Planning with Object Creation

Augusto B. Corrêa, Giuseppe De Giacomo,
Malte Helmert, and Sasha Rubin

University of Basel, Switzerland,
University of Oxford, UK
University of Sydney, Australia
Previous Work

Hoffmann et al. (2009): web service composition (specific case)

Fuentetaja & de la Rosa (2016): compile irrelevant objects to counters

Edelkamp et al. (2019): compile to model-checking
Our Work

synergy between object creation and lifted heuristic search

complete semantics of classical planning with object creation in action effects
( :action buy-largest-possible-truck
  (:parameters (?C - city))
  (:precondition ( has-garage ?C))
  (:effect ( and
               ( when ( large-garage ?C)
                     (:new (?T - large-truck)
                           (at ?T ?C)))
               ( when ( not ( large-garage ?C))
                     (:new (?T - small-truck)
                           (at ?T ?C))))))
How to plan?

if task is solvable:
lifted breadth-first search finds a plan!

otherwise:
search might go on forever

planning with object creation is semidecidable
How to search?

```python
def bfs(task):
    queue = Queue(task.initial_state)
    visited = set()
    while not queue.empty():
        s = queue.pop()
        if is_goal(succ):
            return extract_plan(succ)
        for a in s.get_applicable_actions():
            succ = get_successor(s, a)
            if succ not in visited:
                queue.add(succ)
                visited.add(succ)
    return UNSOLVABLE
```

Applicable Actions

state = database
precondition = CSP/query

if objects change, it does not matter:
new CSP at every state anyway

[Francès 2017; ABC et al. 2020; Horčík & Fišer 2021; Ståhlberg 2023]
How to search?

```python
def bfs(task):
    queue = Queue(task.initial_state)
    visited = set()
    while not queue.empty():
        s = queue.pop()
        if is_goal(succ):
            return extract_plan(succ)
        for a in s.get_applicable_actions():
            succ = get_successor(s, a)
            if succ not in visited:
                queue.add(succ)
                visited.add(succ)
    return UNSOLVABLE
```
Generate Successors

effects evaluated as in classical planning except when we have object creation

main difference:
assign names to created objects

different ways: numbers, strings, etc.
How to search?

```python
def bfs(task):
    queue = Queue(task.initial_state)
    visited = set()
    while not queue.empty():
        s = queue.pop()
        if is_goal(succ):
            return extract_plan(succ)
        for a in s.get_applicable_actions():
            succ = get_successor(s, a)
            if succ not in visited:
                queue.add(succ)
                visited.add(succ)
    return UNSOLVABLE
```
Duplicated States
Duplicated States

#7

#8
Duplicated States

#7

#8

#7
Duplicated States

#7

#8

#8

#7
Duplicated States

not enough to check equality

need to check state isomorphism

similar to ideas in orbit search

our implementation: simple duplicate detection
Experimental Results

implemented on top of Powerlifted

four STRIPS-like domains:

1. logistics company
2. cluster management
3. commutative rings (Petrov & Muise 2023)
4. settlers (Long & Fox 2003, IPC 2002)
Experimental Results

model problems simulating object creation:
objects pre-declared; extra predicates

two baselines without object creation:
Powerlifted & Fast Downward (FD)
# Breadth-First Search

<table>
<thead>
<tr>
<th>Category</th>
<th>w/ creation</th>
<th>no creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistics Comp.</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Cluster Manag.</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Comm. Rings</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>Settlers</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>11</strong></td>
<td><strong>19</strong></td>
</tr>
</tbody>
</table>
Problems

3/4 domains have unrestricted creation

branching factor increases at each layer

settlers:
only domain where this is not the case, and all methods perform similarly
Best-First Width Search

compute novelty and run BFWS


every time an object is created, the successor state has novel tuples!

solution:
partition on # of new objects
<table>
<thead>
<tr>
<th></th>
<th>w/ creation</th>
<th>no creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistics Comp.</td>
<td>18</td>
<td>8</td>
</tr>
<tr>
<td>Cluster Manag.</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>Comm. Rings</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>Settlers</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>46</strong></td>
<td><strong>42</strong></td>
</tr>
<tr>
<td></td>
<td>w/ creation</td>
<td>FD</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>----</td>
</tr>
<tr>
<td>Logistics Comp.</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td>Cluster Manag.</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Comm. Rings</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Settlers</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>46</strong></td>
<td><strong>32</strong></td>
</tr>
</tbody>
</table>
Conclusion

object creation in action effects implemented on top of a lifted planner

a long way to go...
Conclusion

object creation in action effects

implemented on top of a lifted planner

a long way to go...

Thank you!
## Breadth-First Search

<table>
<thead>
<tr>
<th></th>
<th>w/ creation</th>
<th>FD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistics Comp.</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Cluster Manag.</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Comm. Rings</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Settlers</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>11</strong></td>
<td><strong>21</strong></td>
</tr>
</tbody>
</table>