Mechanising Upper Bounds in Planning: First Steps Towards a Verified Planner

Mohammad Abdulaziz[1]
NICTA and ANU

30 June 2014

NICTA Funding and Supporting Members and Partners

[1] Joint work with Michael Norrish and Charles Gretton

Outline

- Verified Planner
- Preliminaries
  - The Planning Problem
  - The Bound on Plan Length
- Theorems
- Verification
- Bounds and Decompositions

Verified Planner: Why?

- Planning systems are informally designed and implemented.
- Limits on critical applications
- We want to create a planner with
  - Correctness: soundness and completeness
  - Bounds memory and time consumption

Bounds: Why?

- A verified planner requires the proof of fundamental theorems about planning problems
- We verify upper bounds on lengths of plans
- If a problem is solvable, some plan satisfies that bound
- How do bounds help in a verified planner?
  - Give a guarantee on resources needed and efficiency
  - Provide correctness guarantees

Verification: HOL4

- We used HOL4, an interactive higher-order logic proof assistant
- Definitions, theorems and proofs are encoded in HOL
- Proofs are checked on top of a small trusted code base
  - Failed proofs may suggest counter-examples

Mechanising Bounds

In this work we report on

- Verifying bounds from Rintanen and Gretton's IJCAI 2013 paper (R&G)
- A mistake in their formalisation of bounds and its repair
- A new theorem that led to tighter bounds

The Planning Problem Definition

A planning problem (Π) can be defined as:

- Domain (D) : a set of Boolean variables representing the planning problem states.
- Actions (A) : a set of tuples (p, e)
- Initial state (I) : a map from the domain to Boolean
- Goal state (G) : a map from the domain to Boolean

A solution is a sequence of actions (π) whose members are in A.

An example problem: Logistics

NICTA

- 3 cities $\{C_1, C_2, C_3\}$ where objects can be located
- 1 truck $\{T\}$ that drives from between any pair of different cities
- 2 parcels $\{P_1, P_2\}$ that can be loaded or unloaded onto trucks

Problem Π

$$D = \left\{ \begin{array}{l} \\ T@C_1, T@C_2, T@C_3, \\ P_1@C_1, P_1@C_2, P_1@C_3, P_1@T, \\ P_2@C_3, , P_2@C_1, P_2@C_2, P_2@T \end{array} \right\}$$

An example problem: Logistics

- 3 cities $\{C_1, C_2, C_3\}$ where objects can be located
- 1 truck $\{T\}$ that drives from between any pair of different cities
- 2 parcels $\{P_1, P_2\}$ that can be loaded or unloaded onto trucks

Problem Π

I = {
$T@C_1, \overline{T@C_2}, \overline{T@C_3}, P_1@C_1, \overline{P_1@C_2}, \overline{P_1@C_3}, \overline{P_1@T}, P_2@C_3, , \overline{P_2@C_1}, \overline{P_2@C_2}, \overline{P_2@T}\}$

An example problem: Logistics

- 3 cities $\{C_1, C_2, C_3\}$ where objects can be located
- 1 truck $\{T\}$ that drives from between any pair of different cities
- 2 parcels $\{P_1, P_2\}$ that can be loaded or unloaded onto trucks

Problem Π

A =

$$\{Load(p, c) = (\{p@c, T@c\}, \{p@T, \overline{p@c}\}) \quad | p \in \{P_1, P_2\} \wedge c \in \{C_1, C_2, C_3\}\}$$

$$\cup$$

$$\{UnLoad(p, c) = (\{p@T, T@c\}, \{\overline{p@T}, p@c\}) | p \in \{P_1, P_2\} \wedge c \in \{C_1, C_2, C_3\}\}$$

$$\cup$$

$$\{Drive(c_i, c_j) = (\{T@c_i\}, \{T@c_j, \overline{T@c_i} | c_i \in \{C_1, C_2, C_3\} \wedge c_j \in \{C_1, C_2, C_3\} \wedge c_i \neq$$

An example problem: Logistics

- 3 cities $\{C_1, C_2, C_3\}$ where objects can be located
- 1 truck $\{T\}$ that drives from between any pair of different cities
- 2 parcels $\{P_1, P_2\}$ that can be loaded or unloaded onto trucks

Problem Π

G =
$\{\mathrm{T@C}_1, \overline{T@C_2}, \overline{T@C_3}, P_1@C_3, \overline{P_1@C_1}, \overline{P_1@C_2}, \overline{P_1@T}, P_2@C_1, \overline{P_2@C_2}, \overline{P_2@C_3}, \overline{P_2@T}\}$

An example problem: Logistics

- 3 cities $\{C_1, C_2, C_3\}$ where objects can be located
- 1 truck $\{T\}$ that drives from between any pair of different cities
- 2 parcels $\{P_1, P_2\}$ that can be loaded or unloaded onto trucks

Problem Π

Solution:
$[Load(P_1, C_1); DriveC_1, C_2); Drive(C_2, C_3), UnLoad(P_1, C_3); Load(P_2, C_3);$
Drive(C_3, C_2); $Drive(C_2, C_1); UnLoad(P_2, C_1)]$

A Bound on Plan Length

Definition

The bound is defined as:

$$\ell(\Pi) = \max_{s \in \mathcal{S}} \min_{\pi \in \Pi(s)} |\pi|$$

Our contribution builds on definitions of bounds by R&G

- A valid bound is the longest shortest execution between the initial state and any other state in $\mathcal{P}(D)$
- It is the diameter of the state transition graph, **but**, only from I.

Theorems

- One way to deduce theorems about $\ell(\Pi)$ is to appeal to state space cardinality arguments.
- So, the most basic theorem that can be stated about $\ell(\Pi)$ is:

Theorem

$$\ell(\Pi) < 2^{|D|}$$

Theorems on Bounds

- R&G suggested a hierarchical decomposition of Π to get tighter bounds

- Their decomposition yields subexponential bounds

- The details require we introduce the concepts of **dependency graph** and **projection**.

Dependency graph

NICTA

Definition

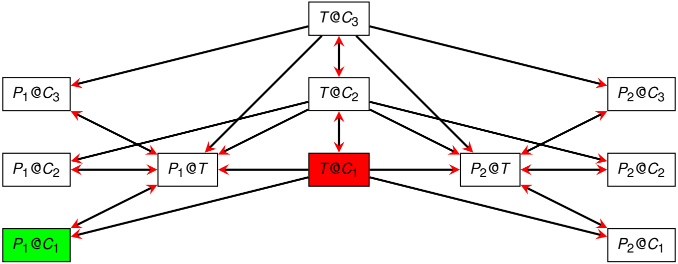
A dependency graph is a directed graph which:

- *Has a node for each variable in D*
- *Has an edge from v_1 to v_2 ($v_1 \rightarrow v_2$) iff*
  - $v_1 = v_2$ *; or*
  - *there is an action a in A such that v_1 is a precondition of a and v_2 is an effect of a; or*
  - *there is an action a in A such that both v_1 and v_2 are effects of a.*

Dependency graph: Example

[Reflexive arcs omitted]

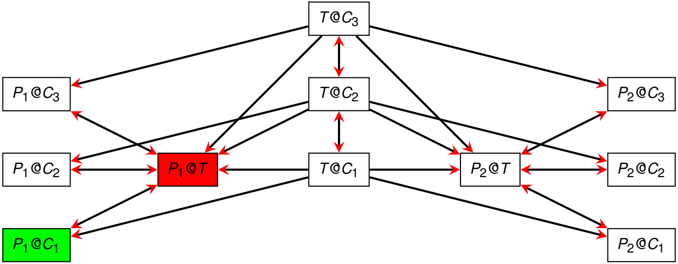
$Load(P_1, C_1) = (\{P_1@C_1, T@C_1\}, \{P_1@T, \overline{P_1@C_1}\})$



Dependency graph: Example

[Reflexive arcs omitted]

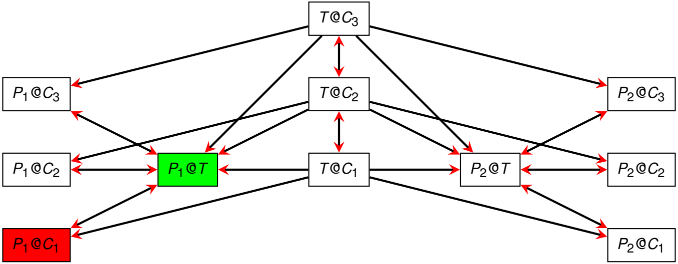
$Load(P_1, C_1) = (\{P_1@C_1, T@C_1\}, \{P_1@T, \overline{P_1@C_1}\})$



Dependency graph: Example

[Reflexive arcs omitted]

$$Load(P_1, C_1) = (\{P_1@C_1, T@C_1\}, \{P_1@T, \overline{P_1@C_1}\})$$

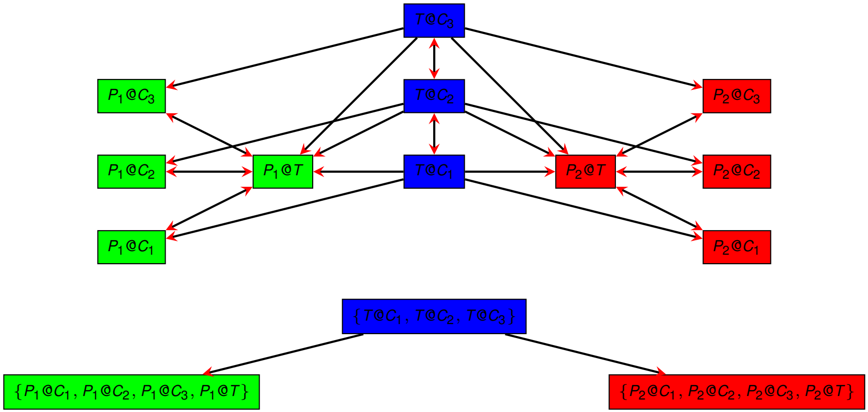


Lifted Dependency Graphs

Definition

*A **lifted dependency graph***

- *is a lifting (contraction) of the dependency graph;*
- *is a directed graph such that, for a partition P of D:*
  - *there is a node for each $s \in P$*
  - *An edge from node s_1 to node s_2 ($s_1 \rightarrow s_2$) iff*
    - *s_1 is disjoint from s_2; and*
    - *$\exists v_1 \in s_1,\ v_2 \in s_2$ such that $v_1 \rightarrow v_2$.*


$T@C_3$
$P_1@C_3$
$T@C_2$
$P_2@C_3$
$P_1@C_2$
$P_1@T$
$T@C_1$
$P_2@T$
$P_2@C_2$
$P_1@C_1$
$P_2@C_1$
$\{T@C_1, T@C_2, T@C_3\}$
$\{P_1@C_1, P_1@C_2, P_1@C_3, P_1@T\}$
$\{P_2@C_1, P_2@C_2, P_2@C_3, P_2@T\}$

Projection

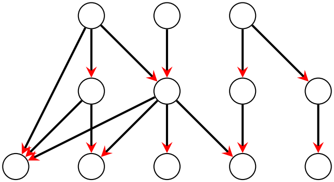
- **Projection** "limits" an x to a specific set of variables vs, written $x\downharpoonright_{vs}$.
- Can be applied to:
  - An action ($a\downharpoonright_{vs}$)
  - An action sequence ($\pi\downharpoonright_{vs}$)
  - A state ($s\downharpoonright_{vs}$)
  - A problem ($\Pi\downharpoonright_{vs}$)

Leaf Ancestor Theorem

- Used when there is a branching structure in the dependency graph.
- Can obtain subexponential bounds on plan lengths.

Theorem

$$\forall\ \Pi\ G_{vs}.\ \ DAG(G_{vs})\ \Rightarrow\ \ell(\Pi) \leq \Sigma_{vs\ \in leaves(G_{vs})}\ell(\Pi\downharpoonright_{vs\ \cup ancestors(vs)})$$

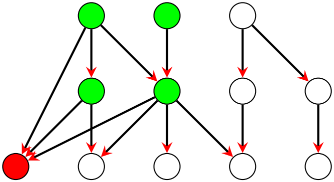


Leaf Ancestor Theorem

- Used when there is a branching structure in the dependency graph.
- Can obtain subexponential bounds on plan lengths.

Theorem

$$\forall\, \Pi\, G_{vs}.\ \ DAG(G_{vs}) \ \Rightarrow\ \ell(\Pi) \leq \Sigma_{vs\ \in leaves(G_{vs})} \ell(\Pi\downharpoonright_{vs\ \cup ancestors(vs)})$$

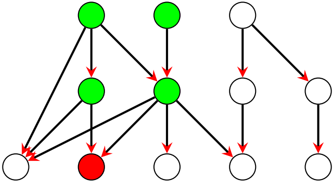


Leaf Ancestor Theorem

- Used when there is a branching structure in the dependency graph.
- Can obtain subexponential bounds on plan lengths.

Theorem

$$\forall\ \Pi\ G_{vs}.\ \ DAG(G_{vs})\ \Rightarrow\ \ell(\Pi) \leq \Sigma_{vs\ \in leaves(G_{vs})}\ell(\Pi\downharpoonright_{vs\ \cup ancestors(vs)})$$

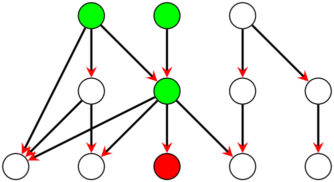


Leaf Ancestor Theorem

- Used when there is a branching structure in the dependency graph.
- Can obtain subexponential bounds on plan lengths.

Theorem

$$\forall\ \Pi\ G_{vs}.\ \ DAG(G_{vs})\ \Rightarrow\ \ell(\Pi) \leq \Sigma_{vs\ \in leaves(G_{vs})}\ell(\Pi\downharpoonright_{vs\ \cup ancestors(vs)})$$

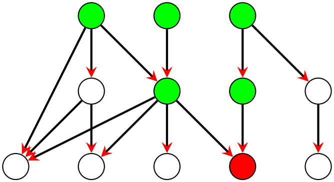


Leaf Ancestor Theorem

- Used when there is a branching structure in the dependency graph.
- Can obtain subexponential bounds on plan lengths.

Theorem

$$\forall\ \Pi\ G_{vs}.\ \ DAG(G_{vs})\ \Rightarrow\ \ell(\Pi) \leq \Sigma_{vs\ \in leaves(G_{vs})}\ell(\Pi\downharpoonright_{vs\ \cup ancestors(vs)})$$

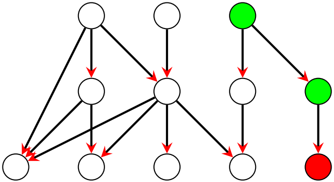


Leaf Ancestor Theorem

- Used when there is a branching structure in the dependency graph.
- Can obtain subexponential bounds on plan lengths.

Theorem

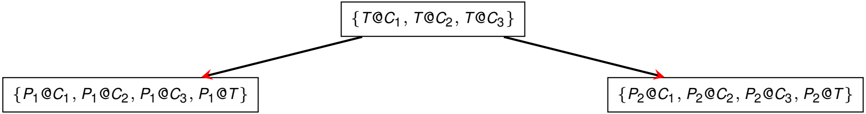
$$\forall\, \Pi\; G_{vs}.\;\; DAG(G_{vs}) \;\Rightarrow\; \ell(\Pi) \leq \Sigma_{vs\, \in leaves(G_{vs})} \ell(\Pi\downharpoonright_{vs\, \cup ancestors(vs)})$$



Leaf Ancestor Theorem: Example

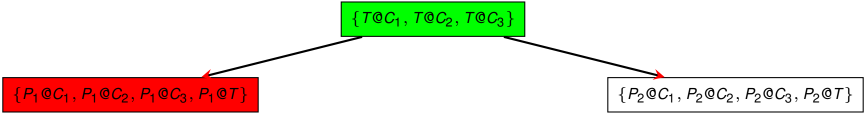
NICTA

- Using cardinality arguments in Logistics, the bound $(2^{7+1} - 2)$ is obtained, which is much tighter than $(2^{11} - 1)$.
- Appealing further to problem constraints—**e.g.**, a parcel cannot be in two locations at once—yields even tighter bounds.



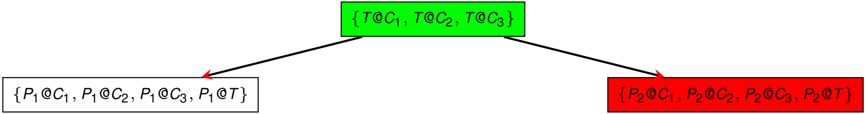
Leaf Ancestor Theorem: Example

- Using cardinality arguments in Logistics, the bound $(2^{7+1} - 2)$ is obtained, which is much tighter than $(2^{11} - 1)$.
- Appealing further to problem constraints—**e.g.**, a parcel cannot be in two locations at once—yields even tighter bounds.

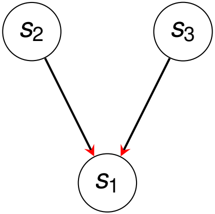


Leaf Ancestor Theorem: Example

- Using cardinality arguments in Logistics, the bound $(2^{7+1} - 2)$ is obtained, which is much tighter than $(2^{11} - 1)$.
- Appealing further to problem constraints—**e.g.**, a parcel cannot be in two locations at once—yields even tighter bounds.



Some problems are not decomposable by the leaf ancestor theorem

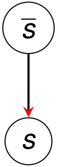


- The leaf ancestor theorem only leads to:

$$\ell(\Pi) < 2^{|s_1|+|s_2|+|s_3|}$$

Child Parent Theorem

[Note: $\overline{s} = D \setminus s$]



Theorem

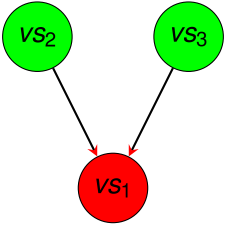
$$\forall\, \Pi\, s.\ \ s \not\rightarrow \overline{s} \ \Rightarrow\ \ell(\Pi) < (\ell(\Pi\downharpoonright_{\overline{s}}) + 1)(\ell(\Pi\downharpoonright_{s}) + 1)$$

- This theorem is for when there is a “child parent relation” in the lifted graph
- There are two actions sets:
  1. child actions with *dom e* $\subseteq s$; and
  2. parent actions with *dom p* $\subseteq \overline{s} \wedge$ *dom e* $\subseteq \overline{vs}$

Child Parent Theorem

NICTA

Some cases are not decomposable by the leaf ancestor theorem

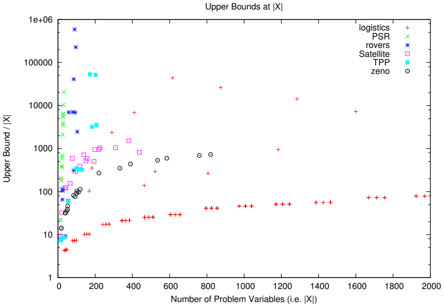


- The leaf ancestor and child parent theorems lead to:

$$\ell(\Pi) \leq (2^{|vs_1|} - 1)(2^{|vs_2|} + 2^{|vs_3|} - 2) + 2^{|vs_1|} + 2^{|vs_2|} + 2^{|vs_4|} - 3$$

Experiments

- R&G's approach can give tight bounds in domains like LOGISTICS, SATELLITE, and ZENO
- Solves previously open instances
  - Closed open instance of ROVERS with Qualitative Preferences from IPC 2006



Verification: Basic Theorem

Theorem

$$\ell(\Pi) < 2^{|D|}$$

- Verifying the first theorem is relatively straight-forward:
  - Prove that number of states traversed by any cycle—a repetition of the same state—free plan is at most $2^{|D|}$
    - Then employ the pigeonhole principle
  - Prove that for any plan, if there is a cycle, its removal gives an admissible plan

Verification: Child Parent Theorem

- BUG!

  ~~$\ell(\Pi) = \max_{s \in S} \min_{\pi \in \Pi(s)} |\pi|$~~

- Problems:
  - Unreachable states
  - Unrefinable action sequences

Problem

$$\Pi = \left[\begin{array}{l} I = \{\overline{w}, \overline{x}, \overline{y}, \overline{z}\} \\ A = \left\{ \begin{array}{l} a = (\emptyset, \{x\}), b = (\{x\}, \{\overline{x}, y\}), \\ c = (\{x, y\}, \{\overline{x}, \overline{y}, z\}), d = (\{w\}, \{x, y, z\}) \end{array} \right\} \\ G = \{\overline{w}, x, y, z\} \end{array} \right]$$

Verification: Child Parent Theorem

NICTA

- BUG!

~~$\ell(\Pi) = \max_{s \in S} \min_{\pi \in \Pi(s)} |\pi|$~~ $\ell(\Pi) = \max_{s \in S, \pi \in \dot{A}} \min_{\pi' \in \Pi^{\preceq}(\pi, s)} |\pi|$

- Problems:
  - Unreachable states
  - Unrefinable action sequences

Problem

$$\Pi = \left[\begin{array}{l} I = \{\overline{w}, \overline{x}, \overline{y}, \overline{z}\} \\ A = \left\{ \begin{array}{l} a = (\emptyset, \{x\}), b = (\{x\}, \{\overline{x}, y\}), \\ c = (\{x, y\}, \{\overline{x}, \overline{y}, z\}), d = (\{w\}, \{x, y, z\}) \end{array} \right\} \\ G = \{\overline{w}, x, y, z\} \end{array}\right]$$

Verification: Child Parent Theorem: PROOF

NICTA

Our proof is constructive

- Given an existing plan

  ■; ■...; ■; ■; ■; ■...; ■; ■; ■; ■...; ■; ■; .....; ■; ■...; ■; ■; ■; ■...; ■

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan
  ■; ■...; ■; ■; ■; ■...; ■; ■; ■; ■...; ■; ■; .....; ■; ■...; ■; ■; ■; ■...; ■
- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

  ■; ■...; ■; ■; ■; ■...; ■; ■; ■; ■...; ■; ■; .....; ■; ■...; ■; ■; ■; ■...; ■

- Derive a new child plan conforming to $\ell(\Pi\downharpoonleft_s)$

  ■; ■; ■; ■;

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

- Replace the child part

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

- Replace the child part

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

- Replace the child part

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

- Replace the child part

- For each parent plan fragment derive a fragment conforming to $\ell(\Pi\downharpoonright_{\overline{s}})$

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_{s})$

- Replace the child part

- For each parent plan fragment derive a fragment conforming to $\ell(\Pi\downharpoonright_{\overline{s}})$

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

- Replace the child part

- For each parent plan fragment derive a fragment conforming to $\ell(\Pi\downharpoonright_{\overline{s}})$

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

- Replace the child part

- For each parent plan fragment derive a fragment conforming to $\ell(\Pi\downharpoonright_{\overline{s}})$

- Put the new parent fragments in the plan

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

- Replace the child part

- For each parent plan fragment derive a fragment conforming to $\ell(\Pi\downharpoonright_{\overline{s}})$

- Put the new parent fragments in the plan

Verification: Child Parent Theorem: PROOF

Our proof is constructive

- Given an existing plan

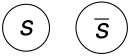
- Derive a new child plan conforming to $\ell(\Pi\downharpoonright_s)$

- Replace the child part

- For each parent plan fragment derive a fragment conforming to $\ell(\Pi\downharpoonright_{\overline{s}})$

- Put the new parent fragments in the plan

Verification: Disconnected Variable Sets Theorem



Theorem

$$\forall\, \Pi\, s.\;\; \overline{s} \not\rightarrow s \wedge s \not\rightarrow \overline{s} \Rightarrow \ell(\Pi) < \ell(\Pi\downharpoonright_{\overline{s}}) + \ell(\Pi\downharpoonright_{s}) + 1$$

- Our proof technique can be applied to this theorem.
- Repeat the first step in theorem 3 proof twice, for s actions, and for $\overline{s}$ actions
- It implies that actions are either
  1. s actions with $dom\ p \subseteq s \wedge dom\ e \subseteq s$; or
  2. $\overline{s}$ actions with $dom\ p \subseteq \overline{s} \wedge dom\ e \subseteq \overline{s}$

Verification: Disconnected Set Theorem: PROOF

NICTA

Our proof was constructive as well

- Given an existing plan

■; ■...; ■; ■; ■; ■...; ■; ■; ■; ■...; ■; ■; .....; ■; ■...; ■; ■; ■; ■...; ■

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

  ■; ■...; ■; ■; ■; ■...; ■; ■; ■; ■...; ■; ■; .....; ■; ■...; ■; ■; ■; ■...; ■
- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

  ■; ■...; ■; ■; ■; ■...; ■; ■; ■; ■...; ■; ■; .....; ■; ■...; ■; ■; ■; ■...; ■

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

  ■; ■; ■...; ■; ■; .....; ■; ■...; ■;

Verification: Disconnected Set Theorem: PROOF

NICTA

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

- Replace the *vs* actions

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

- Replace the *vs* actions

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

- Replace the *vs* actions

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonleft_{vs})$

- Replace the *vs* actions

- Take all $\overline{vs}$ actions and shorten them to conform to $\ell(\Pi\downharpoonleft_{\overline{vs}})$

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

- Replace the *vs* actions

- Take all $\overline{vs}$ actions and shorten them to conform to $\ell(\Pi\downharpoonright_{\overline{vs}})$

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

- Replace the *vs* actions

- Take all $\overline{vs}$ actions and shorten them to conform to $\ell(\Pi\downharpoonright_{\overline{vs}})$

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

- Replace the *vs* actions

- Take all $\overline{vs}$ actions and shorten them to conform to $\ell(\Pi\downharpoonright_{\overline{vs}})$

- Replace $\overline{vs}$ actions in the plan

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonright_{vs})$

- Replace the *vs* actions

- Take all $\overline{vs}$ actions and shorten them to conform to $\ell(\Pi\downharpoonright_{\overline{vs}})$

- Replace $\overline{vs}$ actions in the plan

Verification: Disconnected Set Theorem: PROOF

Our proof was constructive as well

- Given an existing plan

- Take all *vs* actions and shorten them to conform to $\ell(\Pi\downharpoonleft_{vs})$

- Replace the *vs* actions

- Take all $\overline{vs}$ actions and shorten them to conform to $\ell(\Pi\downharpoonleft_{\overline{vs}})$

- Replace $\overline{vs}$ actions in the plan

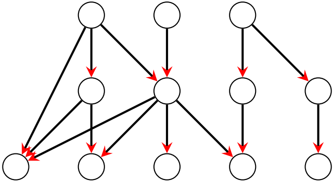
Current Verification: Parent Children Theorem

Theorem

$$\forall\, \Pi\ G_s . DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\ \in G_s} N(s)$$

where,
$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

NICTA

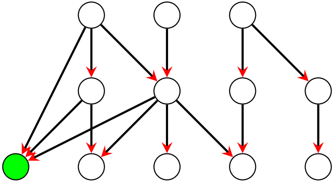
Theorem

$$\forall \Pi\ G_s. DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s \in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

NICTA

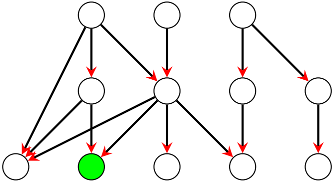
Theorem

$$\forall\, \Pi\; G_s . DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\, \in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

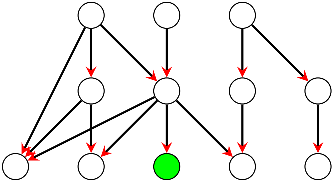
Theorem

$$\forall\, \Pi\; G_s.DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\,\in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

NICTA

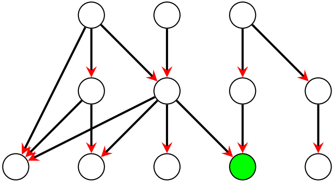
Theorem

$$\forall\, \Pi\ G_s . DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\, \in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

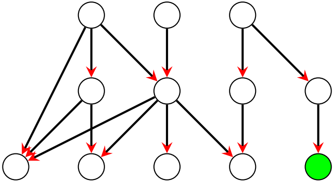
Theorem

$$\forall\, \Pi\; G_s . DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\, \in G_s} N(s)$$

where,

$N(s) = \ell(\Pi|_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

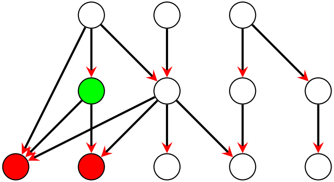
Theorem

$$\forall\, \Pi\ G_s.DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\ \in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

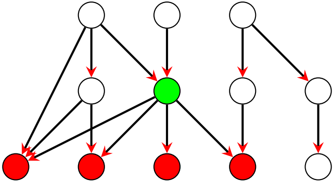
Theorem

$$\forall\, \Pi\ G_s.DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\,\in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



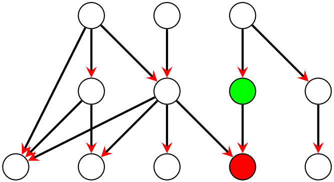
Current Verification: Parent Children Theorem

Theorem

$$\forall\ \Pi\ G_s.DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\ \in G_s} N(s)$$

where,
$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

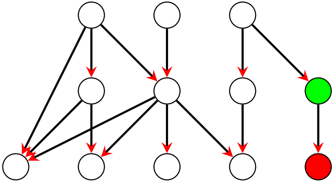
NICTA

Theorem

$$\forall\, \Pi\ G_s.DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\, \in G_s} N(s)$$

where,
$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

NICTA

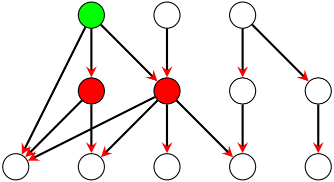
Theorem

$$\forall\, \Pi\ G_s.DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\ \in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

NICTA

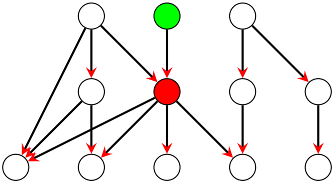
Theorem

$$\forall\, \Pi\ G_s.DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\ \in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Current Verification: Parent Children Theorem

NICTA

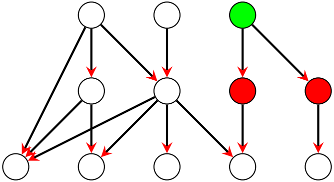
Theorem

$$\forall\, \Pi\ G_s.DAG(G_s) \Rightarrow \ell(\Pi) \leq \Sigma_{s\ \in G_s} N(s)$$

where,

$N(s) = \ell(\Pi\downharpoonright_s)(\Sigma_{s' \in children(s)} N(s') + 1)$

- Verification of this tighter bound is underway



Verification: Parent Children Theorem: PROOF

For a variable set vs_i in the graph

- Given an existing plan

Verification: Parent Children Theorem: PROOF

For a variable set vs_i in the graph

- Given an existing plan

- For each vs_i action fragment, derive a fragment conforming to $\ell(\Pi\downharpoonright_{vs_i})$

Verification: Parent Children Theorem: PROOF

For a variable set vs_i in the graph

- Given an existing plan

- For each vs_i action fragment, derive a fragment conforming to $\ell(\Pi\downharpoonright_{vs_i})$

Verification: Parent Children Theorem: PROOF

For a variable set vs_i in the graph

- Given an existing plan

- For each vs_i action fragment, derive a fragment conforming to $\ell(\Pi\downharpoonright_{vs_i})$

Verification: Parent Children Theorem: PROOF

For a variable set vs_i in the graph

- Given an existing plan

- For each vs_i action fragment, derive a fragment conforming to $\ell(\Pi\downharpoonright_{vs_i})$

- Put the new vs_i actions fragments in the plan

Verification: Parent Children Theorem: PROOF

For a variable set vs_i in the graph

- Given an existing plan

- For each vs_i action fragment, derive a fragment conforming to $\ell(\Pi\downharpoonright_{vs_i})$

- Put the new vs_i actions fragments in the plan

Verification: Parent Children Theorem: PROOF

For a variable set vs_i in the graph

- Given an existing plan

- For each vs_i action fragment, derive a fragment conforming to $\ell(\Pi\downharpoonright_{vs_i})$

- Put the new vs_i actions fragments in the plan

Verification: Parent Children Theorem: PROOF

For a variable set vs_i in the graph

- Given an existing plan

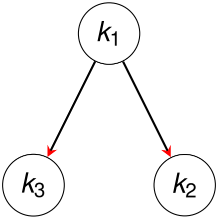
- For each vs_i action fragment, derive a fragment conforming to $\ell(\Pi\downharpoonright_{vs_i})$

- Put the new vs_i actions fragments in the plan

This is repeated recursively on the DAG in reverse topological order

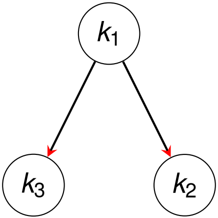
Decompositions: Case 1

- There are multiple ways to decompose a planning problem to get bounds



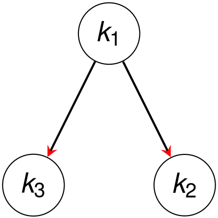
Decompositions: Case 1

- There are multiple ways to decompose a planning problem to get bounds
- With **leaf ancestor** theorem (the algorithm in R&G) $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + 2k_1 + k_3$



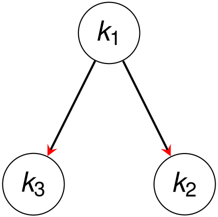
Decompositions: Case 1

- There are multiple ways to decompose a planning problem to get bounds
- With **leaf ancestor** theorem (the algorithm in R&G) $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + 2k_1 + k_3$
- With **child parent** theorem and the **disconnected sets** theorem $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + k_1 + k_3$



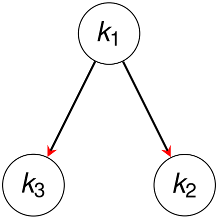
Decompositions: Case 1

- There are multiple ways to decompose a planning problem to get bounds
- With **leaf ancestor** theorem (the algorithm in R&G) $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + 2k_1 + k_3$
- With **child parent** theorem and the **disconnected sets** theorem $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + k_1 + k_3$
- With **parent children** theorem $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + k_1 + k_3$



Decompositions: Case 1

- There are multiple ways to decompose a planning problem to get bounds
- With **leaf ancestor** theorem (the algorithm in R&G) $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + 2k_1 + k_3$
- With **child parent** theorem and the **disconnected sets** theorem $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + k_1 + k_3$
- With **parent children** theorem $\ell(\Pi) \leq k_1 k_2 + k_2 + k_1 k_3 + k_1 + k_3$

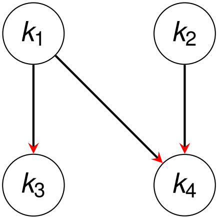


Decompositions: Case 2

- There are multiple ways to decompose a planning problem to get bounds

Decompositions: Case 2

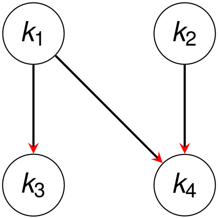
- There are multiple ways to decompose a planning problem to get bounds
- With leaf ancestor theorem (the algorithm in R&G) $\ell(\Pi) \leq k_1 k_3 + k_1 k_4 + k_2 k_4 + 2k_1 + k_2 + k_3 + k_4$



Decompositions: Case 2

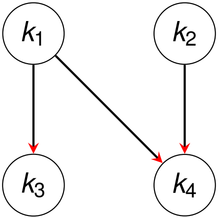
NICTA

- There are multiple ways to decompose a planning problem to get bounds
- With leaf ancestor theorem (the algorithm in R&G) $\ell(\Pi) \leq k_1k_3 + k_1k_4 + k_2k_4 + 2k_1 + k_2 + k_3 + k_4$
- With child parent theorem and the disconnected sets theorem $\ell(\Pi) \leq k_1k_3 + k_2k_3 + k_1k_4 + k_2k_4 + k_1 + k_2 + k_3 + k_4$



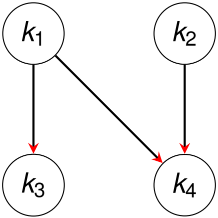
Decompositions: Case 2

- There are multiple ways to decompose a planning problem to get bounds
- With leaf ancestor theorem (the algorithm in R&G)
  $\ell(\Pi) \leq k_1 k_3 + k_1 k_4 + k_2 k_4 + 2k_1 + k_2 + k_3 + k_4$
- With child parent theorem and the disconnected sets theorem
  $\ell(\Pi) \leq k_1 k_3 + k_2 k_3 + k_1 k_4 + k_2 k_4 + k_1 + k_2 + k_3 + k_4$
- With parent children theorem
  $\ell(\Pi) \leq k_1 k_3 + k_1 k_4 + k_2 k_4 + k_1 + k_2 + k_3 + k_4$



Decompositions: Case 2

- There are multiple ways to decompose a planning problem to get bounds
- With leaf ancestor theorem (the algorithm in R&G)
  $\ell(\Pi) \leq k_1k_3 + k_1k_4 + k_2k_4 + 2k_1 + k_2 + k_3 + k_4$
- With child parent theorem and the disconnected sets theorem
  $\ell(\Pi) \leq k_1k_3 + k_2k_3 + k_1k_4 + k_2k_4 + k_1 + k_2 + k_3 + k_4$
- With parent children theorem
  $\ell(\Pi) \leq k_1k_3 + k_1k_4 + k_2k_4 + k_1 + k_2 + k_3 + k_4$



Future Work

- Tighter verified bounds
- Correctness of a planning system
  - Planning algorithms
    - **e.g.**, prove correctness of a SAT encoding
    - **e.g.**, prove correctness of a state space exploration scheme
  - Planning implementations
    - of SAT solvers,
    - of algorithms above

Conclusion

- We verified bounds from Rintanen and Gretton 2013 (R&G)
- Found and fixed a mistake in their formalisation
- Proved a new theorem that leading to novel tighter bounds

- The world's first verified planner awaits!