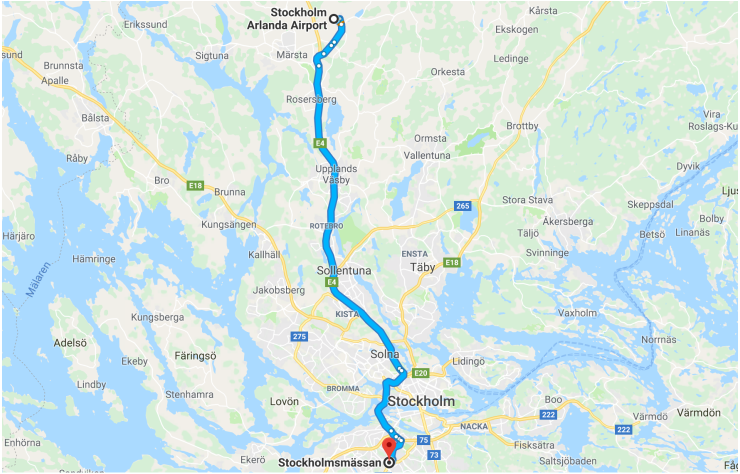
An Introduction to Contraction Hierarchies

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

Joint work with Peter Stuckey

IJCAI Tutorial
2018-07-13

Background: Problem



Background: Literature

Many optimal methods exist for such static shortest path problems including for graphs with millions of nodes. Some highlights:

Method	**Category**	**XT**[1]	**XM**[2]	**Query Time**
Dijkstra	Classic	-	-	seconds
A* (+Euclidean metric)	Classic	-	-	<1 sec
ALT (i.e. Landmarks)	Lowerbounds	mins	$\times 10^1$ MB	msec
True Distance Heuristics	Lowerbounds	mins	$\times 10^1$ MB	msec
Geometric Containers	Goal Pruning	hours	$\times 10^1$ MB	msec
Arc Flags	Goal Pruning	hours	$\times 10^2$ MB	<1 msec
Contraction Hierarchies	Abstraction	mins	$\times 10^1$ MB	μsec
Hub Labels	Oracle	hours	$\times 10^3$ MB	<1 μsec
Compressed Path Databases	Oracle	hours	$\times 10^3$ MB	<1 μsec

[1]XT = Extra Time (preprocessing)
[2]XM = Extra Memory (after preprocessing)

Background: Literature

Many optimal methods exist for such static shortest path problems including for graphs with millions of nodes. Some highlights:

Method	Category	XT[1]	XM[2]	Query Time
Dijkstra	Classic	-	-	seconds
A* (+Euclidean metric)	Classic	-	-	<1 sec
ALT (i.e. Landmarks)	Lowerbounds	mins	$\times 10^1$ MB	msec
True Distance Heuristics	Lowerbounds	mins	$\times 10^1$ MB	msec
Geometric Containers	Goal Pruning	hours	$\times 10^1$ MB	msec
Arc Flags	Goal Pruning	hours	$\times 10^2$ MB	<1 msec
Contraction Hierarchies	Abstraction	mins	$\times 10^1$ MB	μsec
Hub Labels	Oracle	hours	$\times 10^3$ MB	<1 μsec
Compressed Path Databases	Oracle	hours	$\times 10^3$ MB	<1 μsec

[1]XT = Extra Time (preprocessing)
[2]XM = Extra Memory (after preprocessing)

Contraction Hierarchies: Some Definitions

Definition #1 (Algorithmics perspective)

Contraction Hierarchies is a graph augmentation / overlay technique that helps to speed up pathfinding search.

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.

Contraction Hierarchies: Some Definitions

Definition #1 (Algorithmics perspective)

Contraction Hierarchies is a graph augmentation / overlay technique that helps to speed up pathfinding search.

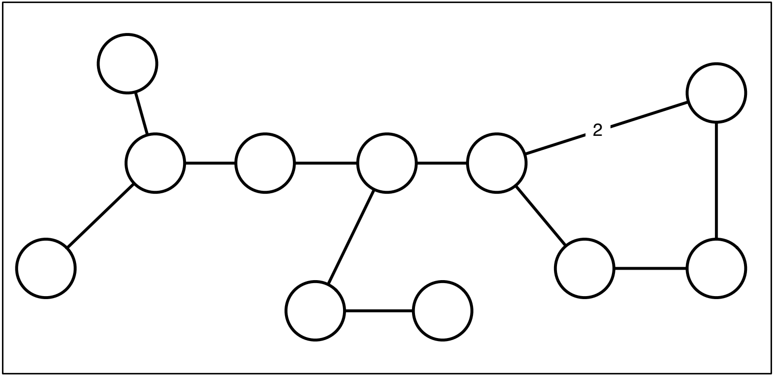
Definition #2 (Heuristic Search perspective)

Contraction Hierarchies is an optimality-preserving abstraction technique where macro edges are embedded in the original graph.

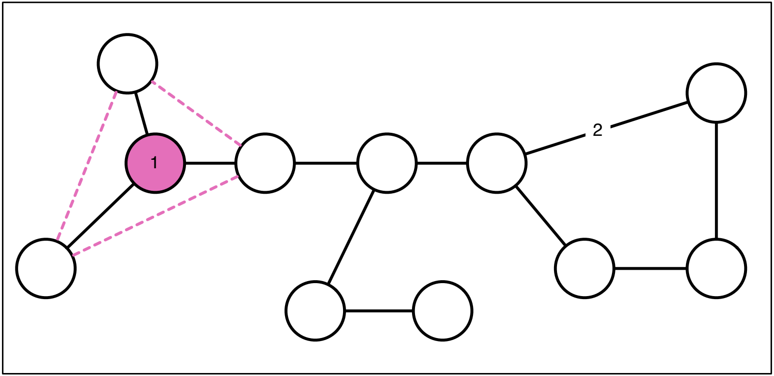
Definition #3 (Practitioner's perspective)

1. (Offline) Add "shortcut edges" between selected pairs of non-adjacent nodes.
2. (Online) Exploit shortcuts to reach the target sooner.

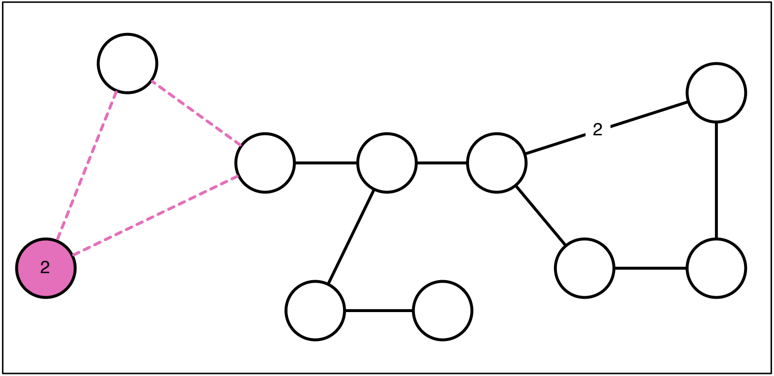
Contraction Hierarchies: Example



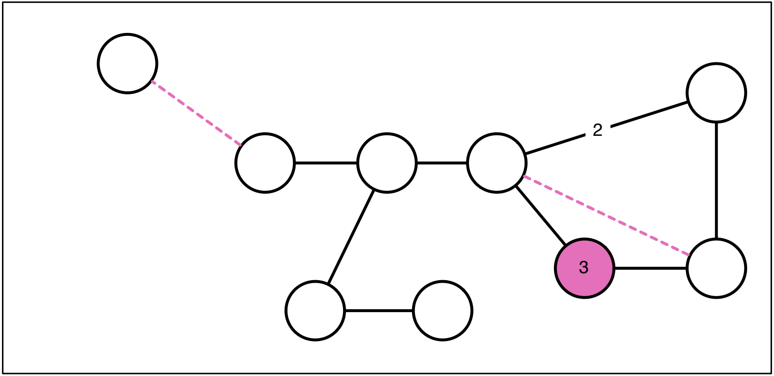
Contraction Hierarchies: Example



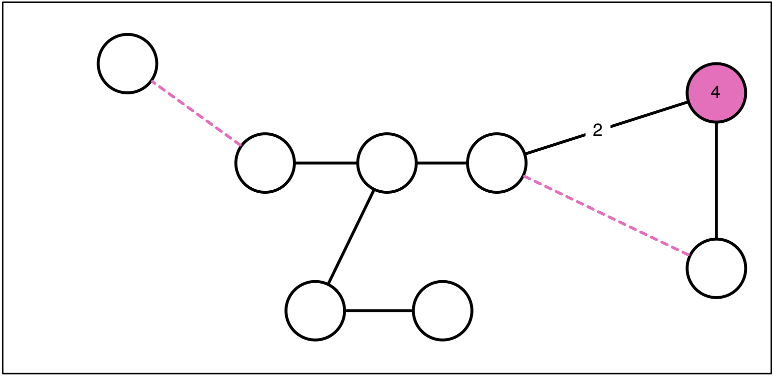
Contraction Hierarchies: Example



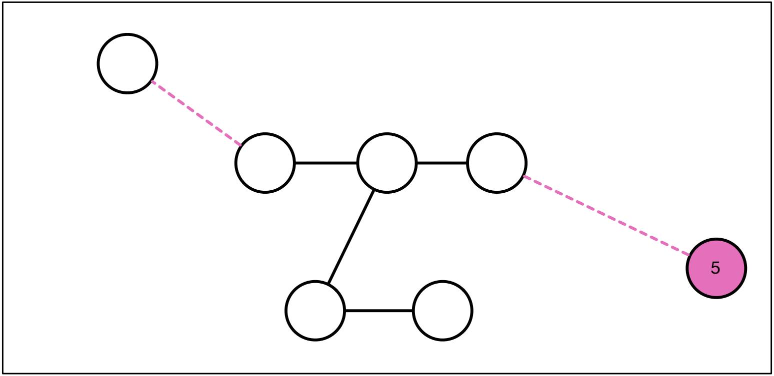
Contraction Hierarchies: Example



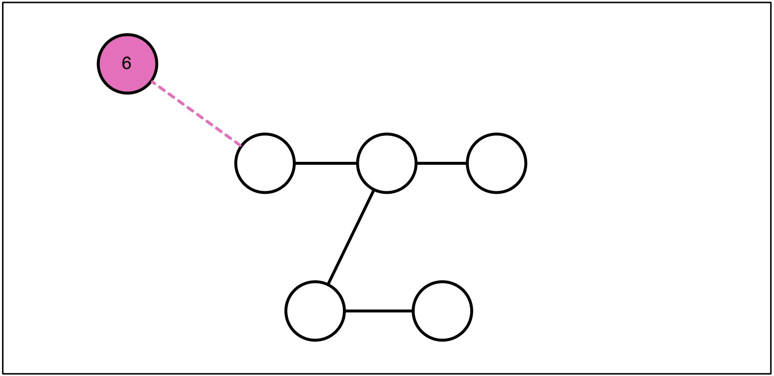
Contraction Hierarchies: Example



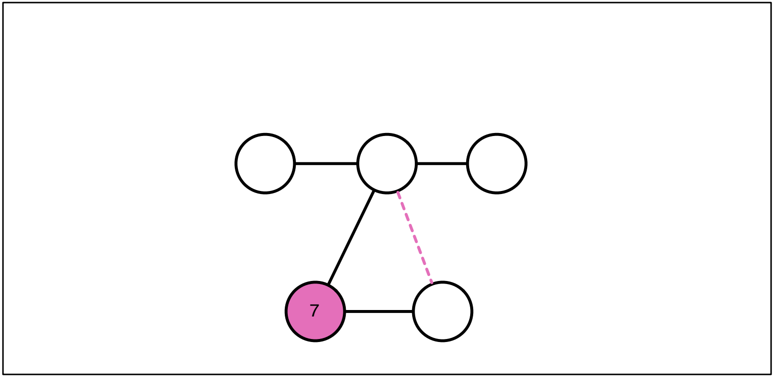
Contraction Hierarchies: Example



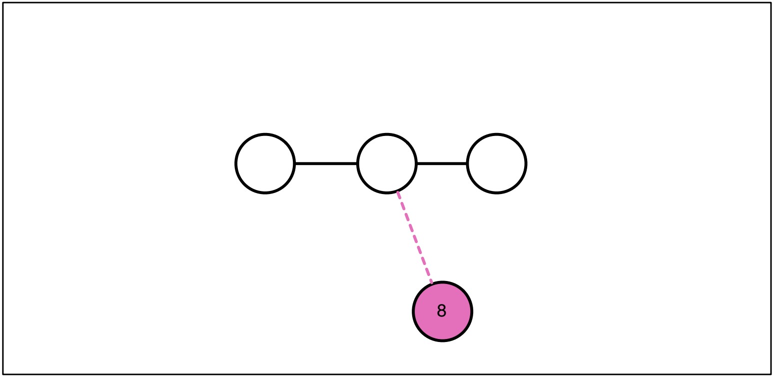
Contraction Hierarchies: Example



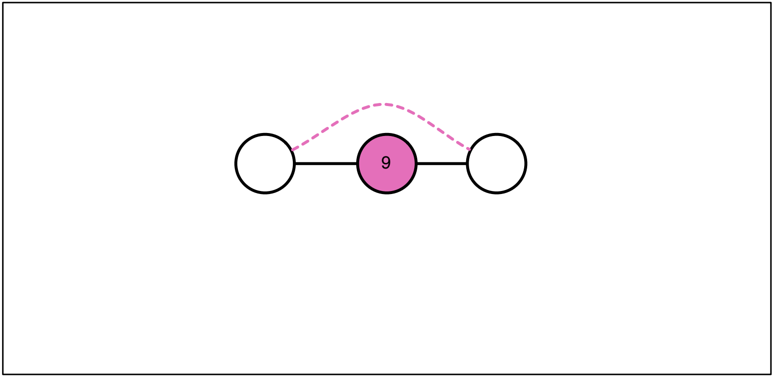
Contraction Hierarchies: Example



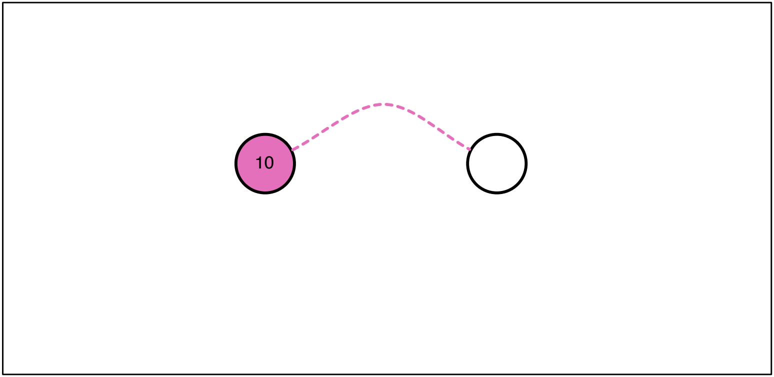
Contraction Hierarchies: Example



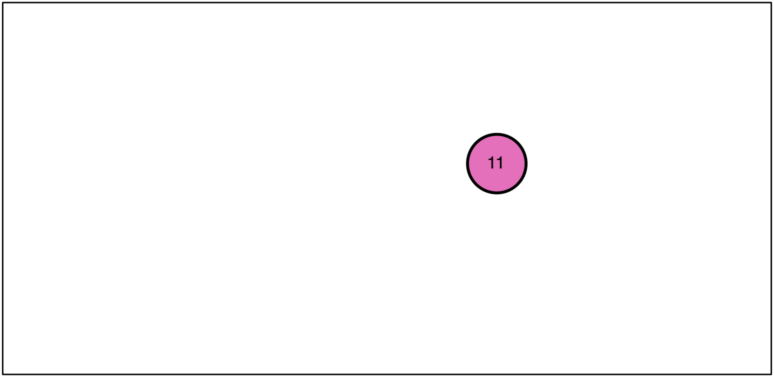
Contraction Hierarchies: Example



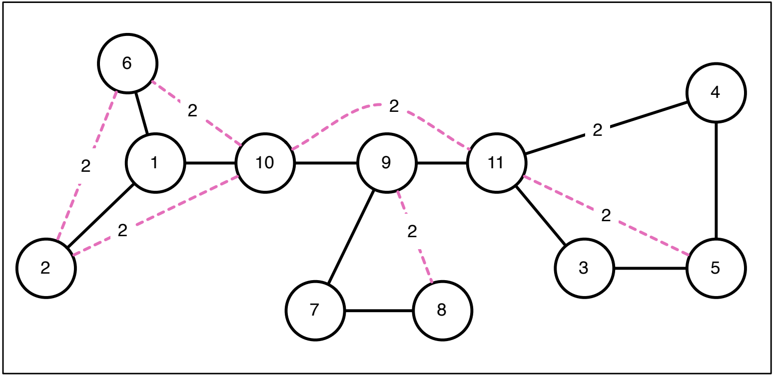
Contraction Hierarchies: Example



Contraction Hierarchies: Example



Contraction Hierarchies: Example



Contraction Hierarchies: In Practice

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

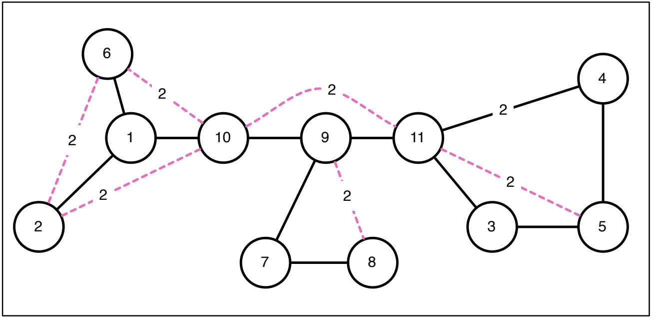
Rules of thumb:

- 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:

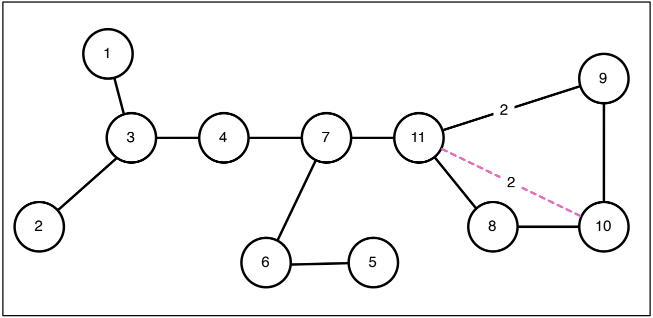
- A "good" node ordering allows a search to reach the topmost node in the hierarchy in a logarithmic number of steps.
- A "bad" node ordering requires a linear number of steps.
- Contract everywhere instead of always from the same place.



Issue 1: How to order the nodes for contraction

Rules of thumb:

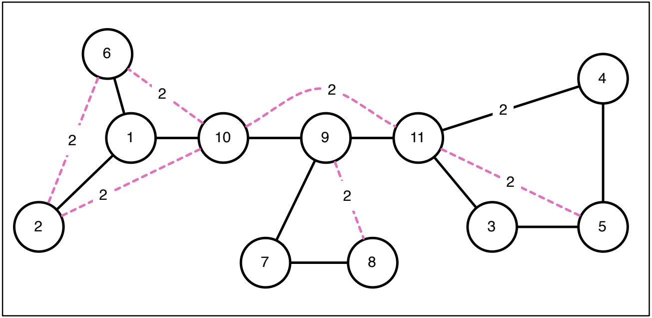
- A "good" node ordering allows a search to reach the topmost node in the hierarchy in a logarithmic number of steps.
- A "bad" node ordering requires a linear number of steps.
- Contract everywhere instead of always from the same place.



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).

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)

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 #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]

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:

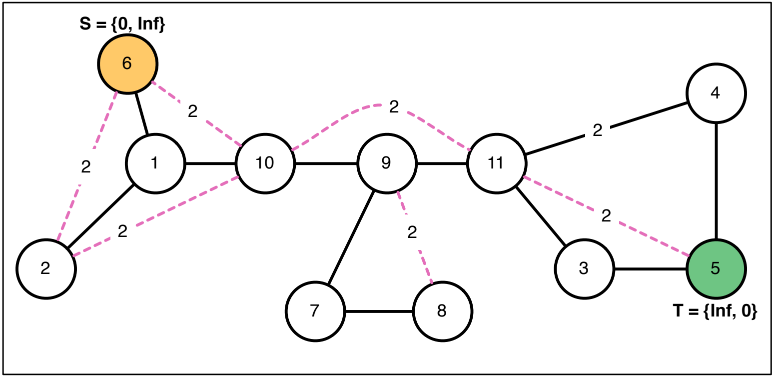
- $G_{\uparrow} = (V, E_{\uparrow} = \{(u, v) \in E \mid u \prec v\})$
- $G_{\downarrow} = (V, E_{\downarrow} = \{(u, v) \in E \mid u \succ v\})$
- $\prec$ and $\succ$ compare the contraction order of pairs of nodes.

Online

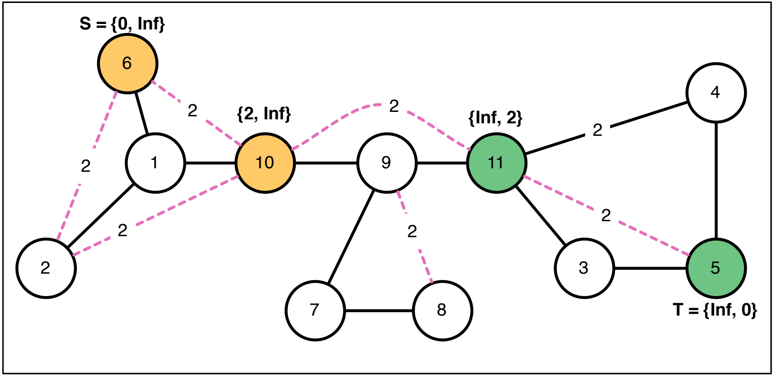
Perform a bi-directional Dijkstra search:

- 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.

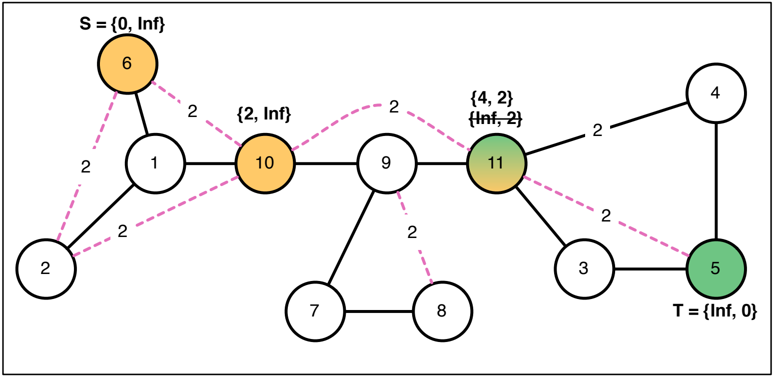
BCH Example



BCH Example



BCH Example



Deconstructing BCH

The following results appear in [Geisberger *et al.*, 2008]:

ch-path

For every *optimal path* 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 $G_\downarrow$.

apex-node

Every *ch-path* has a node which is lexically largest among all nodes in the path.

Deconstructing BCH

CH-SPSP [Harabor and Stuckey, 2018]

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

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

Subject to:

1. $v_i \prec v_{i+1}$ for all $1 \leq i < k$
2. $v_i \succ v_{i+1}$ for all $k \leq i < n$

Graph Traversal Policies

Successor Types

1. up-up successors
2. up-down successors
3. down-down successors
4. down-up successors

Graph Traversal Policies

Successor Types

1. up-up successors
2. up-down successors
3. down-down successors
4. ~~down-up successors~~

Graph Traversal Policies

Successor Types

1. up-up successors
2. up-down successors
3. down-down successors
4. ~~down-up successors~~

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**.

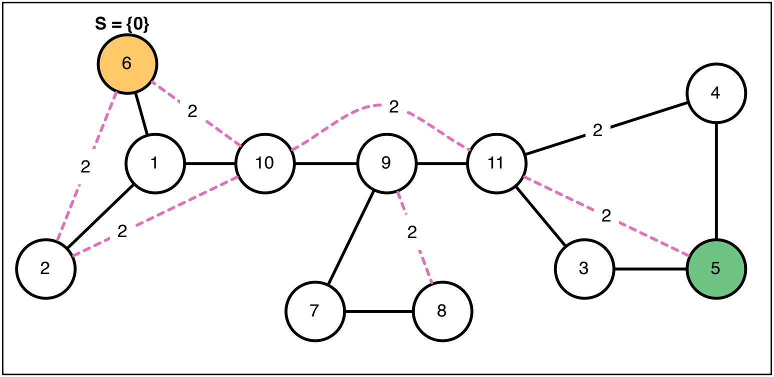
The FCH Query Algorithm

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

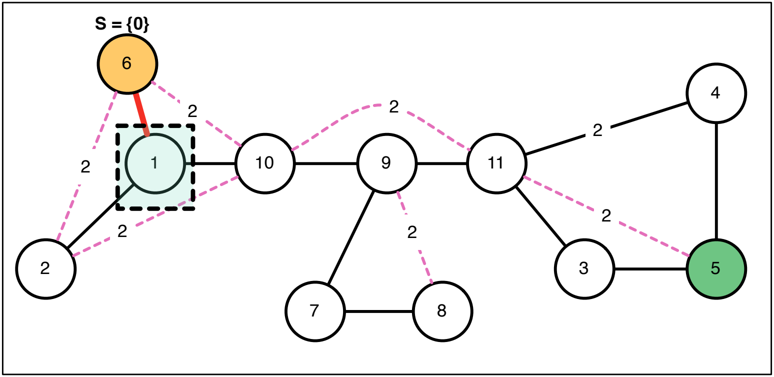
FCH can be easily and effectively combined with many standard pruning methods including Bounding Boxes. This algorithm:

- 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]

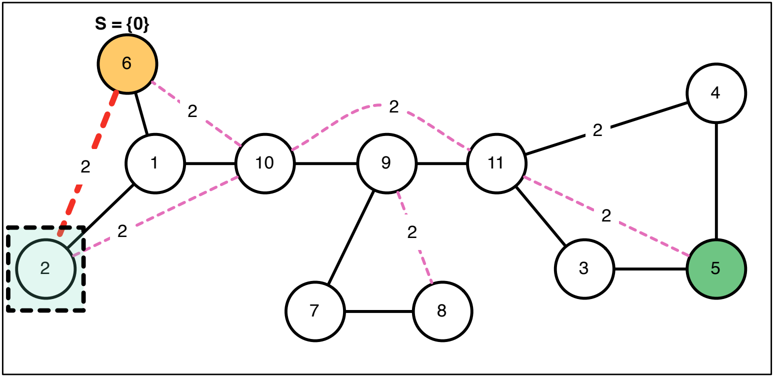
FCH+BB Example



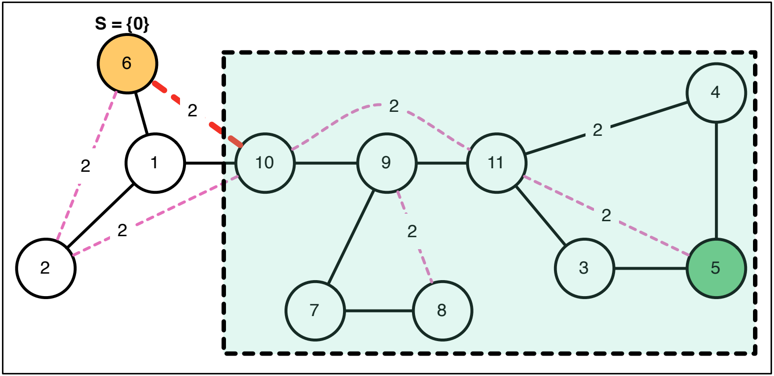
FCH+BB Example



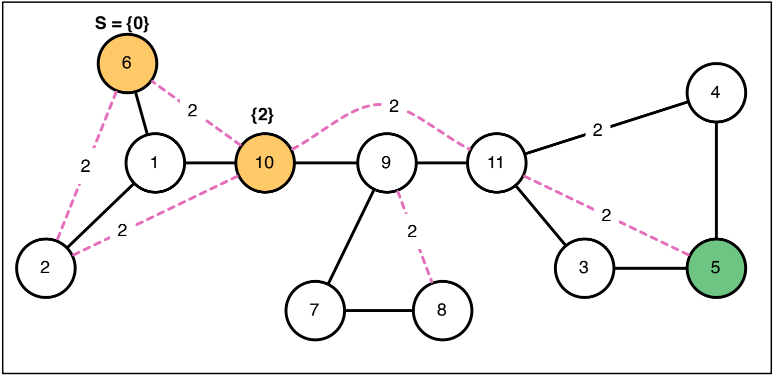
FCH+BB Example



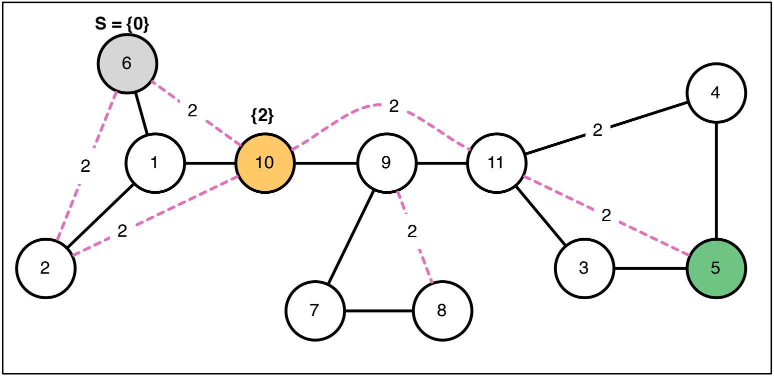
FCH+BB Example



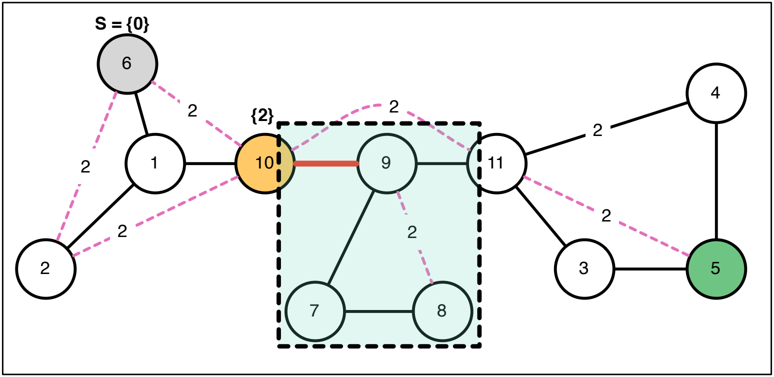
FCH+BB Example



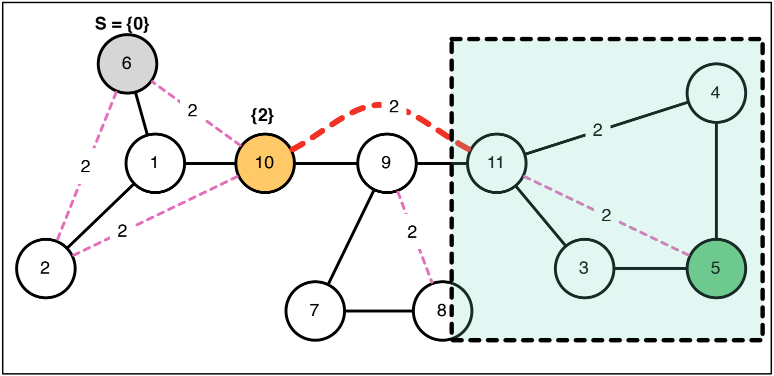
FCH+BB Example



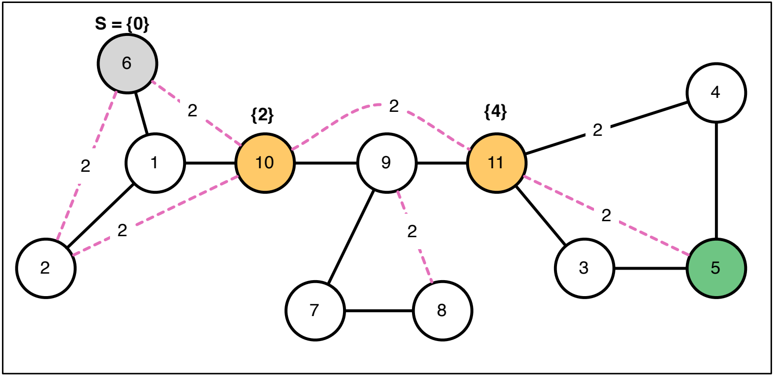
FCH+BB Example



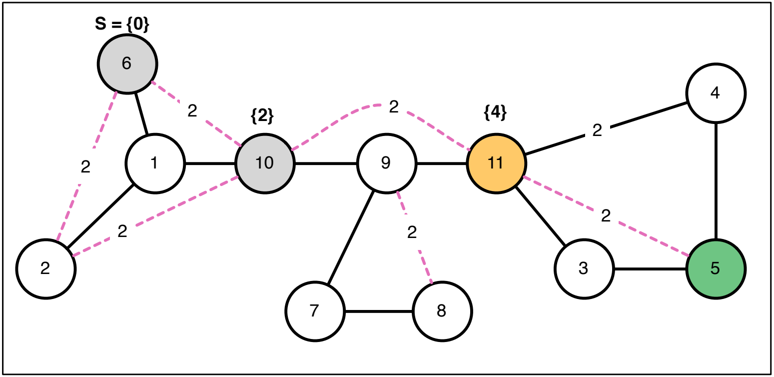
FCH+BB Example



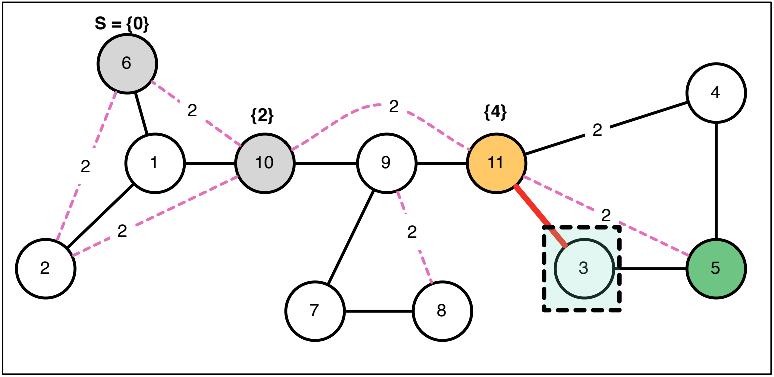
FCH+BB Example



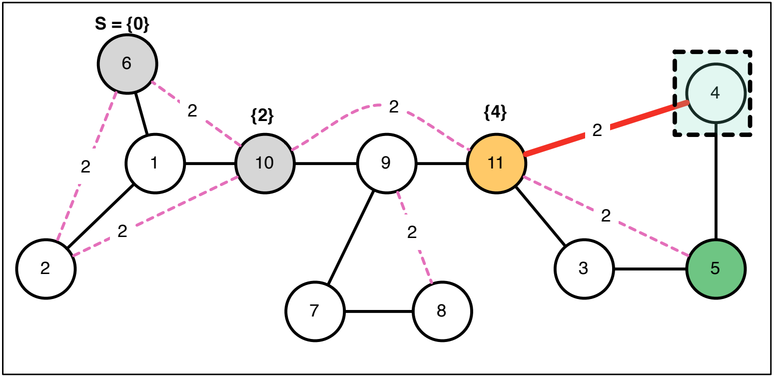
FCH+BB Example



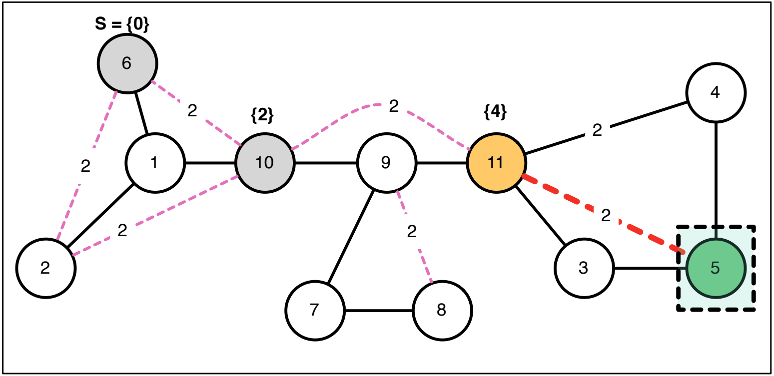
FCH+BB Example



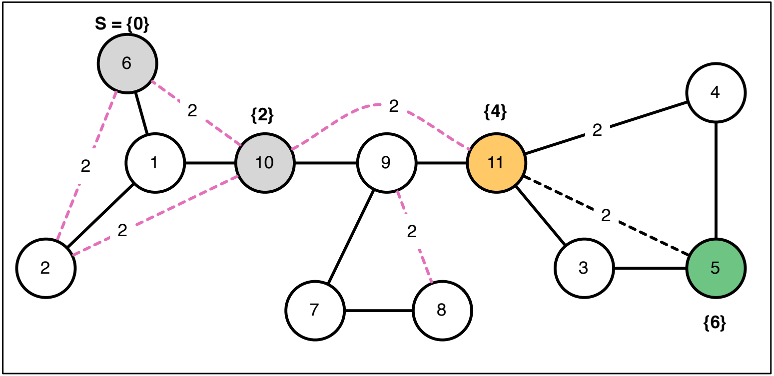
FCH+BB Example



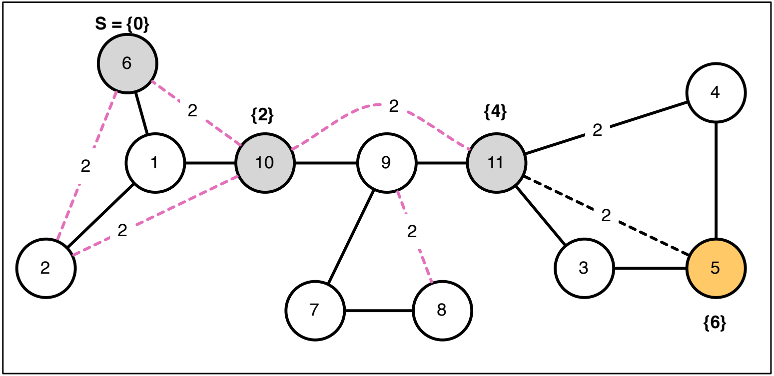
FCH+BB Example



FCH+BB Example



FCH+BB Example



Some Experimental Results

Benchmarks

Graph	$\lvert V \rvert$	$\lvert E \rvert$ #Input	#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`

Comparisons

Uni-directional variants:

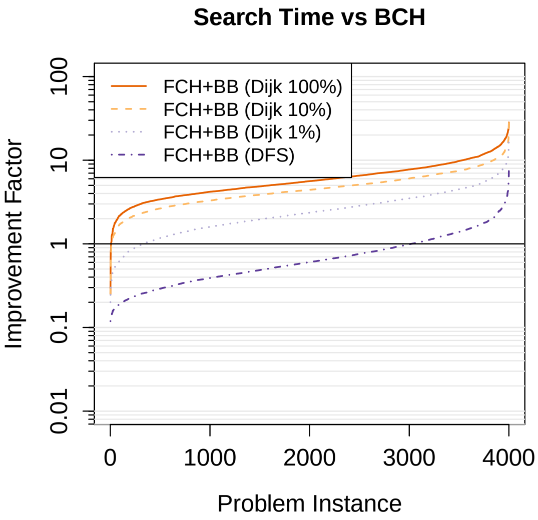
- FCH+BB (DFS) — Very light pre-proc. ($\approx$1 second)
- FCH+BB (Dijk 1%) – Light pre-proc. (minutes)
- FCH+BB (Dijk 10%) – Moderate pre-proc. ($<$ 1 hour)
- FCH+BB (Dijk 100%) – Extensive pre-proc. (many hours)

Bi-directional variants:

- BCH
- BCH+BB (Dijk 100%) – Extensive pre-proc. (many hours)

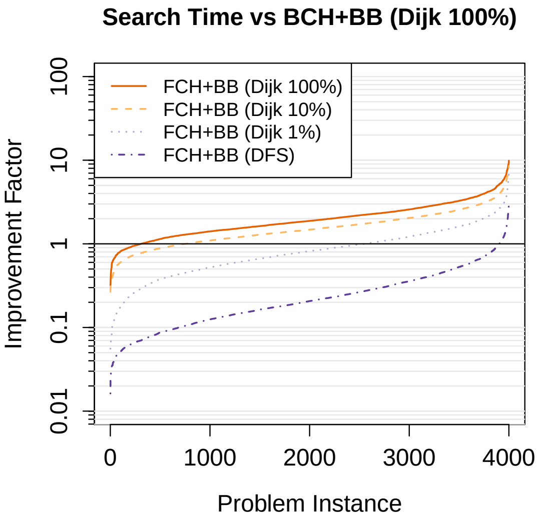
Comparisons vs BCH

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



Comparisons vs BCH+BB

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



Upcoming Talk:

D. Harabor & P. Stuckey

Forward Search in Contraction Hierarchies.

Saturday 14 July @ 09:30. SoCS 2018.

EOF

Optimal Any-angle Pathfinding

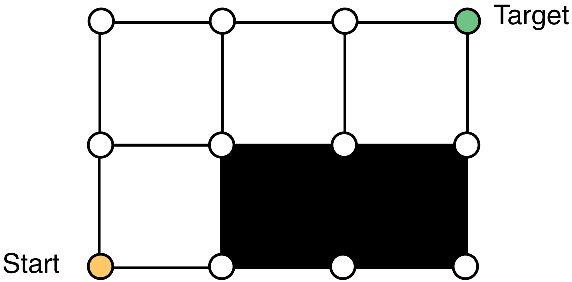
Daniel D. Harabor
Monash University
http://harabor.net/daniel

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

IJCAI Tutorial
2018-07-13

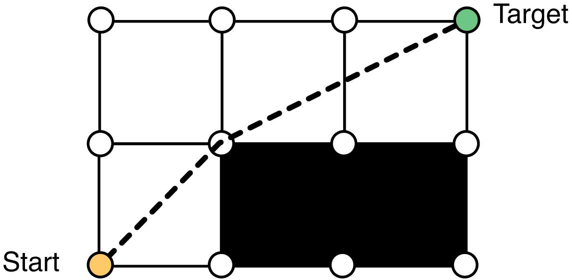
Suboptimality and Theta*

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



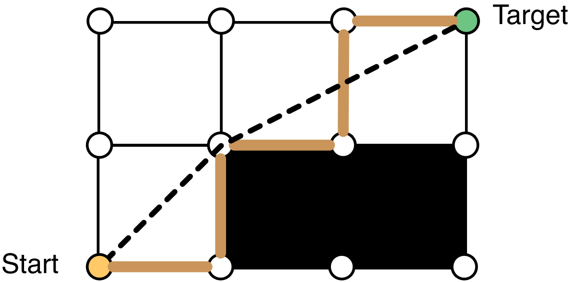
Suboptimality and Theta*

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



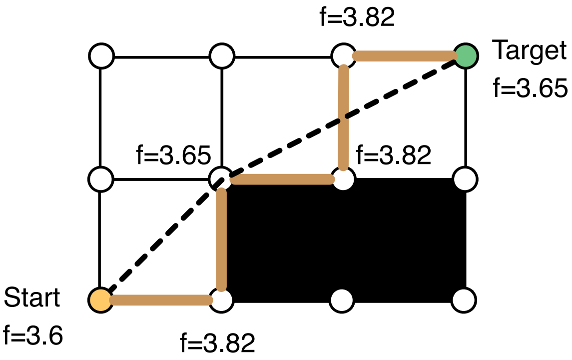
Suboptimality and Theta*

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



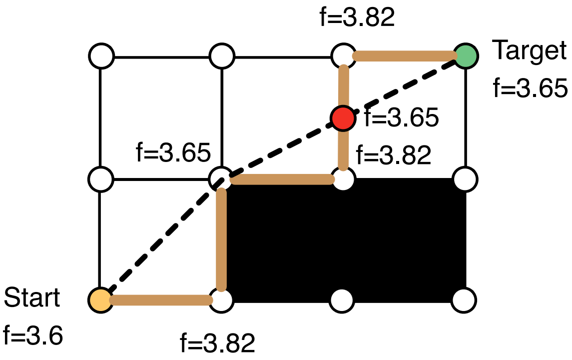
Suboptimality and Theta*

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



Suboptimality and Theta*

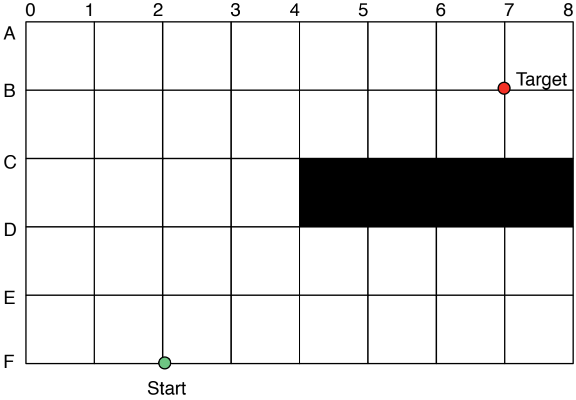
Theta* [Nash *et al.*, 2007] proceeds from one grid vertex to the next.
This strategy is suboptimal since it expands nodes out of order.



Anya: An optimal any-angle pathfinding technique

Intuition

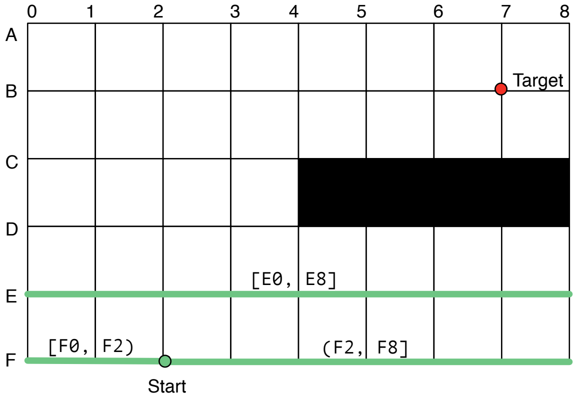
Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.



Anya: An optimal any-angle pathfinding technique

Intuition

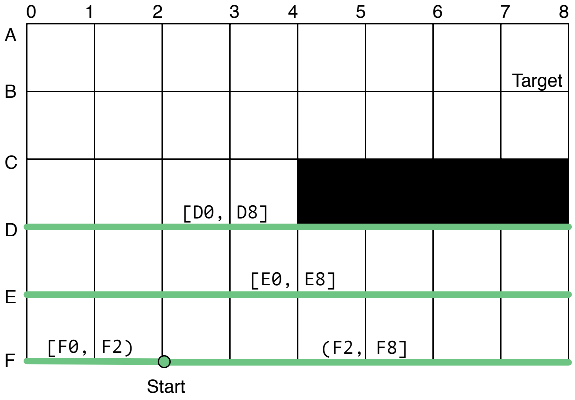
Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.



Anya: An optimal any-angle pathfinding technique

Intuition

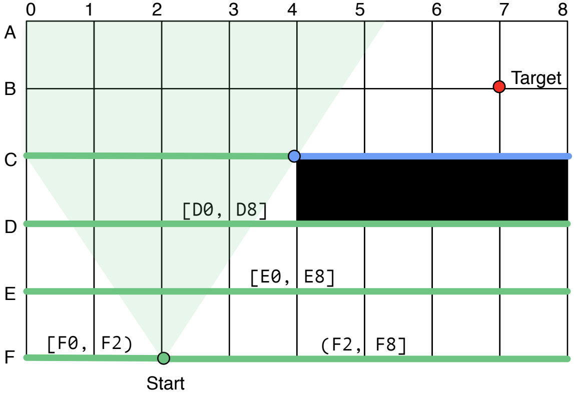
Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.



Anya: An optimal any-angle pathfinding technique

Intuition

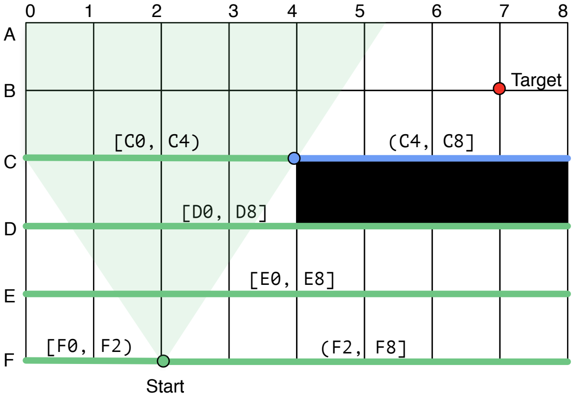
Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.



Anya: An optimal any-angle pathfinding technique

Intuition

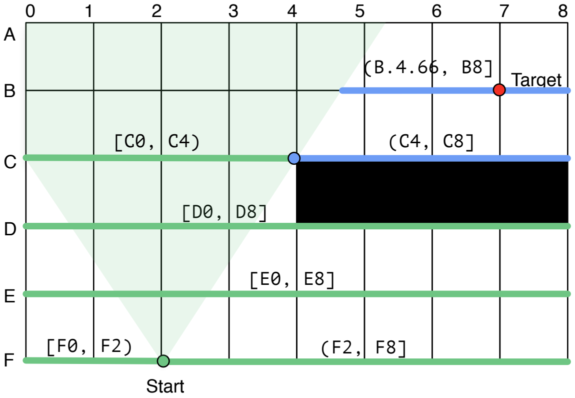
Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.



Anya: An optimal any-angle pathfinding technique

Intuition

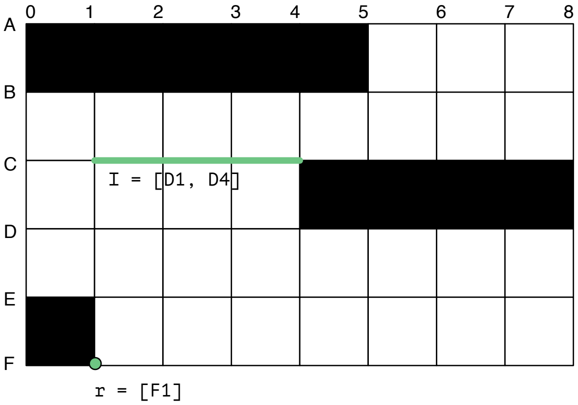
Expand *sets* of nodes together at one time. A set is constructed as a contiguous intervals of points from along a single row.



Definition #1: Search Nodes

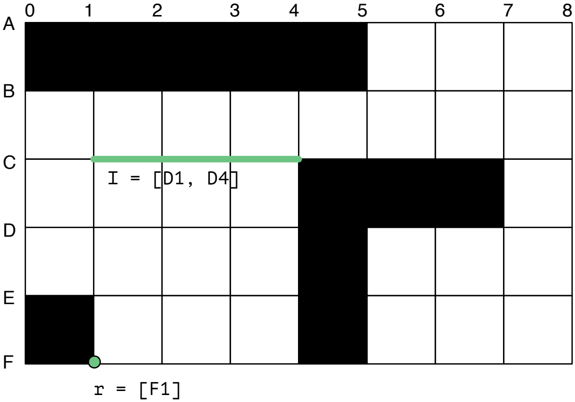
Every node is a tuple (I, r) where:

- r is a *root*; the most recent turning point.
- I is an interval of contiguous points, all visible from r.
- The *start node* has a point interval and a root "off the grid"



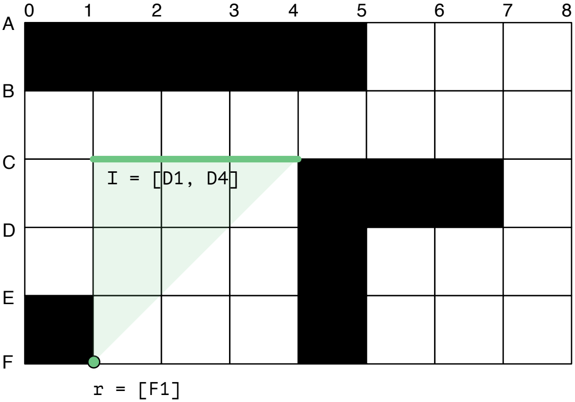
Definition #2: Successors

- 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*



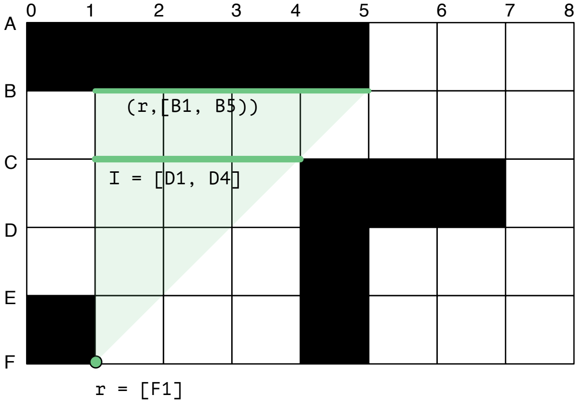
Definition #2: Successors

- 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*



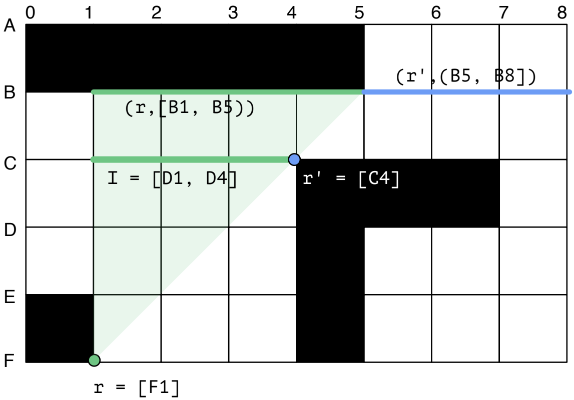
Definition #2: Successors

- 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*



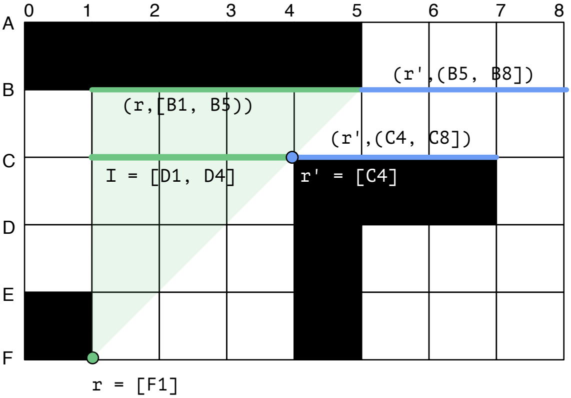
Definition #2: Successors

- 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*



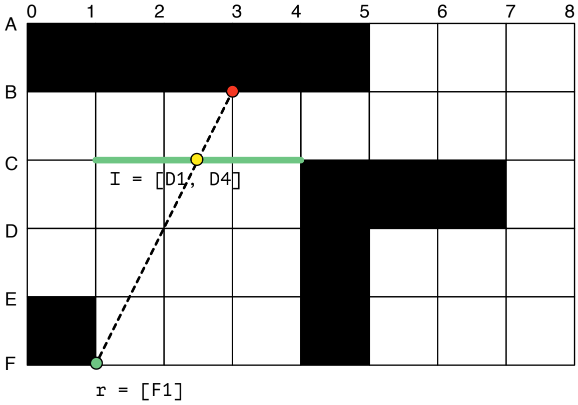
Definition #2: Successors

- 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*



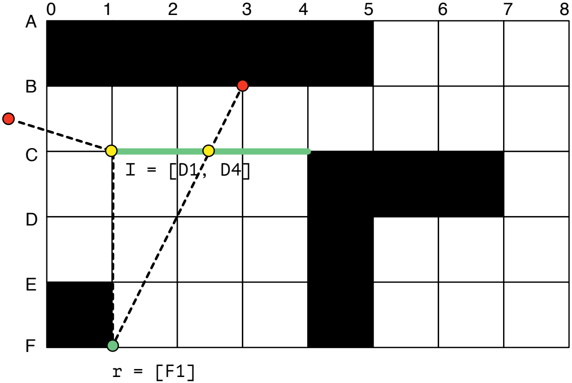
Evaluation Function

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



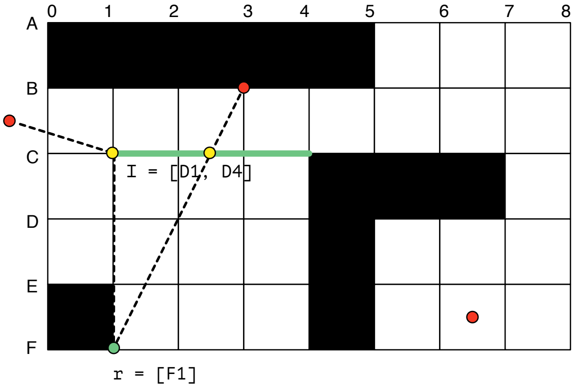
Evaluation Function

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



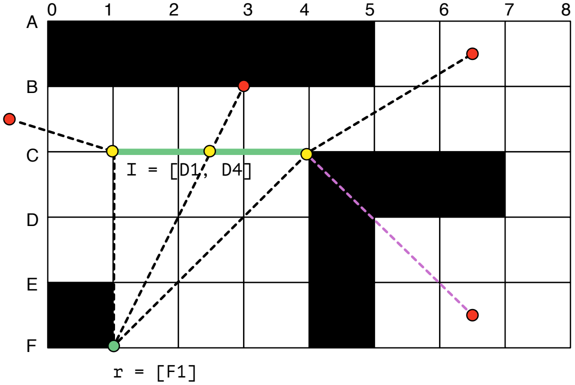
Evaluation Function

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



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.

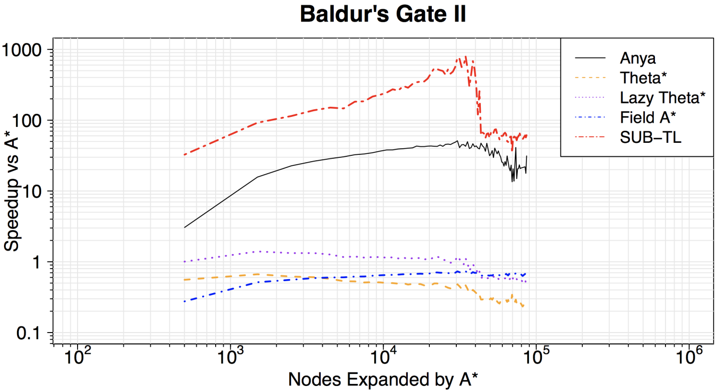
Online

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].

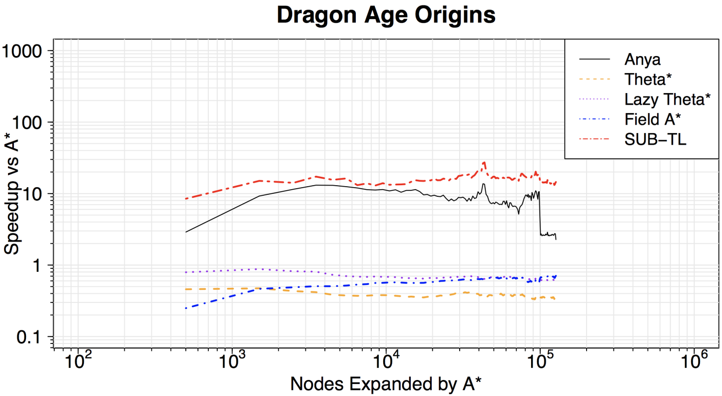
Results on Games Maps

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



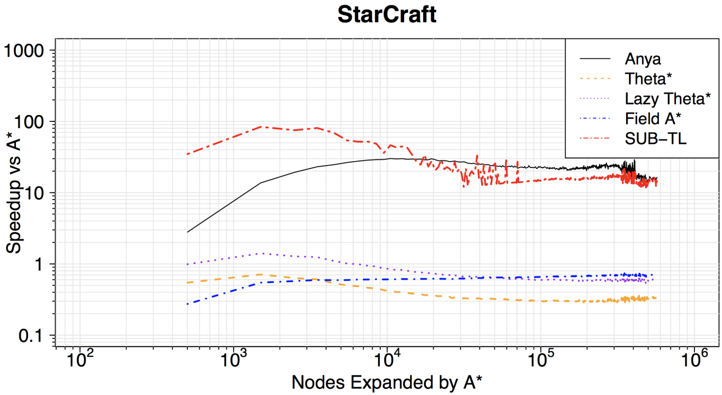
Results on Games Maps

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



Results on Games Maps

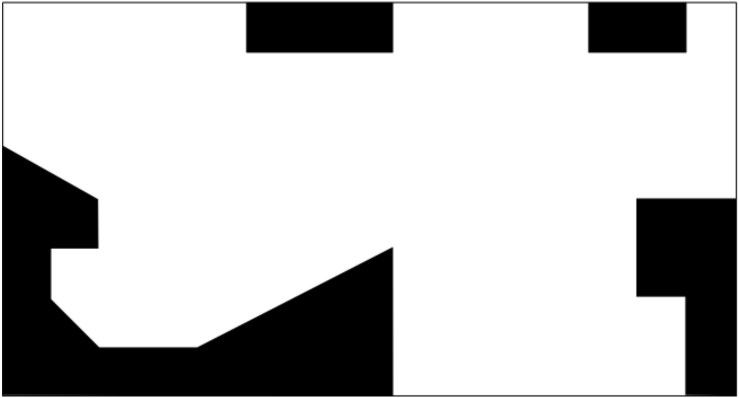
Speedup vs grid A* on a range of benchmarks from real games (all maps and instances from Nathan's repository).



Euclidean Shortest Path Problems in 2D

2D ESPP

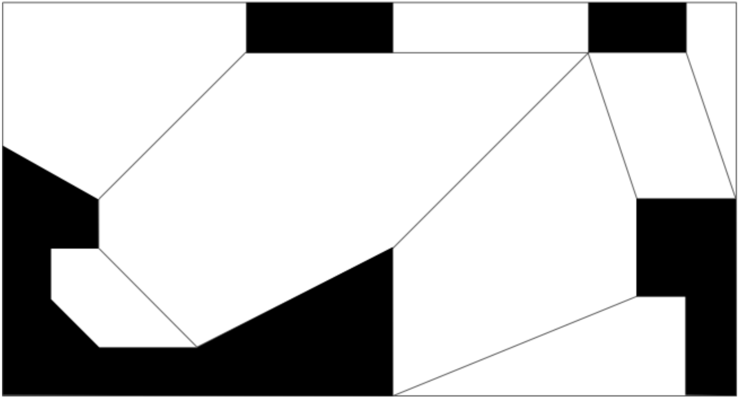
Find a shortest path among a set of polygonal obstacles.



2D ESPP on a Navigation Mesh

Navigation Mesh

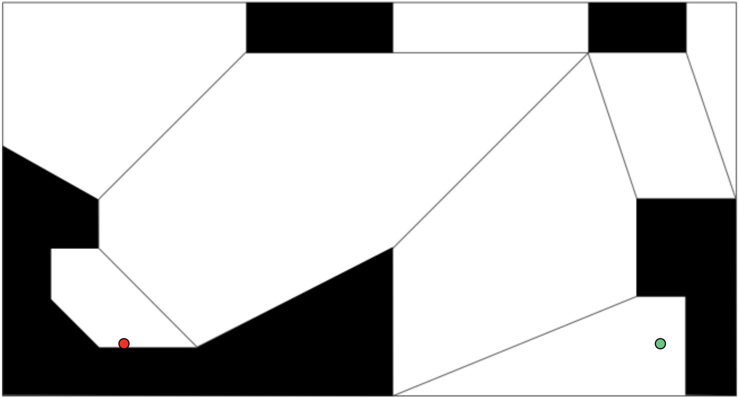
A partitioning of the traversable space into a collection of convex polygons. Cheap to build. Common in many application areas.



From Any-angle Pathfinding to ESPP

Polyanya

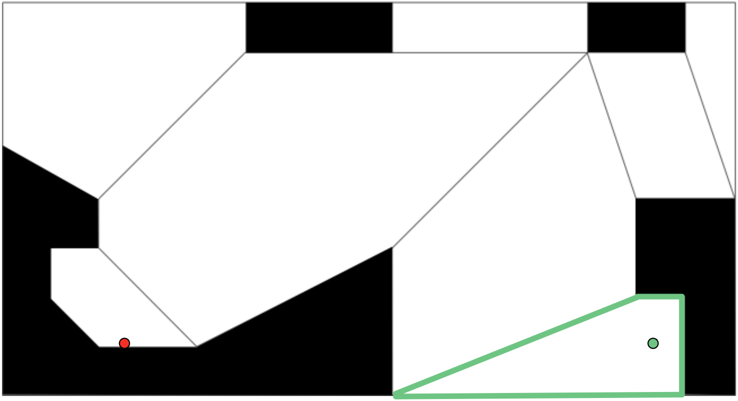
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

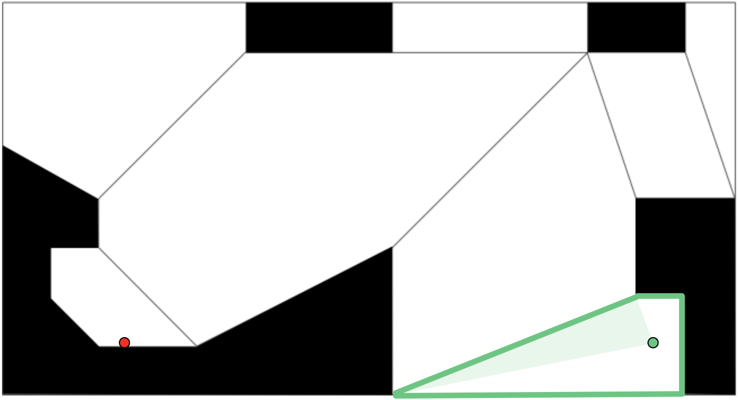
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

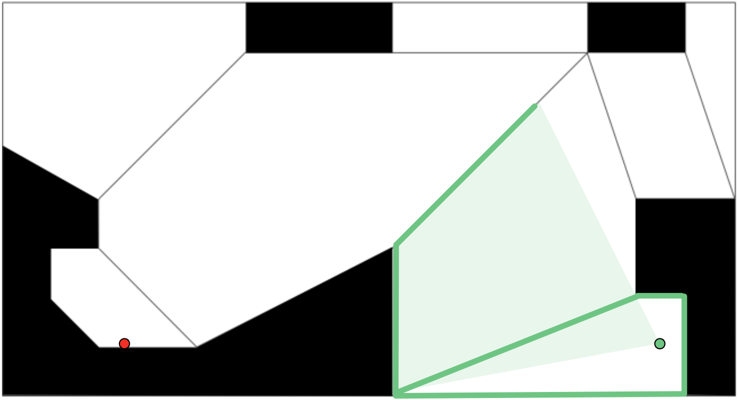
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

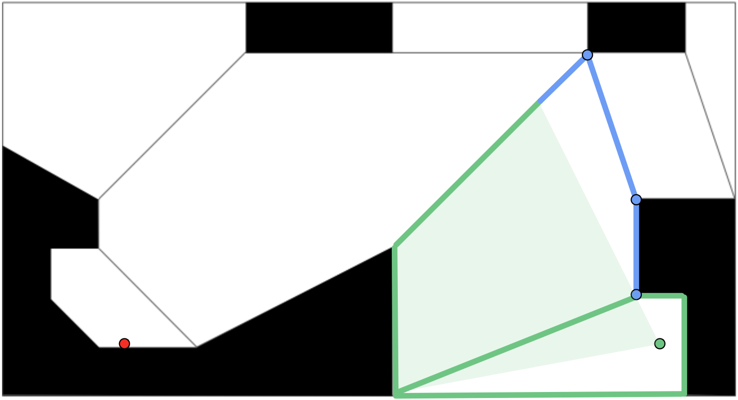
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

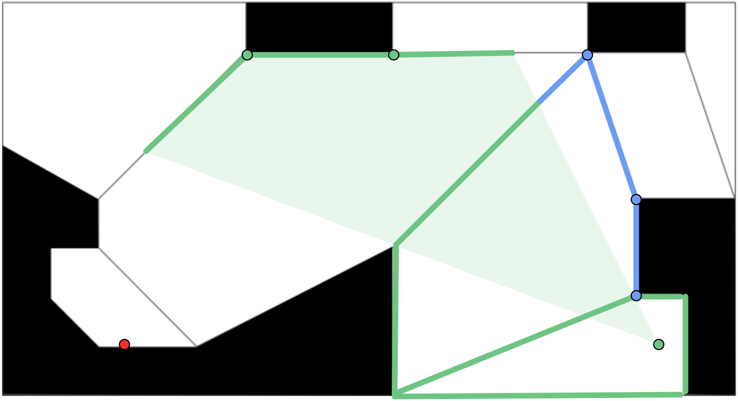
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

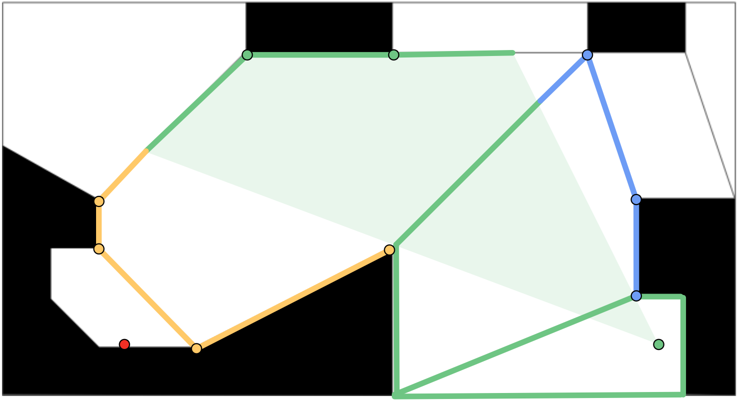
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

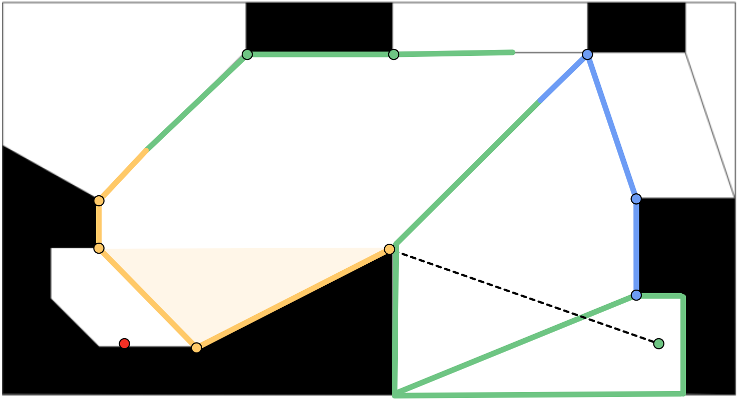
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

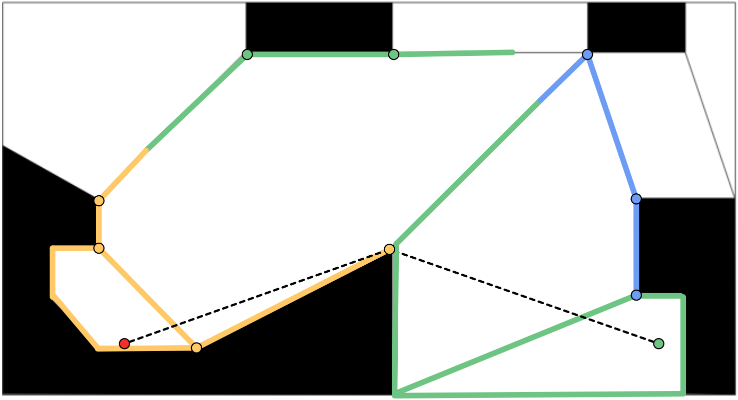
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

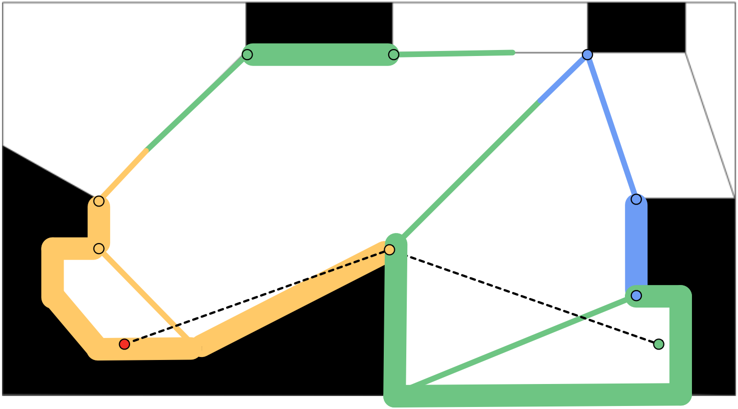
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

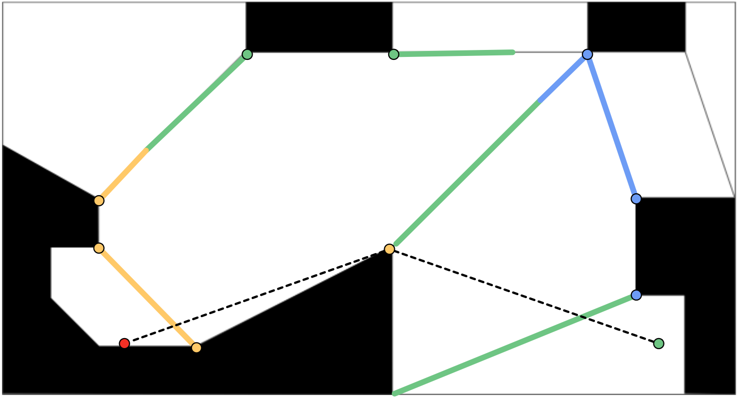
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

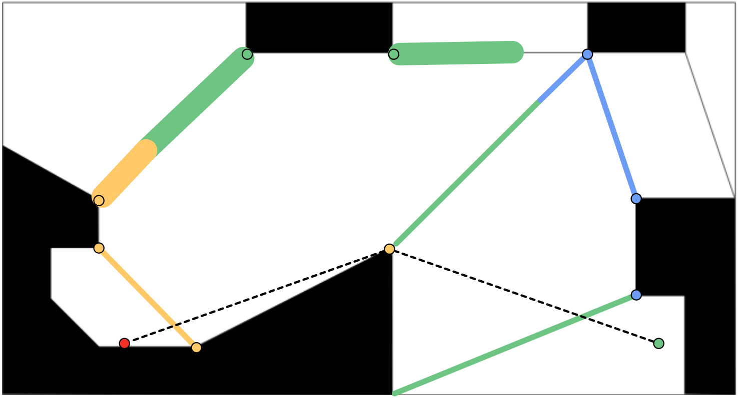
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

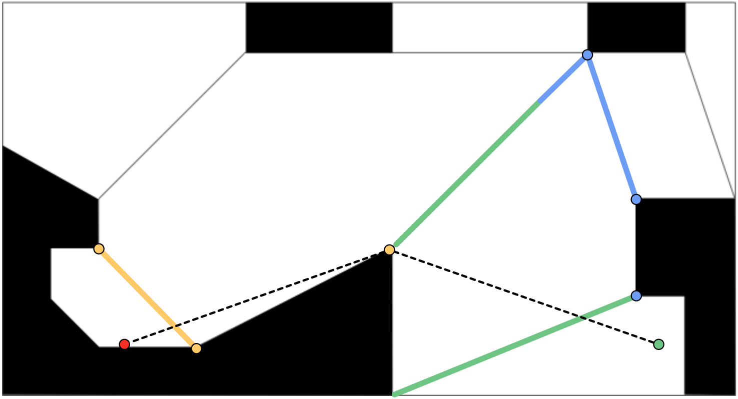
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

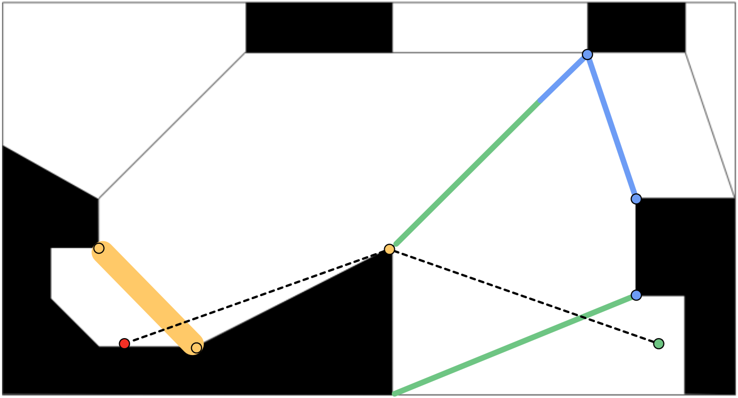
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

Polyanya

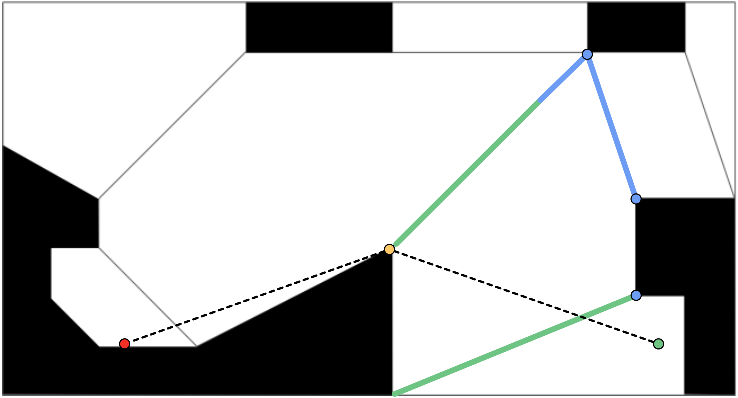
Polyanya [Cui *et al.*, 2017] extends and generalises Anya, from any-angle pathfinding on a grid to ESPP in the plane.



From Any-angle Pathfinding to ESPP

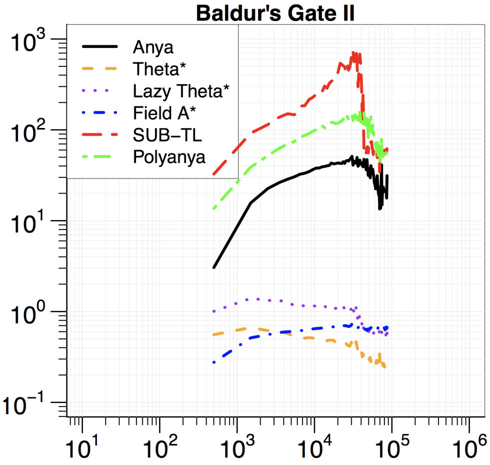
Polyanya

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



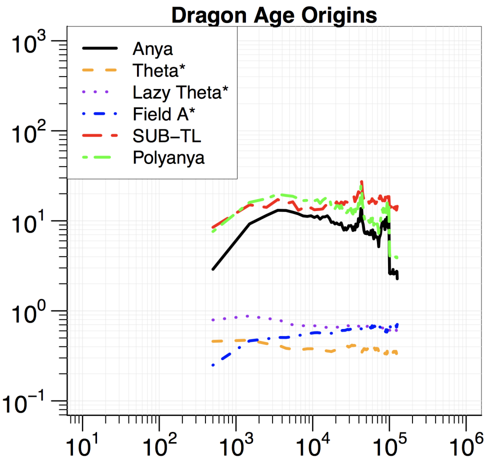
(More) Results on Games Maps

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



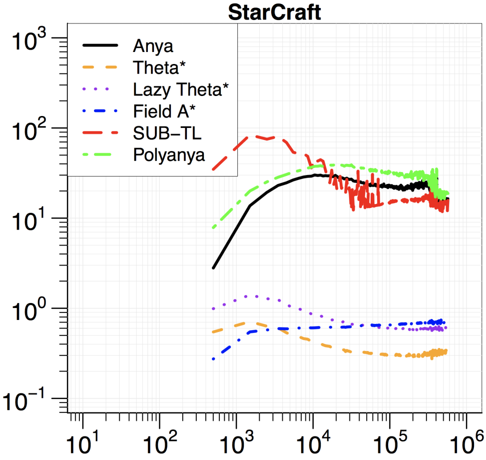
(More) Results on Games Maps

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



(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

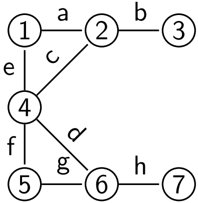
- Build an oracle called a compressed path database (CPD)
- Use the CPD to perform route planning
- The oracle provides optimal moves fast
- It relies on all-pairs pre-processing step
- It requires additional memory to store the pre-processing results

What is a Compressed Path Database (CPD)?

- Given a graph...

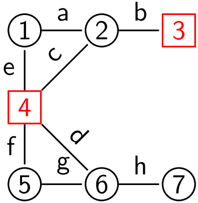
What is a Compressed Path Database (CPD)?

- Given a graph...



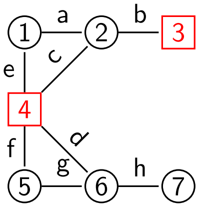
What is a Compressed Path Database (CPD)?

- Given a graph...
- and *any* two nodes s and t...



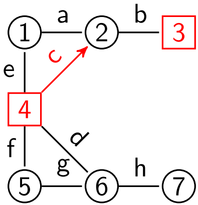
What is a Compressed Path Database (CPD)?

- 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., $\mathrm{CPD}[4, 3] = \mathrm{c}$



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]

Building a CPD

Preprocessing

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

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

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

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

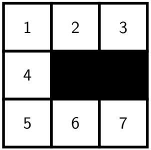
Algorithmic Sketch

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

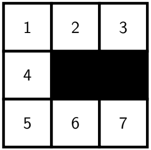
SRC, a system based on these ideas, was the fastest optimal solver in the 2014 Grid-Based Path Planning Competition [Sturtevant *et al.*, 2015].

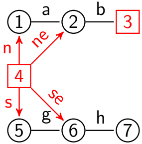
Full technical details in [Strasser *et al.*, 2014; Strasser *et al.*, 2015].

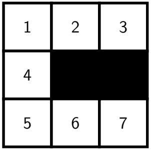
First-move matrix

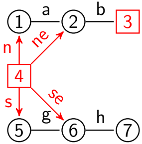


First-move matrix



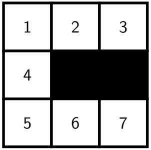


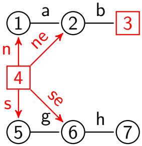




s \ t	1	2	3	4	5	6	7
1							
2							
3							
4	n	ne	ne	*	s	se	se
5							
6							
7							

First-move matrix





s \ t	1	2	3	4	5	6	7
1	*	e	e	s	s	s	s
2	w	*	e	sw	sw	sw	sw
3	w	w	*	w	w	w	w
4	n	ne	ne	*	s	se	se
5	n	n	n	n	*	e	e
6	nw	nw	nw	nw	w	*	e
7	w	w	w	w	w	w	*

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 **ssnnsss** has three runs: **ss**; **nn**; **sss**

s \ t	1	2	3	4	5	6	7
1	*	e	e	s	s	s	s
2	w	*	e	sw	sw	sw	sw
3	w	w	*	w	w	w	w
4	n	ne	ne	*	s	se	se
5	n	n	n	n	*	e	e
6	nw	nw	nw	nw	w	*	e
7	w	w	w	w	w	w	*

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 **ssnnsss** has three runs: **ss**; **nn**; **sss**

s \ t	1	2	3	4	5	6	7
1	*	e	e	s	s	s	s
2	w	*	e	sw	sw	sw	sw
3	w	w	*	w	w	w	w
4	n	ne	ne	*	s	se	se
5	n	n	n	n	*	e	e
6	nw	nw	nw	nw	w	*	e
7	w	w	w	w	w	w	*

Uncompressed matrix has 49 tokens

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			

CPD has 16 runs

Non-Unique Shortest Paths?

Roads

- On roads shortest paths are very often unique.

Game Maps

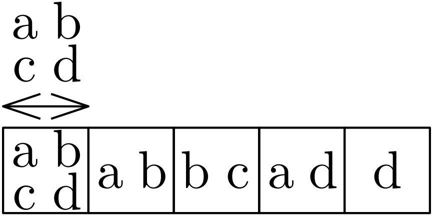
- Game maps often contain unit grids with highly non-unique shortest paths.
- **Idea**: Tie-break paths such that compression is maximised

Tie-Breaking

a b c d	a b	b c	a d	d

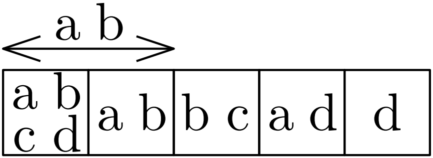
For every row compute all first moves.
(Simple extension of Dijkstra's algorithm)

Tie-Breaking



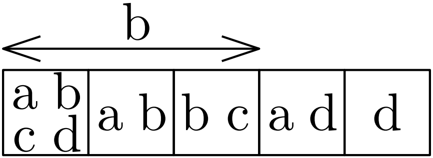
Greedily grow runs from the left to right.

Tie-Breaking



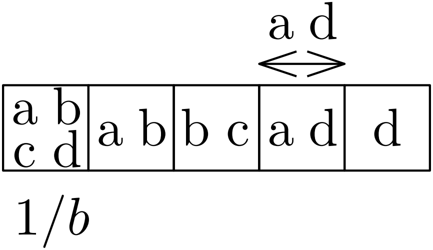
Greedily grow runs from the left to right.

Tie-Breaking



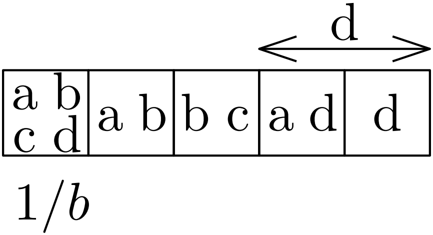
Greedily grow runs from the left to right.

Tie-Breaking



Greedily grow runs from the left to right.

Tie-Breaking



Greedily grow runs from the left to right.

Tie-Breaking

a b c d	a b	b c	a d	d

$$1/b \; 3/d$$

This algorithm produces a minimum number of runs.

The column ordering matters

s \ t	1	2	3	4	5	6	7
1	*	e	e	s	s	s	s
2	w	*	e	sw	sw	sw	sw
3	w	w	*	w	w	w	w
4	n	ne	ne	*	s	se	se
5	n	n	n	n	*	e	e
6	nw	nw	nw	nw	w	*	e
7	w	w	w	w	w	w	*

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

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			

CPD has 16 runs

The column ordering matters

s \ t	1	2	3	4	5	6	7
1	*	e	e	s	s	s	s
2	w	*	e	sw	sw	sw	sw
3	w	w	*	w	w	w	w
4	n	ne	ne	*	s	se	se
5	n	n	n	n	*	e	e
6	nw	nw	nw	nw	w	*	e
7	w	w	w	w	w	w	*

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

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				

CPD has 16 runs

s \ t	1	2	4	5	6	7	3
1	*	e	s	s	s	s	e
2	w	*	sw	sw	sw	sw	e
3	w	w	w	w	w	w	*
4	n	ne	*	s	se	se	ne
5	n	n	n	*	e	e	n
6	nw	nw	nw	w	*	e	nw
7	w	w	w	w	w	*	w

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

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				

CPD has 20 runs

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

SRC in GPPC-14

Entry	Averaged Query Time over All Test Paths (μs) Slowest Move in Path	First 20 Moves of path	Full Path Extraction	Preprocessing Requirements DB Size (MB)	Time (Minutes)
†RA*	282 995	282 995	282 995	**0**	**0.0**
BLJPS	14 453	14 453	14 453	20	0.2
JPS+	7 732	7 732	7 732	947	1.0
BLJPS2	7 444	7 444	7 444	47	0.2
†RA*-Subgoal	1 688	1 688	1 688	264	0.2
JPS+ Bucket	1 616	1 616	1 616	947	1.0
BLJPS2_Sub	1 571	1 571	1 571	524	0.2
NSubgoal	773	773	773	293	2.6
CH	362	362	362	2 400	968.8
SRC-dfs	145	145	**145**	28 000	11649.5
SRC-dfs-i	**1**	**4**	189	28 000	11649.5

SRC-dfs, SRC-dfs-i: two versions of our implemented program

CPD Extensions

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]

CPD Extensions

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.

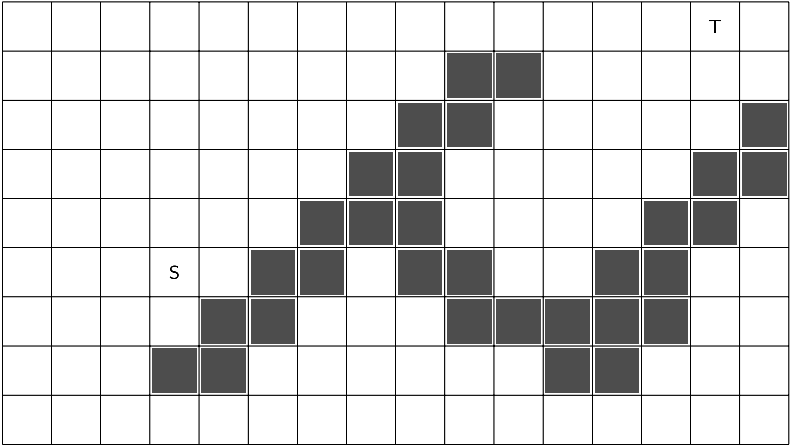
Technical details appear in [Salvetti *et al.*, 2018].

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:

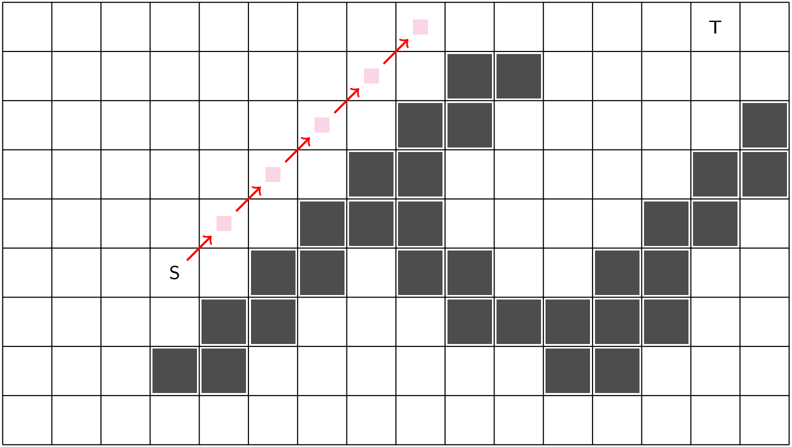
- Ask the CPD which is the next direction to take.
- Ask the JPS database how many steps until the next turning point in the given direction (equiv. the next jump point).

Example

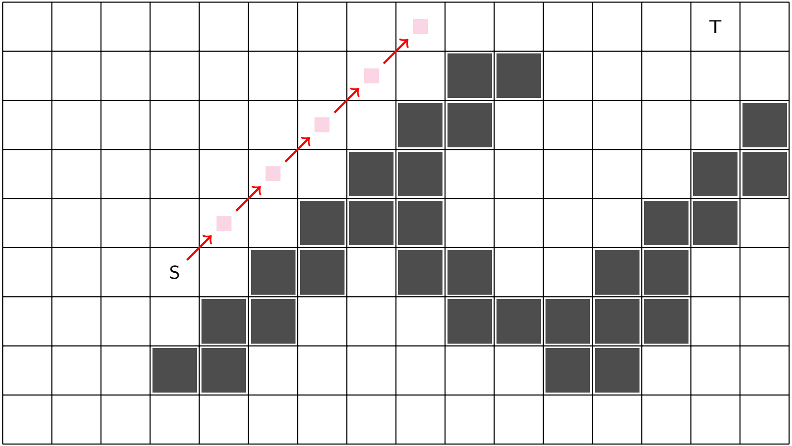


- CPD Oracle: use move $\nearrow$
- JPS+ Oracle: repeat that move for 5 times

Example

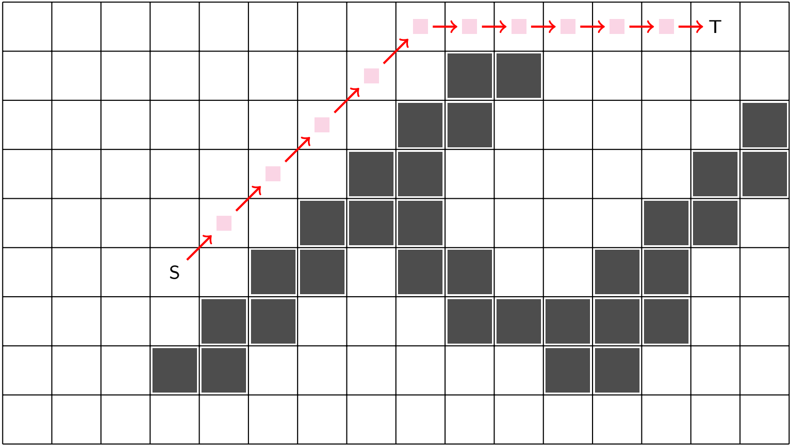


Example



- CPD Oracle: use move →
- JPS+ Oracle: repeat that move for 6 times

Example



- 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

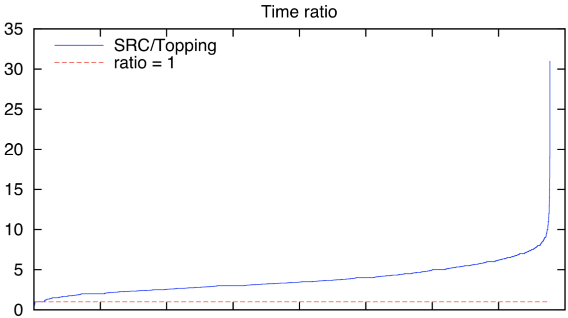
Benchmarks

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

Path length	CPU time A*	CPU time SRC	CPU time JPS+	CPU time Topping	Speedup w.r.t. A*	Speedup w.r.t. SRC	Speedup w.r.t. JPS+
$[0, 150]$	252	4.7	8.2	**1.6**	157	2.9	5.1
$(150, 300]$	1948	9.7	41.5	**2.8**	676	3.3	14.4
$(300, 500]$	5029	14.9	99.8	**4.3**	1160	3.4	23.0
$(500, 750]$	9420	22.6	184.0	**6.2**	1502	3.6	29.4
$(750, 1200]$	17024	35.2	437.0	**6.3**	2664	5.5	63.4
≥ 1200	23039	63.1	527.0	**14.9**	1546	4.2	35.4

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.

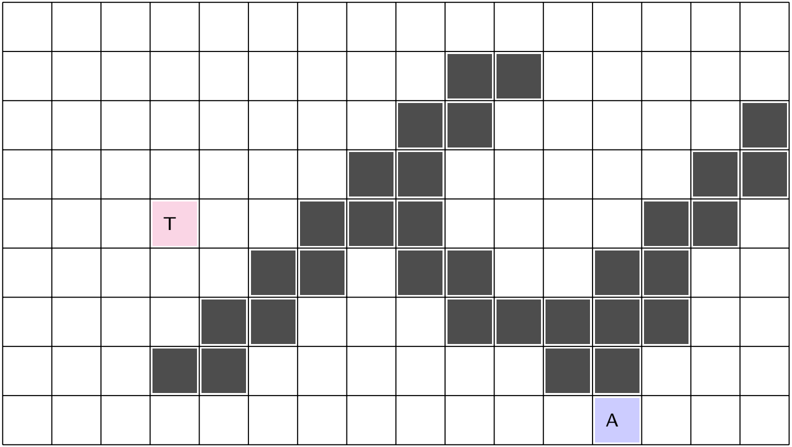
Stats Across All Maps and Queries

Systems	Mem	CPU time [μsec]		Speedup of Topping	
	[MB]	Full path	20 mov.	Full path	20 mov.
SRC	10.40	25.10	3.95	3.84	6.41
JPS+	**3.99**	216.00	216.00	33.00	355.00
Topping	23.00	**6.54**	**0.61**	–	–

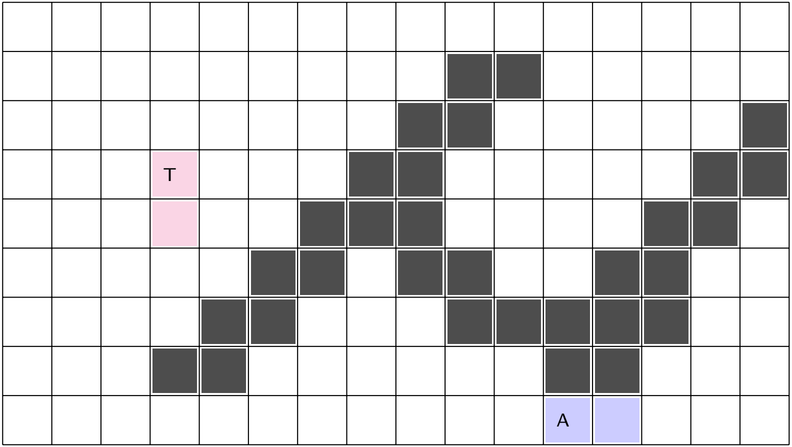
Table: Average memory, average CPU time to compute the full path, average time to compute the first 20 moves of SRC, JPS, and Topping, and average speedup factor of Topping w.r.t. SRC, JPS+ for all maps and queries.

- Compressed Path Databases in Moving-Target Search

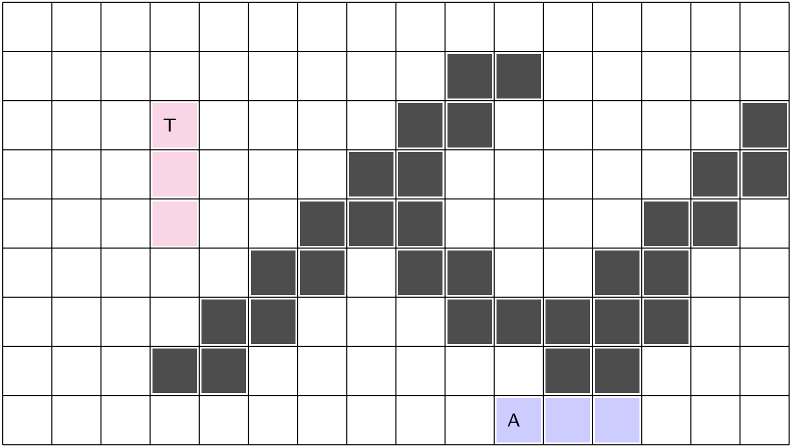
Moving target search (MTS)



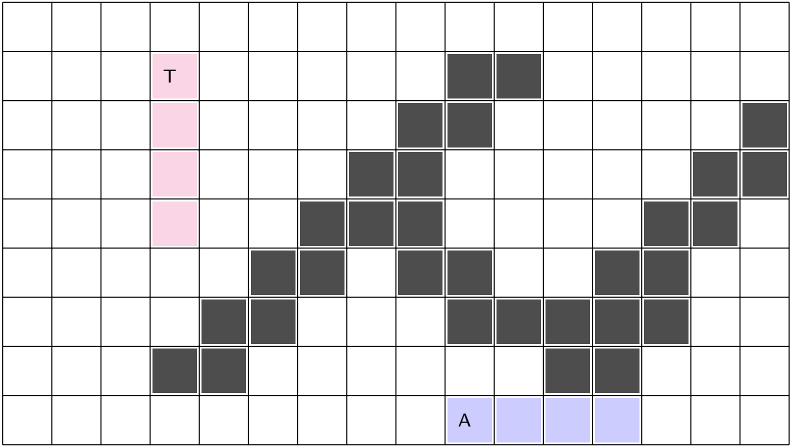
Moving target search (MTS)



Moving target search (MTS)



Moving target search (MTS)



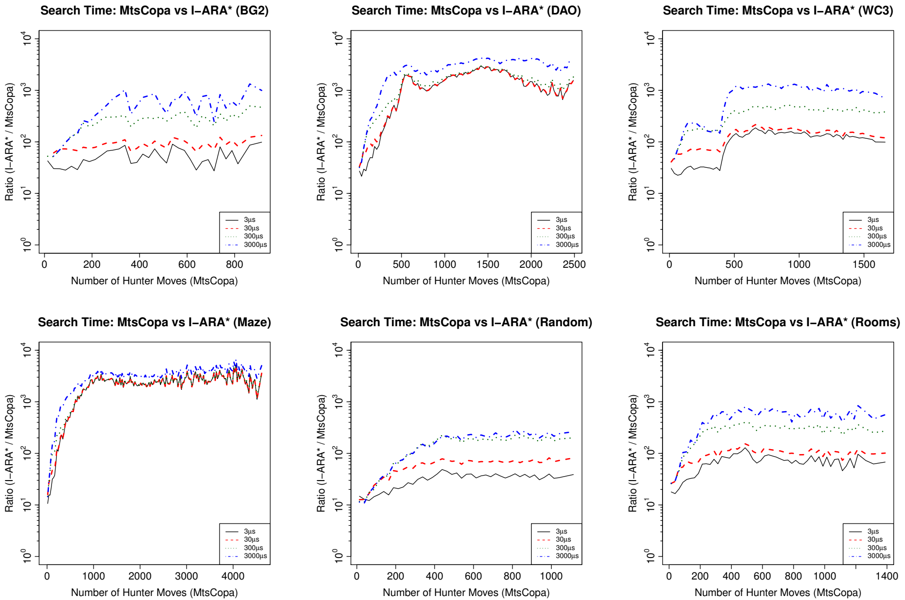
MtsCopa

- 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].

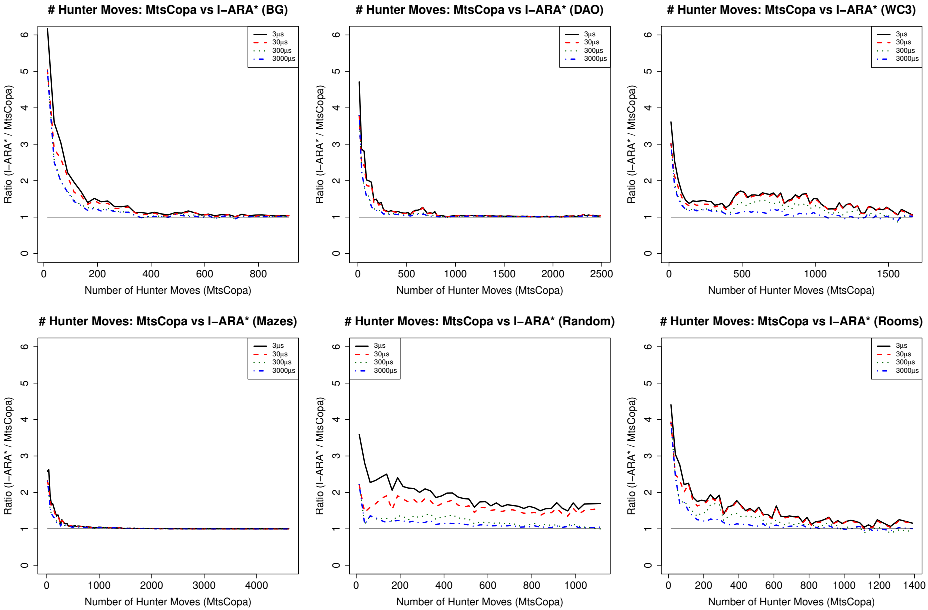
Evaluating MtsCopa

- 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



Solution length (hunter moves)



EOF

References I

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.
In *Proceedings of the SIAM Conference on Algorithm Engineering & Expermiments (ALENEX)*, pages 97–105, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

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.

References II

Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner.
Search-Space Size in Contraction Hierarchies.
In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7965 of *LNCS*, pages 93–104. Springer, 2013.

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.
In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2013.

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.

References III

Michael L. Cui, Daniel Harabor, and Alban Grastien.
Compromise-free Pathfinding on a Navigation Mesh.
In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

R. Geisberger, P. Sanders, D. Schultes, and D. Delling.
Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks.
In *WEA*, pages 319–333, 2008.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter.
Exact Routing in Large Road Networks Using Contraction Hierarchies.
Transportation Science, 46(3):388–404, August 2012.

Daniel Damir Harabor and Alban Grastien.
Improving Jump Point Search.
In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2014.

References IV

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.
Journal of Artificial Intelligence Research, 56(1):89–118, May 2016.

P. E. Hart, N. J. Nilsson, and B. Raphael.
A Formal Basis for the Heuristic Determination of Minimum Cost Paths.
IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.

Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner.
Theta*: Any-Angle Path Planning on Grids.
In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 1177–1183, 2007.

Matteo Salvetti, Adi Botea, Alessandro Saetti, and Alfonso Gerevini.
Compressed Path Databases with Ordered Wildcard Substitutions.
In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2017.

References V

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.
In *Proceedings of ACM International Workshop on Geographic Information Systems (GIS)*, pages 200–209, 2005.

Sabine Storandt.
Contraction Hierarchies on Grid Graphs.
In *Proceedings of the German Conference on Artificial Intelligence (KI)*, pages 236–247, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

Ben Strasser and Dorothea Wagner.
Graph Fill-In, Elimination Ordering, Nested Dissection and Contraction Hierarchies.
In *Gems of Combinatorial Optimization and Graph Algorithms*, pages 69–82. Springer, 2015.

References VI

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.
Journal of Artificial Intelligence Research, 54:593–629, 2015.

Nathan R Sturtevant, Jason Traish, James Tulip, Tansel Uras, Sven Koenig, Ben Strasser, Adi Botea, Daniel Harabor, and Steve Rabin.
The Grid-based Path Planning Competition: 2014 Entries and Results.
In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2015.

N. Sturtevant.
Benchmarks for Grid-Based Pathfinding.
Transactions on Computational Intelligence and AI in Games, 4(2):144 – 148, 2012.

References VII

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.
In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Dorothea Wagner, Thomas Willhalm, and Christos Zaroliagis.
Geometric containers for efficient shortest-path computation.
J. Exp. Algorithmics, 10, December 2005.

Fan Xie, Adi Botea, and Akihiro Kishimoto.
A Scalable Approach to Chasing Multiple Moving Targets with Multiple Agents.
In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

References VIII

Patrick Zumsteg.
Komprimierte pfaddatenbanken durch lauflangenkodierung von optimal permutierten first-move matrizen.
Bachelorarbeit, Universität Basel, 2016.