Simple and Fast Strong Cyclic Planning for Fully-Observable Nondeterministic Planning Problems

Jicheng Fu\textsuperscript{1}, Vincent Ng\textsuperscript{2}, Farokh Bastani\textsuperscript{2}, and I-Ling Yen\textsuperscript{2}

\textsuperscript{1}The University of Central Oklahoma
\textsuperscript{2}The University of Texas at Dallas
Goal

To solve strong cyclic planning problems from a Fully-Observable Nondeterministic planning domain
Goal

- To solve strong cyclic planning problems from a Fully-Observable Nondeterministic planning domain
Goal

- To solve strong cyclic planning problems from a Fully-Observable Nondeterministic planning domain

- A planning problem is a triple \( \langle s_0, g, \Sigma \rangle \), where
  - \( s_0 \) is the initial state,
  - \( g \) is the goal condition, and
  - \( \Sigma \) is the planning domain
Goal

- To solve strong cyclic planning problems from a Fully-Observable Non-deterministic planning domain
Goal

- To solve strong cyclic planning problems from a Fully-Observable Nondeterministic planning domain
- Informally, in a nondeterministic planning domain, an action may generate multiple effects
Goal

- To solve strong cyclic planning problems from a Fully-Observable Nondeterministic planning domain

- Informally, in a nondeterministic planning domain, an action may generate multiple effects

Formally, a nondeterministic domain is a 4-tuple \( \Sigma = (P, S, A, \gamma) \):

- \( P \) is a finite set of propositions;
- \( S \subseteq 2^P \) is a finite set of states in the system;
- \( A \) is a finite set of actions; and
- \( \gamma: S \times A \to 2^S \) is the state-transition function
Goal

- To solve strong cyclic planning problems from a Fully-Observable Nondeterministic planning domain
Goal

- To solve strong cyclic planning problems from a **Fully-Observable** Nondeterministic planning domain

- Full observability
  The states of the world are fully observable
Goal

- To solve strong cyclic planning problems from a Fully-Observable Nondeterministic planning domain
Goal

- To solve **strong cyclic planning** problems from a Fully-Observable Nondeterministic planning domain

- **Strong cyclic planning**
  - refers to a particular type of solutions to nondeterministic problems
  - different from so-called **weak planning**
Weak Planning Solutions

- Solutions where there is a chance to achieve the goal

In fact, non-goal leaf states are not part of the weak plan!

In the weak plan, there is no path from a non-goal leaf state to the goal

Non-deterministic actions

Non-goal leaf states
Strong Cyclic Planning Solutions

- Prescribe actions for all possible non-goal leaf states
  - Find a path for each non-goal leaf state to the goal state
  - May loop indefinitely
  - But contain no dead-ends
  - More difficult than finding weak planning solutions

Then a strong cyclic plan is found!
Representing a Plan

- Regardless of whether a plan is weak or strong cyclic, we can represent it as a **policy** $\pi$
  - a partial function mapping states to actions

- More formally, policy $\pi: S_{\pi} \rightarrow A$
  - consists of state action pairs $(s, a)$ such that $\pi(s) = a$
  - defines which action to take under state $s$
How to Generate a Strong Cyclic Plan

Given a planning problem with initial state $s_0$ and goal state $g$, we employ a 3-step Basic algorithm motivated by work in Incremental contingency planning [Dearden et al, 2003], FF-replan [Yoon, Fern, and Givan, 2007], and NDP [Kuter et al., 2008]
Basic Algorithm: Step 1

- Find a path (i.e., weak plan) from $s_0$ to $g$
  using a classic planner
Basic Algorithm: **Step 1**

- Find a **path** (i.e., weak plan) from \( s_0 \) to \( g \) using a classic planner
Basic Algorithm: **Step 1**

- Find a **path** (i.e., weak plan) from $s_0$ to $g$ using a classic planner

![Diagram showing a path from $s_0$ to $g$ with nondeterministic actions indicated by dotted arrows.](image)
Basic Algorithm: Step 1

- Find a **path** (i.e., weak plan) from \( s_0 \) to \( g \) using a classic planner

Given a nondeterministic action, the effect included in the weak plan is its **Intended Effect**
Basic Algorithm: Step 1

- Find a **path** (i.e., weak plan) from $s_0$ to $g$ using a classic planner

Given a nondeterministic action, the effect included in the weak plan is its **Intended Effect**; the effects not included in the weak plan are its **Failed Effects**.
Basic Algorithm: Step 1

- Find a path (i.e., weak plan) from $s_0$ to $g$ using a classic planner.

Given a nondeterministic action, the effect included in the weak plan is its **Intended Effect**; the effects not included in the weak plan are its **Failed Effects**. Put them all in a set $L$. 

Diagram: Nondeterministic actions.
Basic Algorithm: Step 2

- For every failed effect $e$, find a path from $e$ to $g$. 
Basic Algorithm: Step 2

- For every failed effect $e$, find a path from $e$ to $g$.

Then a strong cyclic plan is found!
But sometimes we may encounter a dead end. In this case, we need to **backtrack**

Action $a_2$ will be disabled at state $s_1$ since it leads to a dead end.
Basic Algorithm: Step 3

But sometimes we may encounter a dead end. In this case, we need to backtrack.

State $s_1$ will try some other actions to find another path to goal $g$.

Action $a_2$ will be disabled at state $s_1$ since it leads to a dead end.
Basic Algorithm: Application to Beam Domain

- A set of **positions** in one of two levels: *up* or *down*
- **Goal:** have the agent move from *down* \(_0\) to *up* \(_n\)
- **Three** possible actions can be used to move around
Action 1: **Climb**

- Climb is deterministic
  - it moves the agent from $\text{down}_0$ to $\text{up}_0$.
  - can only be applied to $\text{down}_0$
Action 2: Jump

- Jump is nondeterministic
  - can be applied only to a position in the upper level
  - if successful, agent moves to up position to its right
  - if unsuccessful, agent moves to down position to its right

Diagram:

0 -> 1

Jump to up₁ successfully

0 -> 1

Fall to down₁
Action 3: **Moveback**

- Moveback is deterministic
  - can be applied only to a position in the lower level
  - moves agent one step to the left
Applying the Basic Strong Cyclic Algorithm

- Initially, \( L = \{\text{down}_0\} \)
- **Step 1**: Find a path from \( \text{down}_0 \) to \( \text{up}_n \)

This is a weak plan because it does not consider the possibility of falling!
Applying the Basic Strong Cyclic Algorithm

- Initially, \( L = \{\text{down}_0\} \)
- **Step 1:** Find a path from \( \text{down}_0 \) to \( \text{up}_n \)

\[ L = \{\text{down}_1, \text{down}_2, ..., \text{down}_n\}, \] the set of failed effects that are not considered in the weak plan.
Applying the Basic Strong Cyclic Algorithm

- \( L = \{down_1, down_2, \ldots, down_n\} \)
- **Step 2**: Find a path from each position in \( L \) to \( up_n \).
- E.g., for \( down_1 \), the path to \( up_n \) is:
Applying the Basic Strong Cyclic Algorithm

- $L = \{\text{down}_1, \text{down}_2, \ldots, \text{down}_n\}$
- **Step 2**: Find a path from each position in $L$ to $\text{up}_n$.
- E.g., for $\text{down}_2$, the path to $\text{up}_n$ is:

```
Re-explored the states that have already been solved in the path for $\text{down}_1$
```

```
0 → 1 → 2 → 3 → … → n-1 → n
```

```
0 ← 1 ← 2 ← 3 ← … ← n-1 ← n
```

---

- $\text{down}_1$
- $\text{down}_2$
Applying the Basic Strong Cyclic Algorithm

- $L = \{\text{down}_1, \text{down}_2, \ldots, \text{down}_n\}$
- Step 2: Find a path from each position in $L$ to $\text{up}_n$.
- E.g., for $\text{down}_3$, the path to $\text{up}_n$ is:

Re-explored the states that have already been solved in the path for $\text{down}_2$

The same pattern repeats for other down positions.

**Issue:** Many states (e.g., $\text{down}_0$, $\text{up}_0$, etc.) are repeatedly explored!
So ...

- The basic algorithm can be *inefficient*
  many states can be repeatedly explored

- Goal
  improve the Basic algorithm w.r.t. *planning efficiency*
  and *plan size* by proposing *two extensions*
Extension 1: Goal Alternative

- Observation
  The Basic algorithm attempts to find a path from each failed effect to goal state \( g \), which can be far away.

- With goal alternative, we attempt to find a path from each failed effect to an alternative goal.
  An alternative goal is presumably closer to the associated failed effect than the overall goal \( g \).
  - Could improve planning efficiency and reduce plan size.
  - Each failed effect has its own alternative goal.
  - I.e., an alternative goal is associated with a failed effect.
  - We use the corresponding intended effect as alternative goal.
For each failed effect $down_i$

instead of using $up_n$ as the search goal, we use the intended effect $up_i$ of action $\text{Jump}(up_{i-1}, up_i)$ as the search goal.
For each failed effect $down_i$

instead of using $up_n$ as the search goal, we use the intended effect $up_i$ of action $\text{Jump}(up_{i-1}, up_i)$ as the search goal.
For each failed effect $down_i$

instead of using $up_n$ as the search goal, we use the intended effect $up_i$ of action Jump($up_{i-1}$, $up_i$) as the search goal.
For failed effect $down_1$, 

- If we use goal alternative, the intended effect $up_1$ is the search goal

If we use the ultimate goal $up_n$ as the search goal

$S$ The generated weak plan is lengthy
Goal Alternative: A Caveat

- It is possible that a path cannot be established between a failed effect and its alternative goal.
- If this happens, we resort to establishing a path from the failed effect to the original goal $g$. 
Why is Goal Alternative correct?

- By definition, an intended effect $\hat{s}$ is included in some path $wp$ to goal $g$, while a failed effect $s$ is ignored in $wp$.

- Since we have already found a path from $\hat{s}$ to $g$, if we can find a path from $s$ to $\hat{s}$, then the path from $s$ to $\hat{s}$ can be the solution to $\langle s, g, \Sigma \rangle$.

- Hence, much effort is saved by avoiding the search from $\hat{s}$ to $g$. 
Observation

Even if a state is solved (i.e., a path has been found from $s$ to the goal $g$), the Basic algorithm still attempts to solve it every time it is encountered.

State reuse aims to improve planning efficiency by not re-sol veling a state.

When searching for a weak plan, if a solved state is encountered, the search stops.
For failed effect $down_1$, goal alternative generates the plan:

Moveback($down_1$, $down_0$); Climb($down_0$, $up_0$); Jump($up_0$, $up_1$)

State reuse will make the plan even more concise:

Moveback($down_0$)

Reached a solved state $down_0$. Hence, the search stops!
Note that …

- State reuse and goal alternative can be applied independently of each other
  In particular, goal alternative does NOT rely on state reuse to improve planning efficiency

- In our poster we use the blocksworld example to show that goal alternative plays a more critical role than state reuse
Evaluation

- All problem instances belong to the benchmark domains of the IPC2008 FOND track
  Blocksworld, Forest, Faults, and First-responders

- Goal
  compare FIP, our planner that implements the Basic algorithm with the two extensions, against two state-of-the-art planners, MBP and Gamer
give each planner 1200 seconds to solve each problem instance
Evaluation 1: Problem Coverage

<table>
<thead>
<tr>
<th>Domain</th>
<th>Gamer</th>
<th>MBP</th>
<th>FIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocksworld (30)</td>
<td>10</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>faults (55)</td>
<td>38</td>
<td>16</td>
<td>55</td>
</tr>
<tr>
<td>first-responders (100)</td>
<td>21</td>
<td>11</td>
<td>75</td>
</tr>
<tr>
<td>forest(90)</td>
<td>7</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Total (275)</td>
<td>76</td>
<td>28</td>
<td>167</td>
</tr>
</tbody>
</table>

FIP solves more problems than Gamer and MBP within the time limit
Evaluation 2: Efficiency

Among the Basic Algorithm

- FIP is on average more than 8 times faster than Basic
- As the problem complexity increases, FIP could be more than 100 times faster than Basic
- FIP’s plans are 3.4 times smaller than Basic on average

Comparing with MBP and Gamer

- FIP is on average more than three orders of magnitude faster
- FIP’s plans are 2.8 times smaller than Gamer’s
Evaluation 3: Which of the two extensions makes a more critical contribution?

<table>
<thead>
<tr>
<th>Problem</th>
<th>FIP</th>
<th>FIP-SR-only</th>
<th>FIP-GA-only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t</td>
<td>s</td>
<td>t</td>
</tr>
<tr>
<td>bw-1</td>
<td>0.007</td>
<td>8</td>
<td>0.010</td>
</tr>
<tr>
<td>bw-2</td>
<td>0.006</td>
<td>7</td>
<td>0.008</td>
</tr>
<tr>
<td>bw-3</td>
<td>0.008</td>
<td>10</td>
<td>0.008</td>
</tr>
<tr>
<td>bw-4</td>
<td>0.011</td>
<td>14</td>
<td>0.012</td>
</tr>
<tr>
<td>bw-5</td>
<td>0.010</td>
<td>12</td>
<td>0.014</td>
</tr>
<tr>
<td>bw-6</td>
<td>0.009</td>
<td>10</td>
<td>0.010</td>
</tr>
<tr>
<td>bw-7</td>
<td>0.010</td>
<td>17</td>
<td>0.016</td>
</tr>
<tr>
<td>bw-8</td>
<td>0.285</td>
<td>45</td>
<td>0.725</td>
</tr>
<tr>
<td>bw-9</td>
<td>59.230</td>
<td>312</td>
<td>112.235</td>
</tr>
<tr>
<td>bw-10</td>
<td>3.126</td>
<td>48</td>
<td>7.458</td>
</tr>
<tr>
<td>faults-6-6</td>
<td>0.005</td>
<td>20</td>
<td>0.021</td>
</tr>
<tr>
<td>faults-7-7</td>
<td>0.005</td>
<td>23</td>
<td>0.045</td>
</tr>
<tr>
<td>faults-8-8</td>
<td>0.007</td>
<td>26</td>
<td>0.111</td>
</tr>
<tr>
<td>faults-9-9</td>
<td>0.007</td>
<td>29</td>
<td>0.246</td>
</tr>
<tr>
<td>faults-10-7</td>
<td>0.007</td>
<td>32</td>
<td>0.960</td>
</tr>
<tr>
<td>faults-10-10</td>
<td>0.009</td>
<td>32</td>
<td>1.101</td>
</tr>
<tr>
<td>f-r-1-8</td>
<td>0.002</td>
<td>10</td>
<td>0.003</td>
</tr>
<tr>
<td>f-r-2-3</td>
<td>0.003</td>
<td>11</td>
<td>0.003</td>
</tr>
<tr>
<td>f-r-4-2</td>
<td>0.003</td>
<td>6</td>
<td>0.003</td>
</tr>
<tr>
<td>f-r-6-2</td>
<td>0.003</td>
<td>7</td>
<td>0.003</td>
</tr>
<tr>
<td>forest-2-5</td>
<td>0.008</td>
<td>56</td>
<td>0.007</td>
</tr>
<tr>
<td>forest-2-6</td>
<td>0.008</td>
<td>50</td>
<td>0.007</td>
</tr>
<tr>
<td>forest-2-7</td>
<td>0.008</td>
<td>44</td>
<td>0.007</td>
</tr>
<tr>
<td>forest-2-8</td>
<td>0.008</td>
<td>56</td>
<td>0.007</td>
</tr>
<tr>
<td>forest-2-9</td>
<td>0.007</td>
<td>42</td>
<td>0.006</td>
</tr>
<tr>
<td>forest-2-10</td>
<td>0.008</td>
<td>44</td>
<td>0.006</td>
</tr>
</tbody>
</table>

FIP-SR-only: extends Basic with state reuse only

FIP-GA-only: extends Basic with goal alternative only

So, goal alternative plays a more crucial role than state reuse in improving planning efficiency and reducing plan size!

On average, FIP-GA-only runs more than 5 times faster than FIP-SR-only

FIP-GA-only creates plans that are 3.4 times smaller than FIP-SR-only.
Summary

- Proposed two extensions to the Basic strong cyclic planning algorithm, goal alternative and state reuse

- FIP significantly outperforms state-of-the-art planners in terms of problem coverage, efficiency, and solution size.