Recent Developments in Learning Real-time Search

&

Their Applications to Real-time Path-finding

Vadim Bulitko
Acknowledgments

- Pedro Meseguer
- Carlos Hernandez
- Nathan Sturtevant
- IRCL group members
- NSERC & iCORE
- Jonathan Schaeffer
- Rich Korf
- Sven Koenig
Outline

• Motivation
• Incremental Search
• Learning Real-time Search
• State Abstraction
Anecdotal Example

- A robot moves in town
- Daily commute to work
- Unknown:
  - traffic conditions
  - speed traps
  - road construction
- Learns while driving
- Each day is a new trial
Simultaneous Planning & Learning

• We are interested in domains that are:
  • initially unknown
  • “real-time” (thinking matters)

• Examples:
  • RTS games
  • Robotics
Problem Formulation

- path-planning in a game-like environment
- 2D grid
- commercial maps
- initially unknown
- static goal/start of known coordinates
Agent Operation

- **Trial**
  - agent is deployed at the start state
  - takes actions until the goal state is reached

- **Convergence Run**
  - a sequence of trials until no changes in the agent’s heuristic/map are observed
Objectives

• Minimize first-move delay (lag)

• Minimize convergence travel

• Minimize suboptimality of final solution
Incremental Search

- Incremental A*
- Freespace assumption
- A* with Euclidean distance
- Replan when plan fails

- D* [Stenz 95], D* Lite [Koenig/Likhachev 02]
Incremental A*

- First move lag: **2904** nodes
- Convergence travel: **298**
First Move Delay

- Small map from an RPG
- 6176 states
- Shortest path of 115
- Incremental A*
- several orders of magnitude slower than greedy agent

- Important with many units
Real-time Search

• Pioneered by [Korf 90]

• LRTA*
  • greedy with respect to the heuristic function
  • update heuristic on each move

• Complete & optimal (with some fine print)
LRTA* in action

The process of abstracting a graph involves creating an abstraction of the states connected to a clique. Because we only pay this cost when we are making changes that affect the connectivity of the entire map. In practice, a clique can be abstracted of its state abstraction in a single map via its corresponding clique in the abstracted graph. This abstraction is state abstraction in the abstractions of the connected states. In the case of the figure, we were able to demonstrate that because the graph was being studied as a classical abstracting algorithm, we only pay this cost when we are making changes that affect the connectivity of the entire map. When we consider the group of the process of abstracting a graph, we can see that the abstraction of a clique can be abstracted of its state abstraction in a single map via its corresponding clique in the abstracted graph. This abstraction is state abstraction in the abstractions of the connected states. In the case of the figure, we were able to demonstrate that because the graph was being studied as a classical abstracting algorithm, we only pay this cost when we are making changes that affect the connectivity of the entire map.
LRTA* LRTA*

Start

Goal

- First move lag: 6 nodes (484x faster)
- Convergence travel: 47630 (160x slower)
Following

- RTA*/LRTA* [Korf 90]
- MTS [Ishida, Korf 91]
- SRTA* [Russell, Wefald 91]
- DTA*/LDTA* [Russell, Wefald 91]
- SLA* [Shue, Zamani 93]
- RTDP [Barto, et al. 95]
- Weighted LRTA* [Ishida, Shimbo 96]
- Bounded LRTA* [Ishida, Shimbo 96]
- RTBS [Ishida, 97]
- SLA*T [Shue, Zamani 98]
- FALCONS [Furcy, Koenig 00]
- eFALCONS [Furcy, Koenig 01]
- CB-LRTA* [Shang, et al. 03]
- Multi-update LRTA* [Koenig 04]
- Υ-Trap [Bulitko 04]
- LRTS [Bulitko 05]
- LRTA*(k) [Hernandez, Meseguer 05]
Three Extensions

- Deeper lookahead  
  [Korf 90]

- Controlled suboptimality  
  [Shimbo/Ishida 96, Bulitko 04]

- Learning quota, backtracking  
  [Shue/Zamani 93, Shue et al. 01, Bulitko 04]

- Learning Real-time Search (LRTS)

- LRTS(d,ϒ,Τ)  
  [Bulitko 05]
Illustration of lookahead

Figure 4: Initial and final heuristics of LRTA* with the lookahead of 1 (left) and 2 (right).

Both LRTA* with the lookahead of 1 and 2 converge to their final heuristics in one trial. However, the additional ply of the lookahead takes advantage of the two extra heuristic values and, as a result, reduces the execution cost from 5 to 3 moves. On the other hand, the planning cost of each moves increases from 2 to 4 nodes.
## Effects of lookahead

<table>
<thead>
<tr>
<th>$d$</th>
<th>Deliberation per move (ms)</th>
<th>Travel per trial</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0087</td>
<td>661.5</td>
</tr>
<tr>
<td>3</td>
<td>0.0215</td>
<td>241.8</td>
</tr>
<tr>
<td>5</td>
<td>0.0360</td>
<td>193.3</td>
</tr>
<tr>
<td>7</td>
<td>0.0514</td>
<td>114.9</td>
</tr>
<tr>
<td>9</td>
<td>0.0715</td>
<td>105.8</td>
</tr>
</tbody>
</table>
Illustration of gamma

Figure 6: Heuristic values over successive trials of: LRTA* (left) and 1-LRTA* (right).
**Effects of gamma**

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>Suboptimality</th>
<th>Convergence travel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>6.19%</td>
<td>9,300</td>
</tr>
<tr>
<td>0.3</td>
<td>4.92%</td>
<td>8,751</td>
</tr>
<tr>
<td>0.5</td>
<td>2.41%</td>
<td>9,435</td>
</tr>
<tr>
<td>0.7</td>
<td>1.23%</td>
<td>13,862</td>
</tr>
<tr>
<td>0.9</td>
<td>0.20%</td>
<td>25,507</td>
</tr>
<tr>
<td>1.0</td>
<td>0.00%</td>
<td>31,336</td>
</tr>
</tbody>
</table>
**Illustration of backtracking**

Figure 10: Heuristic values over successive trials of: LRTA* (left) and SLA* (right).

The underlying intuition can be illustrated with a simple example in Figure 10. Once again, consider a one-dimensional five-state domain. Each action has the execution cost of one. The initial heuristic is accurate in the left two states and one lower in the right three states. On each trial, LRTA* will raise the heuristic value of a single state. Therefore, three trials (12 moves) are needed to make the heuristic perfect. SLA*, on the other hand, gets to the middle state in 2 moves, updates its value from 1 to 2, backtracks to the second state from the right, increases its value from 2 to 3, backtracks to the right most state, and increases its value from 3 to 4. The agent will then take 4 moves towards the goal, following the now perfect heuristic. As a result, the first trial is longer (8 vs. 4 moves) but the overall number of moves until convergence is reduced (from 12 to 8).
## Effects of T

<table>
<thead>
<tr>
<th>$T$</th>
<th>First trial travel</th>
<th>Convergence travel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>434</td>
<td>457</td>
</tr>
<tr>
<td>10</td>
<td>413</td>
<td>487</td>
</tr>
<tr>
<td>50</td>
<td>398</td>
<td>592</td>
</tr>
<tr>
<td>1,000</td>
<td>390</td>
<td>810</td>
</tr>
<tr>
<td>5,000</td>
<td>235</td>
<td>935</td>
</tr>
</tbody>
</table>
LRTS in action

heuristic update
• First move lag: 789 nodes (132x slower)
• Convergence travel: 1217 (39x faster)
Outstanding Issues

• Slow convergence
• $\Theta(n)$ memory required
• Why?
  • Large state space
  • Tabular representation of heuristic function
• No generalization
Generalization

• heuristic function approximation
  • e.g., ANN with tile-coding

• state aggregation
  • tiling, clique abstraction, triangulation
Clique Abstraction

Figure 30 The process of abstracting a graph: we only pay this cost when we are making changes that affect the connectivity of the entire map. In practice, we rely on abstracted repairs under abstraction. We perform these repairs in a bottom-up manner, removing nodes and edges that are no longer necessary. The abstraction is then built on top of this information. A single repair may result in multiple changes to the graph. These changes can be encoded in a matrix, and in the worst case, the algorithm is O(n^2).
Illustration

- Map from an RTS game
Level 0 (base level)

- 16807 nodes
Level 1

• 5212 nodes
Level 2

• 1919 nodes
Level 3

- 771 nodes
Path Refinement

• When tasked to go from A to B
  • see if there is a previously built path from \texttt{parent}(A) to \texttt{parent}(B)
  • if not then call ourselves recursively to plan from \texttt{parent}(A) to \texttt{parent}(B)
  • otherwise plan from A to B in the corridor
• Dynamically build/repair abstraction while exploring the map
Illustration

- LRTS
- A*
Path Refinement LRTS

• First move lag:
  262 nodes (3x faster)

• Convergence travel:
  619 (2x faster)
Evaluation

• 10 maps from a role-playing video game
• 1000 convergence runs total

• \text{LRTS}(d=10, \gamma=0.5, T=0)

• PR \text{LRTS}
  • \text{A}^* \text{ at level } 0
  • \text{LRTS}(d=5, \gamma=0.5, T=0) \text{ at level } 1
First Move Delay

![Graph showing First Move Delay](image)

- **A***
- **LRTA***
- **LRTS***
- **PR-LRTS***

**First move lag (nodes)**

**Solution Length**
Effects of Abstraction

- PR LRTS over LRTS
  - 40% reduction in convergence travel
  - the same time lag
  - the same suboptimality

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>1st move time</th>
<th>Conv. travel</th>
<th>Suboptimality</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>5.01 ms</td>
<td>186</td>
<td>0.0%</td>
</tr>
<tr>
<td>LRTA*</td>
<td>0.02 ms</td>
<td>25,868</td>
<td>0.0%</td>
</tr>
<tr>
<td>LRTS</td>
<td>0.93 ms</td>
<td>555</td>
<td>2.07%</td>
</tr>
<tr>
<td>PR-LRTS</td>
<td>0.95 ms</td>
<td>345</td>
<td>2.19%</td>
</tr>
</tbody>
</table>
Why Does it Work?

• At higher abstraction levels

• A* spends less time planning

• LRTS has fewer heuristic values to learn

• Each heuristic value learnt at an abstract level affects several ground states

• Search at the lower level is restricted
Summary

- Agent-centered decision-making
- Unknown and “real-time” environments
- Incremental A* learns a model
- Real-time search learns a heuristic function
- We add generalization via state aggregation
- PR LRTS learns both: model + heuristic

http://ircl.cs.ualberta.ca/lrts
LRTA*(k)

- More than one heuristic update per move
- More planning per move
- Faster learning

- Similar manifestations arise with deeper lookahead
Joint Work

• experiments in commercial maps
• comparisons to backtracking in LRTS (SLA*)
• sweet spot analysis
• comparisons to deeper lookahead
• a hybrid algorithm
  • deeper lookahead
  • backpropagation from LRTA*(k)
• backtracking from LRTS/SLA*
LRTA*\((d,k)\) in action
heuristic update

heuristic update

heuristic update

\(d\)
<table>
<thead>
<tr>
<th>k/d</th>
<th>I</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2250</td>
<td>808</td>
<td>345</td>
</tr>
<tr>
<td>3</td>
<td>1309</td>
<td>541</td>
<td>270</td>
</tr>
<tr>
<td>5</td>
<td>1231</td>
<td>541</td>
<td>270</td>
</tr>
<tr>
<td>∞</td>
<td>1231</td>
<td>541</td>
<td>270</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>k/d</th>
<th>I</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14061</td>
<td>31881</td>
<td>18632</td>
</tr>
<tr>
<td>3</td>
<td>10307</td>
<td>30880</td>
<td>21635</td>
</tr>
<tr>
<td>5</td>
<td>9861</td>
<td>31113</td>
<td>21635</td>
</tr>
<tr>
<td>∞</td>
<td>9878</td>
<td>31113</td>
<td>21635</td>
</tr>
</tbody>
</table>