An Introduction to Contraction Hierarchies

Daniel D. Harabor
Monash University
http://harabor.net/daniel

Joint work with Peter Stuckey

IJCAI Tutorial
2018-07-13
Background: Problem
Many optimal methods exist for such static shortest path problems including for graphs with millions of nodes. Some highlights:

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
<th>XT¹</th>
<th>XM²</th>
<th>Query Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra</td>
<td>Classic</td>
<td>-</td>
<td>-</td>
<td>seconds</td>
</tr>
<tr>
<td>A* (+Euclidean metric)</td>
<td>Classic</td>
<td>-</td>
<td>-</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>ALT (i.e. Landmarks)</td>
<td>Lowerbounds</td>
<td>mins</td>
<td>×10¹ MB</td>
<td>msec</td>
</tr>
<tr>
<td>True Distance Heuristics</td>
<td>Lowerbounds</td>
<td>mins</td>
<td>×10¹ MB</td>
<td>msec</td>
</tr>
<tr>
<td>Geometric Containers</td>
<td>Goal Pruning</td>
<td>hours</td>
<td>×10² MB</td>
<td>&lt; 1 msec</td>
</tr>
<tr>
<td>Arc Flags</td>
<td>Goal Pruning</td>
<td>hours</td>
<td>×10¹ MB</td>
<td>µsec</td>
</tr>
<tr>
<td>Contraction Hierarchies</td>
<td>Abstraction</td>
<td>mins</td>
<td>×10¹ MB</td>
<td>&lt; 1 µsec</td>
</tr>
<tr>
<td>Hub Labels</td>
<td>Oracle</td>
<td>hours</td>
<td>×10³ MB</td>
<td>&lt; 1 µsec</td>
</tr>
<tr>
<td>Compressed Path Databases</td>
<td>Oracle</td>
<td>hours</td>
<td>×10³ MB</td>
<td></td>
</tr>
</tbody>
</table>

¹XT = Extra Time (preprocessing)
²XM = Extra Memory (after preprocessing)
Many optimal methods exist for such static shortest path problems including for graphs with millions of nodes. Some highlights:

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
<th>XT(^1)</th>
<th>XM(^2)</th>
<th>Query Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra</td>
<td>Classic</td>
<td>-</td>
<td>-</td>
<td>seconds</td>
</tr>
<tr>
<td>A* (+Euclidean metric)</td>
<td>Classic</td>
<td>-</td>
<td>-</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>ALT (i.e. Landmarks)</td>
<td>Lowerbounds</td>
<td>mins</td>
<td>×10(^1) MB</td>
<td>msec</td>
</tr>
<tr>
<td>True Distance Heuristics</td>
<td>Lowerbounds</td>
<td>mins</td>
<td>×10(^1) MB</td>
<td>msec</td>
</tr>
<tr>
<td>Geometric Containers</td>
<td>Goal Pruning</td>
<td>hours</td>
<td>×10(^1) MB</td>
<td>msec</td>
</tr>
<tr>
<td>Arc Flags</td>
<td>Goal Pruning</td>
<td>hours</td>
<td>×10(^2) MB</td>
<td>&lt; 1 msec</td>
</tr>
<tr>
<td>Contraction Hierarchies</td>
<td>Abstraction</td>
<td>mins</td>
<td>×10(^1) MB</td>
<td>μsec</td>
</tr>
<tr>
<td>Hub Labels</td>
<td>Oracle</td>
<td>hours</td>
<td>×10(^3) MB</td>
<td>&lt; 1 μsec</td>
</tr>
<tr>
<td>Compressed Path Databases</td>
<td>Oracle</td>
<td>hours</td>
<td>×10(^3) MB</td>
<td>&lt; 1 μsec</td>
</tr>
</tbody>
</table>

\(^1\)XT = Extra Time (preprocessing)

\(^2\)XM = Extra Memory (after preprocessing)
Definition #1 (Algorithmics perspective)

Contraction Hierarchies is a graph augmentation / overlay technique that helps to speed up pathfinding search.
Definition #1 (Algorithmics perspective)
Contraction Hierarchies is a graph augmentation / overlay technique that helps to speed up pathfinding search.

Definition #2 (Heuristic Search perspective)
Contraction Hierarchies is an optimality-preserving abstraction technique where macro edges are embedded in the original graph.
Contraction Hierarchies: Some Definitions

Definition #1 (Algorithmics perspective)
Contraction Hierarchies is a graph augmentation / overlay technique that helps to speed up pathfinding search.

Definition #2 (Heuristic Search perspective)
Contraction Hierarchies is an optimality-preserving abstraction technique where macro edges are embedded in the original graph.

Definition #3 (Practitioner’s perspective)
1. (Offline) Add “shortcut edges” between selected pairs of non-adjacent nodes.
2. (Online) Exploit shortcuts to reach the target sooner.
Contraction Hierarchies: Example

![Graph Example]
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Contraction Hierarchies: Example
Common issues that arise when building a contraction hierarchy

1. How to order the nodes for contraction?
2. What criteria to use for adding shortcuts?
3. How to search the resulting hierarchy?
Issue 1: How to order the nodes for contraction

Rules of thumb:

- A “good” node ordering allows a search to reach the topmost node in the hierarchy in a logarithmic number of steps.
- A “bad” node ordering requires a linear number of steps.
- Contract everywhere instead of always from the same place.
Issue 1: How to order the nodes for contraction

Rules of thumb:
- A “good” node ordering allows a search to reach the topmost node in the hierarchy in a logarithmic number of steps.
- A “bad” node ordering requires a linear number of steps.
- Contract everywhere instead of always from the same place.
Issue 1: How to order the nodes for contraction

Rules of thumb:
- A “good” node ordering allows a search to reach the topmost node in the hierarchy in a logarithmic number of steps.
- A “bad” node ordering requires a linear number of steps.
- Contract everywhere instead of always from the same place.
Issue 1: How to order the nodes for contraction

Rules of thumb:

- A “good” node ordering allows a search to reach the topmost node in the hierarchy in a logarithmic number of steps.
- A “bad” node ordering requires a linear number of steps.
- Contract everywhere instead of always from the same place.
Issue 1: How to order the nodes for contraction

Rules of thumb:
- A “good” node ordering allows a search to reach the topmost node in the hierarchy in a logarithmic number of steps.
- A “bad” node ordering requires a linear number of steps.
- Contract everywhere instead of always from the same place.

Greedy orderings
Lazily maintained priorities decide which node is contracted next. Some heuristics (lower values means contract sooner):
- Edge difference
- Hop count
- Voronoi region size
- Other single heuristics and also weighted combinations.

More on greedy heuristics:  [Geisberger et al., 2008].
Theoretical results:  [Bauer et al., 2013; Strasser and Wagner, 2015].
Issue 2: What criteria to use for adding shortcuts?

Contraction

- Contracting a node means adding shortcuts between pairs of neighbours that are “more important” (i.e. higher ranked).

Shortcuts are added when they help to maintain some specific graph property. Some interesting examples:

- Cost optimality [Geisberger et al., 2008]
- Freespace reachability [Uras and Koenig, 2018]
- Something else (you decide)

How to decide if a shortcut is strictly necessary?

- Naive: add all possible shortcuts.
- Strict: invoke Dijkstra and perform a "witness search" proof.
- Pragmatic: invoke Dijkstra with cutoffs (max cost, max hops).
Issue 2: What criteria to use for adding shortcuts?

**Contraction**

- Contracting a node means adding shortcuts between pairs of neighbours that are “more important” (i.e. higher ranked).
- Shortcuts are added when they help to maintain some specific graph property. Some interesting examples:
  - Cost optimality [Geisberger et al., 2008]
  - Freespace reachability [Uras and Koenig, 2018]
  - Something else (you decide)

**How to decide if a shortcut is strictly necessary?**

- Naive: add all possible shortcuts.
- Strict: invoke Dijkstra and perform a "witness search" proof.
- Pragmatic: invoke Dijkstra with cutoffs (max cost, max hops).
Issue 2: What criteria to use for adding shortcuts?

**Contraction**
- Contracting a node means adding shortcuts between pairs of neighbours that are “more important” (i.e. higher ranked).
- Shortcuts are added when they help to maintain some specific graph property. Some interesting examples:
  - Cost optimality [Geisberger et al., 2008]
  - Freespace reachability [Uras and Koenig, 2018]
  - Something else (you decide)

**How to decide if a shortcut is strictly necessary?**
- Naive: add all possible shortcuts.
- Strict: invoke Dijkstra and perform a “witness search” proof.
- Pragmatic: invoke Dijkstra with cutoffs (max cost, max hops).
Existing literature

Contraction Hierarchies are typically combined with some variant of bi-directional Dijkstra search.

- Bi-directional search (two directions simultaneously)
  - Technical descriptions in [Batz et al., 2009; Storandt, 2013]
- Bi-directional search (one direction at a time)
  - Technical descriptions in [Bauer et al., 2010]
- Hybrid search (bi-directional first, then something else)
Issue #3: How to search the contraction hierarchy?

**Existing literature**

Contraction Hierarchies are typically combined with some variant of bi-directional Dijkstra search.
- Bi-directional search (two directions simultaneously)
- Bi-directional search (one direction at a time)
  - Technical descriptions in [Batz et al., 2009; Storandt, 2013]
- Hybrid search (bi-directional first, then something else)
  - Technical descriptions in [Bauer et al., 2010]

**Cool new stuff!**

Contraction hierarchies can also be combined with your favourite uni-directional search scheme (since 2018!)
- Technical descriptions in [Harabor and Stuckey, 2018]
The BCH Query Algorithm

BCH = Bi-directional Dijkstra Search + Contraction Hierarchies. Introduced in [Geisberger et al., 2008; Geisberger et al., 2012].

Offline

Divide the contracted graph $G = (V, E)$ into two:

- $G_{\uparrow} = (V, E_{\uparrow} = \{(u, v) \in E \mid u \prec v\})$
- $G_{\downarrow} = (V, E_{\downarrow} = \{(u, v) \in E \mid u \succ v\})$
- $\prec$ and $\succ$ compare the contraction order of pairs of nodes.

Online

Perform a bi-directional Dijkstra search:

- Forward search in $G_{\uparrow}$ (relax only outgoing “up” edges).
- Backward search in $G_{\downarrow}$; (relax only incoming “up” edges).
- Expansions can be interleaved or sequential.
BCH Example

S = \{0, \text{Inf}\}

T = \{\text{Inf, 0}\}
BCH Example

S = \{0, \text{Inf}\}

T = \{\text{Inf, 0}\}

\{\text{Inf, 2}\}

\{2, \text{Inf}\}

\begin{align*}
2 & \quad 2 & \quad 2 & \quad 2 & \quad 2 & \quad 2 & \quad 2 & \quad 2
\end{align*}
BCH Example

$S = \{0, \text{Inf}\}$

$T = \{\text{Inf}, 0\}$

$\{\text{Inf}, 2\} \quad \{2, \text{Inf}\} \quad \{4, 2\} \quad \{\text{Inf}, 2\}$
The following results appear in [Geisberger et al., 2008]:

<table>
<thead>
<tr>
<th>ch-path</th>
</tr>
</thead>
<tbody>
<tr>
<td>For every <em>optimal path</em> in $G$ there exists a cost equivalent path with prefix $\langle s, ..., k \rangle$ found in $G^\uparrow$ and suffix $\langle k, ..., t \rangle$ found in $G^\downarrow$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>apex-node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every <em>ch-path</em> has a node which is lexically largest among all nodes in the path.</td>
</tr>
</tbody>
</table>
CH-SPSP [Harabor and Stuckey, 2018]

Find a path $\langle s = v_1, \ldots, v_k, \ldots v_n = t \rangle$ where

$$\min \left( n-1 \right) \sum_{i=1}^{(n-1)} c_{v_i,v_{i+1}}$$

Subject to:

1. $v_i < v_{i+1}$ for all $1 \leq i < k$
2. $v_i \succeq v_{i+1}$ for all $k \leq i < n$
Graph Traversal Policies

<table>
<thead>
<tr>
<th>Successor Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. up-up successors</td>
</tr>
<tr>
<td>2. up-down successors</td>
</tr>
<tr>
<td>3. down-down successors</td>
</tr>
<tr>
<td>4. down-up successors</td>
</tr>
</tbody>
</table>
## Successor Types

1. up-up successors
2. up-down successors
3. down-down successors
4. down-up successors
## Graph Traversal Policies

### Successor Types

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>up-up successors</td>
</tr>
<tr>
<td>2</td>
<td>up-down successors</td>
</tr>
<tr>
<td>3</td>
<td>down-down successors</td>
</tr>
<tr>
<td>4</td>
<td>down-up successors</td>
</tr>
</tbody>
</table>

### Traversal Strategies

- **Always Up**: expand successors of type 1 and 3. This strategy is employed by the bi-directional algorithm **BCH**.
- **Up-then-Down**: expand successors of type 1, 2 and 3. We combine this strategy with A* search to derive the uni-directional query algorithm **FCH**.
The FCH Query Algorithm

FCH is an variation on A* search which computes only optimal *ch-paths*. Query performance is similar to plain A* (seconds).

FCH can be easily and effectively combined with many standard pruning methods including Bounding Boxes. This algorithm:

- Reasons about the target in relation to the current node (e.g. could *this* edge appear on an optimal path?)
- Requires preprocessing (i.e. extra time and extra memory).
- Is a type of Geometric Container [Wagner et al., 2005]
S = \{0\}
FCH+BB Example

S = \{0\}
FCH+BB Example

S = \{0\}

\[ \begin{array}{cccc}
  1 & 10 & 9 & 11 \\
  2 & 2 & 2 & 2 \\
  7 & 8 & 3 & 5 \\
  6 & 2 & 4 & \\
\end{array} \]
FCH+BB Example
FCH+BB Example

S = \{0\}

\{2\}

---

16 / 80
FCH+BB Example

$S = \{0\}$

![Graph example image](image_url)
FCH+BB Example

S = \{0\}

\{2\}

\{4\}

\begin{align*}
S &= \{0\} \\
\{2\} &= \{2\} \\
\{4\} &= \{4\}
\end{align*}
FCH+BB Example

S = \{0\}

\begin{align*}
\{2\} & \rightarrow 10 \\
\{4\} & \rightarrow 11
\end{align*}
$S = \{0\} \quad \{2\} \quad \{4\}$
FCH+BB Example

S = \{0\}

\{2\}

\{4\}

\begin{align*}
S &= \{0\} \\
\{2\} \\
\{4\}
\end{align*}
S = {0} {2} {4}

FCH+BB Example
FCH+BB Example
## Some Experimental Results

### Benchmarks

| Graph | \(|V|\) | \(#\text{Input}\) | \(#\text{Shortcuts}\) | Total |
|-------|--------|----------------|----------------|-------|
| NY-d  | 264346 | 733846        | 920078        | 1653924 |
| BAY-d | 321270 | 800172        | 808952        | 1609124 |
| COL-d | 435666 | 1057066       | 1062850       | 2119916 |
| FLA-d | 1070376| 2712798       | 2697836       | 5410634 |

### Setup

- 1000 instances for each map, 5 runs per instance.
- MacBook Pro 13,2 machine (16GB RAM, OSX 10.12.6).
- From-scratch C++ implementations of all algorithms.
- [https://bitbucket.org/dharabor/pathfinding](https://bitbucket.org/dharabor/pathfinding)
Comparisons

Uni-directional variants:
- FCH+BB (DFS) — Very light pre-proc. (≈1 second)
- FCH+BB (Dijk 1%) – Light pre-proc. (minutes)
- FCH+BB (Dijk 10%) – Moderate pre-proc. (< 1 hour)
- FCH+BB (Dijk 100%) – Extensive pre-proc. (many hours)

Bi-directional variants:
- BCH
- BCH+BB (Dijk 100%) – Extensive pre-proc. (many hours)
**Comparisons vs BCH**

**Experiment:** Time speedup per search (higher is better).

<table>
<thead>
<tr>
<th>Problem Instance</th>
<th>Improvement Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>4000</td>
</tr>
</tbody>
</table>

- FCH+BB (Dijk 100%)
- FCH+BB (Dijk 10%)
- FCH+BB (Dijk 1%)
- FCH+BB (DFS)
Experiment: Time speedup per search (higher is better).

[Comparison graph showing search time vs BCH+BB for different problem instances and improvement factors.]
Upcoming Talk:
D. Harabor & P. Stuckey
Forward Search in Contraction Hierarchies.
Saturday 14 July @ 09:30. SoCS 2018.
EOF
Optimal Any-angle Pathfinding

Daniel D. Harabor
Monash University
http://harabor.net/daniel

Joint work with:
Vural Aksakalli, Michael Cui, Alban Grastien and Dindar Öz

IJCAI Tutorial
2018-07-13
Theta* [Nash et al., 2007] proceeds from one grid vertex to the next. This strategy is suboptimal since it expands nodes out of order.
Suboptimality and Theta*

Theta* [Nash et al., 2007] proceeds from one grid vertex to the next. This strategy is suboptimal since it expands nodes out of order.
Theta* [Nash et al., 2007] proceeds from one grid vertex to the next. This strategy is suboptimal since it expands nodes out of order.
Suboptimality and Theta*

Theta* [Nash et al., 2007] proceeds from one grid vertex to the next. This strategy is suboptimal since it expands nodes out of order.
Theta* [Nash et al., 2007] proceeds from one grid vertex to the next. This strategy is suboptimal since it expands nodes out of order.
Intuition

Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.
Intuition

Expand sets of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.
Intuition

Expand sets of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.
Anyway: An optimal any-angle pathfinding technique

**Intuition**

Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.
Anya: An optimal any-angle pathfinding technique

**Intuition**

Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.

![Diagram](attachment:image.png)
Anya: An optimal any-angle pathfinding technique

**Intuition**

Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.
Definition #1: Search Nodes

Every node is a tuple \((l, r)\) where:

- \(r\) is a *root*; the most recent turning point.
- \(l\) is an interval of contiguous points, all visible from \(r\).
- The *start node* has a point interval and a root “off the grid”
Definition #2: Successors

- Successors of node \((I, r)\) are found by traveling from \(r\) and through \(I\) along a locally taut path.
- Two kinds of successors: observable and non-observable
Definition #2: Successors

- Successors of node \((I, r)\) are found by traveling from \(r\) and through \(I\) along a locally taut path.
- Two kinds of successors: *observable* and *non-observable*
Definition #2: Successors

- Successors of node \((I, r)\) are found by traveling from \(r\) and through \(I\) along a locally taut path.
- Two kinds of successors: *observable* and *non-observable*
Definition #2: Successors

- Successors of node \((I, r)\) are found by traveling from \(r\) and through \(I\) along a locally taut path.
- Two kinds of successors: observable and non-observable

\[I = [D1, D4] \quad r = [F1] \quad (r, [B1, B5]) \quad r' = [C4] \quad (r', [B5, B8])\]
Definition #2: Successors

- Successors of node \((I, r)\) are found by traveling from \(r\) and through \(I\) along a locally taut path.
- Two kinds of successors: observable and non-observable
From each interval \( I \) we choose a single point \( p \) which minimises the cost-to-go (i.e. the \( f \)-value of the node).
From each interval $I$ we choose a single point $p$ which minimises the cost-to-go (i.e. the $f$-value of the node).
From each interval $I$ we choose a single point $p$ which minimises the cost-to-go (i.e. the $f$-value of the node).
Evaluation Function

From each interval $I$ we choose a single point $p$ which minimises the cost-to-go (i.e. the $f$-value of the node).
Theoretical properties

Completeness (Sketch)
- Every point is a corner or belongs to an interval.
- Every interval is visible from some predecessor.

Optimality (Sketch)
- Each representative point has a minimum $f$-value.
- The $f$-value of each successor is monotonically increasing.
- A node whose interval contains the target is eventually expanded.

Online
Each search is performed entirely online and without reference to any pre-computed data structures or heuristics.

Full technical details in [Harabor et al., 2016].
Results on Games Maps

Speedup vs grid A* on a range of benchmarks from real games (all maps and instances from Nathan’s repository).
Results on Games Maps

Speedup vs grid A* on a range of benchmarks from real games (all maps and instances from Nathan’s repository).

![Graph showing speedup vs grid A* for Dragon Age Origins](image-url)
Results on Games Maps

Speedup vs grid A* on a range of benchmarks from real games (all maps and instances from Nathan’s repository).
Euclidean Shortest Path Problems in 2D

2D ESPP
Find a shortest path among a set of polygonal obstacles.
Navigation Mesh

A partitioning of the traversable space into a collection of convex polygons. Cheap to build. Common in many application areas.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
From Any-angle Pathfinding to ESPP

Polyanya

Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
From Any-angle Pathfinding to ESPP

Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
From Any-angle Pathfinding to ESPP

Polyanya [Cui et al., 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.
(More) Results on Games Maps

Speedup vs grid A* on a range of benchmarks from real games (all maps and instances from Nathan’s repository).
Speedup vs grid A* on a range of benchmarks from real games (all maps and instances from Nathan’s repository).
(More) Results on Games Maps

Speedup vs grid A* on a range of benchmarks from real games (all maps and instances from Nathan’s repository).
Upcoming Talk:
Shizhe Zhao, David Taniar and Daniel Harabor
Fast k-Nearest Neighbour on a Navigation Mesh.
Saturday 14 July @ 09:45. SoCS 2018.
EOF
An Introduction to Compressed Path Databases

Daniel D. Harabor
Monash University
http://harabor.net/daniel

Joint work with:
Jorge Baier, Adi Botea, Alfonso Gerevini, Carlos Hérnandez,
Alessandro Saeti and Ben Strasser

IJCAI Tutorial
2018-07-13
The Shortest Path Problem (and its variants)

Several related flavors

1. Compute a *sequence of edges* forming a shortest path
2. Compute the *distance* of a shortest path
3. Compute a first edge of a shortest path (also called *first move*)

- Variants 2 & 3 are not only first steps to solve variant 1!
- Consider for example a driver following turn-by-turn directions or game unit chasing a moving target. Both scenarios are better captured by variant 3 than variant 1.

We focus on first-move queries (i.e. variant 3)
Solving first-move queries

Textbook solutions

- Run one search per query (e.g. using your favourite algorithm).
- This often is too slow
- Preprocessing is a popular way of increasing the speed when answering shortest-path queries
### Solving first-move queries

**Textbook solutions**
- Run one search per query (e.g. using your favourite algorithm).
- This often is too slow
- Preprocessing is a popular way of increasing the speed when answering shortest-path queries

**CPD approach**
- Build an oracle called a compressed path database (CPD)
- Use the CPD to perform route planning
- The oracle provides optimal moves fast
- It relies on all-pairs pre-processing step
- It requires additional memory to store the pre-processing results
What is a Compressed Path Database (CPD)?

- Given a graph...
What is a Compressed Path Database (CPD)?

- Given a graph...
What is a Compressed Path Database (CPD)?

- Given a graph...
- and *any* two nodes $s$ and $t$...
What is a Compressed Path Database (CPD)?

- Given a graph...
- and any two nodes $s$ and $t$...
- A CPD provides the **first edge** of a shortest path from $s$ to $t$
What is a Compressed Path Database (CPD)?

- Given a graph...
- and any two nodes $s$ and $t$...
- A CPD provides the first edge of a shortest path from $s$ to $t$
- E.g., $\text{CPD}[4, 3] = c$

![Graph Diagram](image-url)
What is a Compressed Path Database (CPD)?

- Thus, a CPD is an all-pairs shortest paths (APSPs) oracle

- Having its data compressed, as naive encodings of APSP data are prohibitively large

- Achieved a state-of-the-art speed performance
  - For problems with static targets...
    - Top performers in two Grid-based Path Planning Competitions
      - http://movingai.com/GPPC/
    - and with moving targets [Botea et al., 2013; Baier et al., 2015; Xie et al., 2017]
Preprocessing

1. Run Dijkstra repeatedly, once for each node
2. After each Dijkstra search, store all the first-move data
### Preprocessing

1. Run Dijkstra repeatedly, once for each node
2. After each Dijkstra search, store all the first-move data

### Storage

A naive encoding of APSP data is prohibitively large (often more than the size of available memory). We need to reduce the size, but:

- Without losing any information (i.e. lossless compression) and,
- While maintaining fast lookup performance
Building a CPD

Preprocessing

1. Run Dijkstra repeatedly, once for each node
2. After each Dijkstra search, store all the first-move data

Storage

A naive encoding of APSP data is prohibitively large (often more than the size of available memory). We need to reduce the size, but:
- Without losing any information (i.e. lossless compression) and,
- While maintaining fast lookup performance

Compression Schemes

Some approaches that appear in the literature:
- Quad-tree decomposition [Sankaranarayanan et al., 2005]
- Rectangle decomposition [Botea and Harabor, 2013]
- Run-length encoding [Strasser et al., 2015]
Compression using Run-length Encoding

**Algorithmic Sketch**
- **Input:** A weighted graph
Compression using Run-length Encoding

**Algorithmic Sketch**

- Input: A weighted graph
- Compute a first-move matrix
Compression using Run-length Encoding

Algorithmic Sketch

- Input: A weighted graph
- Compute a first-move matrix
- Run-length encode every row

SRC, a system based on these ideas, was the fastest optimal solver in the 2014 Grid-Based Path Planning Competition [Sturtevant et al., 2015].

Full technical details in [Strasser et al., 2014; Strasser et al., 2015].
Compression using Run-length Encoding

Algorithmic Sketch

- Input: A weighted graph
- Compute a first-move matrix
- Run-length encode every row
- First-move queries answered using a binary search
Compression using Run-length Encoding

**Algorithmic Sketch**

- Input: A weighted graph
- Compute a first-move matrix
- Run-length encode every row
- First-move queries answered using a binary search

*SRC*, a system based on these ideas, was the fastest optimal solver in the 2014 Grid-Based Path Planning Competition [Sturtevant et al., 2015].

Full technical details in [Strasser et al., 2014; Strasser et al., 2015].
First-move matrix

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
First-move matrix
First-move matrix

\[
\begin{array}{c|cccc}
    & 1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 & 5 \\
4 & 6 & 7 & & \\
\end{array}
\]
First-move matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>e</td>
<td>e</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>2</td>
<td>w</td>
<td>*</td>
<td>e</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
</tr>
<tr>
<td>3</td>
<td>w</td>
<td>w</td>
<td>*</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>4</td>
<td>n</td>
<td>ne</td>
<td>ne</td>
<td>*</td>
<td>s</td>
<td>se</td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>*</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>w</td>
<td>*</td>
</tr>
<tr>
<td>7</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
</tbody>
</table>
```
Compressing first-matrix rows

- First-matrix rows are compressed with run-length encoding (RLE)

- *Runs* are repetitions of the same token
  - E.g., the string `ssnnsss` has three runs: `ss`; `nn`; `sss`
Compressing first-matrix rows

- First-matrix rows are compressed with run-length encoding (RLE)

- *Runs* are repetitions of the same token
  - E.g., the string `ssnnssss` has three runs: *ss*; *nn*; *sss*

<table>
<thead>
<tr>
<th></th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>2</td>
<td>w</td>
</tr>
<tr>
<td>3</td>
<td>w</td>
</tr>
<tr>
<td>4</td>
<td>n</td>
</tr>
<tr>
<td>5</td>
<td>n</td>
</tr>
<tr>
<td>6</td>
<td>nw</td>
</tr>
<tr>
<td>7</td>
<td>w</td>
</tr>
</tbody>
</table>

Uncompressed matrix has 49 tokens
Compressing first-matrix rows

- First-matrix rows are compressed with run-length encoding (RLE)
- **Runs** are repetitions of the same token
  - E.g., the string \texttt{ssnssss} has three runs: **ss; nn; sss**

<table>
<thead>
<tr>
<th>( s )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>e</td>
<td>e</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>2</td>
<td>w</td>
<td>*</td>
<td>e</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
</tr>
<tr>
<td>3</td>
<td>w</td>
<td>w</td>
<td>*</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>4</td>
<td>n</td>
<td>ne</td>
<td>ne</td>
<td>*</td>
<td>s</td>
<td>se</td>
<td>se</td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>*</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>w</td>
<td>*</td>
<td>e</td>
</tr>
<tr>
<td>7</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>*</td>
</tr>
</tbody>
</table>

Uncompressed matrix has 49 tokens

CPD has 16 runs
Non-Unique Shortest Paths?

Roads
- On roads shortest paths are very often unique.

Game Maps
- Game maps often contain unit grids with highly non-unique shortest paths.
- **Idea**: Tie-break paths such that compression is maximised
For every row compute all first moves.
(Simple extension of Dijkstra’s algorithm)
Greedily grow runs from the left to right.
Tie-Breaking

Greedily grow runs from the left to right.
Tie-Breaking

Greedily grow runs from the left to right.
Greedily grow runs from the left to right.
Greedily grow runs from the left to right.
This algorithm produces a minimum number of runs.
The column ordering matters

```
<table>
<thead>
<tr>
<th>s</th>
<th>1   2   3   4   5   6   7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*   e   e   s   s   s   s   s</td>
</tr>
<tr>
<td>2</td>
<td>w   *   e   sw  sw  sw  sw  sw</td>
</tr>
<tr>
<td>3</td>
<td>w   w   *   w   w   w   w   w</td>
</tr>
<tr>
<td>4</td>
<td>n   n   ne  ne  *   s   se  se</td>
</tr>
<tr>
<td>5</td>
<td>n   n   n   n   n   *   e   e</td>
</tr>
<tr>
<td>6</td>
<td>nw  nw  nw  nw  w   *   e</td>
</tr>
<tr>
<td>7</td>
<td>w   w   w   w   w   w   w   *</td>
</tr>
</tbody>
</table>
```

Column ordering: 1, 2, 3, 4, 5, 6, 7

```
<table>
<thead>
<tr>
<th>t</th>
<th>1/e 4/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/e 4/s</td>
</tr>
<tr>
<td>2</td>
<td>1/w 3/e 4/sw</td>
</tr>
<tr>
<td>3</td>
<td>1/w</td>
</tr>
<tr>
<td>4</td>
<td>1/n 2/ne 5/s 6/se</td>
</tr>
<tr>
<td>5</td>
<td>1/n 6/e</td>
</tr>
<tr>
<td>6</td>
<td>1/nw 5/w 7/e</td>
</tr>
<tr>
<td>7</td>
<td>1/w</td>
</tr>
</tbody>
</table>
```

CPD has 16 runs
The column ordering matters

<table>
<thead>
<tr>
<th>s</th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>e</td>
<td>e</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>2</td>
<td>w</td>
<td>*</td>
<td>e</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
</tr>
<tr>
<td>3</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>4</td>
<td>n</td>
<td>ne</td>
<td>*</td>
<td>s</td>
<td>se</td>
<td>se</td>
<td>se</td>
<td>se</td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>*</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>w</td>
<td>*</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>7</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>*</td>
</tr>
</tbody>
</table>

Column ordering: 1, 2, 3, 4, 5, 6, 7

CPD has 16 runs

1  1/e  4/s
2  1/w  3/e  4/sw
3  1/w
4  1/n  2/ne  5/s  6/se
5  1/n  6/e
6  1/nw  5/w  7/e
7  1/w

<table>
<thead>
<tr>
<th>s</th>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>e</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>2</td>
<td>w</td>
<td>*</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
<td>sw</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>3</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>4</td>
<td>n</td>
<td>ne</td>
<td>*</td>
<td>s</td>
<td>se</td>
<td>se</td>
<td>se</td>
<td>se</td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>*</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>nw</td>
<td>w</td>
<td>*</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>7</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>*</td>
<td>w</td>
</tr>
</tbody>
</table>

Column ordering: 1, 2, 4, 5, 6, 7, 3

CPD has 20 runs

1  1/e  3/s  7/e
2  1/w  3/sw  7/e
3  1/w
4  1/n  2/ne  4/s  5/se  7/e
5  1/n  5/e  7/n
6  1/nw  4/w  6/e  7/nw
7  1/w
Ordering the matrix columns (nodes)

The bad news

Finding an optimal column ordering is NP-complete (for technical details see [Botea et al., 2015]).
Ordering the matrix columns (nodes)

**The bad news**

Finding an optimal column ordering is NP-complete (for technical details see [Botea et al., 2015]).

**The good news**

Effective orderings exist which are easy to compute. Two examples:

- **DFS heuristic**
  - **Observation:** in sparse graphs many neighbouring nodes can have close ids.
  - **Idea:** Label nodes with ids using a DFS pre-order traversal from a random starting node.

- **CUT heuristic (Nested-Edge-Cut-Order)**
  - **Observation:** for some edges the endpoints must be far away
  - **Idea:** find a small edge cut, that for which we can violate the closeness requirement
<table>
<thead>
<tr>
<th>Entry</th>
<th>Averaged Query Time over All Test Paths ($\mu s$)</th>
<th>Preprocessing Requirements</th>
<th>SRC-dfs, SRC-dfs-i: two versions of our implemented program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slowest Move in Path</td>
<td>First 20 Moves of path</td>
<td>Full Path Extraction</td>
</tr>
<tr>
<td>\textsuperscript{1}RA*</td>
<td>282995</td>
<td>282995</td>
<td>282995</td>
</tr>
<tr>
<td>BLJPS</td>
<td>14453</td>
<td>14453</td>
<td>14453</td>
</tr>
<tr>
<td>JPS+</td>
<td>7732</td>
<td>7732</td>
<td>7732</td>
</tr>
<tr>
<td>BLJPS2</td>
<td>7444</td>
<td>7444</td>
<td>7444</td>
</tr>
<tr>
<td>\textsuperscript{1}RA*-Subgoal</td>
<td>1688</td>
<td>1688</td>
<td>1688</td>
</tr>
<tr>
<td>JPS+ Bucket</td>
<td>1616</td>
<td>1616</td>
<td>1616</td>
</tr>
<tr>
<td>BLJPS2 Sub</td>
<td>1571</td>
<td>1571</td>
<td>1571</td>
</tr>
<tr>
<td>NSubgoal</td>
<td>773</td>
<td>773</td>
<td>773</td>
</tr>
<tr>
<td>CH</td>
<td>362</td>
<td>362</td>
<td>362</td>
</tr>
<tr>
<td>SRC-dfs</td>
<td>145</td>
<td>145</td>
<td>145</td>
</tr>
<tr>
<td>SRC-dfs-i</td>
<td>1</td>
<td>4</td>
<td>189</td>
</tr>
</tbody>
</table>
Since their introduction CPDs have been extended in a variety of ways and applied to a variety of different problems.

**Improved Compression**
- Improved column orderings [Zumsteg, 2016]
- Wildcard (i.e. “don’t care”) symbols [Salvetti et al., 2017]

**Improved Performance**
- Two Oracle Path Planning [Salvetti et al., 2018]

**Moving Target Search**
- In static game maps [Botea et al., 2013]
- In dynamic game maps [Baier et al., 2015]
- For multiple agents chasing multiple targets [Xie et al., 2017]
Since their introduction CPDs have been extended in a variety of ways and applied to a variety of different problems.

**Improved Compression**
- Improved column orderings [Zumsteg, 2016]
- Wildcard (i.e. “don’t care”) symbols [Salvetti et al., 2017]

**Improved Performance**
- Two Oracle Path Planning [Salvetti et al., 2018]

**Moving Target Search**
- In static game maps [Botea et al., 2013]
- In dynamic game maps [Baier et al., 2015]
- For multiple agents chasing multiple targets [Xie et al., 2017]
Two Oracle Path Planning
Topping: **Two Oracle Path Planning**

Topping is a method which combines two successful systems from the 2014 Grid-based Path Planning Competition: SRC and JPS.

Technical details appear in [Salvetti et al., 2018].

**Idea**

One oracle is a first-move database. The other oracle is a jump point database. To solve any shortest path query recurse the following:

- Ask the CPD which is the next direction to take.
- Ask the JPS database how many steps until the next turning point in the given direction (equiv. the next jump point).
- CPD Oracle: use move \( \rightarrow \)
- JPS+ Oracle: repeat that move for 5 times
Example
- CPD Oracle: use move →
- JPS+ Oracle: repeat that move for 6 times
We computed an 11-step optimal path in only 2 iterations.
Experimental Setup

- **Input:**
  - 54 game maps from Dragon Age: Origins, and Baldurs Gate II [Sturtevant, 2012]
  - Sizes from 538 to 137,375 nodes
  - 8-connected
  - 82,850 queries

- **Programs compared:**
  - Topping
  - SRC [Strasser et al., 2015]
    - CPD-based system
    - Fastest optimal solver in the GPPC 2014 [Sturtevant et al., 2015]
    - Used in Topping as a CPD oracle
  - JPS+ [Harabor and Grastien, 2014]
  - A* [Hart et al., 1968]
Speed Results

Benchmarks

82,850 queries on 54 game maps from Dragon Age: Origins, and Baldurs Gate II. All appear in [Sturtevant, 2012]

<table>
<thead>
<tr>
<th>Path length</th>
<th>A* CPU time</th>
<th>Speedup w.r.t.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A* SRC JPS+</td>
<td>Topping</td>
</tr>
<tr>
<td>[0, 150]</td>
<td>252</td>
<td>4.7 8.2 1.6</td>
</tr>
<tr>
<td>(150, 300]</td>
<td>1948</td>
<td>9.7 41.5 2.8</td>
</tr>
<tr>
<td>(300, 500]</td>
<td>5029</td>
<td>14.9 99.8 4.3</td>
</tr>
<tr>
<td>(500, 750]</td>
<td>9420</td>
<td>22.6 184.0 6.2</td>
</tr>
<tr>
<td>(750, 1200]</td>
<td>17024</td>
<td>35.2 437.0 6.3</td>
</tr>
<tr>
<td>≥ 1200</td>
<td>23039</td>
<td>63.1 527.0 14.9</td>
</tr>
</tbody>
</table>

Average CPU time (micro-seconds) and average speedup factor for Topping vs each competitor: A*, SRC, JPS+
Topping vs SRC

Performance gap between SRC and Topping. Queries are ordered so that the curve is monotonic.
Stats Across All Maps and Queries

<table>
<thead>
<tr>
<th>Systems</th>
<th>Mem [MB]</th>
<th>CPU time [µsec]</th>
<th>Speedup of Topping</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Full path 20 mov.</td>
<td>Full path 20 mov.</td>
</tr>
<tr>
<td>SRC</td>
<td>10.40</td>
<td>25.10</td>
<td>3.84</td>
</tr>
<tr>
<td>JPS+</td>
<td>3.99</td>
<td>216.00</td>
<td>33.00</td>
</tr>
<tr>
<td>Topping</td>
<td>23.00</td>
<td>6.54</td>
<td>–</td>
</tr>
</tbody>
</table>

Table: Average memory, average CPU time to compute the full path, average time to compute the first 20 moves of SRC, JPS, and Topping, and average speedup factor of Topping w.r.t. SRC, JPS+ for all maps and queries.
Compressed Path Databases in Moving-Target Search
Moving target search (MTS)
Moving target search (MTS)
Moving target search (MTS)
Moving target search (MTS)
A CPD-based program to compute hunter agent’s moves in MTS
Using older system (Copa) instead of newer version (SRC), for historical reasons
Idea:
- At every time step, query a CPD to obtain the first optimal move from the current position of the agent towards the current position of the target
Orders of magnitude faster than previous approaches to MTS
Full details available in [Botea et al., 2013; Baier et al., 2015].
Evaluating MtsCopa

Data
- 17 grid maps from 6 “domains” in Sturtevant’s collection
- Warcraft III (WC3), Dragon Age: Origins (DAO), Baldur’s Gate II (BG2), Rooms, Mazes, Random (25% obstacles)
- 4-connected, as in related work [Sun et al., 2012]

Benchmark algorithm
- I-ARA* [Sun et al., 2012]
- State-of-the-art MTS solver at that time
- Incremental search, reusing data from previous searches

3.47GHz machine, Red Hat Enterprise
Online search time

Search Time: MtsCopa vs I–ARA* (BG2)

Search Time: MtsCopa vs I–ARA* (DAO)

Search Time: MtsCopa vs I–ARA* (WC3)

Search Time: MtsCopa vs I–ARA* (Maze)

Search Time: MtsCopa vs I–ARA* (Random)

Search Time: MtsCopa vs I–ARA* (Rooms)
Solution length (hunter moves)

# Hunter Moves: MtsCopa vs I–ARA* (BG)

# Hunter Moves: MtsCopa vs I–ARA* (DAO)

# Hunter Moves: MtsCopa vs I–ARA* (WC3)

# Hunter Moves: MtsCopa vs I–ARA* (Mazes)

# Hunter Moves: MtsCopa vs I–ARA* (Random)

# Hunter Moves: MtsCopa vs I–ARA* (Rooms)
EOF
Jorge A Baier, Adi Botea, Daniel Harabor, and Carlos Hernández.
Fast Algorithm for Catching a Prey Quickly in Known and Partially Known Game Maps.
*IEEE Transactions on Computational Intelligence and AI in Games, 7*(2):193–199, 2015.

G. Veit Batz, Daniel Delling, Peter Sanders, and Christian Vetter.
Time-dependent Contraction Hierarchies.

Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner.
Combining Hierarchical and Goal-directed Speed-up Techniques for Dijkstra’s Algorithm.
*Journal of Experimental Algorithmics, 15*:2.3:2.1–2.3:2.31, March 2010.
Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner.  
Search-Space Size in Contraction Hierarchies.  

Adi Botea and Daniel Harabor.  
Path Planning with Compressed All-pairs Shortest Paths Data.  
In Proceedings of International Conference on Automated Planning and Scheduling (ICAPS), 2013.

Adi Botea, Jorge A. Baier, Daniel Harabor, and Carlos Hernández.  
Moving Target Search with Compressed Path Databases.  

Adi Botea, Ben Strasser, and Daniel Harabor.  
Complexity Results for Compressing Optimal Paths.  
In Proceedings of the National Conference on Artificial Intelligence (AAAI), January 2015.
Michael L. Cui, Daniel Harabor, and Alban Grastien.  
Compromise-free Pathfinding on a Navigation Mesh.  

R. Geisberger, P. Sanders, D. Schultes, and D. Delling.  
Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks.  

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter.  
Exact Routing in Large Road Networks Using Contraction Hierarchies.  

Daniel Damir Harabor and Alban Grastien.  
Improving Jump Point Search.  
Daniel Harabor and Peter Stuckey.
Forward Search in Contraction Hierarchies.
In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2018.

Daniel Harabor, Alban Grastien, Dindar Öz, and Vural Aksakalli.
Optimal Any-angle Pathfinding in Practice.

A Formal Basis for the Heuristic Determination of Minimum Cost Paths.

Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner.
Theta*: Any-Angle Path Planning on Grids.

Matteo Salvetti, Adi Botea, Alessandro Saetti, and Alfonso Gerevini.
Compressed Path Databases with Ordered Wildcard Substitutions.
Matteo Salvetti, Adi Botea, Alessandro Saeti, Daniel Harabor, and Alfonso Gerevini.

**Two Oracle Optimal Path Planning on Grid Maps.**
In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet.

**Efficient Query Processing on Spatial Networks.**

Sabine Storandt.

**Contraction Hierarchies on Grid Graphs.**

Ben Strasser and Dorothea Wagner.

**Graph Fill-In, Elimination Ordering, Nested Dissection and Contraction Hierarchies.**
Ben Strasser, Daniel Harabor, and Adi Botea.
Fast First-Move Queries through Run-Length Encoding.
In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2014.

Ben Strasser, Adi Botea, and Daniel Harabor.
Compressing Optimal Paths with Run Length Encoding.

Nathan R Sturtevant, Jason Traish, James Tulip, Tansel Uras, Sven Koenig, Ben Strasser, Adi Botea, Daniel Harabor, and Steve Rabin.
In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2015.

N. Sturtevant.
Benchmarks for Grid-Based Pathfinding.
Xiaoxun Sun, William Yeoh, Tansel Uras, and Sven Koenig.
Incremental ARA*: An Incremental Anytime Search Algorithm for Moving-Target Search.
In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2012.

Tansel Uras and Sven Koenig.
Understanding Subgoal Graphs by Augmenting Contraction Hierarchies.

Dorothea Wagner, Thomas Willhalm, and Christos Zaroliagis.
Geometric containers for efficient shortest-path computation.

Fan Xie, Adi Botea, and Akihiro Kishimoto.
A Scalable Approach to Chasing Multiple Moving Targets with Multiple Agents.
Patrick Zumsteg.
Komprimierte pfaddatenbanken durch lauflangenkodierung von optimal permutierten first-move matrizen.
Bachelorarbeit, Universität Basel, 2016.