University of Basel

Automated Planning using Property-Directed Reachability with Seed Heuristics

Tim Bachmann <tim.bachmann@stud.unibas.ch>

Department Mathematics and Computer Science, University of Basel

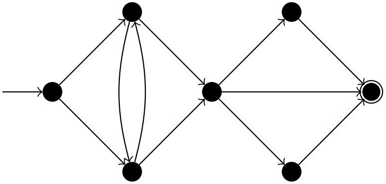
May 22, 2023

Table of Content

Planning

Planning Task $\langle \mathcal{V}, \mathcal{O}, s_I, s_* \rangle$

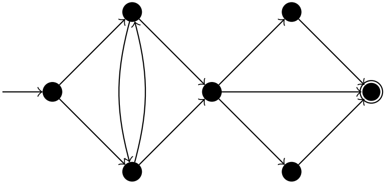
- Finite set of **variables** $\mathcal{V} = \{A, B, C, \ldots\}$.
- A **state** is a full assignment of variables to truth values.
- Finite set of **operators** $\mathcal{O}$ for transitions between states.
- **Initial state** s_I.
- **Goal formula** s_*.



Planning

Planning Task $\langle \mathcal{V}, \mathcal{O}, s_I, s_* \rangle$

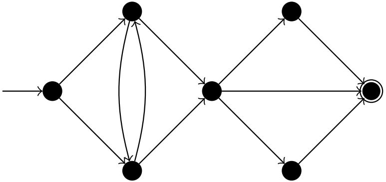
- Finite set of **variables** $\mathcal{V} = \{A, B, C, \dots\}$.
- A **state** is a full assignment of variables to truth values.
- Finite set of **operators** $\mathcal{O}$ for transitions between states.
- **Initial state** s_I.
- **Goal formula** s_*.



Planning

Planning Task $\langle \mathcal{V}, \mathcal{O}, s_I, s_* \rangle$

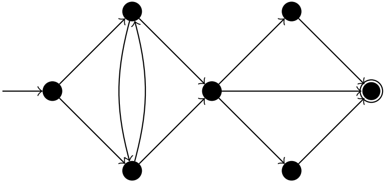
- Finite set of **variables** $\mathcal{V} = \{A, B, C, \dots\}$.
- A **state** is a full assignment of variables to truth values.
- Finite set of **operators** $\mathcal{O}$ for transitions between states.
- **Initial state** s_I.
- **Goal formula** s_*.



Planning

Planning Task $\langle \mathcal{V}, \mathcal{O}, s_I, s_* \rangle$

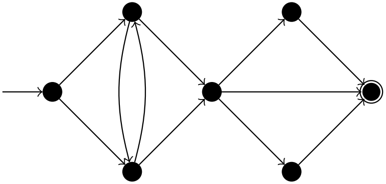
- Finite set of **variables** $\mathcal{V} = \{A, B, C, \dots\}$.
- A **state** is a full assignment of variables to truth values.
- Finite set of **operators** $\mathcal{O}$ for transitions between states.
- **Initial state** s_I.
- **Goal formula** s_*.



Planning

Planning Task $\langle \mathcal{V}, \mathcal{O}, s_I, s_* \rangle$

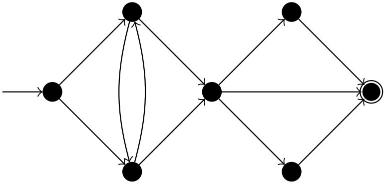
- Finite set of **variables** $\mathcal{V} = \{A, B, C, \dots\}$.
- A **state** is a full assignment of variables to truth values.
- Finite set of **operators** $\mathcal{O}$ for transitions between states.
- **Initial state** s_I.
- **Goal formula** s_*.



Planning

Planning Task $\langle \mathcal{V}, \mathcal{O}, s_I, s_* \rangle$

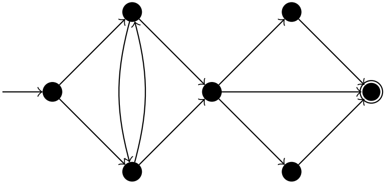
- Finite set of **variables** $\mathcal{V} = \{A, B, C, \dots\}$.
- A **state** is a full assignment of variables to truth values.
- Finite set of **operators** $\mathcal{O}$ for transitions between states.
- **Initial state** s_I.
- **Goal formula** s_*.



Planning

Planning Task $\langle \mathcal{V}, \mathcal{O}, s_I, s_* \rangle$

- Finite set of **variables** $\mathcal{V} = \{A, B, C, \dots\}$.
- A **state** is a full assignment of variables to truth values.
- Finite set of **operators** $\mathcal{O}$ for transitions between states.
- **Initial state** s_I.
- **Goal formula** s_*.



Notation

- Variables: A, B, C
- States: $p, q, v, \ldots$
- States as a formula: $p = A \wedge \neg B \wedge C$
- Inverse: $\neg p = \neg A \vee B \vee \neg C$

Notation

- Variables: A, B, C
- States: $p, q, v, \ldots$
- States as a formula: $p = A \wedge \neg B \wedge C$
- Inverse: $\neg p = \neg A \vee B \vee \neg C$

Notation

- Variables: A, B, C
- States: $p, q, v, \ldots$
- States as a formula: $p = A \wedge \neg B \wedge C$
- Inverse: $\neg p = \neg A \vee B \vee \neg C$

Notation

- Variables: A, B, C
- States: $p, q, v, \ldots$
- States as a formula: $p = A \wedge \neg B \wedge C$
- Inverse: $\neg p = \neg A \vee B \vee \neg C$

Notation

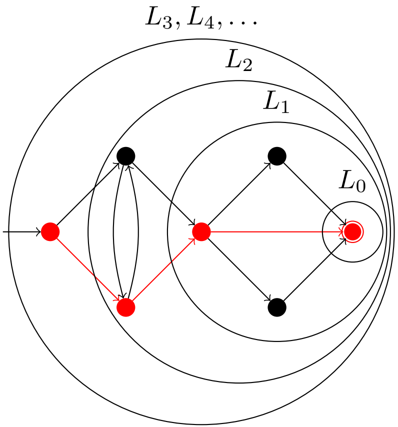
- Variables: A, B, C
- States: $p, q, v, \ldots$
- States as a formula: $p = A \land \neg B \land C$
- Inverse: $\neg p = \neg A \lor B \lor \neg C$

Overview

- **Property-Directed Reachability (PDR)**
  - Planning Algorithm
  - Based on a series of nested “Layers”
- **Goal**
  - Adding a pre-computation step
  - Improving the performance of the Algorithm

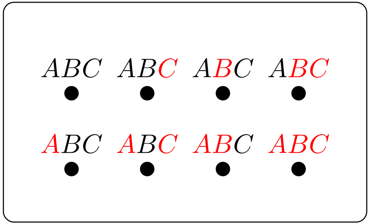
What is Property-Directed Reachability

- Planning algorithm
- Based on layers
- Iterative strengthening of the layers



Layers

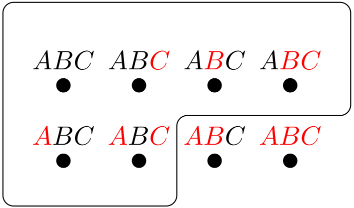
- CNF Formula
- Nested
  - $L_i \subseteq L_{i-1}$
  - $S_i \supseteq S_{i-1}$
- Higher layer index i
  - → fewer clauses
  - → more states



- $L_3 = \top$
- $L_2 = \top \wedge (A \vee B)$
- $L_1 = \top \wedge (A \vee B) \wedge (A)$
- $L_0 = \top \wedge (A \vee B) \wedge (A) \wedge (B \vee C)$

Layers

- CNF Formula
- Nested
  - $L_i \subseteq L_{i-1}$
  - $S_i \supseteq S_{i-1}$
- Higher layer index i
  - → fewer clauses
  - → more states



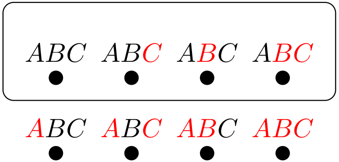
- $L_3 = \top$
- $L_2 = \top \wedge (A \vee B)$
- $L_1 = \top \wedge (A \vee B) \wedge (A)$
- $L_0 = \top \wedge (A \vee B) \wedge (A) \wedge (B \vee C)$

Layers

- CNF Formula
- Nested
  - $L_i \subseteq L_{i-1}$
  - $S_i \supseteq S_{i-1}$
- Higher layer index i

  → fewer clauses

  → more states



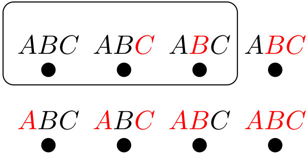
- $L_3 = \top$
- $L_2 = \top \wedge (A \vee B)$
- $L_1 = \top \wedge (A \vee B) \wedge (A)$
- $L_0 = \top \wedge (A \vee B) \wedge (A) \wedge (B \vee C)$

Layers

- CNF Formula
- Nested
  - $L_i \subseteq L_{i-1}$
  - $S_i \supseteq S_{i-1}$
- Higher layer index i

  → fewer clauses

  → more states



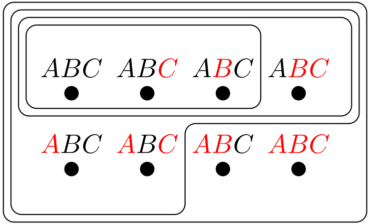
- $L_3 = \top$
- $L_2 = \top \wedge (A \vee B)$
- $L_1 = \top \wedge (A \vee B) \wedge (A)$
- $L_0 = \top \wedge (A \vee B) \wedge (A) \wedge (B \vee C)$

Layers

- CNF Formula
- Nested
  - $L_i \subseteq L_{i-1}$
  - $S_i \supseteq S_{i-1}$
- Higher layer index i

  → fewer clauses

  → more states



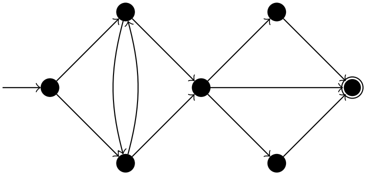
- $L_3 = \top$
- $L_2 = \top \wedge (A \vee B)$
- $L_1 = \top \wedge (A \vee B) \wedge (A)$
- $L_0 = \top \wedge (A \vee B) \wedge (A) \wedge (B \vee C)$

Algorithm Structure

```
Initialization
for $k = 0, 1, 2, \ldots$ do
    Path Construction Phase
    Clause Propagation Phase
```

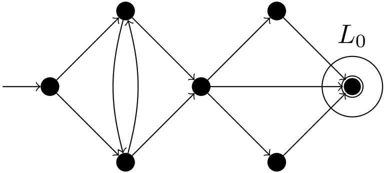
Initialization

- Initialize layer L_0 with the goal formula.
- Initialize layers L_i with $i > 0$ with the formula $\top$.



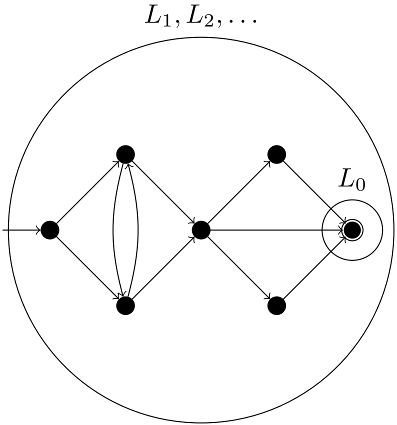
Initialization

- Initialize layer L_0 with the goal formula.
- Initialize layers L_i with $i > 0$ with the formula $\top$.



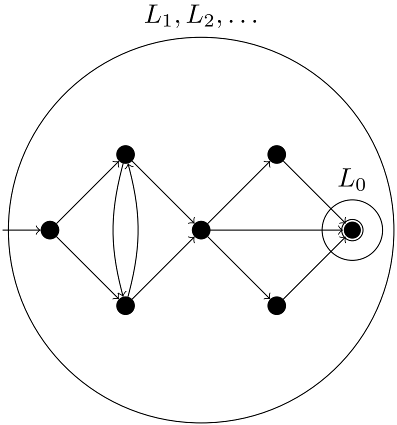
Initialization

- Initialize layer L_0 with the goal formula.
- Initialize layers L_i with $i > 0$ with the formula $\top$.



Path Construction Phase

Goal
Path from the initial state to a goal state.

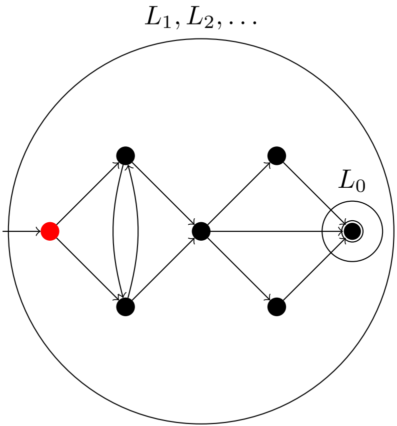


Iteration step $k = 1$

Path Construction Phase

Check if the initial state is in layer L_k.

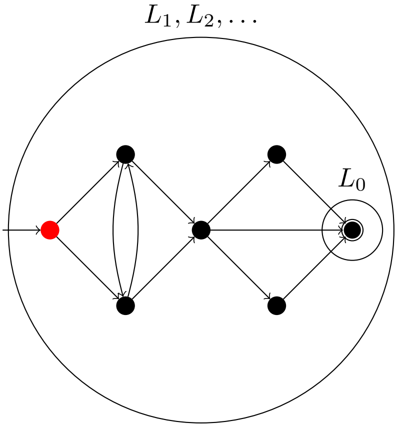
$\Rightarrow$ **true**



Iteration step $k = 1$

Path Construction Phase

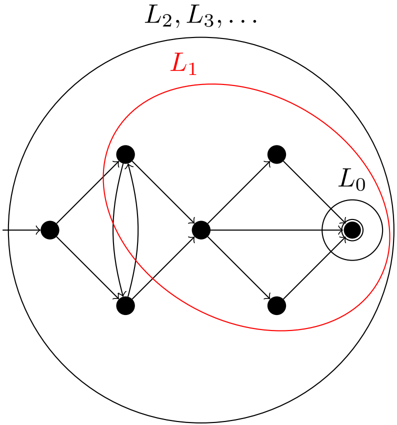
Find a successor state in L_0.
$\Rightarrow$ there is none.



Iteration step $k = 1$

Path Construction Phase

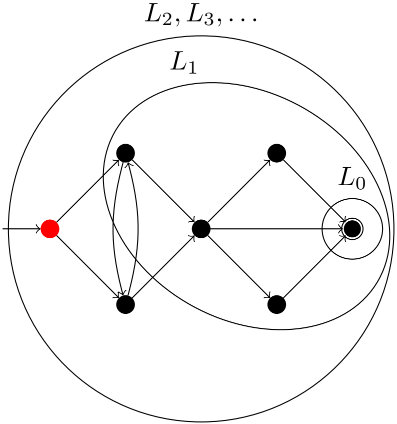
Strengthen the layer L_1 by adding a clause.



Iteration step $k = 1$

Path Construction Phase

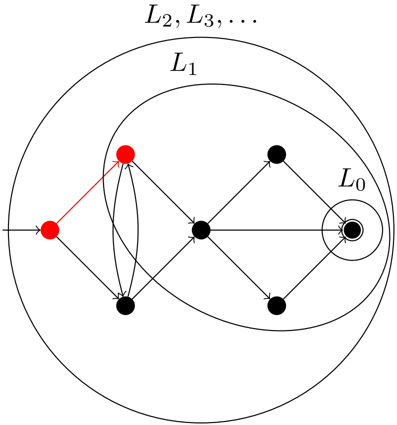
Next iteration.
Check if the initial state is in layer L_k.
$\Rightarrow$ **true**



Iteration step $k = 2$

Path Construction Phase

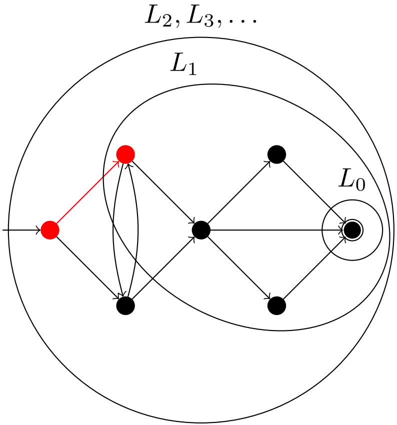
Find a successor state in L_1.
⇒ found one.



Iteration step $k = 2$

Path Construction Phase

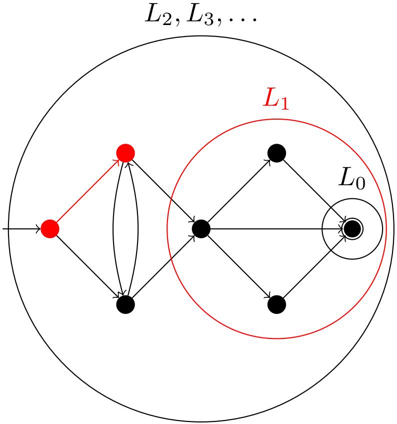
Find a successor state in L_0.
$\Rightarrow$ there is none.



Iteration step $k = 2$

Path Construction Phase

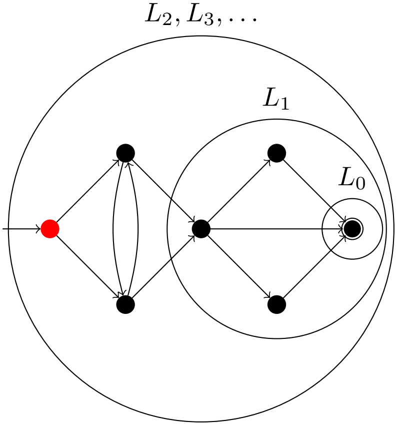
Strengthen the layer L_1 by adding a clause.



Iteration step $k = 2$

Path Construction Phase

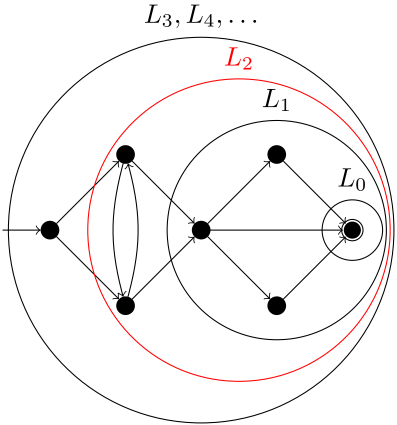
Find a successor state in L_1.
$\Rightarrow$ there is none.



Iteration step $k = 2$

Path Construction Phase

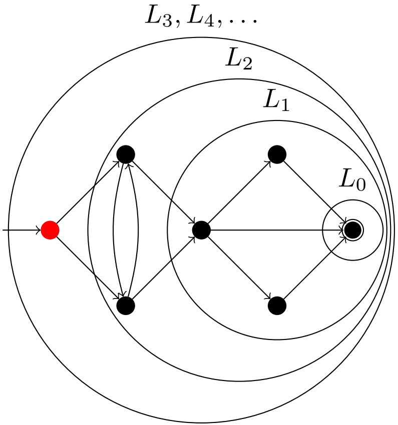
Strengthen the layer L_2 by adding a clause.



Iteration step $k = 2$

Path Construction Phase

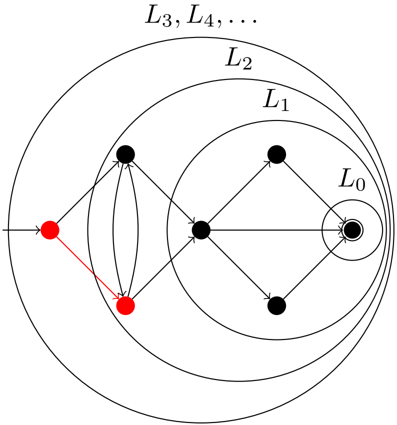
Next iteration.
Check if the initial state is in layer L_k.
$\Rightarrow$ **true**



Iteration step $k = 3$

Path Construction Phase

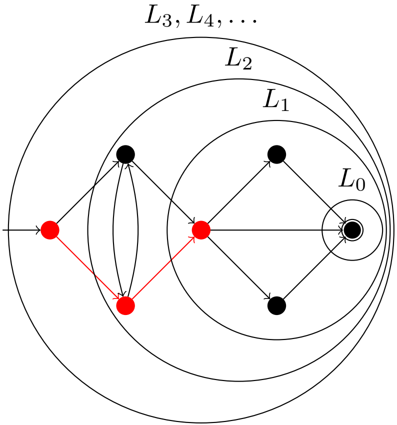
Find a successor state in L_2.
$\Rightarrow$ found one.



Iteration step $k = 3$

Path Construction Phase

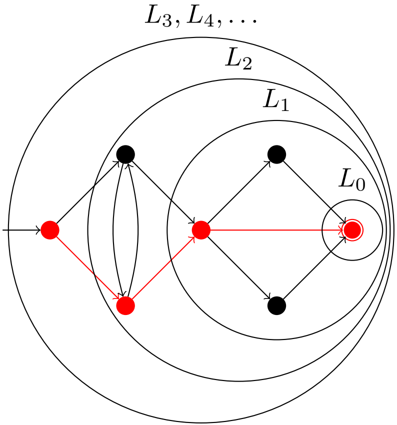
Find a successor state in L_1.
$\Rightarrow$ found one.



Iteration step $k = 3$

Path Construction Phase

Find a successor state in L_0.
⇒ found one.
⇒ path found.



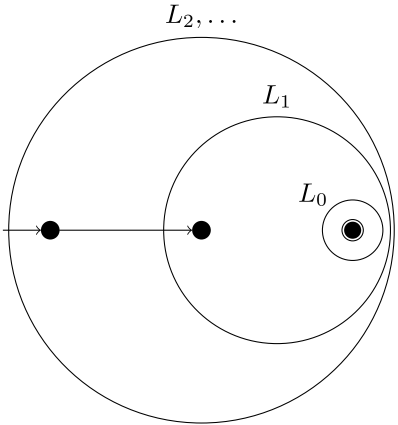
Iteration step $k = 3$

Clause Propagation Phase

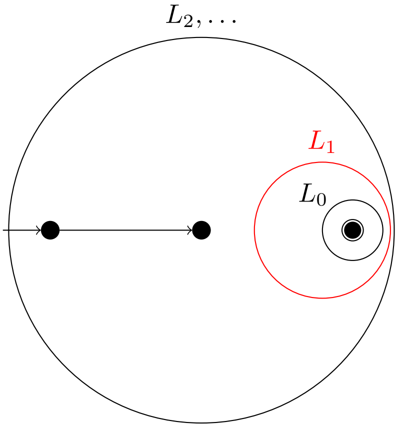
- Push clauses from lower index layers to higher index layers.
- Detects if a problem is unsolvable.

Clause Propagation Phase – Unsolvability

- If two layers identical ⇔ Problem unsolvable.



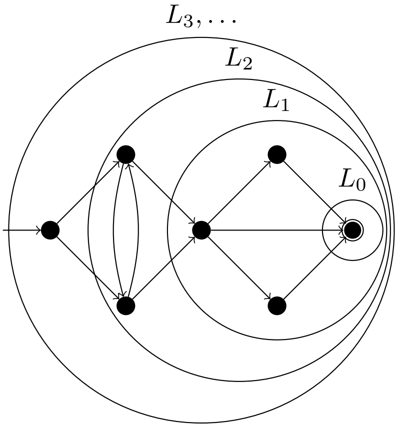
Clause Propagation Phase – Unsolvability



- If two layers identical ⇔ Problem unsolvable.

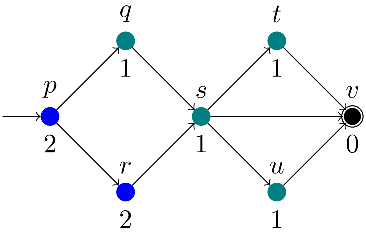
What is Seeding, and why might we want it?

- Populate layers in a preprocessing step.
- Saves a lot of work in the path construction phase.



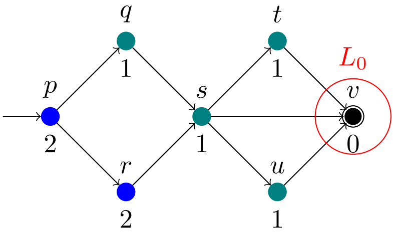
Seeding using a generic Heuristic

- Admissible heuristic h.
- Exclude state s from layer L_i if $h(s) > i$.
- $L_0 = \neg p \wedge \neg q \wedge \neg r \wedge \neg s \wedge \neg t \wedge \neg u \wedge s_*$
- $L_1 = \neg p \wedge \neg r$
- $L_2 = \top$



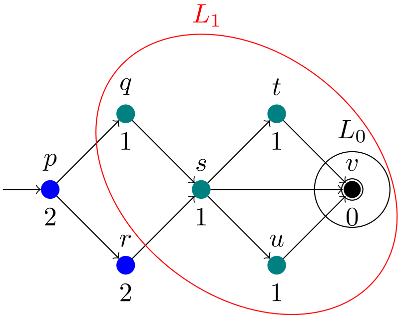
Seeding using a generic Heuristic

- Admissible heuristic h.
- Exclude state s from layer L_i if $h(s) > i$.
- $L_0 = \neg p \wedge \neg q \wedge \neg r \wedge \neg s \wedge \neg t \wedge \neg u \wedge s_*$
- $L_1 = \neg p \wedge \neg r$
- $L_2 = \top$



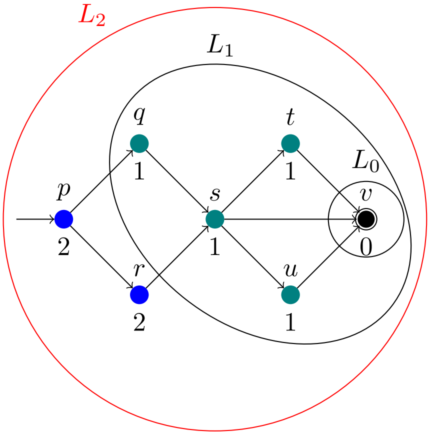
Seeding using a generic Heuristic

- Admissible heuristic h.
- Exclude state s from layer L_i if $h(s) > i$.
- $L_0 = \neg p \wedge \neg q \wedge \neg r \wedge \neg s \wedge \neg t \wedge \neg u \wedge s_*$
- $L_1 = \neg p \wedge \neg r$
- $L_2 = \top$



Seeding using a generic Heuristic

- Admissible heuristic h.
- Exclude state s from layer L_i if $h(s) > i$.
- $L_0 = \neg p \wedge \neg q \wedge \neg r \wedge \neg s \wedge \neg t \wedge \neg u \wedge s_*$
- $L_1 = \neg p \wedge \neg r$
- $L_2 = \top$



Seeding using a generic Heuristic

Problems:

- States have many variables.
- Many states (even if they are not reachable).

⇒ The layers are big formulas.

$$p = A \land B \land C \land D \land \dots$$

$$q = \neg A \land B \land C \land D \land \dots$$

$$v = A \land \neg B \land C \land D \land \dots$$

$$\dots$$

$$L_0 = \neg p \land \neg q \land \neg v \land \dots$$

Seeding using a generic Heuristic

Problems:

- States have many variables.
- Many states (even if they are not reachable).

⇒ The layers are big formulas.

$$p = A \wedge B \wedge C \wedge D \wedge \dots$$

$$q = \neg A \wedge B \wedge C \wedge D \wedge \dots$$

$$v = A \wedge \neg B \wedge C \wedge D \wedge \dots$$

$$\dots$$

$$L_0 = \neg p \wedge \neg q \wedge \neg v \wedge \dots$$

Seeding using a generic Heuristic

Problems:

- States have many variables.
- Many states (even if they are not reachable).

$\Rightarrow$ The layers are big formulas.

$$\begin{aligned} p &= A \wedge B \wedge C \wedge D \wedge \dots \\ q &= \neg A \wedge B \wedge C \wedge D \wedge \dots \\ v &= A \wedge \neg B \wedge C \wedge D \wedge \dots \\ &\dots \end{aligned}$$

$$L_0 = \neg p \wedge \neg q \wedge \neg v \wedge \dots$$

Seeding using the Pattern Database Heuristic

Solution:

- The pattern database heuristic h_{pdb} with a pattern P.
- Seeding using projected states $\pi_P(s)$.
- Exclude projected state $\pi_P(s)$ if $h_{pdb}(s) > i$.
- Smaller and fewer clauses.

Example:

$$\begin{aligned} P &= \{A, B, C\} \\ v &= A \wedge B \wedge \neg C \wedge D \wedge \ldots \\ \pi_P(v) &= A \wedge B \wedge \neg C \\ \neg\pi_P(v) &= \neg A \vee \neg B \vee C \end{aligned}$$

Seeding using the Pattern Database Heuristic

Solution:

- The pattern database heuristic h_{pdb} with a pattern P.
- Seeding using projected states $\pi_P(s)$.
- Exclude projected state $\pi_P(s)$ if $h_{pdb}(s) > i$.
- Smaller and fewer clauses.

Example:

$$\begin{aligned} P &= \{A, B, C\} \\ v &= A \wedge B \wedge \neg C \wedge D \wedge \ldots \\ \pi_P(v) &= A \wedge B \wedge \neg C \\ \neg \pi_P(v) &= \neg A \vee \neg B \vee C \end{aligned}$$

Seeding using the Pattern Database Heuristic

Solution:

- The pattern database heuristic h_{pdb} with a pattern P.
- Seeding using projected states $\pi_P(s)$.
- Exclude projected state $\pi_P(s)$ if $h_{pdb}(s) > i$.
- Smaller and fewer clauses.

Example:

$$
\begin{aligned}
P &= \{A, B, C\} \\
v &= A \land B \land \neg C \land D \land \ldots \\
\pi_P(v) &= A \land B \land \neg C \\
\neg\pi_P(v) &= \neg A \lor \neg B \lor C
\end{aligned}
$$

Seeding using the Pattern Database Heuristic

Solution:

- The pattern database heuristic h_{pdb} with a pattern P.
- Seeding using projected states $\pi_P(s)$.
- Exclude projected state $\pi_P(s)$ if $h_{pdb}(s) > i$.
- Smaller and fewer clauses.

Example:

$$P = \{A, B, C\}$$
$$v = A \wedge B \wedge \neg C \wedge D \wedge \dots$$
$$\pi_P(v) = A \wedge B \wedge \neg C$$
$$\neg\pi_P(v) = \neg A \vee \neg B \vee C$$

Seeding using the Pattern Database Heuristic

Solution:

- The pattern database heuristic h_{pdb} with a pattern P.
- Seeding using projected states $\pi_P(s)$.
- Exclude projected state $\pi_P(s)$ if $h_{pdb}(s) > i$.
- Smaller and fewer clauses.

Example:

$$P = \{A, B, C\}$$

$$v = A \wedge B \wedge \neg C \wedge D \wedge \ldots$$

$$\pi_P(v) = A \wedge B \wedge \neg C$$

$$\neg\pi_P(v) = \neg A \vee \neg B \vee C$$

Seeding using the Pattern Database Heuristic

Solution:

- The pattern database heuristic h_{pdb} with a pattern P.
- Seeding using projected states $\pi_P(s)$.
- Exclude projected state $\pi_P(s)$ if $h_{pdb}(s) > i$.
- Smaller and fewer clauses.

Example:

$$\begin{aligned} P &= \{A, B, C\} \\ v &= A \wedge B \wedge \neg C \wedge D \wedge \ldots \\ \pi_P(v) &= A \wedge B \wedge \neg C \\ \neg\pi_P(v) &= \neg A \vee \neg B \vee C \end{aligned}$$

Seeding using the Pattern Database Heuristic

Solution:

- The pattern database heuristic h_{pdb} with a pattern P.
- Seeding using projected states $\pi_P(s)$.
- Exclude projected state $\pi_P(s)$ if $h_{pdb}(s) > i$.
- Smaller and fewer clauses.

Example:

$$\begin{aligned} P &= \{A, B, C\} \\ v &= A \land B \land \neg C \land D \land \ldots \\ \pi_P(v) &= A \land B \land \neg C \\ \neg\pi_P(v) &= \neg A \lor \neg B \lor C \end{aligned}$$

Implementation

- As a new search engine in Fast Downward.
- Uses Fast Downwards Pattern Database, Pattern Generator.
- Can be extended for other heuristics.

Results

Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- "Overseeding" leads to performance decrease.
- Strictly fewer total expansion per task.

Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- "Overseeding" leads to performance decrease.
- Generally fewer expansions, but not strictly.

Results

Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- "Overseeding" leads to performance decrease.
- Strictly fewer total expansion per task.

Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- "Overseeding" leads to performance decrease.
- Generally fewer expansions, but not strictly.

Results

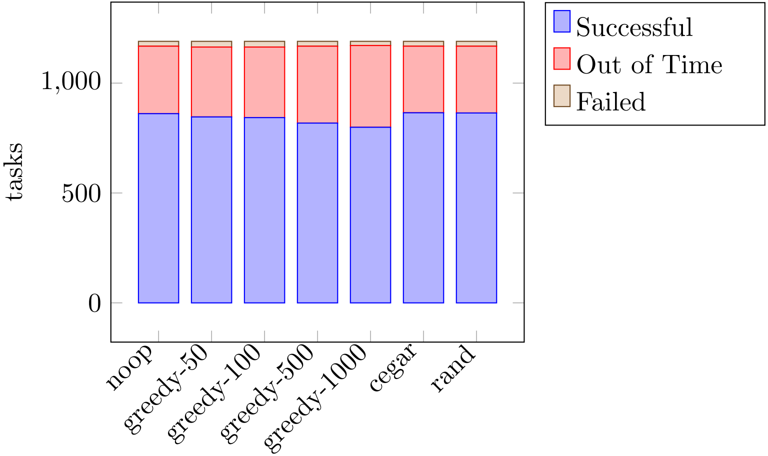
Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- "Overseeding" leads to performance decrease.
- Strictly fewer total expansion per task.

Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- "Overseeding" leads to performance decrease.
- Generally fewer expansions, but not strictly.

Number of Solved Tasks



Results

Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- “Overseeding” leads to performance decrease.
- Strictly fewer total expansion per task.

Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- “Overseeding” leads to performance decrease.
- Generally fewer expansions, but not strictly.

Results

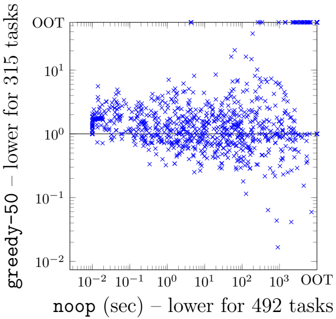
Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- “Overseeding” leads to performance decrease.
- Strictly fewer total expansion per task.

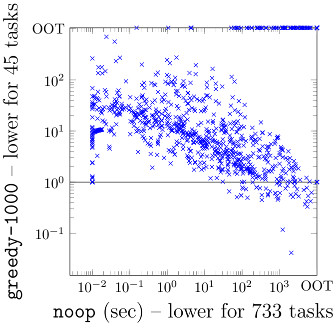
Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- “Overseeding” leads to performance decrease.
- Generally fewer expansions, but not strictly.

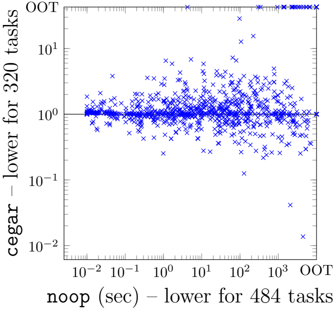
Planning Time



noop vs. **greedy-50**



noop vs. **greedy-1000**



noop vs. **cegar**

Results

Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- “Overseeding” leads to performance decrease.
- Strictly fewer total expansion per task.

Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- “Overseeding” leads to performance decrease.
- Generally fewer expansions, but not strictly.

Results

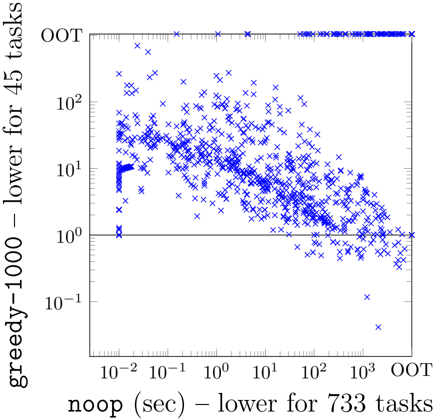
Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- “Overseeding” leads to performance decrease.
- Strictly fewer total expansion per task.

Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- “Overseeding” leads to performance decrease.
- Generally fewer expansions, but not strictly.

Overseeding



`noop` vs. `greedy-1000`

Results

Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- "Overseeding" leads to performance decrease.
- Strictly fewer total expansion per task.

Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- "Overseeding" leads to performance decrease.
- Generally fewer expansions, but not strictly.

Results

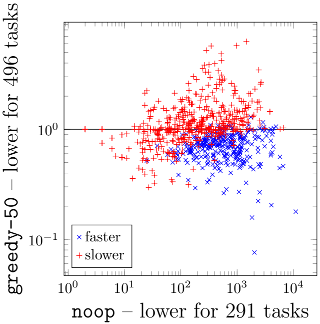
Expectation

- More solved tasks within time constraints.
- Tasks on average solved faster.
- "Overseeding" leads to performance decrease.
- Strictly fewer total expansion per task.

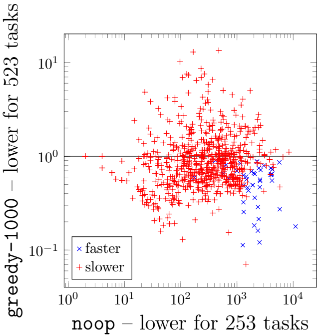
Result

- Few more tasks solved with light seeding, otherwise fewer tasks solved.
- Tasks on average solved slower.
- "Overseeding" leads to performance decrease.
- Generally fewer expansions, but not strictly.

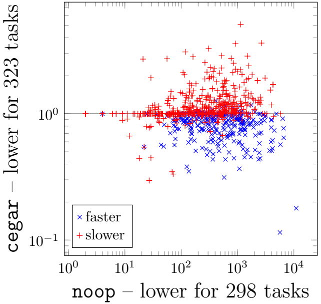
Expansions



`noop` vs. `greedy-50`

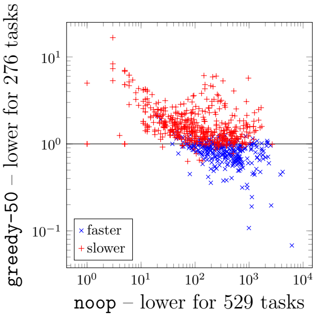


`noop` vs. `greedy-1000`

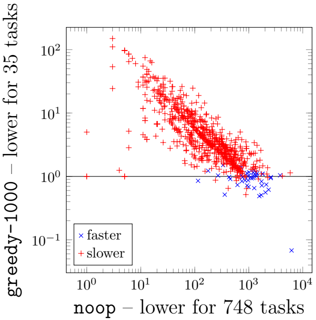


`noop` vs. `cegar`

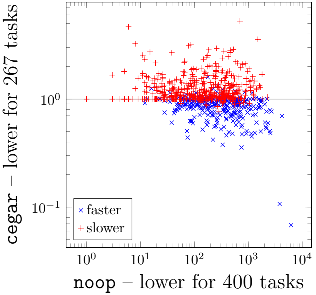
Size of Layer L_0



`noop` vs. `greedy-50`



`noop` vs. `greedy-1000`



`noop` vs. `cegar`

Conclusion

- Implemented PDR seeding
- Performance not consistently improved
- Expansions not consistently reduced

Questions?

Summary

- PDR in Fast Downward
- Seeding using Pattern Database

- Performance benefit less than expected

Configuration Outcomes

PDR Configuration	Successful	Out of Time	Failed
`noop`	861	307	22
`greedy-50`	846	318	26
`greedy-100`	843	321	26
`greedy-500`	818	350	22
`greedy-1000`	799	372	19
`cegar`	865	303	22
`rand`	864	304	22

Failed Tasks

Observations

- Unable to reproduce locally
- Happens to big tasks
- Durations of `step()` calls is high
  (single iteration of pdr takes a long time)

Hypothesis

- Sigkill from slurm, because process did not stop after timeout
  Wall-Clock time is not calculated properly