Deeply Preferred Operators: Lazy Search Meets Lookahead

Roei Bahumi Carmel Domshlak Erez Karpas

Faculty of Industrial Engineering and Management,
Technion — Israel Institute of Technology

May 14, 2012

Outline

1. Background & Motivation
2. Lazy Lookahead: Deeply Preferred Operators
3. Empirical Evaluation

Heuristic Search

- Search algorithm
  - Chooses which state to expand next
  - Choice is based on heuristic evaluation function
- Heuristic
  - Used to estimate distance from state s to the goal
  - Can also prefer some successors of s

Heuristic Search

- Search algorithm
  - Chooses which state to expand next
  - Choice is based on heuristic evaluation function
- Heuristic
  - Used to estimate distance from state s to the goal
  - Can also prefer some successors of s

Preferred Operators

- FF's relaxed plan heuristic (Hoffmann & Nebel, 2001) uses the relaxed planning graph to construct a relaxed plan
- The chosen actions in the first layer of the relaxed planning graph are denoted as helpful
- Later generalized to preferred operators
    - Causal graph heuristic (Helmert, 2006)
    - Landmark count heuristic (Richter, Helmert & Westphal, 2008)
    - Structural pattern heuristic (Bahumi, Domshlak & Katz, 2011)

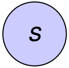
Using Preferred Operators

- Originally, used in FF by pruning all non-preferred operators in EHC search
    - Incomplete, but very effective
    - If first search fails, uses complete GBFS search, ignoring preferred operators
- Fast Downward uses “alternating dual queues”
    - Two open lists: one containing all states, the other only preferred states
    - Alternate between the open lists
    - Preserves completeness

Lazy Search

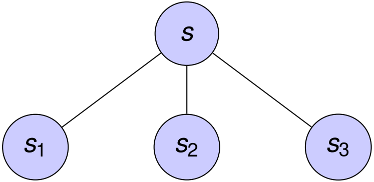
- Lazy search (deferred evaluation) changes the search: a state is inserted into the open list with the heuristic value of its parent
  - A state is only evaluated when it is removed from the open list and expanded
  - This reduces the number of heuristic evaluations at the last layer of the search
- Found by Richter & Helmert (2009) to work especially well when using preferred operators

Lazy Search with Preferred Operators: Illustration



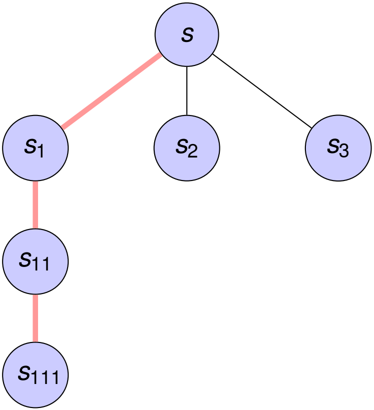
1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1
4. Insert s_1, s_2, s_3 into open list with $h = 3$

Lazy Search with Preferred Operators: Illustration



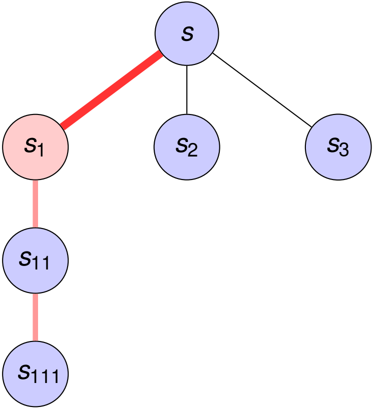
1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1
4. Insert s_1, s_2, s_3 into open list with $h = 3$

Lazy Search with Preferred Operators: Illustration



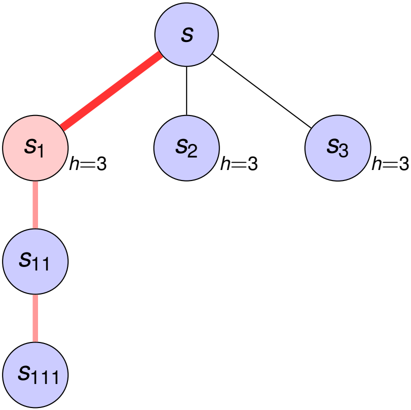
1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1
4. Insert s_1, s_2, s_3 into open list with $h = 3$

Lazy Search with Preferred Operators: Illustration



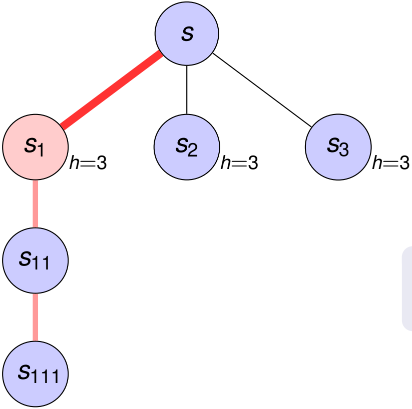
1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1
4. Insert s_1, s_2, s_3 into open list with $h = 3$

Lazy Search with Preferred Operators: Illustration



1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1
4. Insert s_1, s_2, s_3 into open list with $h = 3$

Lazy Search with Preferred Operators: Illustration



1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1
4. Insert s_1, s_2, s_3 into open list with $h = 3$

Information about the rest of the plan is lost

Lookahead

- Vidal (2004) proposed lookahead:
    1. Attempt to follow FF's relaxed plan
    2. Add the last state reached by the relaxed plan to the open list
- Uses a sophisticated procedure for following the relaxed plan
- This lookahead was integrated into eager search

Outline

1. Background & Motivation

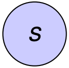
2. Lazy Lookahead: Deeply Preferred Operators

3. Empirical Evaluation

Lazy Lookahead

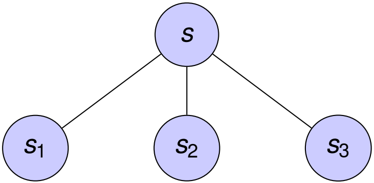
- We propose combining lookahead with lazy search
  1. Expand and evaluate state *s*
  2. A heuristically suggested path is generated
  3. Follow the heuristically suggested path, adding every state along it to the open list
  4. The heuristic estimate of each of these states is adjusted by the cost along the heuristically suggested path to reach it

Lazy Search with Deeply Preferred Operators: Illustration



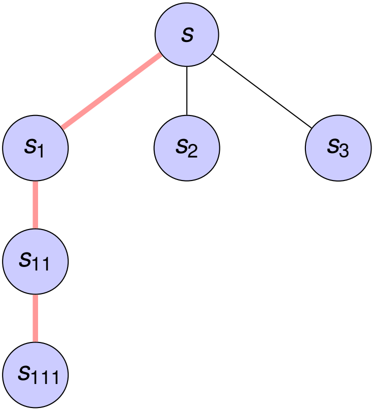
1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1, s_{11}, s_{111}
4. Insert into open list:
   - s_2, s_3 with $h = 3$
   - s_1 with $h = 2$
   - s_{11} with $h = 1$
   - s_{111} with $h = 0$

Lazy Search with Deeply Preferred Operators: Illustration



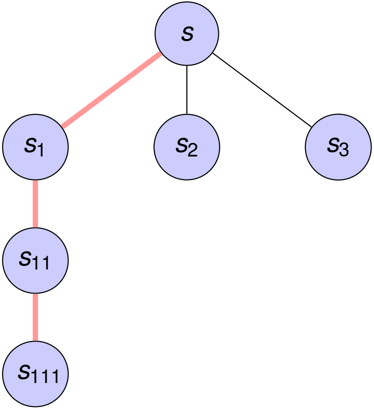
1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1, s_{11}, s_{111}
4. Insert into open list:
   - s_2, s_3 with $h = 3$
   - s_1 with $h = 2$
   - s_{11} with $h = 1$
   - s_{111} with $h = 0$

Lazy Search with Deeply Preferred Operators: Illustration



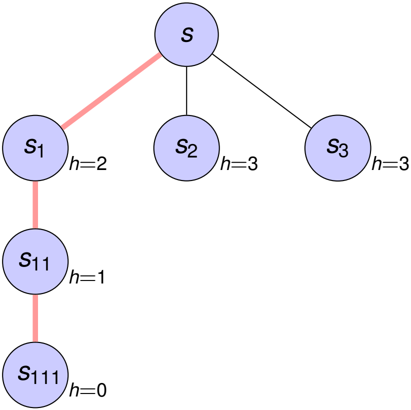
1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1, s_{11}, s_{111}
4. Insert into open list:
   - s_2, s_3 with $h = 3$
   - s_1 with $h = 2$
   - s_{11} with $h = 1$
   - s_{111} with $h = 0$

Lazy Search with Deeply Preferred Operators: Illustration



1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1, s_{11}, s_{111}
4. Insert into open list:
   - s_2, s_3 with $h = 3$
   - s_1 with $h = 2$
   - s_{11} with $h = 1$
   - s_{111} with $h = 0$

Lazy Search with Deeply Preferred Operators: Illustration



1. Expand s
2. Evaluate s: $h(s) = 3$
3. Preferred: s_1, s_{11}, s_{111}
4. Insert into open list:
   - s_2, s_3 with $h = 3$
   - s_1 with $h = 2$
   - s_{11} with $h = 1$
   - s_{111} with $h = 0$

Heuristically Suggeted Paths

- Not always easy to generate from a plan for an abstraction
    - Might be partially ordered
    - Might not be applicable
    - Might not reach the goal

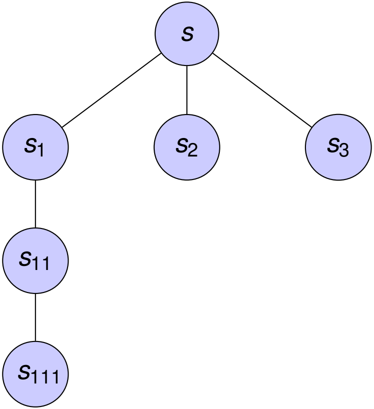
Generating Heuristically Suggeted Paths

- We repeatedly attempt to apply actions according to the partial order, until no more actions can be applied
- When more than one action is applicable, we choose according to some arbitrary order (LL), or choose at random (rnd-LL)
- In the implementation for the relaxed plan heuristic, we also order actions according to the layers in the relaxed planning graph
- It is possible to use more sophisticated reasoning (Vidal, 2004)

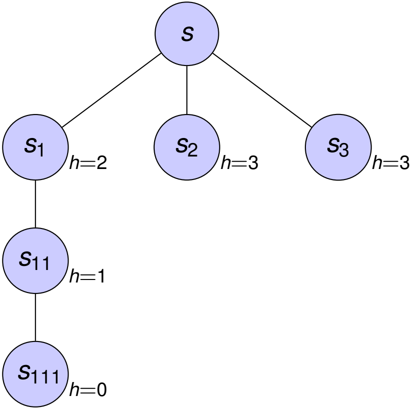
Lazy Lookahead: Pros & Cons

- Pros
  - Can drastically reduce number of heuristic evaluations
  - Can provide guidance even if only first part of heuristically suggested path is good
- Cons
  - Could lead to expansion of huge heuristic plateaus

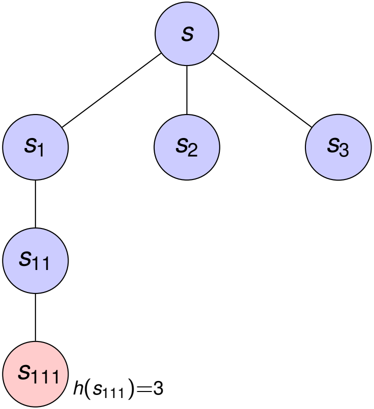
Lazy Lookahead: Bad Behavior



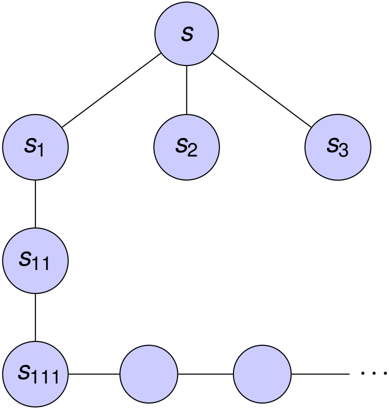
Lazy Lookahead: Bad Behavior



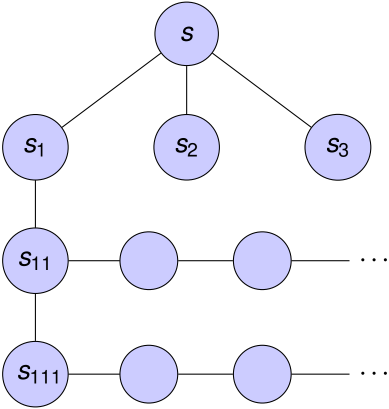
Lazy Lookahead: Bad Behavior



Lazy Lookahead: Bad Behavior



Lazy Lookahead: Bad Behavior



Conditional Lookahead (CLL)

- Addressed the bad behaior above
- Only performs look ahead from state *s* when:
    - State *s* was reached "normally" (not from look ahead), or
    - The true heuristic value state *s* (computed when *s* is expanded) is lower than the true heuristic value of the ancestor where the look ahead started
- Requires keeping track of extra information at each search node

Outline

1. Background & Motivation
2. Lazy Lookahead: Deeply Preferred Operators
3. Empirical Evaluation

Experiment Setup

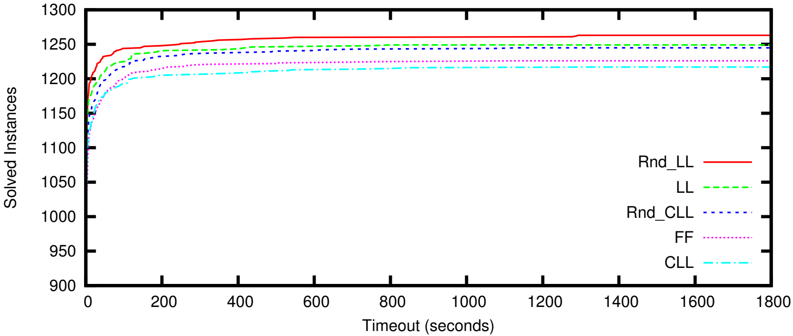
- Implemented on top of Fast Downward
- We use the relaxed plan heuristic in the evaluation
- Lazy greedy best first search, boosted dual queues with preferred operators
  - LL — lazy lookahead with arbitrary action ordering
  - rnd-LL — lazy lookahead with random action ordering
  - CLL — conditional lazy lookahead with arbitrary action ordering
  - rnd-CLL — conditional lazy lookahead with random action ordering
  - FF — baseline relaxed plan heuristic (FD implementation)
- 1.5 GB memory limit, 30 minute time limit

Empirical Evaluation: Solved Instances

domain	LL	rnd-LL	CLL	rnd-CLL	FF
airport (50)	37	**38**	**38**	35	37
depot (22)	19	**20**	19	18	19
logistics98 (35)	33	**35**	**35**	**35**	33
mystery (30)	15	**16**	**16**	**16**	**16**
openstacks (30)	6	6	6	6	6
optical-telegraphs (48)	**3**	**3**	**3**	**3**	2
parcprinter (30)	**27**	17	26	18	21
pathways (30)	20	22	21	22	**29**
philosophers (48)	**48**	**48**	20	40	42
pw-notankage (50)	43	**44**	43	43	41
pw-tankage (50)	41	**43**	40	41	40
psr-large (50)	15	**16**	15	**16**	15
psr-middle (50)	43	42	43	**44**	42
schedule (150)	**150**	**150**	**150**	**150**	149
sokoban (30)	28	**29**	28	28	28
storage (30)	17	19	17	19	**20**
transport (30)	**30**	**30**	**30**	**30**	21
trucks-strips (30)	16	17	16	17	**18**
woodworking (30)	**30**	29	**30**	**30**	27
TOTAL	1260	**1263**	1235	1250	1245

Only domains where there was any difference in the results are shown.

Empirical Evaluation: Anytime Results



Empirical Evaluation: Generated/Evaluated States

- We computed the metric score of the number of generated states and the number of evaluated states.
- The metric score of configuration c is $\frac{v^*}{v_c}$
- Averages over problems solved by all configurations in each domain
- Here we report only the overall average (over domain scores)

Attribute	LL	rnd-LL	CLL	rnd-CLL	FF
Generated States	0.70	**0.73**	0.62	0.66	0.40
Evaluated States	**0.76**	0.72	0.66	0.63	0.36

Summary

- Presented a method of combining lookahead with lazy search
- Random action ordering helps
- Conditional lookahead might be too costly
- Lazy lookahead performs better than the baseline

Thank You