A Guide to Budgeted Tree Search

Nathan R. Sturtevant
University of Alberta
Amii Fellow, CIFAR Chair

Malte Helmert
Universität Basel
Talk Overview

• Budgeted Tree Search (BTS) is a new algorithm with better worst-case guarantees than IDA*
Talk Overview

• Budgeted Tree Search (BTS) is a new algorithm with better worst-case guarantees than IDA*

• Companion work to original paper on IBEX
  (Helmert, Lattimore, Lelis, Orseau, Sturtevant, IJCAI 2019)
Talk Overview

• Budgeted Tree Search (BTS) is a new algorithm with better worst-case guarantees than IDA*

• Companion work to original paper on IBEX
  
  (Helmert, Lattimore, Lelis, Orseau, Sturtevant, IJCAI 2019)

• Why do we need BTS?
Talk Overview

• Budgeted Tree Search (BTS) is a new algorithm with better worst-case guarantees than IDA*

• Companion work to original paper on IBEX
  (Helmert, Lattimore, Lelis, Orseau, Sturtevant, IJCAI 2019)

• Why do we need BTS?
• How does BTS work?
Start


Goal
Start

\[
\begin{array}{ccc}
5 & 4 & 3 \\
2 & 1 & \text{Goal} \\
\end{array}
\]

Goal

\[
\begin{array}{ccc}
3 & 4 & 5 \\
1 & 2 & \text{Goal} \\
\end{array}
\]

\( h = 11 \)
Search Problem

• Given:
Search Problem

• Given:
  • Start state
Search Problem

• Given:
  • Start state
  • Goal state
Search Problem

• Given:
  • Start state
  • Goal state
  • Successor function
Search Problem

- Given:
  - Start state
  - Goal state
  - Successor function
  - Cost function
Search Problem

• Given:
  • Start state
  • Goal state
  • Successor function
  • Cost function

← Defines implicit graph
Search Problem

• Given:
  • Start state
  • Goal state
  • Successor function
  • Cost function
  • Heuristic function

← Defines implicit graph
Search Problem

• Given:
  - Start state
  - Goal state
  - Successor function
  - Cost function
  - Heuristic function

← Defines implicit graph

• Find:
Search Problem

• Given:
  • Start state
  • Goal state
  • Successor function
  • Cost function
  • Heuristic function

← Defines implicit graph

• Find:
  • Optimal path between start/goal
Why do we need BTS?
IDA* Refresher

• IDA* does *iterative deepening* search on $f$-costs
  • $f(n) = g(n) + h(n)$
IDA* Refresher

• IDA* does iterative deepening search on $f$-costs
  • $f(n) = g(n) + h(n)$
• Next iteration $f$-cost:
  • Smallest unexplored from previous iteration
IDA* - Unit Costs
IDA* - Unit Costs
2 States
f-cost 11
IDA* Worst Case
IDA* Worst Case

- $f$-cost layers grow exponentially
IDA* Worst Case

- $f$-cost layers grow exponentially
  - 1
IDA* Worst Case

• $f$-cost layers grow exponentially
  • $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$
IDA* Worst Case

- $f$-cost layers grow exponentially
  - $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$
- What if $f$-cost layers grew linearly?
IDA* Worst Case

• $f$-cost layers grow exponentially
  • $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$

• What if $f$-cost layers grew linearly?
  • $1$
IDA* Worst Case

• $f$-cost layers grow exponentially
  
  $\cdot 1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$

• What if $f$-cost layers grew linearly?
  
  $\cdot 1 + 2 + 3 + 4 + \ldots + b^d \approx (b^d)^2$
IDA* Worst Case

• $f$-cost layers grow exponentially
  • $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$
• What if $f$-cost layers grew linearly?
  • $1 + 2 + 3 + 4 + \ldots + b^d \approx (b^d)^2$

• Happens with non-unit edge costs:
  • STP: Cost of moving tile $t$: \[ \frac{t + 2}{t + 1} \]
IDA* Worst Case

• $f$-cost layers grow exponentially
  
  \[ 1 + b + b^2 + b^3 + \ldots + b^d \approx b^d \]

• What if $f$-cost layers grew linearly?
  
  \[ 1 + 2 + 3 + 4 + \ldots + b^d \approx (b^d)^2 \]

• Happens with non-unit edge costs:

  • STP: Cost of moving tile $t$: \( \frac{t + 2}{t + 1} \)

<table>
<thead>
<tr>
<th>Tile</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \frac{1 + 2}{1 + 1} = 1.5 )</td>
</tr>
<tr>
<td>3</td>
<td>( \frac{3 + 2}{3 + 1} = 1.25 )</td>
</tr>
<tr>
<td>7</td>
<td>( \frac{7 + 2}{7 + 1} = 1.125 )</td>
</tr>
<tr>
<td>9</td>
<td>( \frac{9 + 2}{9 + 1} = 1.1 )</td>
</tr>
</tbody>
</table>
f-cost 11

11.25

13.45

13.62

13.7

13.87

13.5

13.83
Why do we need BTS?

If the nodes in each iteration do not grow exponentially
Getting the next bound
Getting the next bound

• Can be conservative:
  • IDA* (Korf, 1985)
Getting the next bound

• Can be conservative:
  • IDA* (Korf, 1985)

• Can try to build a predictor based on past:
  • IDA*$_{CR}$ (Sarkar et al, 1990)
  • IDA*$_{IM}$ (Burns & Ruml, 2013)
Getting the next bound

• Can be conservative:
  • IDA* (Korf, 1985)

• Can try to build a predictor based on past:
  • IDA*$_{CR}$ (Sarkar et al, 1990)
  • IDA*$_{IM}$ (Burns & Ruml, 2013)

• Can model the state space growth:
  • EDA* (Sharon et al, 2014)
Getting the next bound

- Can be conservative:
  - IDA* (Korf, 1985)
- Can try to build a predictor based on past:
  - IDA*\textsubscript{CR} (Sarkar et al, 1990)
  - IDA*\textsubscript{IM} (Burns & Ruml, 2013)
- Can model the state space growth:
  - EDA* (Sharon et al, 2014)
- Want to guarantee exponential growth in expansions
$f = 11$

1 node
\[ f = 11.25 \]

2 nodes
\( f = 11.25 \)

2 nodes

\([4, 16]\)
$f = 13.97$

11 nodes

[22, 88]
$f = 17.17$

47 nodes
$f = 18.32$

99 nodes
$f = 18.32$
99 nodes
[198, 495]
Exponential Search

• Bentley and Yao, 1976
• Algorithm for searching sorted/unbounded array
Exponential Search

• Bentley and Yao, 1976
• Algorithm for searching sorted/unbounded array
Exponential Search

• Bentley and Yao, 1976
• Algorithm for searching sorted/unbounded array
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array
Exponential Search

• Bentley and Yao, 1976

• Algorithm for searching sorted/unbounded array
Exponential Search

• Bentley and Yao, 1976
• Algorithm for searching sorted/unbounded array
Exponential Search

• Bentley and Yao, 1976
• Algorithm for searching sorted/unbounded array

< < < <
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array
Exponential Search

• Bentley and Yao, 1976
• Algorithm for searching sorted/unbounded array
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array
Exponential Search

• Bentley and Yao, 1976
• Algorithm for searching sorted/unbounded array

Binary Search
Exponential Search

• Bentley and Yao, 1976
• Algorithm for searching sorted/unbounded array

• Running time: $\log(i)$
Nodes and $f$-costs

• Exponential Search:
  • Find *value* in *unbounded sorted array*

• Tree Search:
  • Find *(node expansions)* in *(f-costs)*

• Nodes expansions non-decreasing with *f-cost*
How does BTS work?
Budgeted search

• Like exponential search on f-costs
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \quad \text{vs.} \quad f = \infty \]
Budgeted search

- Like exponential search on f-costs

\[
\begin{align*}
\text{f} &= 10 \\
\text{f} &= 27.2 \ (2\times) \\
\text{f} &= 30.7 \ (8\times) \\
\text{f} &= \infty
\end{align*}
\]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]
\[ n_i = 100 \]
\[ f = 27.2 \ (2\times) \]
\[ f = 30.7 \ (8\times) \]
\[ f = \infty \]
Budgeted search

• Like exponential search on f-costs
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2 \times) \]

\[ f = 30.7 \ (8 \times) \]

\[ f = \infty \]

\[ n_i = 100 \]

200 800
Budgeted search

• Like exponential search on f-costs

\[
\begin{align*}
f &= 10 \\
n_i &= 100 \\
f &= 27.2 \ (2 \times) \\
f &= 30.7 \ (8 \times) \\
f &= \infty
\end{align*}
\]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2\times) \]

\[ f = 30.7 \ (8\times) \]

\[ f = \infty \]

\[ n_i = 100 \]

\[ < 200 \]

\[ 200 \]

\[ 800 \]

\[ \geq 800 \]
Budgeted search

- Like exponential search on f-costs

\[ f = 10 \]
\[ n_i = 100 \]
\[ f = 27.2 \ (2\times) \]
\[ f = 30.7 \ (8\times) \]
\[ f = \infty \]

Budget:
Stop when exceeded
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]
\[ n_i = 100 \]

\[ f = 27.2 \ (2\times) \]
\[ f = 30.7 \ (8\times) \]

Budget:
Stop when exceeded

\[ f = \infty \]
\[ \leq 800 \]
\[ \geq 800 \]
Budgeted search

• Like exponential search on f-costs
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \quad \quad \quad f = \infty \]
Budgeted search

- Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2\times) \]

\[ f = 30.7 \ (8\times) \]

\[ f = \infty \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2\times) \]

\[ f = 30.7 \ (8\times) \]

\[ f = \infty \]
Budgeted search

- Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2x) \]

\[ f = 30.7 \ (8x) \]

\[ f = \infty \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2x) \quad f = 30.7 \ (8x) \]

\[ f = \infty \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2\times) \]

\[ f = 30.7 \ (8\times) \]

\[ f = \infty \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \text{ (2x)} \]

\[ f = 30.7 \text{ (8x)} \]

\[ f = \infty \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2\times) \]

\[ f = 30.7 \ (8\times) \]

\[ f = \infty \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \quad (2\times) \]

\[ f = 30.7 \quad (8\times) \]

\[ f = \infty \]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2\times) \]

\[ f = 30.7 \ (8\times) \]

\[ f = \infty \]
Budgeted search

- Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2\times) \]
\[ f = 30.7 \ (8\times) \]

\[ f = \infty \]

\[ n_i \log(f_i) \]
Budgeted search

• Like exponential search on f-costs

\[
\sum_{i=0}^{i} n_i \log(f_i)
\]
Budgeted search

• Like exponential search on f-costs

\[ f = 10 \]

\[ f = 27.2 \ (2 \times) \]

\[ f = 30.7 \ (8 \times) \]

\[ f = \infty \]

\[ \sum_{i} n_i \log(f_i) < (\sum_{i} n_i) \log(C^*) \approx N \log(C^*) \]
BTS Phases

Find next $f$-cost bound:
BTS Phases

Find next $f$-cost bound:

1. Search with conservative $f$, $\infty$ budget (IDA*)
BTS Phases

Find next $f$-cost bound:
1. Search with conservative $f$, $\infty$ budget (IDA*)
2. Grow $f$ exponentially, constant budget
BTS Phases

Find next $f$-cost bound:

1. Search with conservative $f$, $\infty$ budget (IDA*)
2. Grow $f$ exponentially, constant budget
3. Do binary search on $f$, constant budget
BTS Phases

Find next $f$-cost bound:

1. Search with conservative $f$, $\infty$ budget (IDA*)
2. Grow $f$ exponentially, constant budget
3. Do binary search on $f$, constant budget
f-limit: (13.50+14.45)/2=13.97
nodes: [4,16]
expand: 11
f-limit: \( (13.50 + 14.45)/2 = 13.97 \)

nodes: [4,16]

expand: 11
f-limit: \((13.50 + 14.45) / 2 = 13.97\)

nodes: \([4, 16]\)
expand: 11
f-limit: 14.00
nodes: [22, ∞]
expand: 0
IDA*

f-limit: 14.00
nodes: [22, \infty]
expand: 0
f-limit: 14.00
nodes: [22, \infty]
expand: 0
f-limit: 14.00
nodes: [22, ∞]
expand: 0
f-limit: 14.20 + 2^0 = 15.20
nodes: [22, 88]
expand: 0
f-limit: $14.20 + 2^{0} = 15.20$

nodes: [22, 88]

expand: 0
f-limit: 14.20 + 2^0 = 15.20
nodes: [22, 88]
expand: 0
f-limit: 14.20+2^0=15.20
nodes: [22,88]
expand: 0
f-limit: $16.20 + 2^1 = 18.20$

nodes: [22, 88]

expand: 0
f-limit: 16.20 + 2^1 = 18.20
nodes: [22, 88]
expand: 0
f-limit: \((16.20 + 18.15)/2 = 17.17\)

nodes: [22, 88]

expand: 0
f-limit: \(\frac{16.20+18.15}{2}=17.17\)

nodes: [22, 88]

expand: 0

BIN
f-limit: (16.20 + 18.15)/2 = 17.17
nodes: [22, 88]
expand: 0
How does BTS work?

ID*

With budget

\{ Exponential Search, Binary Search \}
IDA*

BTS*
Conclusions

• BTS reduces worst case of IDA*
  • Same performance as IDA* if tree grows exponentially
• IBEX solves similar problems in different contexts
Conclusions

• BTS reduces worst case of IDA*
  • Same performance as IDA* if tree grows exponentially
• IBEX solves similar problems in different contexts

• Demos & videos online:
  • https://www.movingai.com/SAS/
  • https://www.movingai.com/SAS/BTS/