Fast Path-finding through Subgoaling

Vadim Bulitko & Ramon Lawrence

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

- Heuristic search
  - learning in heuristic search
- How to *avoid* on-line learning
Heuristic Search Problems

- search on a finite weighted graph
- goal and start states are known
- heuristic guidance:
  - estimated distance to goal
Pathfinding

- search on a finite weighted graph
- **goal** and **start** states are known
- **heuristic** guidance:
  - estimated distance to goal
A* computes an entire path before taking its first action. It does not scale well with problem size.

Various techniques/tricks are used to mitigate this issue:

- weighted A*
- PRA* “Dragon Age: Origins”
- scalability?

We want true real-time planning:

- time per move is constant-bounded
  - constant is independent of the number of states
Performance Measures

- Planning time per move (under a cap)
  - CPU seconds

- Path quality
  - ratio of path cost to optimal path cost

- Memory (per agent / per all agents on a map)
  - bytes

- Pre-computation time
  - CPU seconds

- 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
The Problem

- Heuristic is inaccurate
- Misleads the agent
- Fixing the heuristic takes a long time

LRTA*. Korf, 1990

Vadim Bulitko & Ramon Lawrence

February 27, 2012
The Problem

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

LRTA*. Korf, 1990
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
HCDPS off-line

- Partition the space into HC regions
  - compress the partitioning (RLE)
  - Compute optimal paths between nearby regions
  - use dynamic programming for distant regions
  - Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
HCDPS off-line

1. Partition the space into HC regions
   - Compress the partitioning (RLE)
2. Compute optimal paths between nearby regions
3. Use dynamic programming for distant regions
4. Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
  - compress the partitioning (RLE)
- Compute optimal paths between nearby regions
  - use dynamic programming for distant regions
- Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
  - Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
Partition the space into HC regions

- compress the partitioning (RLE)

- Compute optimal paths between nearby regions

- Use dynamic programming for distant regions

- Compress paths into subgoals
Partition the space into HC regions

- compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- use dynamic programming for distant regions
- Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
Partition the space into HC regions

- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
Partition the space into HC regions

- compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- use dynamic programming for distant regions
- Compress paths into subgoals
Partition the space into HC regions

- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
Partition the space into HC regions

compress the partitioning (RLE)

Compute optimal paths between nearby regions

use dynamic programming for distant regions

Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
  - use dynamic programming for distant regions
- Compress paths into subgoals
Partition the space into HC regions
compress the partitioning (RLE)
Compute optimal paths between nearby regions
use dynamic programming for distant regions
Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
  - Use dynamic programming for distant regions
- Compress paths into subgoals
Partition the space into HC regions

compress the partitioning (RLE)

Compute optimal paths between nearby regions

use dynamic programming for distant regions

Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
  - Use dynamic programming for distant regions
- Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
  - Use dynamic programming for distant regions
  - Compress paths into subgoals
Partition the space into HC regions

- compress the partitioning (RLE)

Compute optimal paths between nearby regions

- use dynamic programming for distant regions

Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions

Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
HCDPS off-line

- Partition the space into HC regions
- Compress the partitioning (RLE)
- Compute optimal paths between nearby regions
- Use dynamic programming for distant regions
- Compress paths into subgoals
Modern Art?
HCDPS on-line

- Look up the regions
- Retrieve the path
- Follow it with HCing
HCDPS on-line

- Look up the regions
- Retrieve the path
- Follow it with HCing
HCDPS on-line

- Look up the regions
- Retrieve the path
- Follow it with HCing
Look up the regions

Retrieve the path

Follow it with HCing
HCDPS on-line

- Look up the regions
- Retrieve the path
- Follow it with HCing
HCDPS on-line

- Look up the regions
- Retrieve the path
- Follow it with HCing
HCDPS on-line

- Look up the regions
- Retrieve the path
- Follow it with HCing
HCDPS on-line

- Look up the regions
- Retrieve the path
- Follow it with HCing
Look up the regions
Retrieve the path
Follow it with HCing
HCDPS on-line

- Look up the regions
- Retrieve the path
- Follow it with HCing
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
More Theory

- Worst case suboptimality

\[ O(\omega^{-\Omega}) \]

- Special case

\[ O(\Omega^{-\omega}) \]
More Theory

- Worst case suboptimality

\[ O\left(c\frac{\Omega}{\omega}\right) \]

- Special case

\[ O\left(\frac{\Omega}{\omega}\right) \]

region cutoff
More Theory

- Worst case suboptimality

\[ O(\Omega/c\omega) \]

- Special case

\[ O(\Omega/\omega) \]

- max edge cost

- region cutoff
More Theory

- Worst case suboptimality
  \[ O \left( \frac{\Omega}{c \omega} \right) \]
  max edge cost

- Special case
  \[ O \left( \frac{\Omega}{\omega} \right) \]
  min edge cost
  region cutoff
More Theory

- Worst case suboptimality
  \[O \left( c \frac{\Omega}{\omega} \right)\]
  - max edge cost
  - region cutoff

- Special case
  \[O \left( \frac{\Omega}{\omega} \right)\]
  - min edge cost
  - start and goal more than c edges apart
Empirical Evaluation

- Grid-based pathfinding
- 8-connected grids
- 1 and 1.4 costs (cardinal / diagonal)
- 10 maps from Dragon Age: Origins
- 97K states on average
- 1000 problems
- 100 per map
Empirical Evaluation
Empirical Evaluation
Empirical Evaluation
Empirical Evaluation

- HCDPS
- D LRTA*
- kNN LRTA*
- LSS LRTA*
- TBA*
- A*; weighted A*
- BEAM

Lawrence & Bulitko, 2010
Bulitko et al., 2008
Bulitko et al., 2010
Koenig & Sun, 2009
Bjornsson et al., 2009
Hart et al., 1968; Pohl, 1970
Lowerre, 1976
On Line
Sample Comparisons

**HCDPS (L=3, c=25):**

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- Median suboptimality 2.6%
- Max suboptimality 8%
- 0Kb per agent (average)
- Scrubbing 0.006 %
Sample Comparisons

<table>
<thead>
<tr>
<th>HCDPS (L=3, c=25)</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.69 microseconds per move</td>
<td>100 times worse</td>
</tr>
<tr>
<td>0.7 milliseconds per problem</td>
<td>95 times worse</td>
</tr>
<tr>
<td>median suboptimality 2.6%</td>
<td>2.6% better</td>
</tr>
<tr>
<td>max suboptimality 8%</td>
<td>8% better</td>
</tr>
<tr>
<td>0Kb per agent (average)</td>
<td>734 KBytes</td>
</tr>
<tr>
<td>scrubbing 0.006%</td>
<td>0.006% better</td>
</tr>
</tbody>
</table>
Sample Comparisons

**HCDPS (L=3, c=25):**

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- Median suboptimality 2.6%
- Max suboptimality 8%
- 0Kb per agent (average)
- Scrubbing 0.006 %
Sample Comparisons

HCDPS (L=3, c=25):

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- median suboptimality 2.6%
- max suboptimality 8%
- 0Kb per agent (average)
- scrubbing 0.006 %
### Sample Comparisons

<table>
<thead>
<tr>
<th>HCDPS (L=3, c=25):</th>
<th>LSS LRTA* (10K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.69 microseconds per move</td>
<td>207 times worse</td>
</tr>
<tr>
<td>0.7 milliseconds per problem</td>
<td>403 times worse</td>
</tr>
<tr>
<td>median suboptimality 2.6%</td>
<td>16 times worse</td>
</tr>
<tr>
<td>max suboptimality 8%</td>
<td>68 times worse</td>
</tr>
<tr>
<td>0Kb per agent (average)</td>
<td>516 KBytes</td>
</tr>
<tr>
<td>scrubbing 0.006 %</td>
<td>1207 times worse</td>
</tr>
</tbody>
</table>
Sample Comparisons

- **HCDPS (L=3, c=25):**
  - 0.69 microseconds per move
  - 0.7 milliseconds per problem
  - Median suboptimality 2.6%
  - Max suboptimality 8%
  - 0Kb per agent (average)
  - Scrubbing 0.006 %
Sample Comparisons

HCDPS (L=3, c=25):

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- median suboptimality 2.6%
- max suboptimality 8%
- 0Kb per agent (average)
- scrubbing 0.006 %
### Sample Comparisons

<table>
<thead>
<tr>
<th>HCDPS (L=3, c=25):</th>
<th>TBA* (500)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.69 microseconds per move</td>
<td>100 times worse</td>
</tr>
<tr>
<td>0.7 milliseconds per problem</td>
<td>103 times worse</td>
</tr>
<tr>
<td>median suboptimality 2.6%</td>
<td>1.7 times better</td>
</tr>
<tr>
<td>max suboptimality 8%</td>
<td>3.5 times worse</td>
</tr>
<tr>
<td>0Kb per agent (average)</td>
<td>688 KBytes</td>
</tr>
<tr>
<td>scrubbing 0.006 %</td>
<td>448 times worse</td>
</tr>
</tbody>
</table>
Sample Comparisons

HCDPS (L=3, c=25):

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- median suboptimality 2.6%
- max suboptimality 8%
- 0Kb per agent (average)
- scrubbing 0.006 %
Sample Comparisons

HCDPS (L=3, c=25):

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- median suboptimality 2.6%
- max suboptimality 8%
- 0Kb per agent (average)
- scrubbing 0.006 %
Sample Comparisons

<table>
<thead>
<tr>
<th>HCDPS ( (L=3, c=25) ):</th>
<th>D LRTA* ( (5) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.69 microseconds per move</td>
<td>4 times worse</td>
</tr>
<tr>
<td>0.7 milliseconds per problem</td>
<td>58 times worse</td>
</tr>
<tr>
<td>median suboptimality 2.6%</td>
<td>4 times worse</td>
</tr>
<tr>
<td>max suboptimality 8%</td>
<td>3.9K times worse</td>
</tr>
<tr>
<td>0Kb per agent (average)</td>
<td>9.8 KBytes</td>
</tr>
<tr>
<td>scrubbing 0.006 %</td>
<td>16.5K times worse</td>
</tr>
</tbody>
</table>
Sample Comparisons

HCDPS (L=3, c=25):

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- median suboptimality 2.6%
- max suboptimality 8%
- 0Kb per agent (average)
- scrubbing 0.006 %
Sample Comparisons

HCDPS (L=3, c=25):

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- Median suboptimality 2.6%
- Max suboptimality 8%
- 0Kb per agent (average)
- Scrubbing 0.006 %
## Sample Comparisons

<table>
<thead>
<tr>
<th>HCDPS (L=3, c=25):</th>
<th>kNN LRTA* (80K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.69 microseconds per move</td>
<td>46 times worse</td>
</tr>
<tr>
<td>0.7 milliseconds per problem</td>
<td>47 times worse</td>
</tr>
<tr>
<td>median suboptimality 2.6%</td>
<td>2 times better</td>
</tr>
<tr>
<td>max suboptimality 8%</td>
<td>12K times worse</td>
</tr>
<tr>
<td>0Kb per agent (average)</td>
<td>1 KBytes</td>
</tr>
<tr>
<td>scrubbing 0.006 %</td>
<td>10K times worse</td>
</tr>
</tbody>
</table>
Sample Comparisons

HCDPS (L=3, c=25):

- 0.69 microseconds per move
- 0.7 milliseconds per problem
- Median suboptimality 2.6%
- Max suboptimality 8%
- 0Kb per agent (average)
- Scrubbing 0.006 %
Off Line
Sample Comparisons

- HCDPS ($L=3, c=25$):
  - 73K database cases
  - 3.3Mbytes database size
  - 25.2 seconds to build
Sample Comparisons

HCDPS (L=3, c=25):

- 73K database cases
- 3.3Mbytes database size
- 25.2 seconds to build

Average map size: 630.4 Kbytes
5.4 times bigger
## Sample Comparisons

<table>
<thead>
<tr>
<th>HCDPS (L=3, c=25):</th>
<th>D LRTA* (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>73K database cases</td>
<td>1.5 times worse</td>
</tr>
<tr>
<td>3.3Mbytes database size</td>
<td>1.8 times better</td>
</tr>
<tr>
<td>25.2 seconds to build</td>
<td>94 times worse</td>
</tr>
</tbody>
</table>

Average map size: 630.4 Kbytes
5.4 times bigger
Sample Comparisons

HCDPS (L=3, c=25):

- 73K database cases
- 3.3Mbytes database size
- 25.2 seconds to build

Average map size: 630.4 Kbytes
5.4 times bigger
Sample Comparisons

HCDPS (L=3, c=25):

- 73K database cases
- 3.3Mbytes database size
- 25.2 seconds to build
Sample Comparisons

<table>
<thead>
<tr>
<th>HCDPS (L=3, c=25):</th>
<th>kNN LRTA(^*) (80K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>73K database cases</td>
<td>1.1 times worse</td>
</tr>
<tr>
<td>3.3Mbytes database size</td>
<td>2 times worse</td>
</tr>
<tr>
<td>25.2 seconds to build</td>
<td>67 times worse</td>
</tr>
</tbody>
</table>
Sample Comparisons

- HCDPS (L=3, c=25):
  - 73K database cases
  - 3.3Mbytes database size
  - 25.2 seconds to build
## Multi-agent Pathfinding

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory</th>
<th>Break-even agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCDPS (L=3, c=25):</td>
<td>3.3 Mb</td>
<td></td>
</tr>
<tr>
<td>A*</td>
<td>734 Kb</td>
<td>5</td>
</tr>
<tr>
<td>TBA*(500)</td>
<td>688 Kb</td>
<td>5</td>
</tr>
<tr>
<td>LSS LRTA*(10K)</td>
<td>516 Kb</td>
<td>7</td>
</tr>
</tbody>
</table>
Current Limitations

- Requires partitioning of the entire space
- may not be applicable to general planning

- Poor hill-climability excessive number of regions

- Database may be too large for console games
  - unless streamed in
Future Work

- Database parameters
  - number of records
    - how many to guarantee certain suboptimality?
  - placement of records
    - random/non-random? How?
  - selection of the best record on-line
    - optimal selectors?
Conclusions

- A new algorithm
- the coverage of D LRTA*
- hill-climability of kNN LRTA*
- eliminates learning and scrubbing

- On modern video game maps:
  - 0.7 microseconds per move
  - 2.6% median suboptimality
  - 3.3 Mbytes total memory (5.4 times of map size)
  - computed in 25.2 seconds