

Factored Planning: How, When, and When Not

Carmel Domshlak - Technion, Israel

Joint work with *Ronen Brafman* - Stanford/NASA (on leave from Ben-Gurion Univ.)

State model for Classical AI Planning

- finite state space S
- an initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- applicable actions $A(s) \subseteq A$ for $s \in S$
- a transition function $s' = f(a, s)$ for $a \in A(s)$
- cost function $c : A^* \rightarrow [0, \infty)$

A **solution** is a sequence of applicable actions that maps s_0 into S_G

An **optimal solution** minimizes c

Planning Languages

Key issue:

- state models represented **implicitly** in a **declarative language**
- representations use **state variables** to represent different aspects of the overall state of the system

Play two roles

- **specification**: concise model description
- **computation**: reveal useful info about problem's *structure*

The (simplified) SAS⁺ language (Bäckström & Nebel, '95)

A **problem** in SAS⁺ is a tuple $\langle V, A, I, G \rangle$

- V is a set of n finite-domain **variables**
- A is a set of **actions**
- I is a complete assignment to V (**initial state**)
- G is a partial assignment to V (**goal situation**)

Actions $a \in A$ represented by two partial assignments to V :

- the **Precondition** $\text{pre}(a)$
- the **Effect** $\text{eff}(a)$

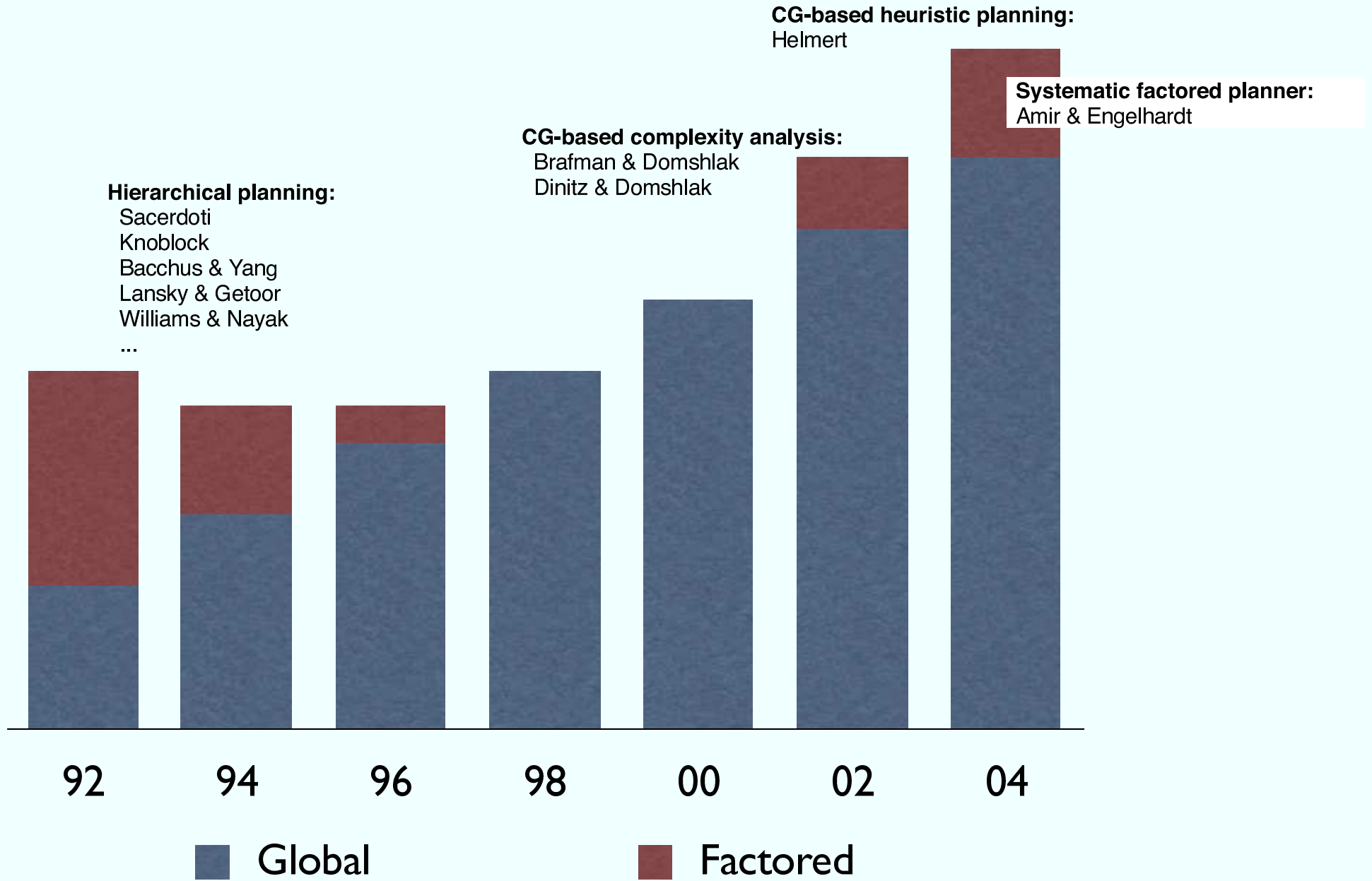
A problem in SAS⁺ determines a state model.

Factored planning

- *Basic problem*: state space of size $\exp(n)$
- *Basic idea*: **divide & conquer**
 1. reformulate the problem as a set of smaller problems over the subspaces of the overall state space
 2. solve each of these subproblems
 3. combine the local solutions to obtain a solution to the original problem
- *Space division*:
 - **vertical** (in contrast to *horizontal*)
 - each subspace $S_{(i)} \subseteq S$ is defined in terms of a variable subset $V_{(i)} \subseteq V$
 - $V_{(1)}, \dots, V_{(r)}$ are called the **factors** of the division
- *Potential gain*: sub-problems are each exponentially easier

Factored planning

- *Basic problem*: state space of size $exp(n)$
- *Basic idea*: **divide & conquer**
- *Potential gain*: sub-problems are each exponentially easier
- *Potential problems*:
 - sub-problems not informed about global structure
 - backtracking between sub-problems possible
 - composing sub-solutions may be expensive
- *Main research questions*:
 - How to factor and merge? Complexity for a given factorization? How does it compare to unfactored planning?
 - Guaranteed properties for FP-generated plans? How does it compare to non-factored planning?



(Undirected) Causal Graph CG

Nodes of CG are problem variables V , and $(v, u) \in CG$ iff:

- some action that changes v is preconditioned by a value of u , or
- some action changes both v and u

Example:

- R rocket location
- E, M possible package locations
- F fuel tank state (full/empty)
- Actions:
 - $fly-@l1-@l2$
 - $(un)load-@p-@l$
 - $fuel$



First things first

How to factor and merge?

- Factoring \Rightarrow **factor = variable** (for now)
- Focus on:
 1. how to combine a set of *given* local plans for different factors,
 2. how to generate these local plans.

Step 1: Sequences Combination Problem

- **Assumption:** For every $v \in V$, we are given a set of **pre-scheduled action sequences** $SPlan(v)$ where
 - each $\rho \in SPlan(v)$ is a finite sequence of pairs (a, t) , such that
 - a is an action *affecting* v , and
 - t is the time point at which a is to be performed.
- **Task:** Construct a global plan using these n sets of action sequences $SPlan(v)$.

Sequence-based Planning as CSP

- **Assumption:** For every $v \in V$, we are given a set of **pre-scheduled action sequences** $SPlan(v)$ where
 - each $\rho \in SPlan(v)$ is a finite sequence of pairs (a, t) , such that
 - a is an action *affecting* v , and
 - t is the time point at which a is to be performed.
- **Task:** Construct a global plan using these n sets of action sequences $SPlan(v)$.
- **Solution:** CSP (denoted *SeqCSP*) over n variables X_1, \dots, X_n where:
 1. The domain of X_i is exactly $SPlan(v_i)$, and
 2. The constraints of *SeqCSP* *bijectionally* correspond to the edges of the causal graph *CG*.

From Sequences Combination to Planning

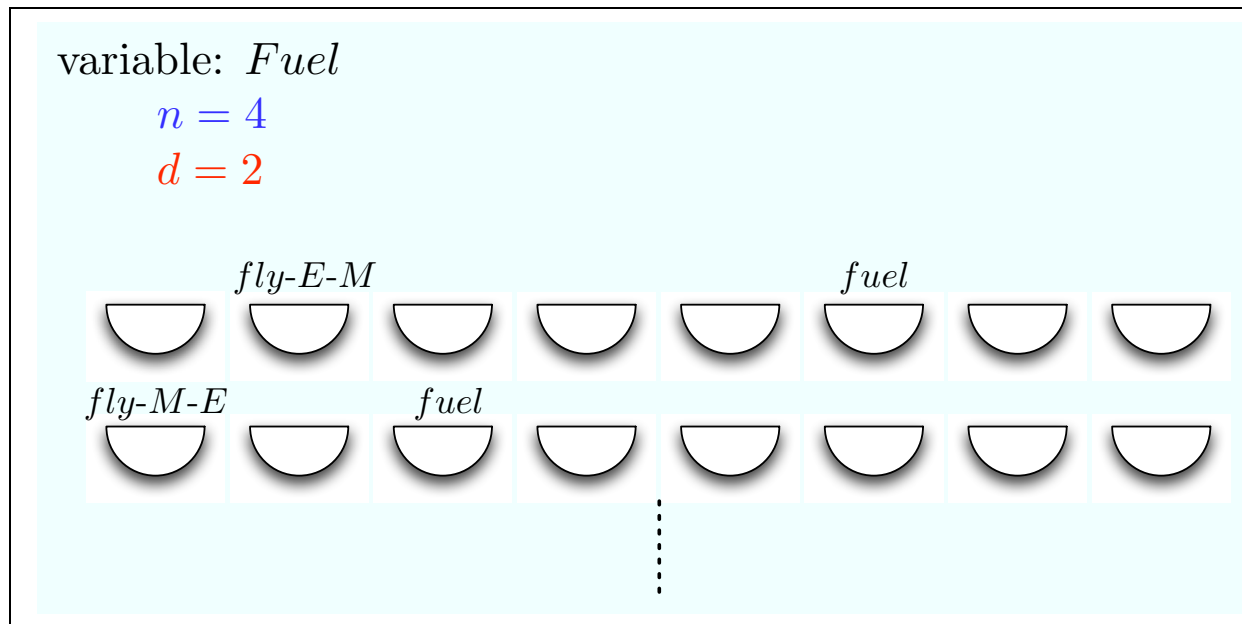
procedure LID

$d := 1$

loop

for $i := 1 \dots n$ **do**

$Dom(X_i) :=$ all sub-plans for v_i of length up to d ,
over all schedules across nd time points.



From Sequences Combination to Planning

procedure LID

$d := 1$

loop

for $i := 1 \dots n$ **do**

$Dom(X_i) :=$ all sub-plans for v_i of length up to d ,
over all schedules across nd time points.

Construct $SeqCSP(d)$ over X_1, \dots, X_n .

if (solve-csp($SeqCSP(d)$)) **then**

Reconstruct a plan ρ from the solution of $SeqCSP(d)$.

return ρ

else

$d := d + 1$

endloop

Correctness and *Local Optimality*

Theorem 1

- LID is a sound and complete planning algorithm.
- If LID terminates with a plan ρ at iteration δ , then, for any plan ρ' , there exists a state variable that changes its value on ρ' at least δ times.

Intuition behind δ :

Π cannot be solved if we force *all* problem variables to be changed less than δ times *each*.

Complexity

Theorem 2 Given a planning problem Π , it can be solved using LID in time

$$O(n(n\delta a)^{w\delta+\delta}) \quad (1)$$

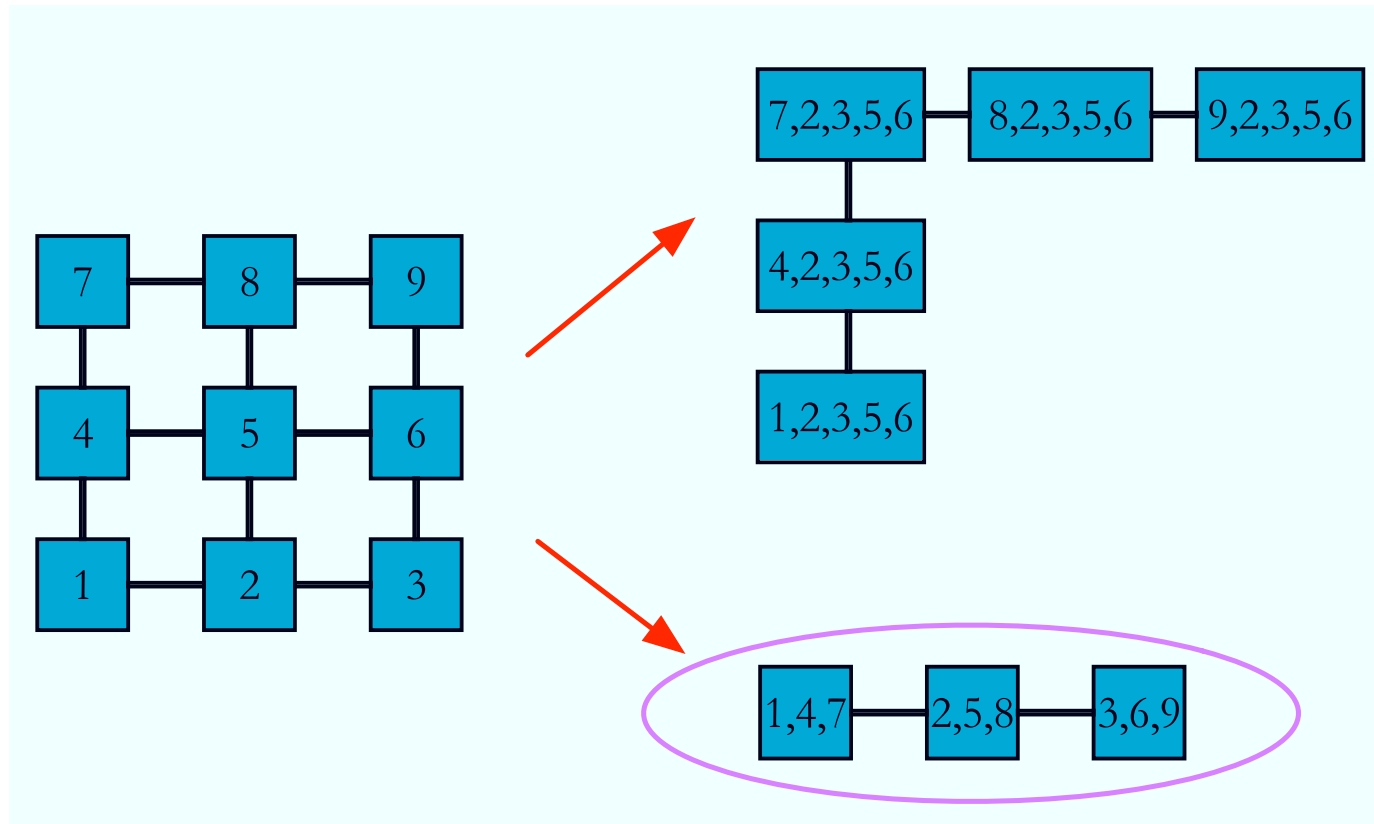
where $a = \max_i \{|A_i|\}$, w is the *tree-width* of CG , and δ is the *local depth* of Π , defined as:

$$\delta = \min_{\rho \in Plan(\Pi)} \max_{1 \leq i \leq n} \{|\rho_i|\} \quad (2)$$

Intuition behind δ :

Π cannot be solved if we force *all* problem variables to be changed less than δ times *each*.

Tree-width of graphs by example



Factored vs. Non-factored planning: Plan Optimality

Standard notions of **plan optimality**

- Sequential optimality (OP): minimize number of actions
- Step-optimality (SOP): minimize number of concurrent steps
 - can be of interest on its own, or
 - a reasonable compromise when OP is beyond reach.

Factored vs. Non-factored planning: Plan Optimality

Standard notions of **plan optimality**

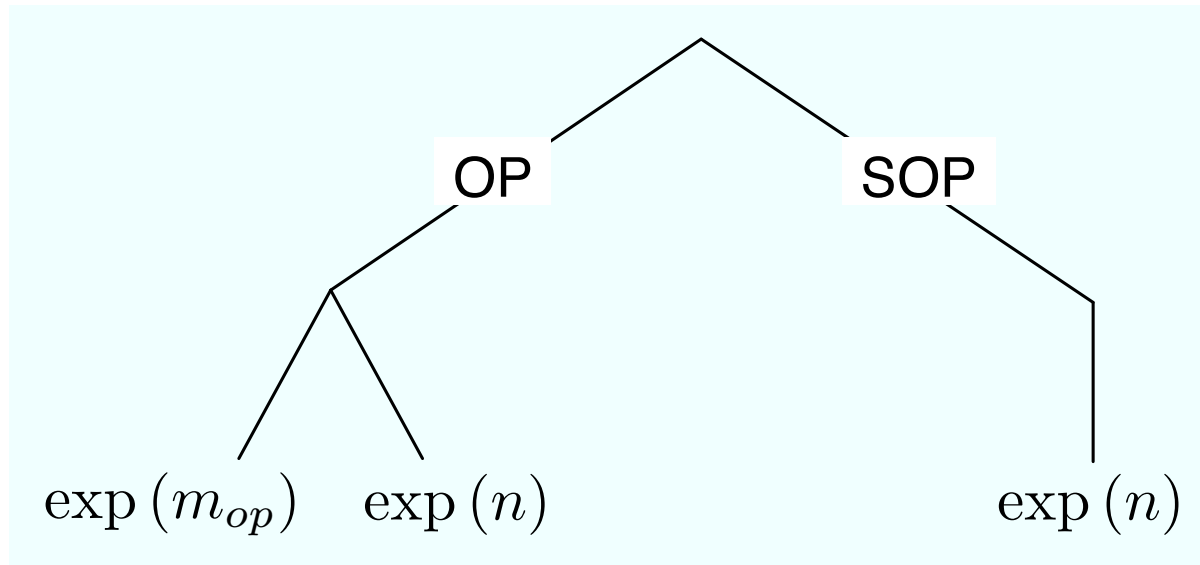
- Sequential optimality (OP): minimize number of actions
- Step-optimality (SOP): minimize number of concurrent steps
 - can be of interest on its own, or
 - a reasonable compromise when OP is beyond reach.
- Local optimality (LOP) guaranteed by LID is not any different.

Lemma 1 Given a planning problem Π , let m_{op} , m_{sop} , m_{lop} denote the number of actions in an optimal, step-optimal, and locally optimal plan, respectively.

We have that $m_{sop} \leq n \cdot m_{op}$ and $m_{lop} \leq n \cdot m_{op}$, and both these bounds are tight.

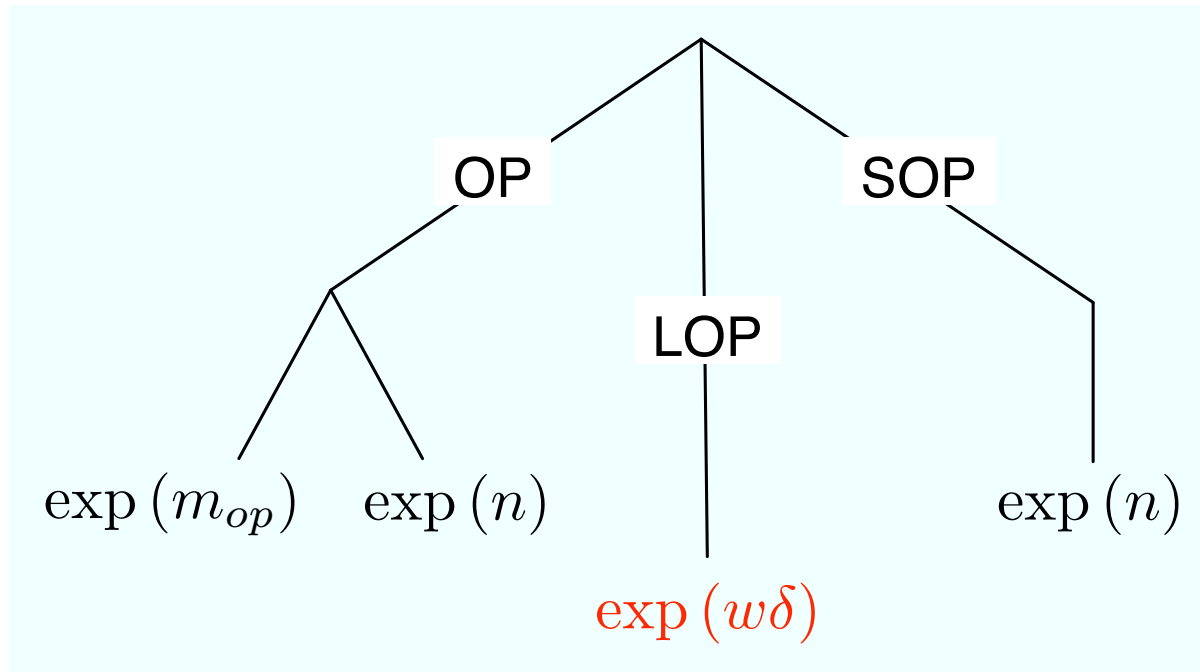
Time Complexity

Worst-case **time complexity** guarantees



Factored vs. Non-factored planning: Complexity

Worst-case **time complexity** guarantees



Causal graph in standardly optimal planning?

The DK encoding for planning-as-CSP (Do & Kambhampati 01)

- Fix upper bound m on plan step-length
- CSP variables $v_i^{[k]}$ correspond to problem variables v_i at time points $k \in 1, \dots, m$
- CSP variable domains: actions affecting corresponding problem variables, plus noops
- CSP constraints (informally)
 - Initial state holds at $t = 0$, goal situation at $t = m$
 - If $v_i^{[k]} = a$, then $Pre(a)$ are provided at $t = k - 1$ and preserved at $t = k$
 - If a affects both v_i and v_j , then $v_i^{[k]} = a \iff v_j^{[k]} = a$
- We ignore here: additional constraints from planning graph that capture certain reachability information

Complexity of DK-based SOP

The DK encoding + iterative deepening on $m =$ step-optimal plan

Theorem 3 A planning problem with minimal plan step-length m , and causal graph tree-width w can be solved in time:

$$O(\min \{nm \cdot a^{n+1}, nm \cdot a^{wm+1}\}),$$

Complexity of DK-based SOP

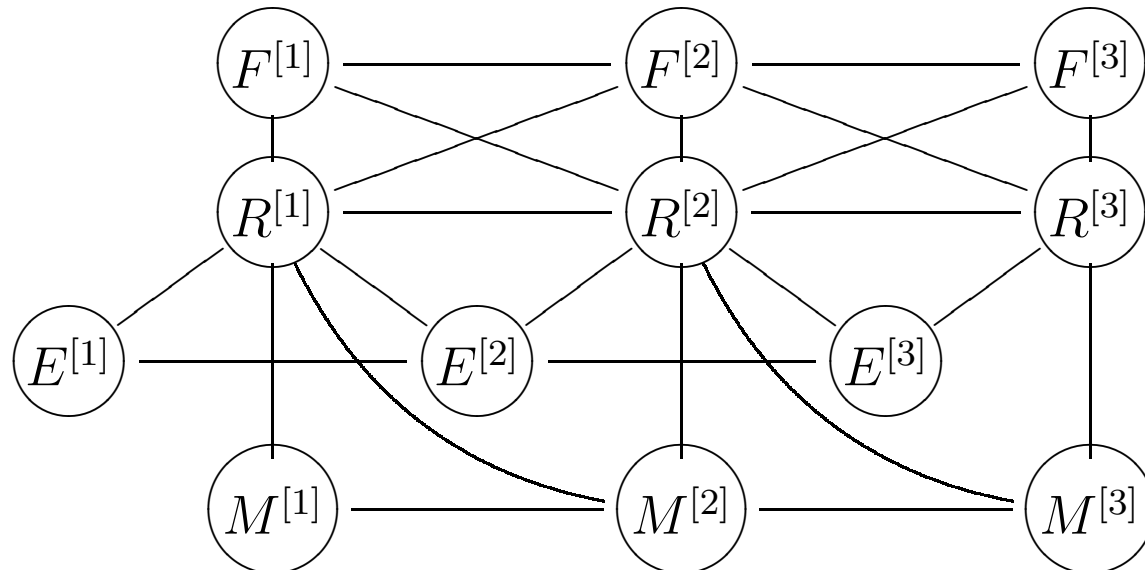
The DK encoding + iterative deepening on $m =$ step-optimal plan

Theorem 4 A planning problem with minimal plan step-length m , and causal graph tree-width w can be solved in time:

$$O(\min \{nm \cdot a^{n+1}, nm \cdot a^{wm+1}\}),$$

Key: Tree-width of DK constraint graph $\leq \min \{wm, n\}$

(+ some tight bounds, e.g., tree-width = n for $m > n$.)



Factored vs. Non-factored: Summary of Complexity Guarantees

- Factored planning is a win when $w\delta = o(n)$
 - Each factor contributes a relatively (to n) small number of actions to the plan
 - Causal graph is not too dense
- Factored planning is more likely to scale up
 - Does not depend on problem size and plan length directly
- Unfactored planning does not scale up with problem size
 - Unless the plans are short . . . in which case factored planning is better, provided CG is not too dense

Is LID practically interesting?

- LID is appealing when
 - plans not too long in relation to domain size
 - relatively well balanced
- Is this setting of any relevance?

Is LID practically interesting?

- LID is appealing when
 - plans not too long in relation to domain size
 - relatively well balanced
- Is this setting of any relevance?

YES!

- **All** *solved* instances in recent planning competitions have short step-optimal plans relative to n (indirectly from Hoffman, Subharwal, & Domshlak, '06)
- Recent results indicate each action is used only a few times, and typically only once! (Geffner & Vidal, '06)
- Over-subscription and distributed planning problems seem ideal *a priori*

Comparison with (Amir and Engelhardt '03)

- **Similarity**

- factoring + sequence of planning phases in iterative deepening on d
- similar guarantees of local optimality
- factoring is based on a *certain* graphical problem analysis

- **Difference**

- everything else

- **Complexity guarantees?**

Comparison with (Amir and Engelhardt '03)

- **Similarity**

- factoring + sequence of planning phases in iterative deepening on d
- similar guarantees of local optimality
- factoring is based on a *certain* graphical problem analysis

- **Difference**

- everything else

- **Complexity guarantees**

- No advantage to AE, and up to $\exp(n)$ reduction with LID
- **Key:** AE has to rely on graphs that are *strictly* denser than causal graphs

Generalized factoring

Recall: we started with *factor = single variable*.

- Suppose we combine some v_1, \dots, v_k to form a single factor
- Same algorithm LID, but now a single local plan handles v_1, \dots, v_k
- Changes in *SeqCSP*:
 - ☺ Variable domain size increases, but only linearly in k
 - ☹ Local depth δ may increase to $\delta_1 + \dots + \delta_k$.
- So is it **good** or **bad**?

Tree decomposition of the causal graph

Fact: Every graph with tree-width w , can be transformed into a join-tree with node size $w + 1$

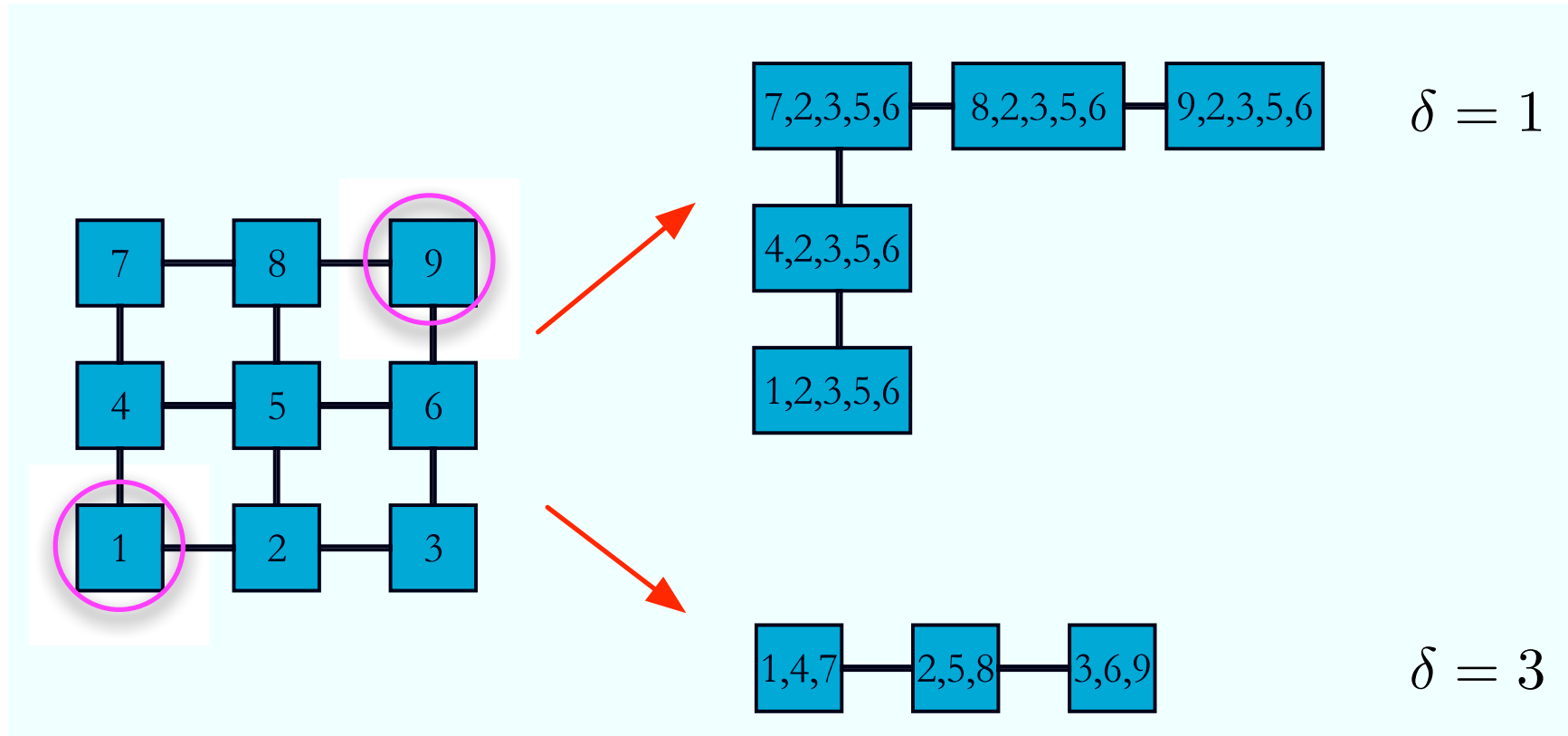
- Factoring = optimal join-tree of CG
 - Join-tree width is 1 (because it is a *tree*)
- NP-complete, but not a “too bad” NP-complete [Amir01]
 - Join-tree with node size $4w + 1$ in time $\exp(w)$
 - Join-tree with node size $\log(n)w + 1$ in time $\text{poly}(n)$

Tree decomposition of the causal graph

Fact: Every graph with tree-width w , can be transformed into a join-tree with node size $w + 1$

- Factoring = optimal join-tree of CG
 - Join-tree width is 1 (because it is a *tree*)
- Local plan length: $(w + 1)\delta_{\text{avg}} \leq (w + 1)\delta_{\text{max}}$
 - Exponent $(w + 1) \cdot \delta \implies 1 \cdot (w + 1)\delta_{\text{avg}}$
 - δ_{avg} is likely to be smaller than δ , **possibly much smaller!**
- Bottom line:
 - Join-tree provides a-priori best factorization.
 - If you have some domain knowledge, seek join-tree that minimizes local plans length

Example



Summary: Insights

Two crucial complexity parameters

- Domain parameter: causal graph's tree-width
 - for factored *and* non-factored planning
- Problem specific parameter:
 - Non-factored: length of (step-)optimal plan
 - Factored: minmax length of local plans

How to best factor domain:

- Use causal graph to create join-tree that minimizes minimax sum of value changes within sub-domains
- Comment: very similar objective in probabilistic reasoning

Contributions in a nutshell

- Complexity-wise best (so far?) **approach** for factored planning
- Tractable planning classes of **practical interest**
- Relation of **local optimality** to standard (to date) notions of optimality
- The star in our movie is the **causal graph**