Lecture *Hierarchical Planning*

Chapter:
Solving Hierarchical Planning Problems via SAT

Gregor Behnke

Institute of Artificial Intelligence,
Ulm University, Germany

Winter Term 2018/2019

(Compiled on: February 20, 2019)

ulm university universität ulm

Overview:

HTN Planning via SAT

In lecture 04 (Solving (Non-Hierarchical) Planning Problems via SAT) we have seen how classical planning problems can be solved via a translation into SAT.

HTN Planning via SAT

In lecture 04 (Solving (Non-Hierarchical) Planning Problems via SAT) we have seen how classical planning problems can be solved via a translation into SAT.

Can this also be done for HTN planning?

Issues with HTN Planning via SAT

Reminder: For a planning problem $\mathcal{P}$ create a CNF formula $\mathcal{F}$ that is satisfiable iff $\mathcal{P}$ has a solution.

Issues with HTN Planning via SAT

Reminder: For a planning problem $\mathcal{P}$ create a CNF formula $\mathcal{F}$ that is satisfiable iff $\mathcal{P}$ has a solution.

(Potential) Issues:

Issues with HTN Planning via SAT

Reminder: For a planning problem $\mathcal{P}$ create a CNF formula $\mathcal{F}$ that is satisfiable iff $\mathcal{P}$ has a solution.

(Potential) Issues:

- HTN planning is undecidable, i.e. there cannot be such a formula $\mathcal{F}$.

Issues with HTN Planning via SAT

Reminder: For a planning problem $\mathcal{P}$ create a CNF formula $\mathcal{F}$ that is satisfiable iff $\mathcal{P}$ has a solution.

(Potential) Issues:

- HTN planning is undecidable, i.e. there cannot be such a formula $\mathcal{F}$.
- Even if we find a way, how do we represent decomposition?

Idea for Transforming Classical Planning

- PLANEX is $\mathbb{PSPACE}$

Idea for Transforming Classical Planning

- PLANEX is $\mathbb{PSPACE}$
- PLANEX “is” $\mathbb{NP}$-“complete” for ℓ-length bounded planning if ℓ is encoded unary.

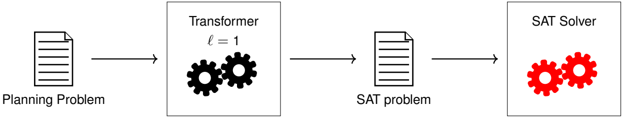
Idea for Transforming Classical Planning

- PLANEX is $\mathbb{PSPACE}$
- PLANEX "is" $\mathbb{NP}$-"complete" for ℓ-length bounded planning if ℓ is encoded unary.
- For full PLANEX: theoretical limit $2^{|V|}$.

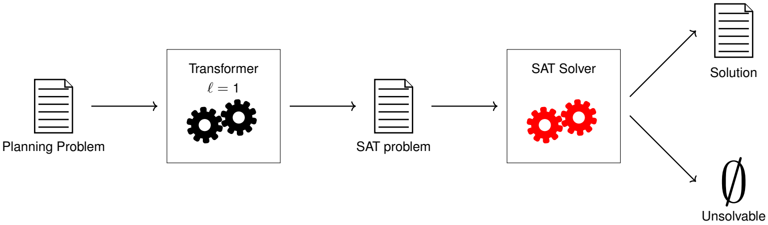
Idea for Transforming Classical Planning

- PLANEX is $\mathbb{PSPACE}$
- PLANEX "is" $\mathbb{NP}$-"complete" for ℓ-length bounded planning if ℓ is encoded unary.
- For full PLANEX: theoretical limit $2^{|V|}$.
- Start with a small ℓ and increase until a solution is found.

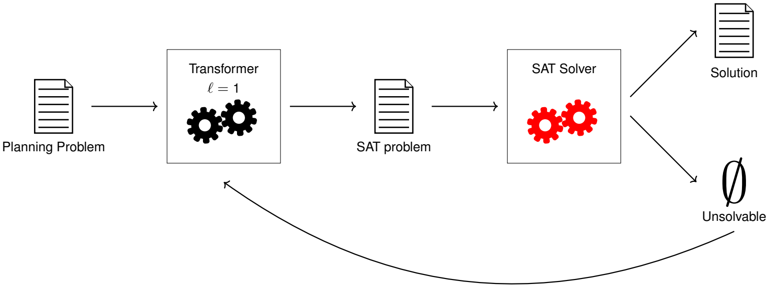
Bound Iteration



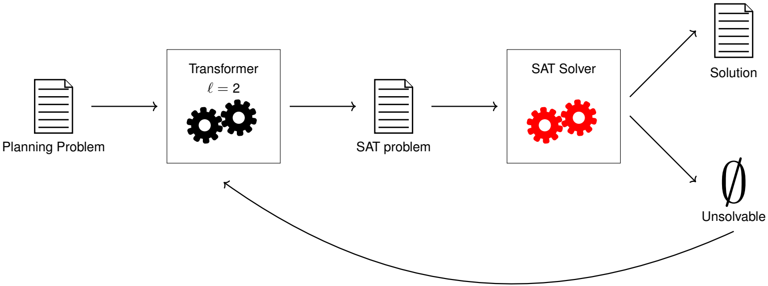
Bound Iteration



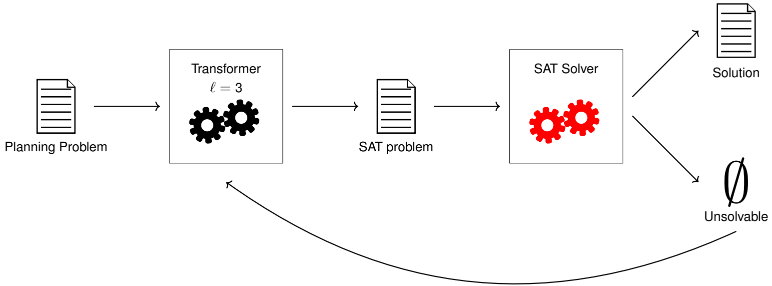
Bound Iteration



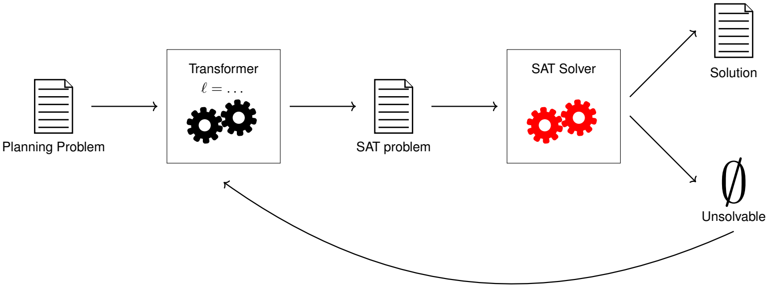
Bound Iteration



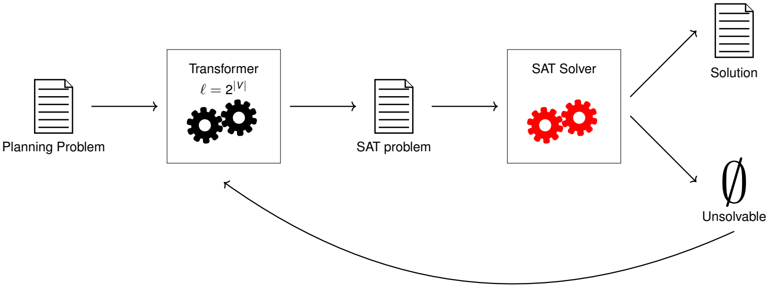
Bound Iteration



Bound Iteration



Bound Iteration



Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.

Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.
- For full PLANEX: upper bound depends on the problem!

Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.
- For full PLANEX: upper bound depends on the problem!
  - acyclic: maximum decomposable length

Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.
- For full PLANEX: upper bound depends on the problem!
  - acyclic: maximum decomposable length
  - totally-ordered: maximum decomposable length with depth $\left(2^{|V|}\right)^2 |C|$

Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.
- For full PLANEX: upper bound depends on the problem!
  - acyclic: maximum decomposable length
  - totally-ordered: maximum decomposable length with depth $\left(2^{|V|}\right)^2 |C|$
  - regular: exercise

Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.
- For full PLANEX: upper bound depends on the problem!
  - acyclic: maximum decomposable length
  - totally-ordered: maximum decomposable length with depth $\left(2^{|V|}\right)^2 |C|$
  - regular: exercise
  - tail-recursive: more complex, use stratification

Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.
- For full PLANEX: upper bound depends on the problem!
  - acyclic: maximum decomposable length
  - totally-ordered: maximum decomposable length with depth $\left(2^{|V|}\right)^2 |C|$
  - regular: exercise
  - tail-recursive: more complex, use stratification
  - general: ∞

Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.
- For full PLANEX: upper bound depends on the problem!
  - acyclic: maximum decomposable length
  - totally-ordered: maximum decomposable length with depth $\left(2^{|V|}\right)^2 |C|$
  - regular: exercise
  - tail-recursive: more complex, use stratification
  - general: ∞
- For general HTNs we can only construct algorithm that terminates on success but not (always) on failure.

Idea for Transforming HTN Planning

Why not do the same for HTN planning?

- Start with a small length bound ℓ and increase until a solution is found.
- For full PLANEX: upper bound depends on the problem!
  - acyclic: maximum decomposable length
  - totally-ordered: maximum decomposable length with depth $\left(2^{|V|}\right)^2 |C|$
  - regular: exercise
  - tail-recursive: more complex, use stratification
  - general: ∞
- For general HTNs we can only construct algorithm that terminates on success but not (always) on failure.
- "Is" PLANEX $\mathbb{NP}$-"complete" for ℓ-length bounded HTN planning if ℓ is encoded unary?

Idea for Transforming HTN Planning

Is *plan length* a good bound for HTN planning?

Idea for Transforming HTN Planning

Is *plan length* a good bound for HTN planning?
Not really.
A length bound follows easily from a depth bound, but not the other way around.

Idea for Transforming HTN Planning

Is *plan length* a good bound for HTN planning?
Not really.
A length bound follows easily from a depth bound, but not the other way around.
We use *decomposition depth* instead.

"Is" PLANEX $\mathbb{NP}$-"complete" for K-depth bounded HTN planning if K is encoded unary?

Idea for Transforming HTN Planning

Is *plan length* a good bound for HTN planning?
Not really.
A length bound follows easily from a depth bound, but not the other way around.
We use *decomposition depth* instead.

"Is" PLANEX $\mathbb{NP}$-"complete" for K-depth bounded HTN planning if K is encoded unary?

Issue?

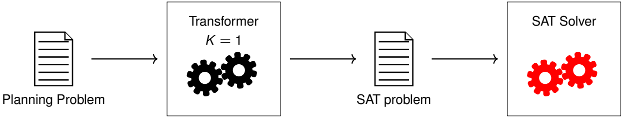
Idea for Transforming HTN Planning

Is *plan length* a good bound for HTN planning?
Not really.
A length bound follows easily from a depth bound, but not the other way around.
We use *decomposition depth* instead.

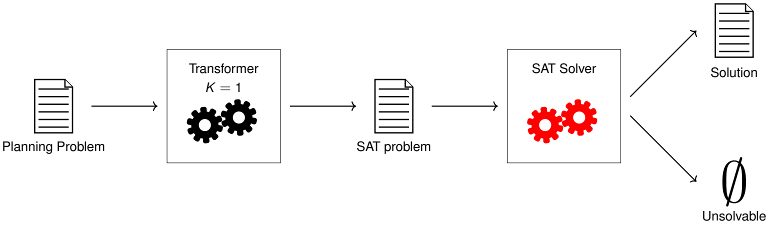
"Is" PLANEX $\mathbb{NP}$-"complete" for K-depth bounded HTN planning if K is encoded unary?

Issue? We loose optimality!

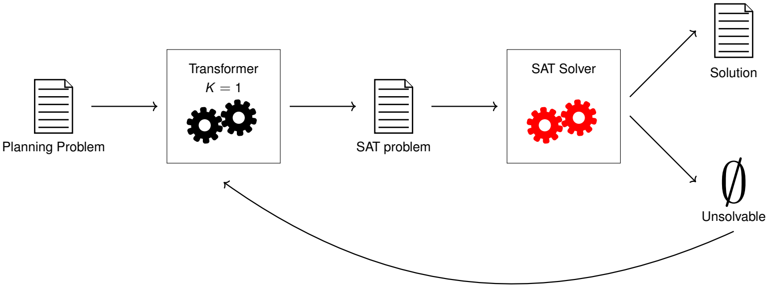
Bound Iteration



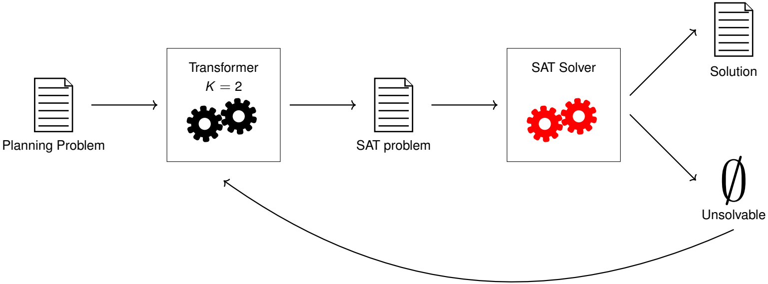
Bound Iteration



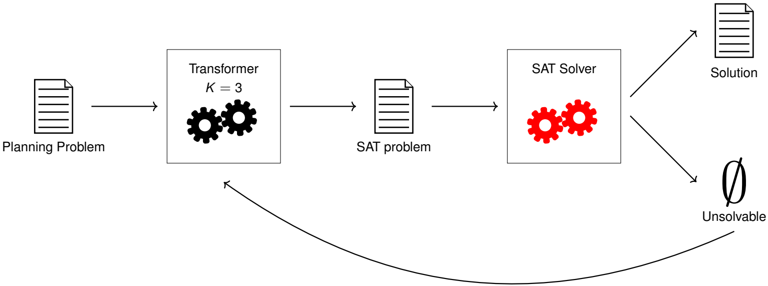
Bound Iteration



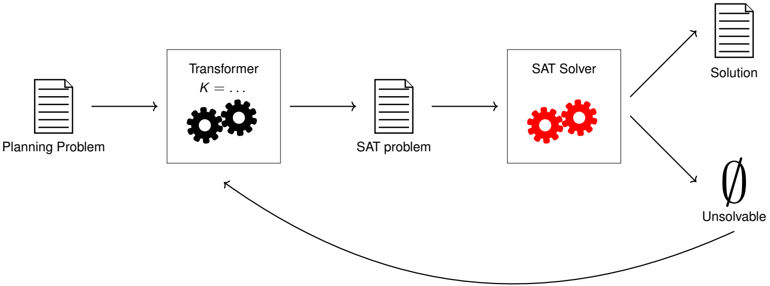
Bound Iteration



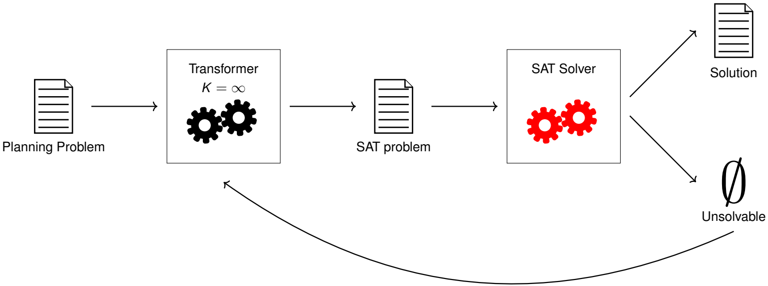
Bound Iteration



Bound Iteration



Bound Iteration



Our Objective

Given an HTN planning problem $\mathcal{P}$ and a depth bound K,
construct a CNF formula $\mathcal{F}$ that is satisfiable iff
$\mathcal{P}$ has a solution whose decomposition depth is $\leq K$

Our Objective

Given an HTN planning problem $\mathcal{P}$ and a depth bound K, construct a CNF formula $\mathcal{F}$ that is satisfiable iff $\mathcal{P}$ has a solution whose decomposition depth is $\leq K$

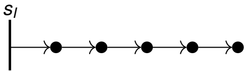
A satisfying valuation β of $\mathcal{F}$ should represent a solution to $\mathcal{P}$.

Solutions in HTN Planning

What is the *solution* to an HTN planning problem?

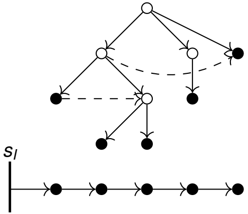
Solutions in HTN Planning

What is the *solution* to an HTN planning problem?



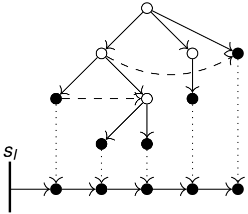
Solutions in HTN Planning

What is the *solution* to an HTN planning problem?



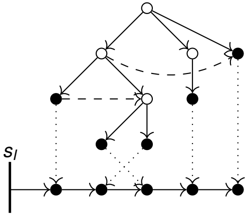
Solutions in HTN Planning

What is the *solution* to an HTN planning problem?



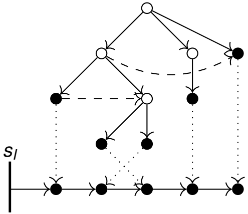
Solutions in HTN Planning

What is the *solution* to an HTN planning problem?



Solutions in HTN Planning

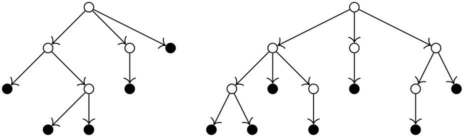
What is the *solution* to an HTN planning problem?



β should represent a *Decomposition Tree*.

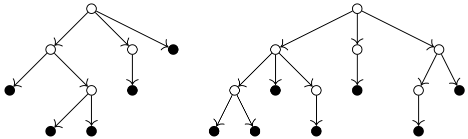
Representing Decomposition Trees Compactly

- There are doubly-exponentially many trees of depth $\leq K$



Representing Decomposition Trees Compactly

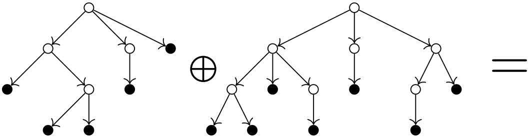
- There are doubly-exponentially many trees of depth $\leq K$
- We need a compact representation of all possible Decomposition Trees of depth $\leq K$



Representing Decomposition Trees Compactly

- There are doubly-exponentially many trees of depth $\leq K$
- We need a compact representation of all possible Decomposition Trees of depth $\leq K$

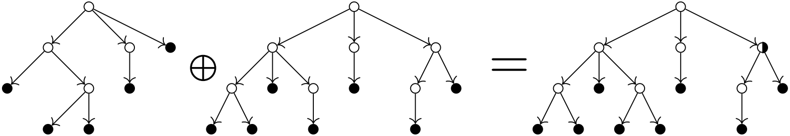
$\Rightarrow$ Construct a super-tree of all possible Decomposition Trees



Representing Decomposition Trees Compactly

- There are doubly-exponentially many trees of depth $\leq K$
- We need a compact representation of all possible Decomposition Trees of depth $\leq K$

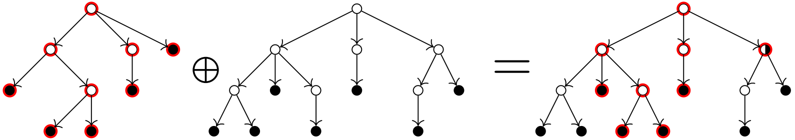
$\Rightarrow$ Construct a super-tree of all possible Decomposition Trees



Representing Decomposition Trees Compactly

- There are doubly-exponentially many trees of depth $\leq K$
- We need a compact representation of all possible Decomposition Trees of depth $\leq K$

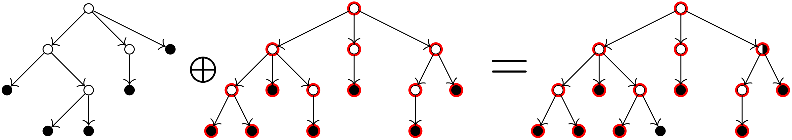
$\Rightarrow$ Construct a super-tree of all possible Decomposition Trees



Representing Decomposition Trees Compactly

- There are doubly-exponentially many trees of depth $\leq K$
- We need a compact representation of all possible Decomposition Trees of depth $\leq K$

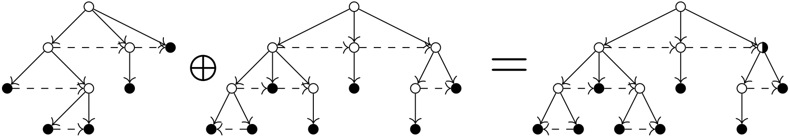
$\Rightarrow$ Construct a super-tree of all possible Decomposition Trees



Representing Decomposition Trees Compactly

- There are doubly-exponentially many trees of depth $\leq K$
- We need a compact representation of all possible Decomposition Trees of depth $\leq K$

$\Rightarrow$ Construct a super-tree of all possible Decomposition Trees



First, we assume *totally-ordered* planning problems.

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

$\Rightarrow$ Compute super-tree by step-wise local expansion

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

⇒ Compute super-tree by step-wise local expansion

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

$\Rightarrow$ Compute super-tree by step-wise local expansion

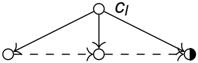
$\circ$ c_I

$c_I \rightarrow ABC$ and $c_I \rightarrow CBp$ and $c_I \rightarrow Ar$

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

⇒ Compute super-tree by step-wise local expansion



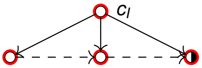
$c_I \rightarrow ABC$ and $c_I \rightarrow CBp$ and $c_I \rightarrow Ar$

$\{A, C\} \quad \{B\} \quad \{C, p, r\}$

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

⇒ Compute super-tree by step-wise local expansion



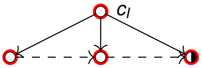
$c_I \to ABC$ and $c_I \to CBp$ and $c_I \to Ar$

$\{A, C\}$ $\{B\}$ $\{C, p, r\}$

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

$\Rightarrow$ Compute super-tree by step-wise local expansion



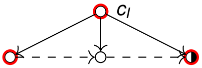
$c_I \rightarrow ABC$ and $c_I \rightarrow CBp$ and $c_I \rightarrow Ar$

$\{A, C\}$ $\{B\}$ $\{C, p, r\}$

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

$\Rightarrow$ Compute super-tree by step-wise local expansion



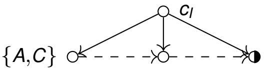
$c_l \rightarrow ABC$ and $c_l \rightarrow CBp$ and $c_l \rightarrow Ar$

$\{A, C\} \quad \{B\} \quad \{C, p, r\}$

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

$\Rightarrow$ Compute super-tree by step-wise local expansion



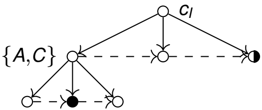
$c_I \rightarrow ABC$ and $c_I \rightarrow CBp$ and $c_I \rightarrow Ar$

$\{A, C\} \quad \{B\} \quad \{C, p, r\}$

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

$\Rightarrow$ Compute super-tree by step-wise local expansion



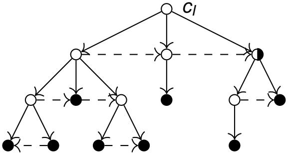
$c_I \rightarrow ABC$ and $c_I \rightarrow CBp$ and $c_I \rightarrow Ar$

$\{A, C\}$ $\{B\}$ $\{C, p, r\}$

Constructing Path Decomposition Trees

It's infeasable to compute **all** Decomposition Trees and **then** to merge them

$\Rightarrow$ Compute super-tree by step-wise local expansion



$c_I \rightarrow ABC$ and $c_I \rightarrow CBp$ and $c_I \rightarrow Ar$

$\{A, C\} \quad \{B\} \quad \{C, p, r\}$

Path Decomposition Trees with Partial Order

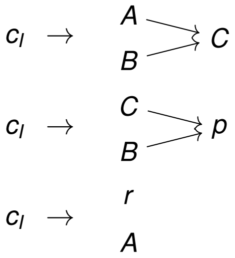
$$c_l \rightarrow ABC \text{ and } c_l \rightarrow CBp \text{ and } c_l \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

Path Decomposition Trees with Partial Order

$$c_l \rightarrow ABC \text{ and } c_l \rightarrow CBp \text{ and } c_l \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

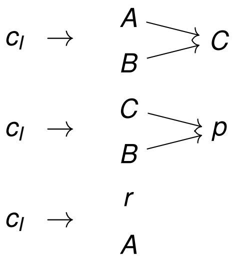


Path Decomposition Trees with Partial Order

$$c_I \rightarrow ABC \text{ and } c_I \rightarrow CBp \text{ and } c_I \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

1. Guess a linearization and check order later (tree encoding)

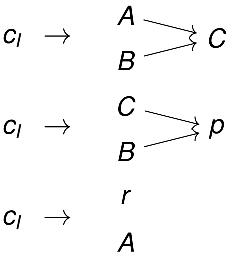


Path Decomposition Trees with Partial Order

$$c_l \rightarrow ABC \text{ and } c_l \rightarrow CBp \text{ and } c_l \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

1. Guess a linearization and check order later (tree encoding)
2. Merge task-labeled DAGs instead of task sequences

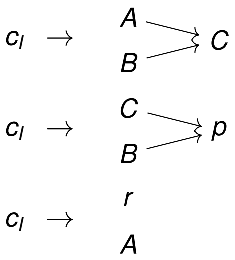


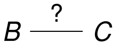
Path Decomposition Trees with Partial Order

$$c_l \rightarrow ABC \text{ and } c_l \rightarrow CBp \text{ and } c_l \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

1. Guess a linearization and check order later (tree encoding)
2. Merge task-labeled DAGs instead of task sequences



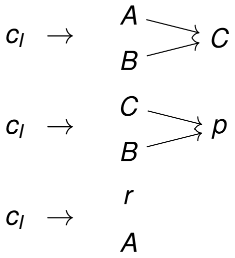


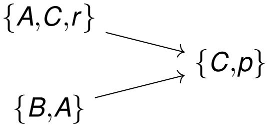
Path Decomposition Trees with Partial Order

$$c_l \rightarrow ABC \text{ and } c_l \rightarrow CBp \text{ and } c_l \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

1. Guess a linearization and check order later (tree encoding)
2. Merge task-labeled DAGs instead of task sequences



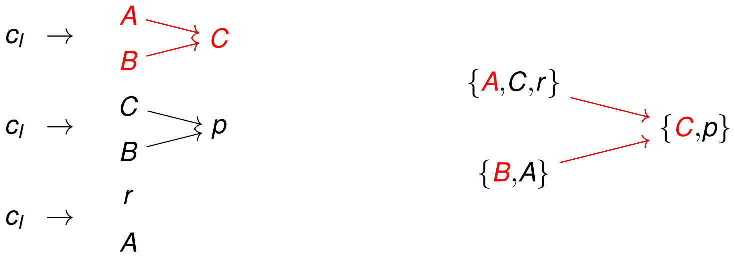


Path Decomposition Trees with Partial Order

$$c_I \rightarrow ABC \text{ and } c_I \rightarrow CBp \text{ and } c_I \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

1. Guess a linearization and check order later (tree encoding)
2. Merge task-labeled DAGs instead of task sequences

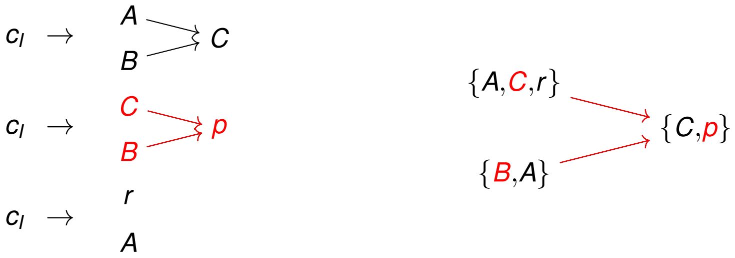


Path Decomposition Trees with Partial Order

$$c_I \rightarrow ABC \text{ and } c_I \rightarrow CBp \text{ and } c_I \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

1. Guess a linearization and check order later (tree encoding)
2. Merge task-labeled DAGs instead of task sequences

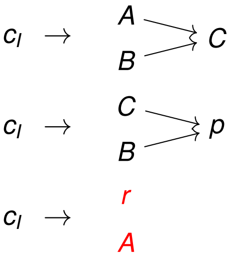


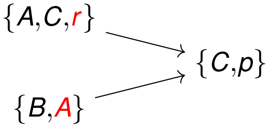
Path Decomposition Trees with Partial Order

$$c_l \rightarrow ABC \text{ and } c_l \rightarrow CBp \text{ and } c_l \rightarrow Ar$$
$$\{A, C\} \quad \{B\} \quad \{C, p, r\}$$

What about methods containing *partial* order?

1. Guess a linearization and check order later (tree encoding)
2. Merge task-labeled DAGs instead of task sequences





Merging Decomposition Methods

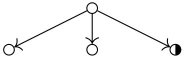
Merging Methods

Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

Merging Decomposition Methods

Merging Methods

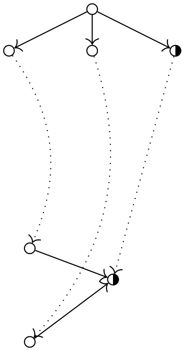
Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.



Merging Decomposition Methods

Merging Methods

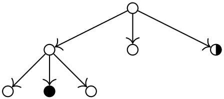
Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

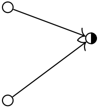


Merging Decomposition Methods

Merging Methods

Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

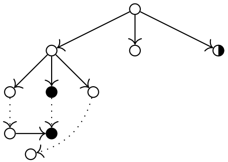


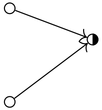


Merging Decomposition Methods

Merging Methods

Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

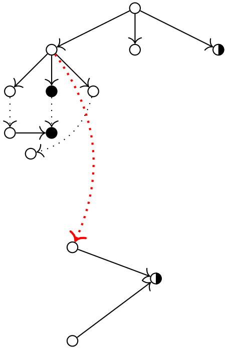




Merging Decomposition Methods

Merging Methods

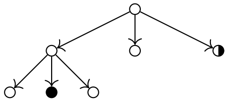
Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

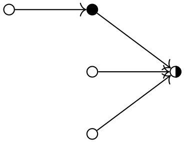


Merging Decomposition Methods

Merging Methods

Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

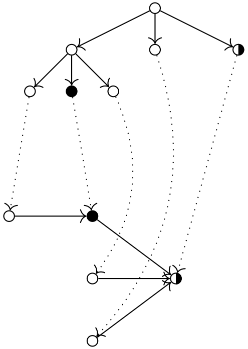




Merging Decomposition Methods

Merging Methods

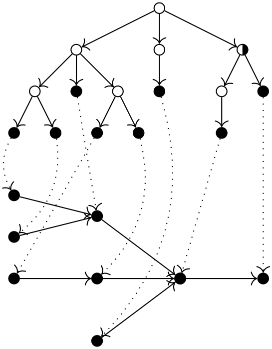
Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.



Merging Decomposition Methods

Merging Methods

Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

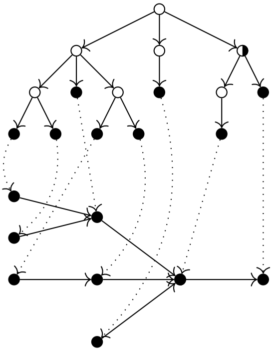


Merging Decomposition Methods

Merging Methods

Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

- Resulting ordering of leafs is called *Solution Order Graph S*

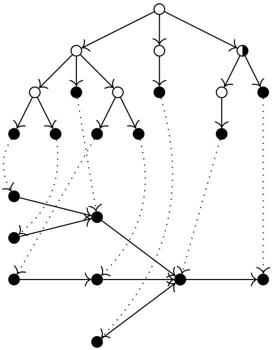


Merging Decomposition Methods

Merging Methods

Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

- Resulting ordering of leafs is called *Solution Order Graph S*
- **Any** task network derivable via decomposition is an induced subgraph of *S*

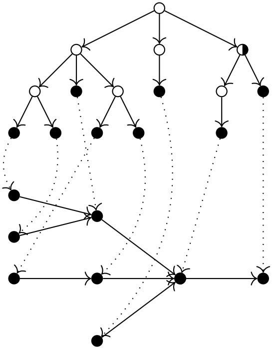


Merging Decomposition Methods

Merging Methods

Given a family G_i of vertex-labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

- Resulting ordering of leafs is called *Solution Order Graph S*
- **Any** task network derivable via decomposition is an induced subgraph of S
- When checking executability, we only have to consider the ordering in S, which is fixed – independent of selected methods



Minimising the SOG

Merging Methods

Given a family G_i of-vertex labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

Minimising the SOG

Merging Methods

Given a family G_i of-vertex labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

Difficult question: How does an optimal PDT look like?

Minimising the SOG

Merging Methods

Given a family G_i of-vertex labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

Difficult question: How does an optimal PDT look like?

- Fewer leafs?
- Fewer tasks per leaf?
- Fewer tasks per inner node?
- Fewer edges in the Solution Order Graph?

Minimising the SOG

Merging Methods

Given a family G_i of-vertex labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

Difficult question: How does an optimal PDT look like?

- Fewer leafs?
- Fewer tasks per leaf?
- Fewer tasks per inner node?
- Fewer edges in the Solution Order Graph?

$\Rightarrow$ Optimising number of children does not lead to global minimum!

Minimising the SOG

Merging Methods

Given a family G_i of-vertex labeled transitively closed DAGs.
Find a vertex set-labeled graph G^* s.t. all G_i are induced subgraphs of G^*.

Difficult question: How does an optimal PDT look like?

- Fewer leafs?
- Fewer tasks per leaf?
- Fewer tasks per inner node?
- Fewer edges in the Solution Order Graph?

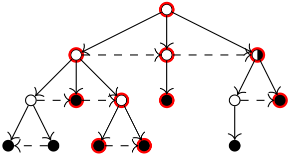
$\Rightarrow$ Optimising number of children does not lead to global minimum!

Theorem

Even minimising the size of G^* is $\mathbb{NP}$-complete.

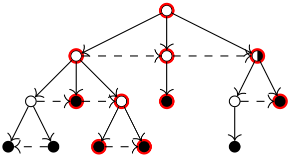
What are PDTs Good for?

- A PDT contains **every** Decomposition Tree of height $\leq K$ as a rooted sub-**tree**



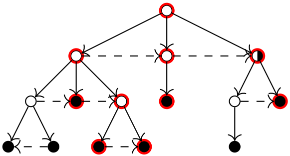
What are PDTs Good for?

- A PDT contains **every** Decomposition Tree of height $\leq K$ as a rooted sub-**tree**
- Let the valuation β of $\mathcal{F}$ describe such a tree



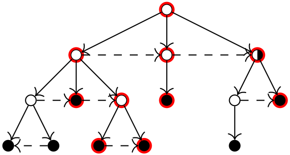
What are PDTs Good for?

- A PDT contains **every** Decomposition Tree of height $\leq K$ as a rooted sub-**tree**
- Let the valuation β of $\mathcal{F}$ describe such a tree
- The formula then asserts that it is a valid DT



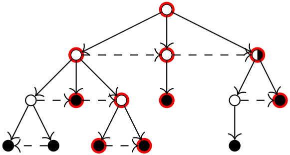
What are PDTs Good for?

- A PDT contains **every** Decomposition Tree of height $\leq K$ as a rooted sub-**tree**
- Let the valuation β of $\mathcal{F}$ describe such a tree
- The formula then asserts that it is a valid DT
- Two types of decision variables:



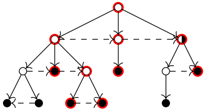
What are PDTs Good for?

- A PDT contains **every** Decomposition Tree of height $\leq K$ as a rooted sub-**tree**
- Let the valuation β of $\mathcal{F}$ describe such a tree
- The formula then asserts that it is a valid DT
- Two types of decision variables:
  - a^v – node v is part of the DT and is labeled with the task a
  - m^v – method m is applied to the task in node v



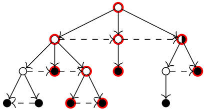
Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the *DT* and is labeled with the task a
  - m^v – method m is applied to the task in node v



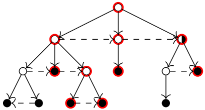
Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the *DT* and is labeled with the task a
  - m^v – method m is applied to the task in node v
- We then have to ensure the following properties for each node v:



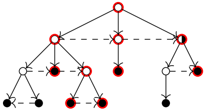
Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the DT and is labeled with the task a
  - m^v – method m is applied to the task in node v
- We then have to ensure the following properties for each node v:
  1. if a^v is true and $a \in C$, then $\bigvee_{m=(a,tn)\in M} m^v$



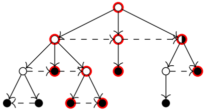
Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the *DT* and is labeled with the task a
  - m^v – method m is applied to the task in node v
- We then have to ensure the following properties for each node v:
  1. if a^v is true and $a \in C$, then $\bigvee_{m=(a,tn)\in M} m^v$
  2. if a^v is true and $a \in P$ or all a^v are false, then $\bigwedge_{m\in M} \neg m^v$



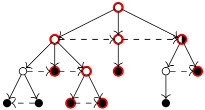
Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the *DT* and is labeled with the task a
  - m^v – method m is applied to the task in node v
- We then have to ensure the following properties for each node v:
  1. if a^v is true and $a \in C$, then $\bigvee_{m=(a,tn)\in M} m^v$
  2. if a^v is true and $a \in P$ or all a^v are false, then $\bigwedge_{m \in M} \neg m^v$
  3. if m^v is true, the children assigned by the PDT contain the correct tasks



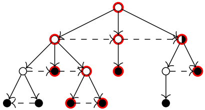
Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the DT and is labeled with the task a
  - m^v – method m is applied to the task in node v
- We then have to ensure the following properties for each node v:
  1. if a^v is true and $a \in C$, then $\bigvee_{m=(a,tn)\in M} m^v$
  2. if a^v is true and $a \in P$ or all a^v are false, then $\bigwedge_{m \in M} \neg m^v$
  3. if m^v is true, the children assigned by the PDT contain the correct tasks
  4. if all m^v are false, all $a^{v'}$ are false for all children v'



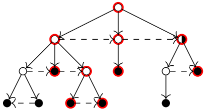
Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the DT and is labeled with the task a
  - m^v – method m is applied to the task in node v
- We then have to ensure the following properties for each node v:
  1. if a^v is true and $a \in C$, then $\bigvee_{m=(a,tn)\in M} m^v$
  2. if a^v is true and $a \in P$ or all a^v are false, then $\bigwedge_{m\in M} \neg m^v$
  3. if m^v is true, the children assigned by the PDT contain the correct tasks
  4. if all m^v are false, all $a^{v'}$ are false for all children v'
  5. at most one a^v and m^v is true



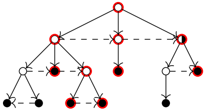
Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the DT and is labeled with the task a
  - m^v – method m is applied to the task in node v
- We then have to ensure the following properties for each node v:
  1. if a^v is true and $a \in C$, then $\bigvee_{m=(a,tn)\in M} m^v$
  2. if a^v is true and $a \in P$ or all a^v are false, then $\bigwedge_{m\in M} \neg m^v$
  3. if m^v is true, the children assigned by the PDT contain the correct tasks
  4. if all m^v are false, all $a^{v'}$ are false for all children v'
  5. at most one a^v and m^v is true
  6. c_l^r is true



Encoding – Overview

- Two types of decision variables:
  - a^v – node v is part of the *DT* and is labeled with the task a
  - m^v – method m is applied to the task in node v
- We then have to ensure the following properties for each node v:
  1. if a^v is true and $a \in C$, then $\bigvee_{m=(a,tn)\in M} m^v$
  2. if a^v is true and $a \in P$ or all a^v are false, then $\bigwedge_{m\in M} \neg m^v$
  3. if m^v is true, the children assigned by the PDT contain the correct tasks
  4. if all m^v are false, all $a^{v'}$ are false for all children v'
  5. at most one a^v and m^v is true *Really?*
  6. c_l^r is true



Encoding PDTs

$$\mathcal{F} \quad = \mathcal{F}(r) \wedge c_I^r \tag{6}$$

$\alpha(v)$ is the set of labels of each vertex of the PDT.
$E(v)$ are the children of v in the PDT.
For every method $m = (c, tn)$, let v_i be the child to which the task $t_{tn,i}$ is assigned.
$\mathbb{M}(A)$ is any encoding of the at-most-one constraint over the set of decision variables A.

Encoding PDTs

$$\mathcal{F} \quad = \mathcal{F}(r) \wedge c_I^r \tag{6}$$

$$\mathcal{F}(v) = \mathbb{M}(\{t^v \mid t \in \alpha(v)\}) \wedge \mathbb{M}(\{m^v \mid M(\alpha(v) \cap C)\}) \wedge \mathit{selectedMethod}(v) \\ \wedge \mathit{applyMethod}(v) \wedge \mathit{nonePresent}(v) \tag{5}$$

$\alpha(v)$ is the set of labels of each vertex of the PDT.
$E(v)$ are the children of v in the PDT.
For every method $m = (c, tn)$, let v_i be the child to which the task $t_{tn,i}$ is assigned.
$\mathbb{M}(A)$ is any encoding of the at-most-one constraint over the set of decision variables A.

Encoding PDTs

$$\mathcal{F} = \mathcal{F}(r) \wedge c_I^r \tag{6}$$

$$\mathcal{F}(v) = \mathbb{M}(\{t^v \mid t \in \alpha(v)\}) \wedge \mathbb{M}(\{m^v \mid M(\alpha(v) \cap C)\}) \wedge \mathit{selectedMethod}(v) \wedge \mathit{applyMethod}(v) \wedge \mathit{nonePresent}(v) \tag{5}$$

$$\mathit{selectedMethod}(v) = \left[\bigwedge_{m \in M(\alpha(v) \cap C)} (m^v \rightarrow t^v)\right] \wedge \left[\bigwedge_{t \in \alpha(v) \cap C} \left(t^v \rightarrow \bigvee_{m \in M(t)} m^v\right)\right] \tag{1\&2\&4}$$

$\alpha(v)$ is the set of labels of each vertex of the PDT.
$E(v)$ are the children of v in the PDT.
For every method $m = (c, tn)$, let v_i be the child to which the task $t_{tn,i}$ is assigned.
$\mathbb{M}(A)$ is any encoding of the at-most-one constraint over the set of decision variables A.

Encoding PDTs

$$\mathcal{F} = \mathcal{F}(r) \wedge c_I^r \quad (6)$$

$$\mathcal{F}(v) = \mathbb{M}(\{t^v \mid t \in \alpha(v)\}) \wedge \mathbb{M}(\{m^v \mid M(\alpha(v) \cap C)\}) \wedge \mathit{selectedMethod}(v) \wedge \mathit{applyMethod}(v) \wedge \mathit{nonePresent}(v) \quad (5)$$

$$\mathit{selectedMethod}(v) = \left[\bigwedge_{m \in M(\alpha(v) \cap C)} (m^v \rightarrow t^v)\right] \wedge \left[\bigwedge_{t \in \alpha(v) \cap C} \left(t^v \rightarrow \bigvee_{m \in M(t)} m^v\right)\right] \quad (1\&2\&4)$$

$$\mathit{applyMethod}(v) = \bigwedge_{m=(t,tn) \in M(\alpha(v))} \left[m^v \rightarrow \left(\bigwedge_{i=1}^{|tn|} t_{tn,i}^{v_i} \wedge \bigwedge_{v_i \in E(v) \setminus \{v_1,\dots,v_{|tn|}\}} \bigwedge_{t_* \in \alpha(v)} \neg t_*^{v_i}\right)\right] \quad (3)$$

$\alpha(v)$ is the set of labels of each vertex of the PDT.
$E(v)$ are the children of v in the PDT.
For every method $m = (c, tn)$, let v_i be the child to which the task $t_{tn,i}$ is assigned.
$\mathbb{M}(A)$ is any encoding of the at-most-one constraint over the set of decision variables A.

Encoding PDTs

$$\mathcal{F} = \mathcal{F}(r) \land c_I^r \tag{6}$$

$$\mathcal{F}(v) = \mathbb{M}(\{t^v \mid t \in \alpha(v)\}) \land \mathbb{M}(\{m^v \mid M(\alpha(v) \cap C)\}) \land \mathit{selectedMethod}(v) \land \mathit{applyMethod}(v) \land \mathit{nonePresent}(v) \tag{5}$$

$$\mathit{selectedMethod}(v) = \left[\bigwedge_{m \in M(\alpha(v)\cap C)} (m^v \to t^v)\right] \land \left[\bigwedge_{t \in \alpha(v)\cap C} \left(t^v \to \bigvee_{m \in M(t)} m^v\right)\right] \tag{1\&2\&4}$$

$$\mathit{applyMethod}(v) = \bigwedge_{m=(t,tn)\in M(\alpha(v))} \left[m^v \to \left(\bigwedge_{i=1}^{|tn|} t_{tn,i}^{v_i} \land \bigwedge_{v_i \in E(v)\setminus\{v_1,\ldots,v_{|tn|}\}} \bigwedge_{t_* \in \alpha(v)} \neg t_*^{\,v_i}\right)\right] \tag{3}$$

$$\mathit{nonePresent}(v) = \left(\bigwedge_{t\in\alpha(v)} \neg t^v\right) \to \left(\bigwedge_{v_i \in E(v)} \bigwedge_{t \in C\cup P} \neg t^{v_i}\right) \land \bigwedge_{t \in \alpha(v)\cap P} \left(t^v \to \bigwedge_{v_i\in E(v)} \bigwedge_{t\in C\cup P} \neg t^{v_i}\right) \tag{4\&2}$$

$\alpha(v)$ is the set of labels of each vertex of the PDT.
$E(v)$ are the children of v in the PDT.
For every method $m = (c, tn)$, let v_i be the child to which the task $t_{tn,i}$ is assigned.
$\mathbb{M}(A)$ is any encoding of the at-most-one constraint over the set of decision variables A.

What now?

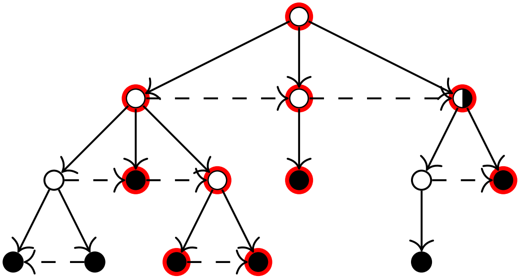
We have a formula $\mathcal{F}$ that is satisfiable iff it represents a valid Decomposition Tree T.

What now?

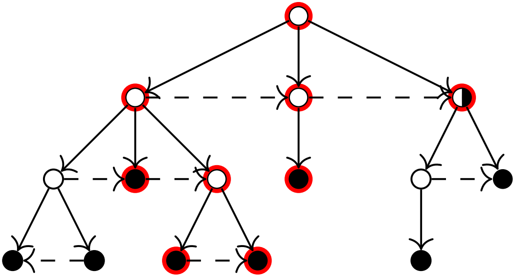
We have a formula $\mathcal{F}$ that is satisfiable iff it represents a valid Decomposition Tree T.

To ensure that it is a solution, we have to check whether the leafs of T are executable in s_I in a valid linearization.

Where are the leafs of T?

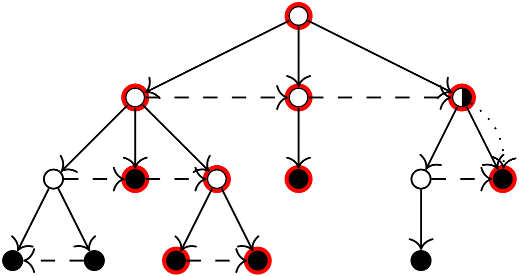


Where are the leafs of T?



A leaf of T could be any vertex of the PDT ...

Where are the leafs of T?



A leaf of T could be any vertex of the PDT ...
“inherit” them towards the leafs!

Encoding PDTs

$$\mathcal{F} = \mathcal{F}(r) \land c_I^r \tag{6}$$

$$\mathcal{F}(v) = \mathbb{M}(\{t^v \mid t \in \alpha(v)\}) \land \mathbb{M}(\{m^v \mid M(\alpha(v) \cap C)\}) \land \mathit{selectedMethod}(v) \land \mathit{applyMethod}(v) \land \mathit{nonePresent}(v) \tag{5}$$

$$\mathit{selectedMethod}(v) = \left[\bigwedge_{m \in M(\alpha(v) \cap C)} (m^v \to t^v) \right] \land \left[\bigwedge_{t \in \alpha(v) \cap C} \left(t^v \to \bigvee_{m \in M(t)} m^v \right) \right] \tag{1\&2\&4}$$

$$\mathit{applyMethod}(v) = \bigwedge_{m=(t,tn) \in M(\alpha(v))} \left[m^v \to \left(\bigwedge_{i=1}^{|tn|} t_{tn,i}^{v_i} \land \bigwedge_{v_i \in E(v) \setminus \{v_1,\ldots,v_{|tn|}\}} \bigwedge_{t_* \in \alpha(v)} \neg t_*^{v_i} \right) \right] \tag{3}$$

$$\mathit{nonePresent}(v) = \left(\bigwedge_{t \in \alpha(v)} \neg t^v \right) \to \left(\bigwedge_{v_i \in E(v)} \bigwedge_{t \in C \cup P} \neg t^{v_i} \right) \land \bigwedge_{t \in \alpha(v) \cap P} \left(t^v \to \bigwedge_{v_i \in E(v)} \bigwedge_{t \in C \cup P} \neg t^{v_i} \right) \tag{4\&2}$$

$\alpha(v)$ is the set of labels of each vertex of the PDT.
$E(v)$ are the children of v in the PDT.
For every method $m = (c, tn)$, let v_i be the child to which the task $t_{tn,i}$ is assigned.
$\mathbb{M}(A)$ is any encoding of the at-most-one constraint over the set of decision variables A.

Encoding PDTs

$$\mathcal{F} = \mathcal{F}(r) \wedge c_I^r \quad (6)$$

$$\mathcal{F}(v) = \mathbb{M}(\{t^v \mid t \in \alpha(v)\}) \wedge \mathbb{M}(\{m^v \mid M(\alpha(v) \cap C)\}) \wedge \mathit{selectedMethod}(v) \wedge \mathit{applyMethod}(v) \wedge \mathit{nonePresent}(v) \wedge \mathit{inheritPrimitive}(v) \quad (5)$$

$$\mathit{selectedMethod}(v) = \left[\bigwedge_{m \in M(\alpha(v) \cap C)} (m^v \to t^v)\right] \wedge \left[\bigwedge_{t \in \alpha(v) \cap C} \left(t^v \to \bigvee_{m \in M(t)} m^v\right)\right] \quad (1\&2\&4)$$

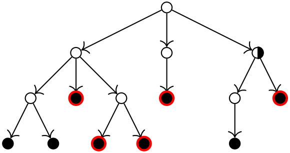
$$\mathit{applyMethod}(v) = \bigwedge_{m=(t,tn) \in M(\alpha(v))} \left[m^v \to \left(\bigwedge_{i=1}^{|tn|} t_{tn,i}^{v_i} \wedge \bigwedge_{v_i \in E(v) \setminus \{v_1,\ldots,v_{|tn|}\}} \bigwedge_{t_* \in \alpha(v)} \neg {t_*}^{v_i}\right)\right] \quad (3)$$

$$\mathit{nonePresent}(v) = \left(\bigwedge_{t \in \alpha(v)} \neg t^v\right) \to \left(\bigwedge_{v_i \in E(v)} \bigwedge_{t \in C \cup P} \neg t^{v_i}\right) \quad (4\&2)$$

$$\mathit{inheritPrimitive}(v) = \bigwedge_{p \in \alpha(v) \cap P} \left[p^v \to \left(p^{v_1} \wedge \bigwedge_{v_i \in E(v) \setminus \{v_1\}} \bigwedge_{k \in \alpha(v)} \neg k^{v_i}\right)\right]$$

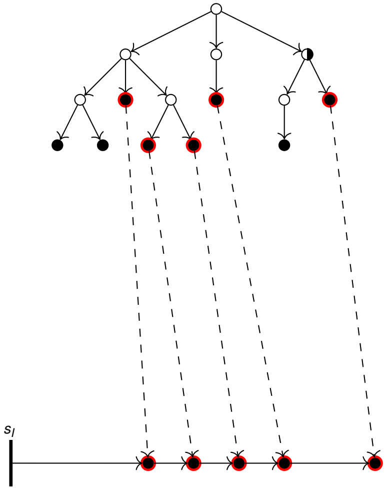
$\alpha(v)$ is the set of labels of each vertex of the PDT.
$E(v)$ are the children of v in the PDT.
For every method $m = (c, tn)$, let v_i be the child to which the task $t_{tn,i}$ is assigned.
$\mathbb{M}(A)$ is any encoding of the at-most-one constraint over the set of decision variables A.

Executability



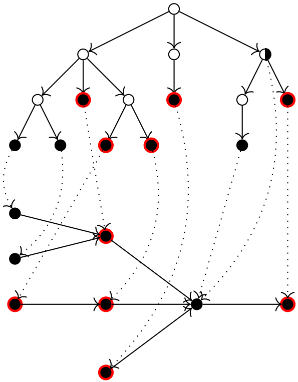
- β assigns primitive tasks to some leafs of the PDT

Executability



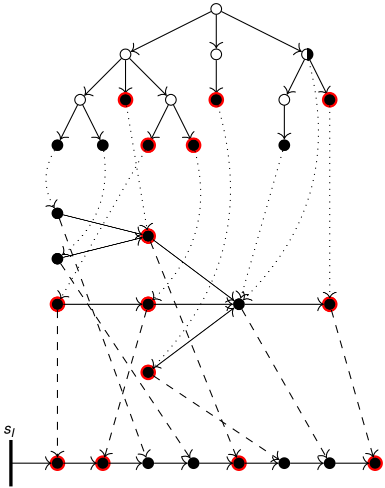
- β assigns primitive tasks to some leafs of the PDT
- A solution is an executable linearization of these tasks

Executability



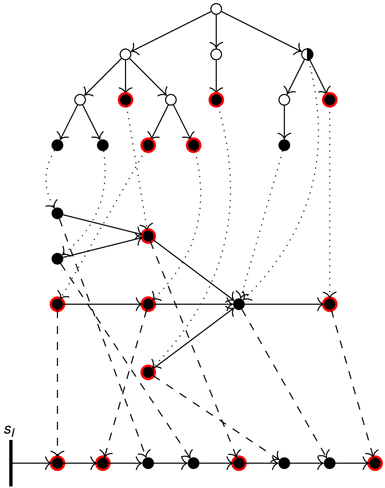
- β assigns primitive tasks to some leafs of the PDT
- A solution is an executable linearization of these tasks
- Linearization has to be compatible with the ordering represented by the SOG

Executability



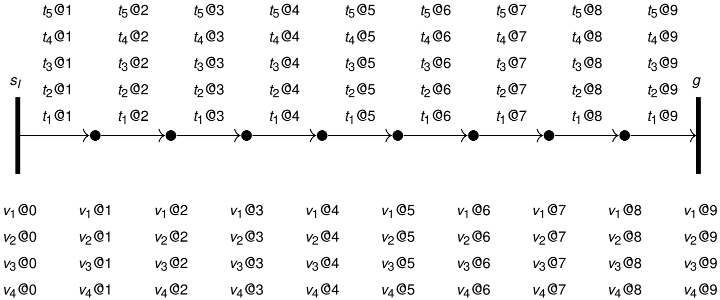
- β assigns primitive tasks to some leafs of the PDT
- A solution is an executable linearization of these tasks
- Linearization has to be compatible with the ordering represented by the SOG
- We represent a matching of the leafs to a sequence of timesteps and assert the correct order

Executability



- β assigns primitive tasks to some leafs of the PDT
- A solution is an executable linearization of these tasks
- Linearization has to be compatible with the ordering represented by the SOG
- We represent a matching of the leafs to a sequence of timesteps and assert the correct order
- We can use any classical encoding of executability!

Reminder: SAT Planning for Classical Problems – Decision Variables

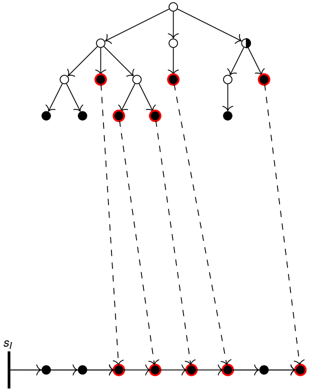


Two types of decision variables!

1. *t*@*i* – Action *t* is executed at time *i*.
2. *v*@*i* – State variable *v* is true at time *i*.

Executability – Matching Leafs to Timesteps

- $\overline{\ell i}$ – the leaf ℓ is matched to timestep i
- a^{ℓ} – the leaf ℓ is active, i.e. a task is assigned to it

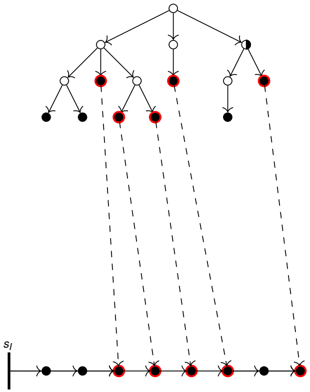


$\mathfrak{L}$ is the set of leafs of the PDT.

Executability – Matching Leafs to Timesteps

- $\overline{\ell i}$ – the leaf ℓ is matched to timestep i
- a^{ℓ} – the leaf ℓ is active, i.e. a task is assigned to it

$$\mathcal{F}_{exe} = F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6$$



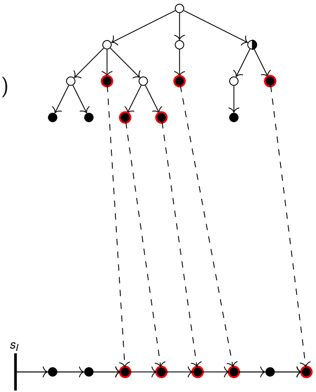
$\mathfrak{L}$ is the set of leafs of the PDT.

Executability – Matching Leafs to Timesteps

- $\overline{\ell i}$ – the leaf ℓ is matched to timestep i
- a^{ℓ} – the leaf ℓ is active, i.e. a task is assigned to it

$$\mathcal{F}_{exe} = F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6$$

$$F_1 = \bigwedge_{i=1}^{|\mathfrak{L}|} \mathbb{M}(\{\overline{\ell i} \mid \ell \in \mathfrak{L}\}) \wedge \bigwedge_{\ell \in \mathfrak{L}} \mathbb{M}(\{\overline{\ell i} \mid 1 \leq i \leq L\})$$



$\mathfrak{L}$ is the set of leafs of the PDT.

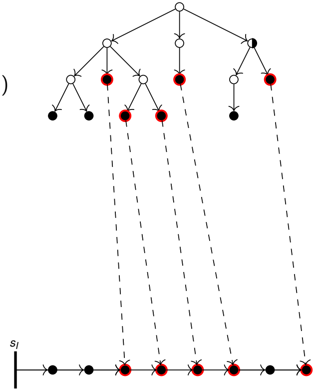
Executability – Matching Leafs to Timesteps

- $\overline{\ell i}$ – the leaf ℓ is matched to timestep i
- a^ℓ – the leaf ℓ is active, i.e. a task is assigned to it

$$\mathcal{F}_{exe} = F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6$$

$$F_1 = \bigwedge_{i=1}^{|\mathfrak{L}|} \mathbb{M}(\{\overline{\ell i} \mid \ell \in \mathfrak{L}\}) \wedge \bigwedge_{\ell \in \mathfrak{L}} \mathbb{M}(\{\overline{\ell i} \mid 1 \leq i \leq L\})$$

$$F_2 = \bigwedge_{\ell \in \mathfrak{L}} \left[\left(\neg \mathrm{a}^\ell \rightarrow \bigwedge_{p \in \alpha(\ell)} \neg p^\ell \right) \wedge \left(\mathrm{a}^\ell \rightarrow \bigvee_{p \in \alpha(\ell)} p^\ell \right) \right]$$



$\mathfrak{L}$ is the set of leafs of the PDT.

Executability – Matching Leafs to Timesteps

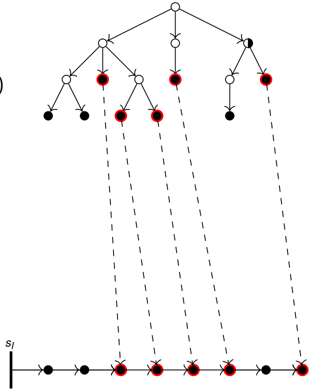
- $\overline{\ell i}$ – the leaf ℓ is matched to timestep i
- a^ℓ – the leaf ℓ is active, i.e. a task is assigned to it

$$\mathcal{F}_{exe} = F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6$$

$$F_1 = \bigwedge_{i=1}^{|\mathfrak{L}|} \mathbb{M}(\{\overline{\ell i} \mid \ell \in \mathfrak{L}\}) \wedge \bigwedge_{\ell \in \mathfrak{L}} \mathbb{M}(\{\overline{\ell i} \mid 1 \leq i \leq L\})$$

$$F_2 = \bigwedge_{\ell \in \mathfrak{L}} \left[\left(\neg \mathrm{a}^\ell \rightarrow \bigwedge_{p \in \alpha(\ell)} \neg p^\ell \right) \wedge \left(\mathrm{a}^\ell \rightarrow \bigvee_{p \in \alpha(\ell)} p^\ell \right) \right]$$

$$F_3 = \bigwedge_{\ell \in \mathfrak{L}} \left[\left(\neg \mathrm{a}^\ell \rightarrow \bigwedge_{1 \leq i \leq |\mathfrak{L}|} \neg \overline{\ell i} \right) \wedge \left(\mathrm{a}^\ell \rightarrow \bigvee_{1 \leq i \leq |\mathfrak{L}|} \overline{\ell i} \right) \right]$$



$\mathfrak{L}$ is the set of leafs of the PDT.

Executability – Matching Leafs to Timesteps

- $\overline{\ell i}$ – the leaf ℓ is matched to timestep i
- a^{ℓ} – the leaf ℓ is active, i.e. a task is assigned to it

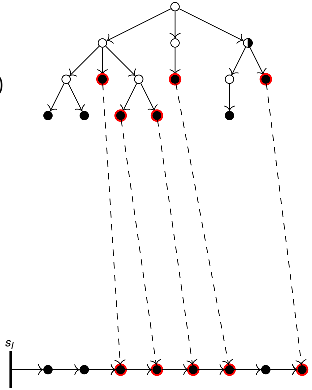
$$\mathcal{F}_{exe} = F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6$$

$$F_1 = \bigwedge_{i=1}^{|\mathfrak{L}|} \mathbb{M}(\{\overline{\ell i} \mid \ell \in \mathfrak{L}\}) \wedge \bigwedge_{\ell \in \mathfrak{L}} \mathbb{M}(\{\overline{\ell i} \mid 1 \leq i \leq L\})$$

$$F_2 = \bigwedge_{\ell \in \mathfrak{L}} \left[\left(\neg \mathrm{a}^{\ell} \rightarrow \bigwedge_{p \in \alpha(\ell)} \neg p^{\ell} \right) \wedge \left(\mathrm{a}^{\ell} \rightarrow \bigvee_{p \in \alpha(\ell)} p^{\ell} \right) \right]$$

$$F_3 = \bigwedge_{\ell \in \mathfrak{L}} \left[\left(\neg \mathrm{a}^{\ell} \rightarrow \bigwedge_{1 \leq i \leq |\mathfrak{L}|} \neg \overline{\ell i} \right) \wedge \left(\mathrm{a}^{\ell} \rightarrow \bigvee_{1 \leq i \leq |\mathfrak{L}|} \overline{\ell i} \right) \right]$$

$$F_4 = \bigwedge_{\ell \in \mathfrak{L}} \bigwedge_{t \in \alpha(\ell)} \bigwedge_{1 \leq i \leq |\mathfrak{L}|} t^{\ell} \wedge \overline{\ell i} \rightarrow t@i$$



$\mathfrak{L}$ is the set of leafs of the PDT.

Executability – Matching Leafs to Timesteps

- $\overline{\ell i}$ – the leaf ℓ is matched to timestep i
- a^ℓ – the leaf ℓ is active, i.e. a task is assigned to it

$$\mathcal{F}_{exe} = F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6$$

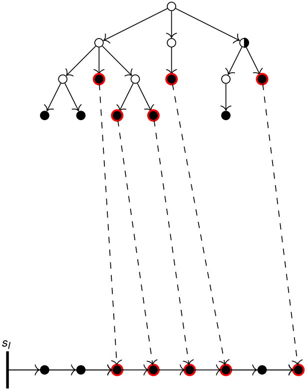
$$F_1 = \bigwedge_{i=1}^{|\mathfrak{L}|} \mathbb{M}(\{\overline{\ell i} \mid \ell \in \mathfrak{L}\}) \wedge \bigwedge_{\ell \in \mathfrak{L}} \mathbb{M}(\{\overline{\ell i} \mid 1 \leq i \leq L\})$$

$$F_2 = \bigwedge_{\ell \in \mathfrak{L}} \left[\left(\neg \mathrm{a}^\ell \rightarrow \bigwedge_{p \in \alpha(\ell)} \neg p^\ell \right) \wedge \left(\mathrm{a}^\ell \rightarrow \bigvee_{p \in \alpha(\ell)} p^\ell \right) \right]$$

$$F_3 = \bigwedge_{\ell \in \mathfrak{L}} \left[\left(\neg \mathrm{a}^\ell \rightarrow \bigwedge_{1 \leq i \leq |\mathfrak{L}|} \neg \overline{\ell i} \right) \wedge \left(\mathrm{a}^\ell \rightarrow \bigvee_{1 \leq i \leq |\mathfrak{L}|} \overline{\ell i} \right) \right]$$

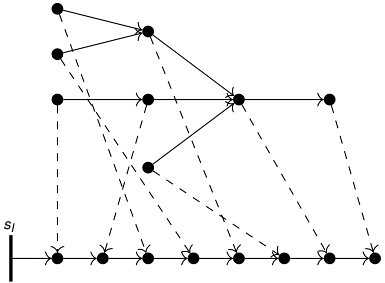
$$F_4 = \bigwedge_{\ell \in \mathfrak{L}} \bigwedge_{t \in \alpha(\ell)} \bigwedge_{1 \leq i \leq |\mathfrak{L}|} t^\ell \wedge \overline{\ell i} \rightarrow t@i$$

$$F_5 = \bigwedge_{1 \leq i \leq |\mathfrak{L}|} \left[\left(\bigwedge_{\ell \in \mathfrak{L}} \neg \overline{\ell i} \right) \rightarrow \left(\bigwedge_{t \in O} \neg t@i \right) \right]$$



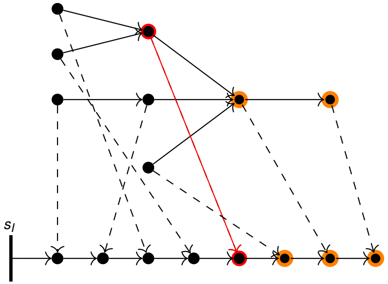
$\mathfrak{L}$ is the set of leafs of the PDT.

что делат? – Checking Order



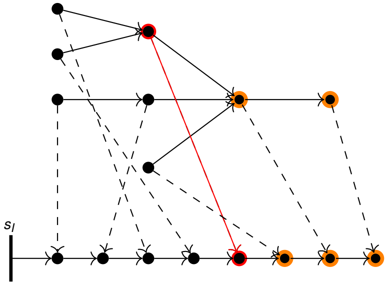
- So far, the matching does not check the order imposed by the methods.

что делат? – Checking Order



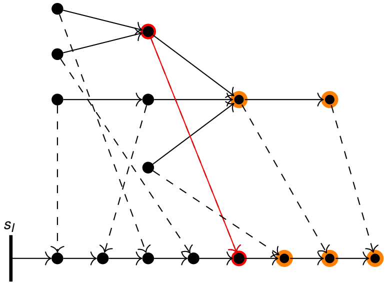
- So far, the matching does not check the order imposed by the methods.
- Since SOG S is fixed:
  If leaf l is matched to time t, all successors of l must be matched to time after t, i.e. cannot be matched to times before t

что делат? – Checking Order



- So far, the matching does not check the order imposed by the methods.
- Since SOG S is fixed:
  If leaf l is matched to time t, all successors of l must be matched to time after t, i.e. cannot be matched to times before t
- Using this property, we can reduce to $\mathcal{O}(n^3)$ clauses

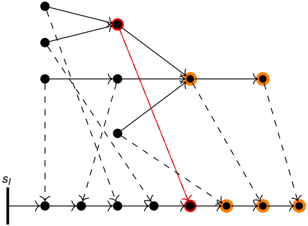
что делат? – Checking Order



- So far, the matching does not check the order imposed by the methods.
- Since SOG *S* is fixed:
  If leaf *l* is matched to time *t*, all successors of *l* must be matched to time after *t*, i.e. cannot be matched to times before *t*
- Using this property, we can reduce to $\mathcal{O}(n^3)$ clauses
- Often degenerates to $\mathcal{O}(n^2)$

Executability

- f_i^ℓ – matching the leaf ℓ to timestep i is forbidden (and implicitly also to any previous timestep)

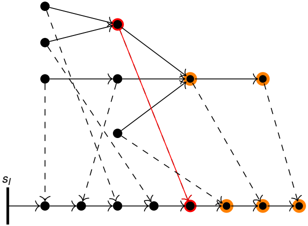


$\mathfrak{L}$ is the set of leafs of the PDT.
$\mathcal{S}$ is the Solution Order Graph.
$N_G^+(\ell)$ are the direct successors of vertex ℓ in the graph G.

Executability

- f_i^ℓ – matching the leaf ℓ to timestep i is forbidden (and implicitly also to any previous timestep)

$$F_6 = \bigwedge_{\ell \in \mathfrak{L}} \bigwedge_{1 \leq i \leq |\mathfrak{L}|} f_1(\ell, i) \wedge f_2(\ell, i) \wedge f_3(\ell, i) \wedge f_4(\ell, i)$$



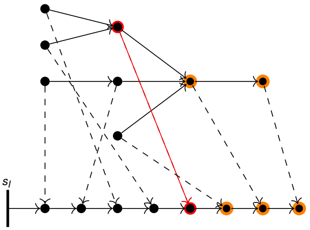
$\mathfrak{L}$ is the set of leafs of the PDT.
$\mathcal{S}$ is the Solution Order Graph.
$N_G^+(\ell)$ are the direct successors of vertex ℓ in the graph G.

Executability

- f_i^ℓ – matching the leaf ℓ to timestep i is forbidden (and implicitly also to any previous timestep)

$$F_6 = \bigwedge_{\ell \in \mathfrak{L}} \bigwedge_{1 \leq i \leq |\mathfrak{L}|} f_1(\ell, i) \wedge f_2(\ell, i) \wedge f_3(\ell, i) \wedge f_4(\ell, i)$$

$$f_1(\ell, i) = \text{if } i = 1 \text{ then } \mathit{true} \text{ else } \bigwedge_{\ell' \in N^+_{\mathcal{S}}(\ell)} \overline{\ell i} \rightarrow f^{\ell'}_{i-1}$$



$\mathfrak{L}$ is the set of leafs of the PDT.
$\mathcal{S}$ is the Solution Order Graph.
$N^+_G(\ell)$ are the direct successors of vertex ℓ in the graph G.

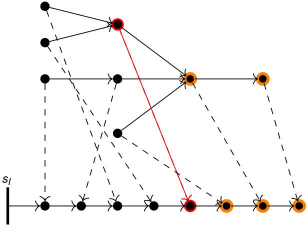
Executability

- f_i^ℓ – matching the leaf ℓ to timestep i is forbidden (and implicitly also to any previous timestep)

$$F_6 = \bigwedge_{\ell \in \mathfrak{L}} \bigwedge_{1 \leq i \leq |\mathfrak{L}|} f_1(\ell, i) \wedge f_2(\ell, i) \wedge f_3(\ell, i) \wedge f_4(\ell, i)$$

$$f_1(\ell, i) = \text{if } i = 1 \text{ then } \textit{true} \text{ else } \bigwedge_{\ell' \in N^+_{\mathcal{S}}(\ell)} \overline{\ell i} \rightarrow f^{\ell'}_{i-1}$$

$$f_2(\ell, i) = \bigwedge_{\ell' \in N^+_{\mathcal{S}}(\ell)} f_i^\ell \rightarrow f_i^{\ell'}$$



$\mathfrak{L}$ is the set of leafs of the PDT.
$\mathcal{S}$ is the Solution Order Graph.
$N^+_G(\ell)$ are the direct successors of vertex ℓ in the graph G.

Executability

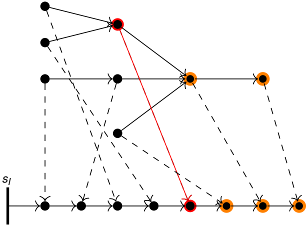
- f_i^ℓ – matching the leaf ℓ to timestep i is forbidden (and implicitly also to any previous timestep)

$$F_6 = \bigwedge_{\ell \in \mathfrak{L}} \bigwedge_{1 \leq i \leq |\mathfrak{L}|} f_1(\ell, i) \wedge f_2(\ell, i) \wedge f_3(\ell, i) \wedge f_4(\ell, i)$$

$$f_1(\ell, i) = \text{if } i = 1 \text{ then } \mathit{true} \text{ else } \bigwedge_{\ell' \in N^+_{\mathcal{S}}(\ell)} \overline{\ell i} \rightarrow f_{i-1}^{\ell'}$$

$$f_2(\ell, i) = \bigwedge_{\ell' \in N^+_{\mathcal{S}}(\ell)} f_i^\ell \rightarrow f_i^{\ell'}$$

$$f_3(\ell, i) = \text{if } i = 1 \text{ then } \mathit{true} \text{ else } f_i^\ell \rightarrow f_{i-1}^\ell$$



$\mathfrak{L}$ is the set of leafs of the PDT.
$\mathcal{S}$ is the Solution Order Graph.
$N^+_G(\ell)$ are the direct successors of vertex ℓ in the graph G.

Executability

- f_i^ℓ – matching the leaf ℓ to timestep i is forbidden (and implicitly also to any previous timestep)

$$F_6 = \bigwedge_{\ell \in \mathfrak{L}} \bigwedge_{1 \leq i \leq |\mathfrak{L}|} f_1(\ell, i) \wedge f_2(\ell, i) \wedge f_3(\ell, i) \wedge f_4(\ell, i)$$

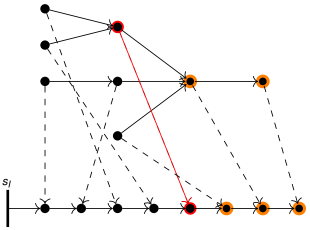
$$f_1(\ell, i) = \text{if } i = 1 \text{ then } \textit{true} \text{ else } \bigwedge_{\ell' \in N^+_{\mathcal{S}}(\ell)} \overline{\ell i} \rightarrow f_{i-1}^{\ell'}$$

$$f_2(\ell, i) = \bigwedge_{\ell' \in N^+_{\mathcal{S}}(\ell)} f_i^\ell \rightarrow f_i^{\ell'}$$

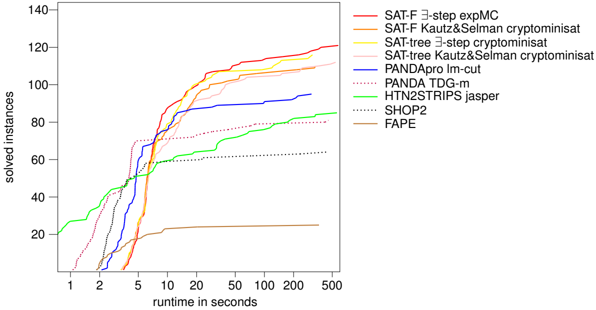
$$f_3(\ell, i) = \text{if } i = 1 \text{ then } \textit{true} \text{ else } f_i^\ell \rightarrow f_{i-1}^\ell$$

$$f_4(\ell, i) = f_i^\ell \rightarrow \neg \overline{\ell i}$$

$\mathfrak{L}$ is the set of leafs of the PDT.
$\mathcal{S}$ is the Solution Order Graph.
$N^+_G(\ell)$ are the direct successors of vertex ℓ in the graph G.



Evaluation – Partially-Ordered Problems [Behnke, Hller, Biundo, 2019]



		SAT-F ∃-step				SAT-F Kautz&Selman				SAT-tree ∃-step				SAT-tree				PANDApro			PANDA		HTN2STRIPS						
	#instances	expMC	MapleLCM	CaDiCaL	cryptominisat	expMC	MapleLCM	CaDiCaL	cryptominisat	expMC	MapleLCM	CaDiCaL	cryptominisat	expMC	MapleLCM	CaDiCaL	cryptominisat	lm-cut	FF	ADD	TDG-m	TDG-c	jasper	FD-SS 2018	Saarplan	LAPKT-BFWS	MpC	SHOP2	FAPE
UM-Translog	22	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	19	17	17	17	6	**22**	-
Satellite	25	**25**	**25**	**25**	**25**	24	24	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	24	23	**25**	21	23	19	14	12	0	22	22
Woodworking	11	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	10	9	9	8	10	5	5	5	5	4	8	0
SmartPhone	7	**7**	**7**	**7**	**7**	**7**	**7**	6	**7**	6	6	**7**	**7**	6	6	6	**7**	5	5	5	5	5	6	6	5	5	4	4	-
PCP	17	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	11	**12**	11	**12**	11	**12**	9	10	11	9	8	3	3	3	3	0	0	-
entertainment	12	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	11	11	**12**	9	9	5	5	5	4	4	5	-
rover	20	10	**11**	9	8	5	6	4	4	4	4	4	6	4	4	4	5	4	3	4	2	2	5	5	4	4	4	3	3
transport	30	**22**	20	20	20	15	14	15	17	**22**	20	19	21	15	15	15	18	9	11	7	1	1	19	17	13	13	3	0	-
total	144	**121**	120	118	117	108	108	107	110	114	112	111	116	106	107	106	112	95	95	93	81	78	85	77	66	63	25	64	25/56

Even undecidable problems can be solved via a translation into SAT.

We have introduced

- Path Decomposition Trees (PDTs)
- Solution Order Graphs (SOGs)
- An encoding for PDTs and SOGs into propositional logic

Behnke,Höller,Biundo, 2018 totSAT – Totally-ordered hierarchical planning through SAT

Behnke,Höller,Biundo, 2018 Tracking Branches in Trees – A Propositional Encoding for Solving Partially-Ordered HTN Planning Problems

Behnke,Höller,Biundo, 2019 Bringing order to chaos – A compact representation of partial order in SAT-based HTN planning