How to Avoid Learning

Vadim Bulitko & Yngvi Björnsson

http://ircl.cs.ualberta.ca
Acknowledgments
Outline

- Heuristic search
  - learning in heuristic search
- How to avoid on-line learning
  - D LRTA*
  - kNN LRTA*
- Recap: Ideas
- Conclusions
search on a finite weighted graph

- goal and start states are known
- heuristic guidance:
  - estimated distance to goal
Algorithm:
- A* vs. LRTA*

Heuristic function:
- naive vs. pre-computed

Goal:
- global vs. subgoal
A* computes an entire path before taking its first action
does not scale well with problem size
Various techniques/tricks are used to mitigate
weighted A*
PRA*

We want \textit{true} real-time planning
time per move is constant-bounded
constant is independent of the number of states
Applications

- Planning
  - in video games
  - in abstract games
  - on robots
Objectives & Measures

- Planning time per move independent of number of states \((\text{real-time-ness})\)
  - \(\text{cap}\)
- Short planning time per move (under the cap)
  - \(\text{states expanded per move}\)
  - \(\text{CPU seconds}\)
- High-quality paths
  - \(\text{ratio of path cost to optimal path cost}\)
Taxonomy of Search

- Algorithm:
  - $A^*$ vs. LRTA$^*$

- Heuristic function:
  - naive vs. sophisticated/pre-computed

- Goal:
  - global vs. subgoal
Algorithm:
- $A^*$ vs. LRTA$^*$

Heuristic function:
- naive vs. sophisticated/pre-computed

Goal:
- global vs. subgoal
## Taxonomy of Search

<table>
<thead>
<tr>
<th>Goal \ h</th>
<th>Naive</th>
<th>Pre-computed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Global</strong></td>
<td>LRTA*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LRTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td><strong>Local</strong></td>
<td>PR LRTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D LRTA*</td>
<td></td>
</tr>
</tbody>
</table>
Conduct limited lookahead around the agent

Update heuristic function

Take one action

A special case of real-time dynamic programming (RTDP)

Similar to simple RL algorithms (e.g., Q-learning)

Planning time per action independent of number of states

Poor paths
LRTA* in action
LRTA* in action
LRTA* in action
LRTA* in action
LRTA* in action
LRTA* in action

Q-learning for state values, no exploration policy
The Problem

- Heuristic is inaccurate
- misleads the agent
- fixing the heuristic takes a long time
The Problem

- Heuristic is inaccurate
- misleads the agent
- fixing the heuristic takes a long time
Solutions

- learn heuristic function more efficiently  
  \( \text{LRTS, etc.} \)

- learn heuristic in a smaller abstract space  
  \( \text{PR LRTS} \)

- learn less by starting with a better initial heuristic

  - pre-compute a better heuristic
  - choose closer goals

  \( \text{D LRTA*} \)

  - heuristic is better closer to a goal
Avoiding Learning

- Use a standard heuristic
  - procedurally specified (e.g., straight-line distance)
- Reduce inaccuracies by using subgoals

How to choose subgoals?

<table>
<thead>
<tr>
<th>Goal \ h</th>
<th>Naive</th>
<th>Pre-computed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>PR LRTS D LRTA*</td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td>LRTA*</td>
<td></td>
</tr>
</tbody>
</table>
For every state pair
- compute an optimal path
- store states on this path as subgoals

Simple algorithm

Intractable precomputation time

Excessive memory
For every state pair:
- compute an optimal path
- store states on this path as subgoals

Simple algorithm

Intractable precomputation time

Excessive memory
For every state pair
  compute an optimal path
  store states on this path as subgoals

Simple algorithm

Intractable precomputation time

Excessive memory
For every *abstract* state pair
- compute an optimal path
- store *first gateway* state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
  - square of abstract states
For every abstract state pair

- compute an optimal path
- store first gateway state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
- square of abstract states
For every abstract state pair
- compute an optimal path
- store first gateway state on this path as subgoal

Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
- square of abstract states
Precomputing Subgoals: D LRTA* 

- For every abstract state pair
  - compute an optimal path
  - store first gateway state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
  - square of abstract states
For every abstract state pair
- compute an optimal path
- store first gateway state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
- square of abstract states
Precomputing Subgoals: D LRTA* 

For every abstract state pair 
- compute an optimal path 
- store first gateway state on this path as subgoal

Complex algorithm 
Reasonable precomputation time 
Memory can still be excessive 
square of abstract states
Precomputing Subgoals: D LRTA*

- For every abstract state pair
  - compute an optimal path
  - store first gateway state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
  - square of abstract states
For every *abstract* state pair
- compute an optimal path
- store *first gateway* state on this path as subgoal

**Complex algorithm**
- **Reasonable precomputation time**
- **Memory can still be excessive**
  - square of abstract states
For every abstract state pair

- compute an optimal path
- store first gateway state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
- square of abstract states
Precomputing Subgoals: D LRTA*

- For every *abstract* state pair
  - compute an optimal path
  - store *first gateway* state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
  - square of abstract states
For every *abstract* state pair
- compute an optimal path
- store *first gateway* state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
- square of abstract states
For every *abstract* state pair

- compute an optimal path
- store *first gateway* state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
  - square of abstract states
For every *abstract* state pair
- compute an optimal path
- store *first gateway* state on this path as subgoal

- Complex algorithm
- Reasonable precomputation time
- Memory can still be excessive
- square of abstract states
Precomputing Subgoals: kNN LRTA*

- For N random state pairs:
  - compute an optimal path
  - compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory O(N)
For $N$ random state pairs:
- compute an optimal path
- compress the path into a series of subgoals

Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory $O(N)$
For $N$ random state pairs:

- compute an optimal path
- compress the path into a series of subgoals

**Simple(r) algorithm**

**Reasonable precomputation time**

**Reasonable memory $O(N)$**
For N random state pairs:
- compute an optimal path
- compress the path into a series of subgoals

Simple(r) algorithm
Reasonable precomputation time
Reasonable memory O(N)
For $N$ random state pairs:

- compute an optimal path
- compress the path into a series of subgoals

Simple(r) algorithm

Reasonable precomputation time

Reasonable memory $O(N)$
For N random state pairs:
- compute an optimal path
- compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory O(N)
Precomputing Subgoals: kNN LRTA*

- For $N$ random state pairs:
  - compute an optimal path
  - compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory $O(N)$
For $N$ random state pairs:

- compute an optimal path
- compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory $O(N)$
Precomputing Subgoals: kNN LRTA*

- For N random state pairs:
  - compute an optimal path
  - compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory O(N)
Precomputing Subgoals: kNN LRTA*

- For N random state pairs:
  - compute an optimal path
  - compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory O(N)
For N random state pairs:

- compute an optimal path
- compress the path into a series of subgoals

Simple(r) algorithm

Reasonable precomputation time

Reasonable memory O(N)
For N random state pairs:

- compute an optimal path
- compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory O(N)
Precomputing Subgoals: kNN LRTA*

- For N random state pairs:
  - compute an optimal path
  - compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory $O(N)$
For N random state pairs:
- compute an optimal path
- compress the path into a series of subgoals

Simple(r) algorithm
Reasonable precomputation time
Reasonable memory O(N)
For $N$ random state pairs:

- compute an optimal path
- compress the path into a series of subgoals

- Simple(r) algorithm
- Reasonable precomputation time
- Reasonable memory $O(N)$
Theoretical Properties

- Guaranteed real-time on-line operation
  - upper-bounded planning time per move regardless of number of states
- Completeness
  - finds goal state for:
    - finite directed weighted graph
    - positive finite edge costs
    - goal reachable from any state reachable from start state

Standard for real-time heuristic search
Empirical Evaluation

1024 random problems       [100,150] optimal cost on each
Empirical Evaluation

Upscaled maps

Real-time cut-off: 10000

Suboptimality (times)

Mean number of states expanded per move

LRTA* (F, G)
LRTA* (F, PDB)
LRTA* (PDB, G)

Want to be here

200
Empirical Evaluation

Upscaled maps

Real-time cut-off: 10000

- LRTA* (F, G)
- LRTA* (F, PDB)
- LRTA* (PDB, G)

Suboptimality (times)

Mean number of states expanded per move

Want to be here

200
Empirical Evaluation

Upscaled maps

Real-time cut-off: 1000

- LRTA* (F, PDB)
- LRTA* (PDB, PDB)
- PR LRTA* (F, G)
- PR LRTA* (F, PDB)
- PR LRTA* (PDB, PDB)
- PR LRTA* (PDB, G)

Suboptimality (times)

Mean number of states expanded per move

Varić Bulićka & Yngvi Björnsson
July 20, 2009
Empirical Evaluation

Upscaled maps

Real-time cut-off: 1000

- \( \text{LRTA}^* \) (F, PDB)
- \( \text{LRTA}^* \) (PDB, PDB)
- \( \text{PR LRTA}^* \) (F, G)
- \( \text{PR LRTA}^* \) (F, PDB)
- \( \text{PR LRTA}^* \) (PDB, PDB)
- \( \text{PR LRTA}^* \) (PDB, G)

Suboptimality (times)

Mean number of states expanded per move
Empirical Evaluation

Upscaled maps

Real-time cut-off: 1000

- LRTA* (F, PDB)
- LRTA* (PDB, PDB)
- PR LRTA* (F, G)
- PR LRTA* (F, PDB)
- PR LRTA* (PDB, PDB)
- PR LRTA* (PDB, G)

Suboptimality (times)

Mean number of states expanded per move

Vadim Bulitko & Yngvi Björnsson

July 20, 2009
Empirical Evaluation

Upscaled maps

Real-time cut-off: 1000

Suboptimality (times)

Mean number of states expanded per move
Empirical Evaluation

- Mean time per move (millisecond)
- Mean suboptimality (%)
Empirical Evaluation

Mean suboptimality (%) vs. Mean time per move (millisecond)

- Orange square: High Mean suboptimality, High Mean time
- Black circle: Low Mean suboptimality, Low Mean time
- Black star: Low Mean suboptimality, High Mean time
- White diamond: High Mean suboptimality, Low Mean time
- Black diamond: Medium Mean suboptimality, Medium Mean time

Data points represent different conditions or scenarios.
Empirical Evaluation

Mean suboptimality (%) vs. Mean time per move (millisecond)

Data points represent different algorithms or conditions.
Empirical Evaluation

![Graph showing mean suboptimality (%) vs mean time per move (millisecond)]
Empirical Evaluation

![Graph showing the relationship between mean time per move (millisecond) and mean suboptimality (%).]
Empirical Evaluation

Mean suboptimality (%) vs Mean database size (map sizes)
Empirical Evaluation

Mean suboptimality (%) vs. Mean database size (map sizes)
Empirical Evaluation

![Graph showing Mean suboptimality (%) vs. Mean database size (map sizes).]
Empirical Evaluation

Mean suboptimality (%)

Mean database size (map sizes)
Empirical Evaluation

![Graph showing the relationship between mean suboptimality and mean database size (map sizes). The x-axis represents the mean database size, ranging from 0 to 10, and the y-axis represents the mean suboptimality, ranging from 0 to 150. There are several data points indicated with different markers.](image-url)
Empirical Evaluation

Mean suboptimality (%)

Mean database size (map sizes)
Recap: Ideas

- Reduce the amount of on-line learning
- Pre-compute a database of paths
  - compress each into a series of subgoals
- Use case-based reasoning on-line
Conclusions

- Proposed an algorithm (kNN LRTA*)
  - improving real-time heuristic search...
  - ...via reducing amount of on-line learning...
  - ...via selecting subgoals dynamically
- Simpler than previous state-of-the-art (D LRTA*)
- Better memory requirements
- Similar on-line performance

Subgoals in Reinforcement Learning?