

Motivation

Lesson Learnt

- Drawback of classical DP on TDs: Always computes all solutions even if only one is required.
- Optimization problems: Sometimes table rows have higher costs than optimal solution.

Idea

- Materialize tables “in parallel”.
- Realization in D-FLAT: modern ASP technology (external atoms)
- Use coexisting ASP solvers that communicate with each other.

Motivation

Lesson Learnt

- Drawback of classical DP on TDs: Always computes all solutions even if only one is required.
- Optimization problems: Sometimes table rows have higher costs than optimal solution.

Idea

- Materialize tables “in parallel”.
- Realization in D-FLAT: modern ASP technology (external atoms)
- Use coexisting ASP solvers that communicate with each other.

Goals

- Anytime behavior (ability to report solutions when interrupted)
- Understand feasibility of this approach

Example: “Lazy” DP on TDs

DP specification in ASP

```
#external childItem(in(X)) : childNode(N), bag(N,X).
#external childAuxItem(dom(X)) : childNode(N), bag(N,X).

item(in(X)) ← childItem(in(X)), not removed(X).
auxItem(dom(X)) ← childAuxItem(dom(X)), not removed(X).
{ item(in(X)) : introduced(X) }.
auxItem(dom(Y)) ← item(in(X)), edge(X,Y), current(X;Y).
← removed(X), not childItem(in(X)), not childAuxItem(dom(X)).
← edge(X,Y), item(in(X)), item(in(Y)).
```

Avoiding Re-grounding via Assumption-based Solving

- The *clingo* system supports external atoms.
- Truth value of externals can be set “from the outside”.
  1. Freeze a certain truth assignment on externals.
  2. Compute all answer sets under this assumption.
  3. Repeat with different assumption.
- Grounding only happens once.

Experimental Results

Search and optimization problems on real-world graphs

"Lazy" vs. "eager"

- Search problems: "Lazy" usually finds a solution much quicker.
- Optimization problems: "Lazy" mostly finds optimum faster (and able to print solutions along the way)

Comparison to clingo (without decomposition)

- Search problems: Clingo finds a solution much quicker.
- DOMINATING SET, VERTEX COVER: Clingo is clearly faster.
- STEINER TREE: "Lazy" is faster ...
  - "Lazy" often finds optimum when clingo times out.
  - "Lazy" offers better suboptimal solutions until timeout.

Discussion

- DP on TDs via “lazy evaluation”
- At each table, an ASP solver is used for computing rows
  - Multiple coexisting ASP solvers that communicate with each other
  - *Assumption-based solving*: avoids excessive re-grounding
- “Lazy” outperforms “eager”
- Outperforms state-of-the-art ASP systems on some problems (w.r.t. anytime performance)

Outline

Motivation

Lesson Learnt

- Bottleneck of D-FLAT (resp. DP in general): size of tables
  - size grows exponentially with treewidth
- Can we find a match to logic (truth-table vs. formula)?

Motivation

Lesson Learnt

- Bottleneck of D-FLAT (resp. DP in general): size of tables
  - size grows exponentially with treewidth
- Can we find a match to logic (truth-table vs. formula)?

Idea

- Employ Binary Decision Diagrams (BDDs):
  - compact representation of truth-tables
  - can be treated like formulas

Motivation

Lesson Learnt

- Bottleneck of D-FLAT (resp. DP in general): size of tables
  - size grows exponentially with treewidth
- Can we find a match to logic (truth-table vs. formula)?

Idea

- Employ Binary Decision Diagrams (BDDs):
  - compact representation of truth-tables
  - can be treated like formulas

Goals

- Understand feasibility of this approach
- Understand limits in describing DPs as formula manipulation

Binary Decision Diagrams

Example (OBDD representation)

Let formula $\varphi = (a \wedge b \wedge c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$.

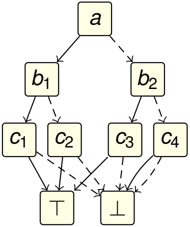


Figure : OBDD of φ.

Binary Decision Diagrams

Example (OBDD representation)

Let formula $\varphi = (a \wedge b \wedge c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$.

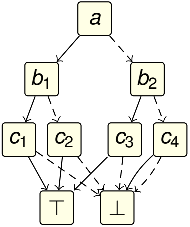


Figure : OBDD of φ.

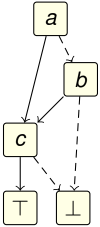


Figure : ROBDD of φ.

Binary Decision Diagrams (ctd.)

Advantages of BDDs:

- Well-studied and mature concepts that are successfully applied to planning, verification, etc.
- Efficient implementations available
- Delegate burden of memory-efficient implementation to data structure
- Logic-based algorithm specification

Comparison

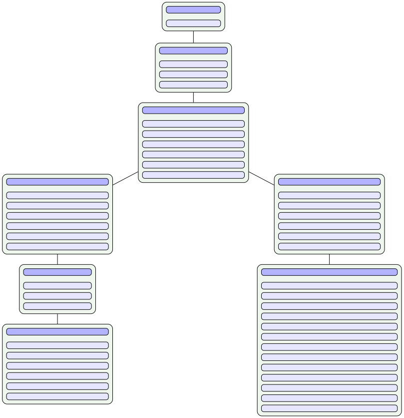


Table-based Dynamic Programming

Comparison

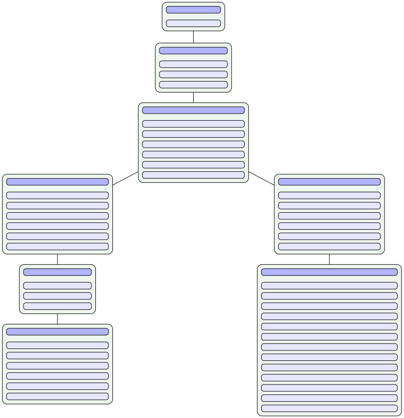
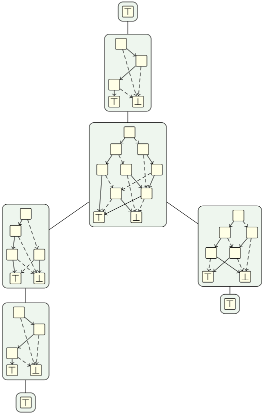


Table-based Dynamic Programming



BDD-based Dynamic Programming

DP of Independent Dominating Set via BDDs

$$\mathcal{B}_t^I = \bigwedge_{(x,y)\in E_t} (\neg i_x \vee \neg i_y) \wedge \bigwedge_{y\in V_t} \left(d_y \leftrightarrow \bigvee_{(x,y)\in E_t} i_x\right)$$

DP of Independent Dominating Set via BDDs

$$\mathcal{B}_t^l = \bigwedge_{(x,y)\in E_t} (\neg i_x \vee \neg i_y) \wedge \bigwedge_{y\in V_t} \left(d_y \leftrightarrow \bigvee_{(x,y)\in E_t} i_x \right)$$

$$\mathcal{B}_t^i = \exists D'_{t'} \Big[\mathcal{B}_{t'}[D_{t'}/D'_{t'}] \wedge \bigwedge_{(u,y)\in E_t} (\neg i_u \vee \neg i_y) \wedge \left(d_u \leftrightarrow \bigvee_{(x,u)\in E_t} i_x \right) \wedge$$

$$\bigwedge_{\substack{(u,y)\in E_t \wedge \\ u\neq y}} \left(d_y \leftrightarrow d'_y \vee i_u \right) \wedge \bigwedge_{y\in V_t \wedge (u,y)\notin E_t} \left(d_y \leftrightarrow d'_y \right) \Big]$$

DP of Independent Dominating Set via BDDs

$$\mathcal{B}^l_t = \bigwedge_{(x,y)\in E_t} (\neg i_x \vee \neg i_y) \wedge \bigwedge_{y\in V_t} \left(d_y \leftrightarrow \bigvee_{(x,y)\in E_t} i_x \right)$$

$$\mathcal{B}^i_t = \exists D'_{t'} \Big[\mathcal{B}_{t'}[D_{t'}/D'_{t'}] \wedge \bigwedge_{(u,y)\in E_t} (\neg i_u \vee \neg i_y) \wedge \left(d_u \leftrightarrow \bigvee_{(x,u)\in E_t} i_x \right) \wedge$$
$$\bigwedge_{\substack{(u,y)\in E_t \wedge \\ u \neq y}} \left(d_y \leftrightarrow d'_y \vee i_u \right) \wedge \bigwedge_{y\in V_t \wedge (u,y)\notin E_t} \left(d_y \leftrightarrow d'_y \right) \Big]$$

$$\mathcal{B}^r_t = \mathcal{B}_{t'}[i_u/\top, d_u/\bot] \vee \mathcal{B}_{t'}[i_u/\bot, d_u/\top]$$

DP of Independent Dominating Set via BDDs

$$\mathcal{B}^l_t = \bigwedge_{(x,y)\in E_t} (\neg i_x \vee \neg i_y) \wedge \bigwedge_{y\in V_t} \left(d_y \leftrightarrow \bigvee_{(x,y)\in E_t} i_x\right)$$

$$\mathcal{B}^i_t = \exists D'_{t'} \Big[\mathcal{B}_{t'}[D_{t'}/D'_{t'}] \wedge \bigwedge_{(u,y)\in E_t} (\neg i_u \vee \neg i_y) \wedge \Big(d_u \leftrightarrow \bigvee_{(x,u)\in E_t} i_x\Big) \wedge \bigwedge_{\substack{(u,y)\in E_t \wedge \\ u\neq y}} (d_y \leftrightarrow d'_y \vee i_u) \wedge \bigwedge_{y\in V_t \wedge (u,y)\notin E_t} (d_y \leftrightarrow d'_y)\Big]$$

$$\mathcal{B}^r_t = \mathcal{B}_{t'}[i_u/\top, d_u/\bot] \vee \mathcal{B}_{t'}[i_u/\bot, d_u/\top]$$

$$\mathcal{B}^j_t = \exists D'_t D''_t \Big[\mathcal{B}_{t'}[D_t/D'_t] \wedge \mathcal{B}_{t''}[D_t/D''_t] \wedge \bigwedge_{x\in V_t} (d_x \leftrightarrow d'_x \vee d''_x)\Big]$$

Dynamic-Programming based QBF-solving

Method

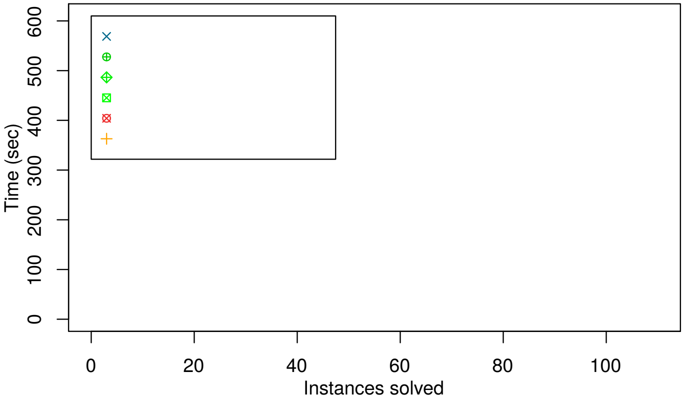
- Use the presented ideas for solving quantified Boolean formulas in prenex CNF form

$$\exists ab \, \forall cd \, \exists ef (a \vee c \vee e) \wedge (\neg b \vee d) \wedge (e \vee f) \wedge (c \vee \neg e) \wedge (\neg d \vee f)$$

- We consider primal graph of the CNF
- Datastructure used is a recursive set of BDDs (recursion depth depends on number of quantifier alternations)
- Some further optimizations required to be competitive

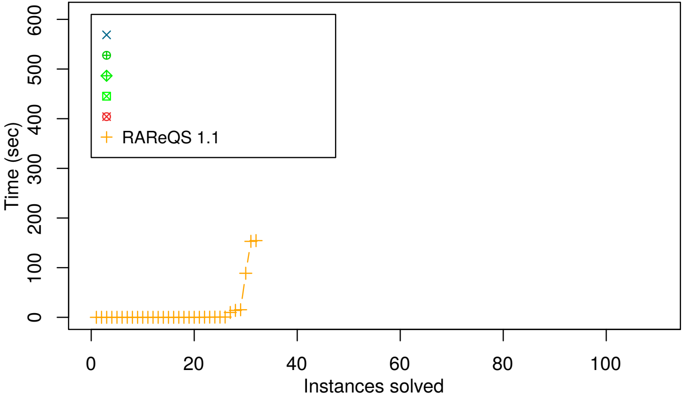
Experimental Results

2-QBF ($\forall\exists$) competition instances ($\#\mathit{instances} = 200$)



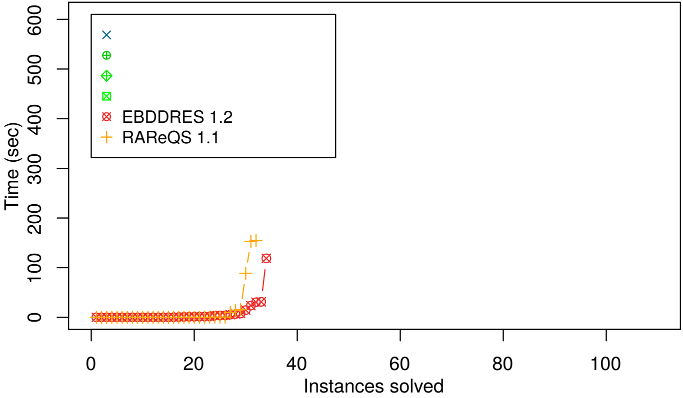
Experimental Results

2-QBF ($\forall\exists$) competition instances ($\#instances = 200$)



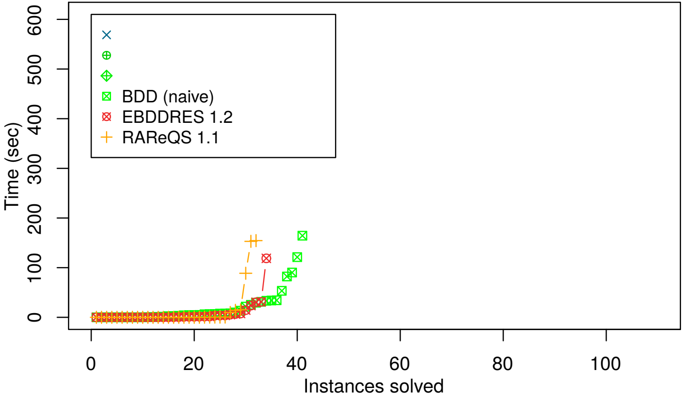
Experimental Results

2-QBF ($\forall\exists$) competition instances ($\#\mathit{instances} = 200$)



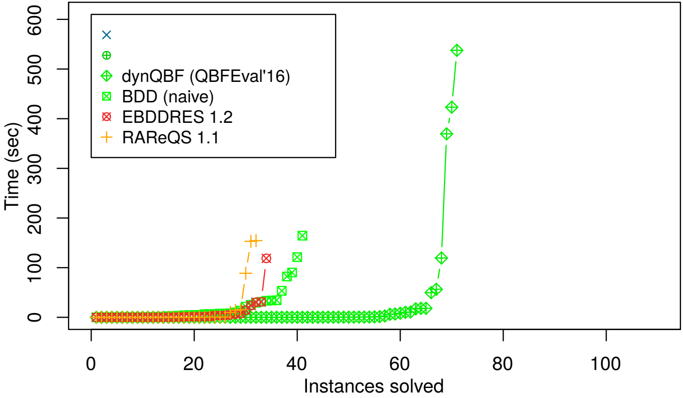
Experimental Results

2-QBF ($\forall\exists$) competition instances ($\#\textit{instances} = 200$)



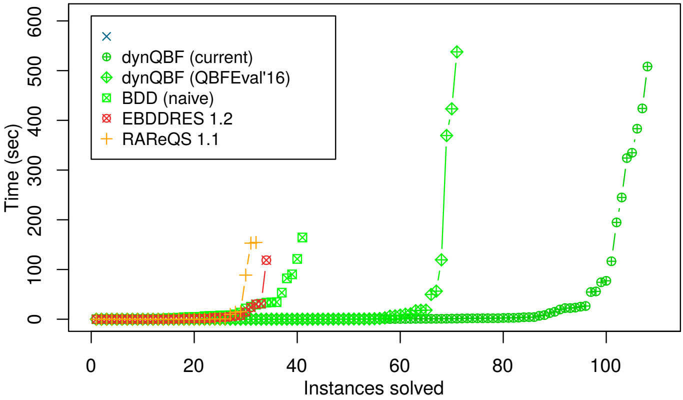
Experimental Results

2-QBF (∀∃) competition instances (#*instances* = 200)



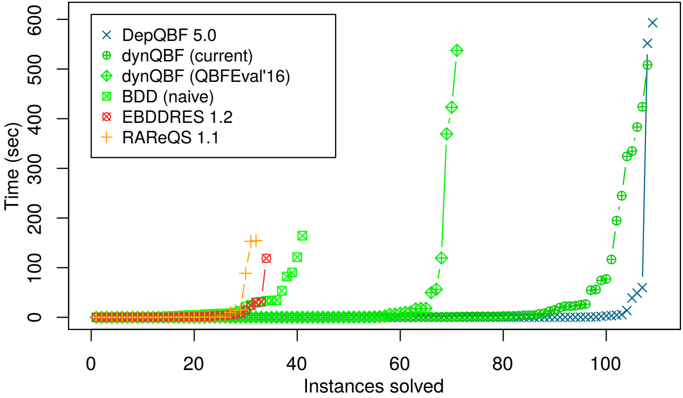
Experimental Results

2-QBF (∀∃) competition instances ($\#\textit{instances} = 200$)



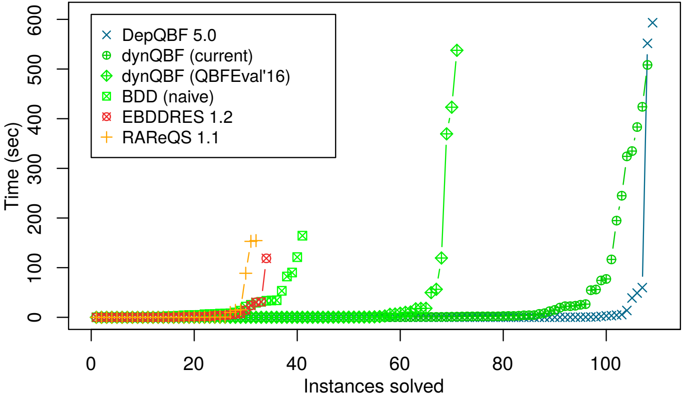
Experimental Results

2-QBF ($\forall\exists$) competition instances ($\#\mathit{instances} = 200$)



Experimental Results

2-QBF (∀∃) competition instances ($\#instances = 200$)

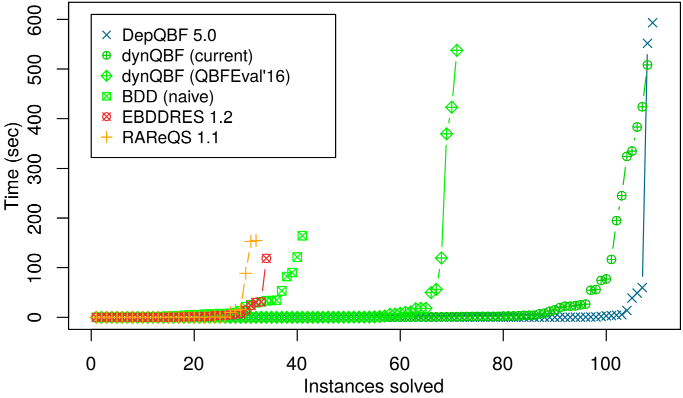


Solved instances with small width ($w \leq 50$, $\#instances = 55$):

- dynQBF: 54, EBDDRES: 31, DepQBF: 28, RAReQS: 19

Experimental Results

2-QBF ($\forall\exists$) competition instances ($\#instances = 200$)



Solved instances with small width ($w \leq 50$, $\#instances = 55$):

- dynQBF: 54, EBDDRES: 31, DepQBF: 28, RAReQS: 19

Uniquely solved ($\#instances = 200$):

- DepQBF: 43, dynQBF: 41, RAReQS: 5, EBDDRES: 2

Experimental Evaluation

QBF Gallery 2014 competition instances ($\#instances = 276$)

System	Solved	SAT	UNSAT	Timeout	Memout	Unique
DepQBF 5.0	103	48	55	169	4	42
RAReQS 1.1	83	36	47	193	0	22
dynQBF (current)	21	6	15	250	5	8
EBDDRES 1.2	7	5	2	4	265	2
BDD (naive)	3	1	2	273	0	0

Experimental Evaluation

QBF Gallery 2014 competition instances ($\#instances = 276$)

System	Solved	SAT	UNSAT	Timeout	Memout	Unique
DepQBF 5.0	103	48	55	169	4	42
RAReQS 1.1	83	36	47	193	0	22
dynQBF (current)	21	6	15	250	5	8
EBDDRES 1.2	7	5	2	4	265	2
BDD (naive)	3	1	2	273	0	0

dynQBF is not yet competitive:

- 27 out of 276 instances were not decomposed within the time limit
- Solved instances have an average width of 55, 3 quantifiers, 4711 atoms and 16409 clauses

Discussion

- dynBDD is a first prototype that performs DP algorithms on tree decompositions via manipulation of BDDs [LPNMR 2015]
- allows for realization of more advanced DP algorithms (“wild cards” etc)
- preliminary results indicate significant decrease of space used
- particularly successful for QBF solving
- currently, algorithms have to be implemented in C++ on top of CUDD
- Systems available:
  - `dbai.tuwien.ac.at/proj/decodyn/dynbdd/`
  - `dbai.tuwien.ac.at/proj/decodyn/dynqbf/`

Outline

Summary

- Tree-Decompositions known as a promising tool to exploit structure in hard problems
- D-FLAT: a system for rapid prototyping of DP algorithms
  - takes care of the decomposition task
  - declarative specifications of dynamic programming via ASP
  - ASP systems used to solve subproblems
  - general applicability
  - able to outperform standard technology
- Many ongoing developments

Ongoing + Future Work

- Automatic generation of D-FLAT code from “standard” encoding
- Exploit smarter ways to store solutions
  - BDDs a promising option
  - easy-to-use interface still missing
- Tighter integration of D-FLAT with ASP solvers
  - communication between D-FLAT and ASP solver is bottleneck
- Incorporation of other decomposition methods
  - Straight forward for clique width, branch width, ...
  - Lack of efficient heuristics for obtaining decomposition

Try it out! D-FLAT is free software, available at

```
http://dbai.tuwien.ac.at/proj/dflat/
```

Try it out! D-FLAT is free software, available at

```
http://dbai.tuwien.ac.at/proj/dflat/
```

...and have fun with decompositions ...

Thanks for your attention!

Main References

JELIA 2014 M. Abseher, B. Bliem, G. Charwat, F. Dusberger, M. Hecher, S. Woltran: "The D-FLAT system for dynamic programming on tree decompositions". *Proceedings JELIA 2014*, pp. 558–572. Springer, 2014.

JLC 2016 B. Bliem, R. Pichler, S. Woltran: "Implementing Courcelle's Theorem in a Declarative Framework for Dynamic Programming". *Journal of Logic and Computation*, 2016.

IJCAI 2015 M. Abseher, F. Dusberger, N. Musliu, S. Woltran: "Improving the efficiency of dynamic programming on tree decompositions via machine learning". *Proceedings IJCAI 2015*, pp. 275–282, AAAI Press, 2015.

IJCAI 2016 B. Bliem, B. Kaufmann, T. Schaub, S. Woltran: "ASP for Anytime Dynamic Programming on Tree Decompositions". *Proceedings IJCAI 2016*, pp. 979–986, AAAI Press, 2016.

LPNMR 2015 G. Charwat, S. Woltran: "Efficient problem solving on tree decompositions using binary decision diagrams". *Proceedings LPNMR 2015*, pp. 213–227, Springer, 2015.

TPLP 2012 B. Bliem, M. Morak, and S. Woltran: "D-FLAT: Declarative problem solving using tree decompositions and answer-set programming". *Theory and Practice of Logic Programming*, vol. 12, pp. 445–464, 2012.