Fast and optimal pathfinding with compressed path databases

Adi Botea, IBM Research, Dublin, Ireland[1]

Brescia, May 31st 2016

[1]Parts of this work in collaboration with Jorge Baier, Daniel Harabor, Carlos Hernandez and Ben Strasser

Table of contents

Introduction

Shortest paths have many applications.

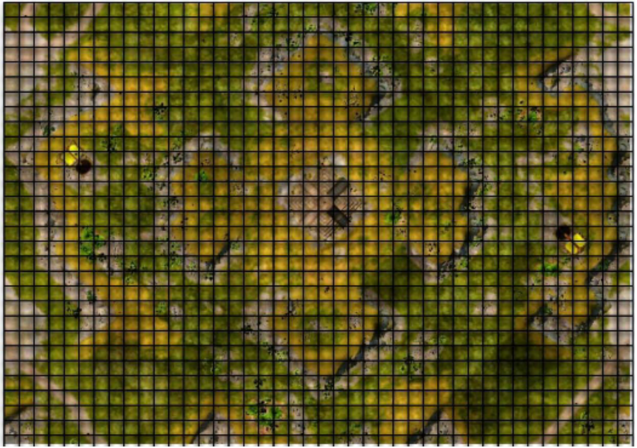
- Games
- Robotics
- Road graphs

In this talk I focus on game maps.

Pathfinding on a game map

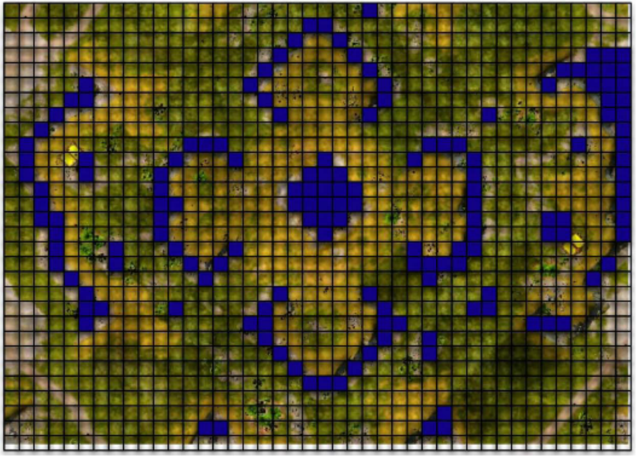
- Convert the map into a search graph
- Search into that graph

Converting the map into a graph



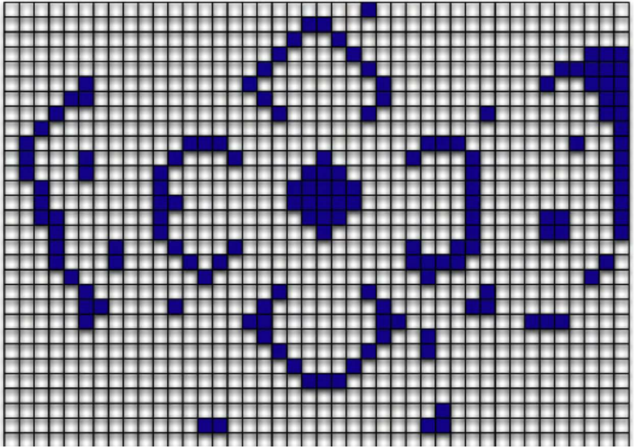
- Gridmaps are a popular choice
- Discretize the map into a two dimensional array

Obstacles and traversable areas



- A cell is either fully traversable or fully blocked
- Blue cells are obstacles in this example

Gridmap as a search graph



- Traversable cells are nodes
- Adjacent cells are connected through an edge
- 4-connected and 8-connected grids are popular choices

Problem Variations

Several variants of the problem setting exist.

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 (called *first-move*)

- Variants 2 & 3 are not only first steps to solve variant 1!
- Consider for example a game unit chasing a moving target. This scenario is better captured by variant 3 than variant 1.

Our algorithm answers first-move-queries (i.e. variant 3).

Textbook solutions

- Search in the graph everytime a new query is posed
- E.g., use A* [HNR68] or Dijkstra's algorithm [Dij59]
- This often is too slow

Our approach

What we do

Precompute and compress All-Pairs Shortest Paths (APSP) data for graphs

Why we do this

APSP can eliminate graph search, providing optimal solutions fast. The memory to cache APSP data is a strong limiting factor. With powerful compression methods, we can extend the use of APSP data to larger graphs.

I present two techniques based on this idea

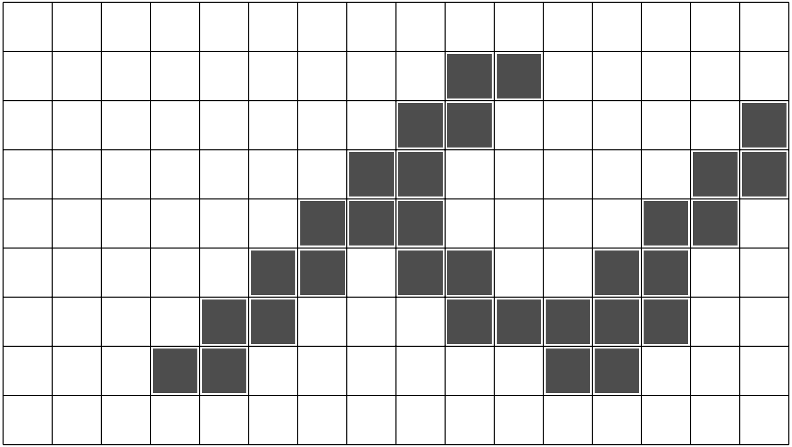
Technique 1: Compression based on rectangular decompositions
Technique 2: Compression based on run-length encoding

Compression with rectangular decompositions

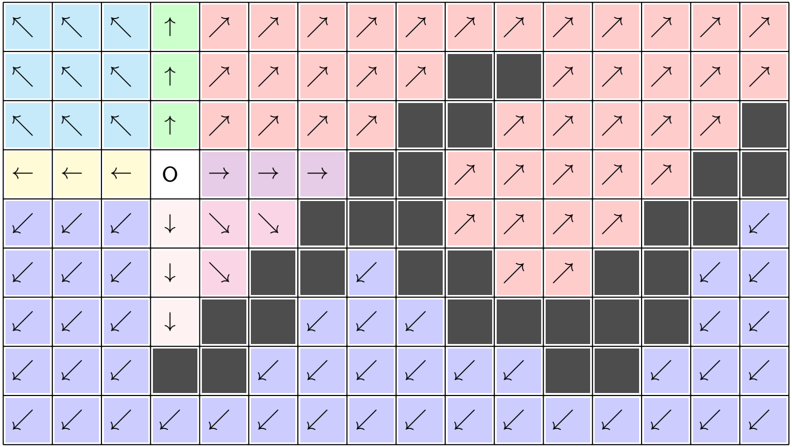
Full technical details in [Bot11, Bot12, BH13].

Copa, a pathfinding system based on these ideas, was a winning entry in the Grid-Based Path Planning Competition, GPPC 2012.

Toy map running example

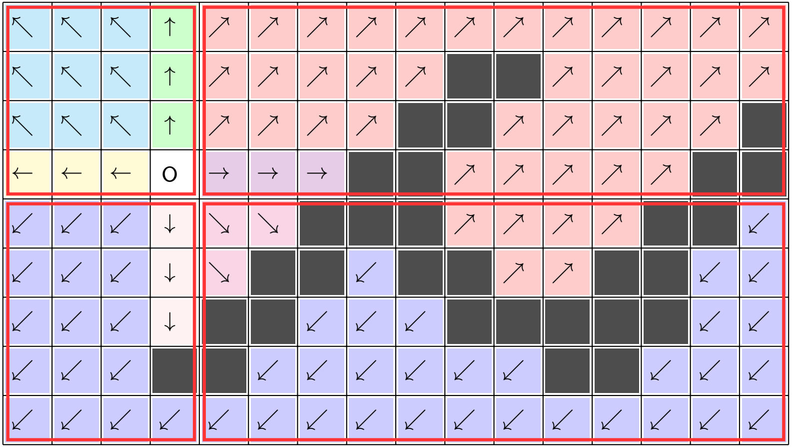


Move table of a given origin node

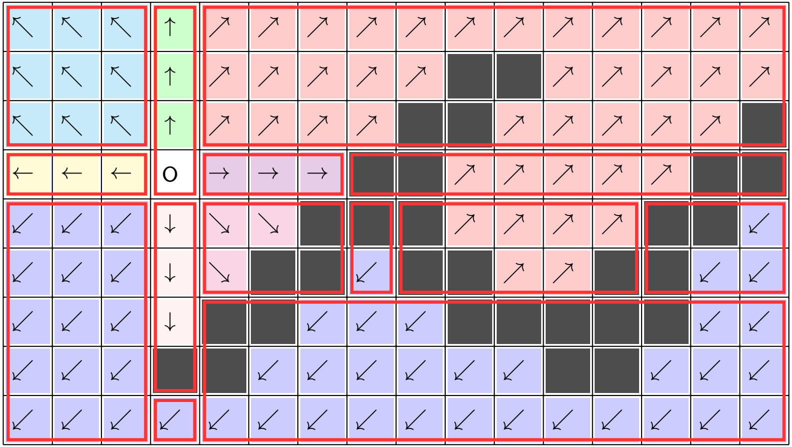


- Assume an origin node O.
- All other nodes are potential targets.
- Many such tables, one for each origin.

The (up to) four sectors around origin



Decomposition into homogeneous rectangles

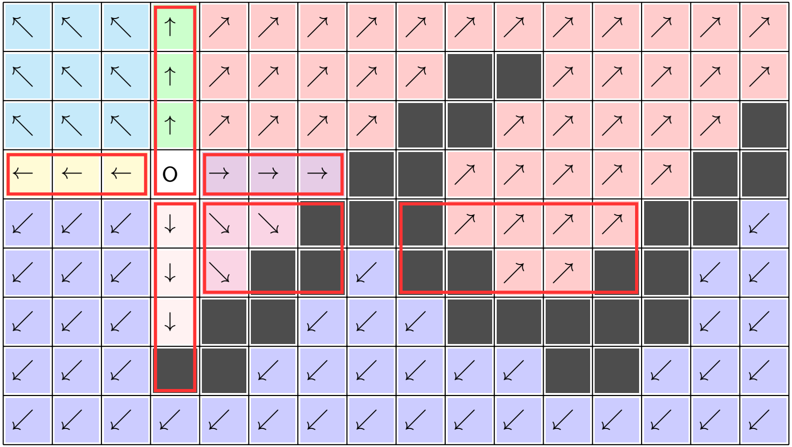


Sector list SL(o, s): list of rectangles for origin node o and sector s. Ordered decreasingly on rectangle “size” (nr contained nodes).

How to retrieve a move

- Given a current node o and a target t
- Identify the rectangle that contains t
  - Identify the sector s containing t
  - Parse list $\mathrm{SL}(o, s)$ until hitting the rectangle that contains t
- The move label of that rectangle is an optimal move from o towards t

List trimming with default moves



Sector *trimmed* list $STL(o, s)$: trimmed list of rectangles for an origin node o and a sector $s \in \{0..3\}$

How to trim a sector list SL(o, s)

- Pick a move label (“default move”)
- Remove 0 or more rectangles with the default move label
- Each removal impacts expected time to retrieve a move, negatively *or* positively

Non-dominated policy on removing rectangles

Go backwards from the end of the list. Don’t skip: remove a rectangle only if all the ones at the right, with the default move label, have been removed. Choose a trimming where the time–memory trade-off is considered acceptable.

The format of the data

- Collection C = concatenation of all STLs
- Index the beginning of each STL, for constant-time access
- Each rectangle has 5 numbers (e.g., left column, width, top row, height, move label)
- Slice C vertically on the 5 columns, obtaining 5 “strings” of symbols.

Collection C

.	.	.	.	.
.	.	.	.	.
3	1	0	4	0
0	3	3	1	6
4	3	3	1	2
8	5	4	2	1
4	3	4	2	3
3	1	4	3	4
.	.	.	.	.
.	.	.	.	.

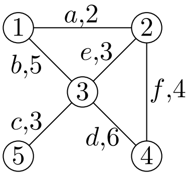
RLE and SWC

- Well known string compression techniques
- RLE: replace “s s s s s t t” with ”1/s 6/t” – you get the idea
- SWC: replace repeated occurrences of a substring with a pointer to an earlier occurrence
- We apply these to each of the 5 strings independently
- We make sure that uncompressing one symbol is done in constant time

Compression based on run-length encoding

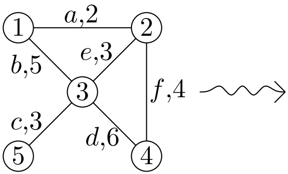
Full technical details in [SHB14, BSH15, STT^{+}15, SBH15].
SRC, a pathfinding system based on these ideas, was the fastest optimal system in the Grid-Based Path Planning Competition, GPPC 2014.

Our Approach



- Input: A weighted graph

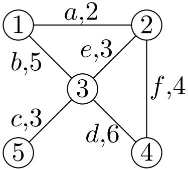
Our Approach



s \ t	1	2	3	4	5
1	–	a	a	a	a
2	a	–	e	f	e
3	e	e	–	d	c
4	f	f	d	–	d
5	c	c	c	c	–

- Input: A weighted graph
- Compute a first-move matrix

Our Approach



s \ t	1	2	3	4	5
1	–	a	a	a	a
2	a	–	e	f	e
3	e	e	–	d	c
4	f	f	d	–	d
5	c	c	c	c	–

1	$1/a$			
2	$1/a$	$3/e$	$4/f$	$5/e$
3	$1/e$	$4/d$	$5/c$	
4	$1/f$	$3/d$		
5	$1/c$			

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

Is this always compact?

- **Problem:** Runs can be very small $\rightarrow$ bad compression
- **Idea:** Reorder the columns to prevent this
- How?

Node-Order

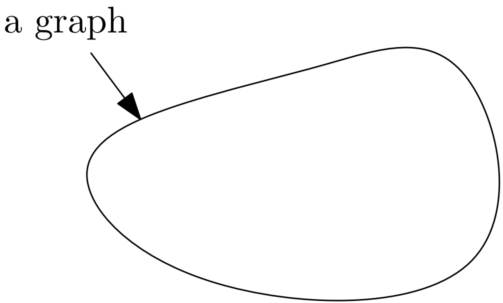
Goal

We want close nodes to have close IDs.

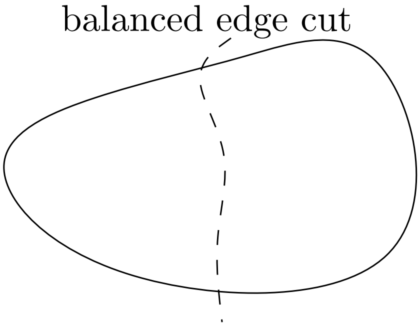
Two ordering-heuristics

- Depth-First Search (idea comes from PHAST [DGNW13])
- 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

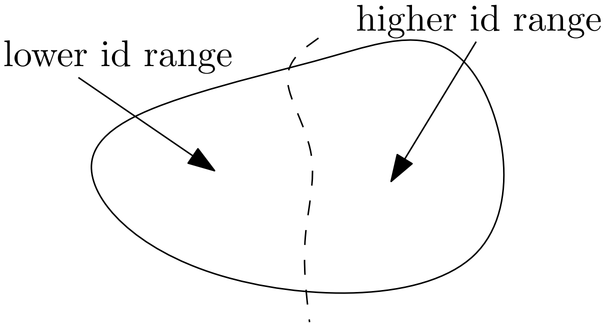
Nested-Edge-Cut-Order



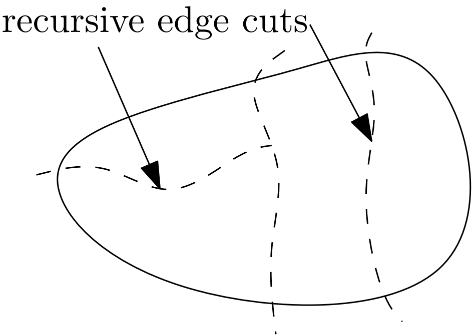
Nested-Edge-Cut-Order



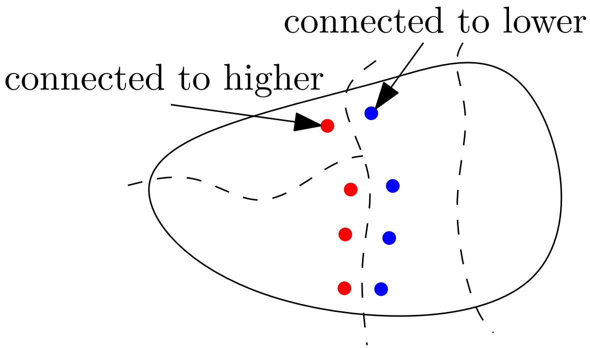
Nested-Edge-Cut-Order



Nested-Edge-Cut-Order

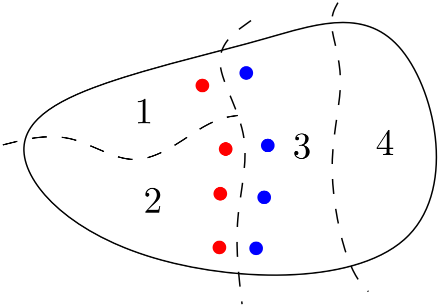


Nested-Edge-Cut-Order



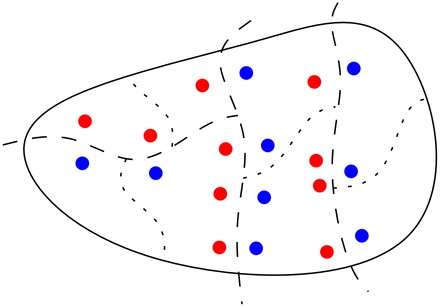
Nested-Edge-Cut-Order

assign IDs in this order



Nested-Edge-Cut-Order

continue recursively



Non-Unique Shortest Paths?

Roads

- On roads shortest paths are unique.

Game Maps

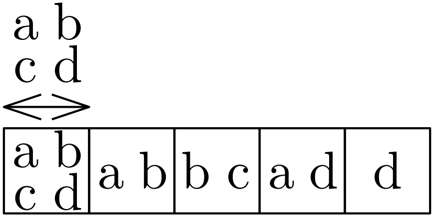
- Game maps often contain unit grids with highly non-unique shortest paths.
- **Idea**: Tie-break paths such that compression is maximized

Tie-Breaking

a b c d	a b	b c	a d	d

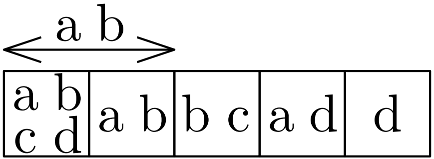
For every row compute all first moves.
(Simple extension of Dijkstra's algorithm)

Tie-Breaking



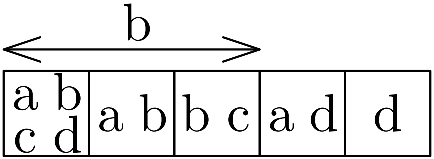
Greedly grow runs from the left to right.

Tie-Breaking



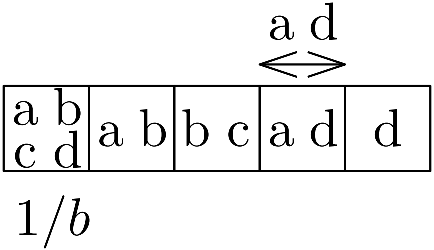
Greedly grow runs from the left to right.

Tie-Breaking



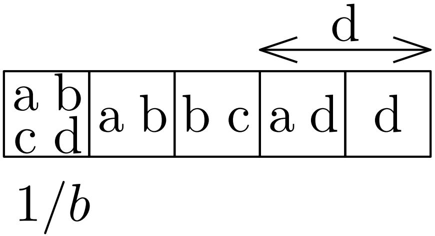
Greedly grow runs from the left to right.

Tie-Breaking



Greedly grow runs from the left to right.

Tie-Breaking



Greedly 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.

Multi-Row-Compression

- **Observation:** Rows of adjacent nodes are similar.
- **Idea:** Store redundancy only once.

- Works, but is only constant tuning.
- In the following:
  - SRC: Single-Row Compression
  - MRC: Multi-Row Compression

Evaluation

- Copa in GPPC-12
- MtsCopa: Copa applied to moving target search [BBHH13, BBHH14]
- SRC in GPPC-14
- Joint evaluation

Copa in GPPC-12

Comparison to optimal entries

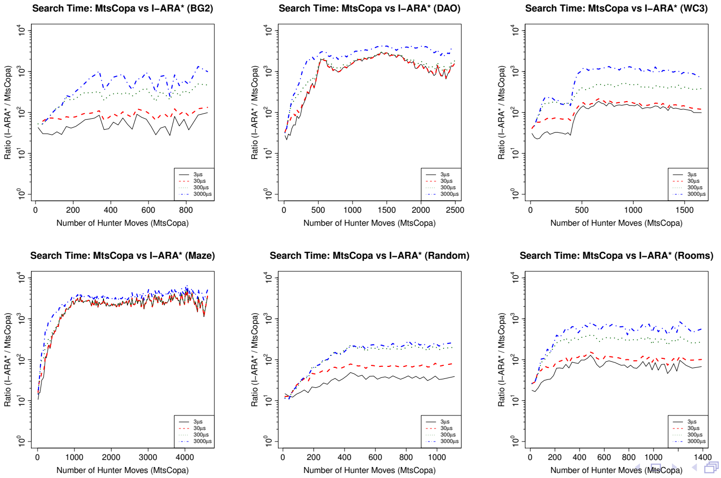
Entry	Year	Total Time (s)	Average Time (ms) (per path)	Average Start Time (ms) (first 20 moves)	Max Time (ms) (per segment)	Problems Solved	RAM Usage (MB) (before)	RAM Usage (MB) (after)	Storage
Compressed Path Databases (move by move)	2012	1348.5	0.798	0.011	0.005	1689740	468.32	571.36	52GB
Compressed Path Databases (20 moves)	2012	1190.0	0.704	0.029	0.683	1689740	468.32	494.77	52GB
SubgoalGraph (variant 2)	2012	1944.0	1.118	1.118	1.118	1739340	17.33	43.78	554MB
SubgoalGraph (variant 1)	2012	2079.5	1.196	1.196	1.196	1739340	15.08	41.53	259MB
Jump Point Search (plus)	2012	36307.5	20.874	20.874	20.874	1739340	356.28	383.05	3.0G
Jump Point Search	2012	108749.9	62.524	62.524	62.524	1739340	252.18	278.97	0
Manhattan Cohesive Areas	2012	130.8	216.235	216.235	216.235	605	267.01	483.27	0

Copa = "Compressed Path Databases"
http://movingai.com/GPPC/detail.html

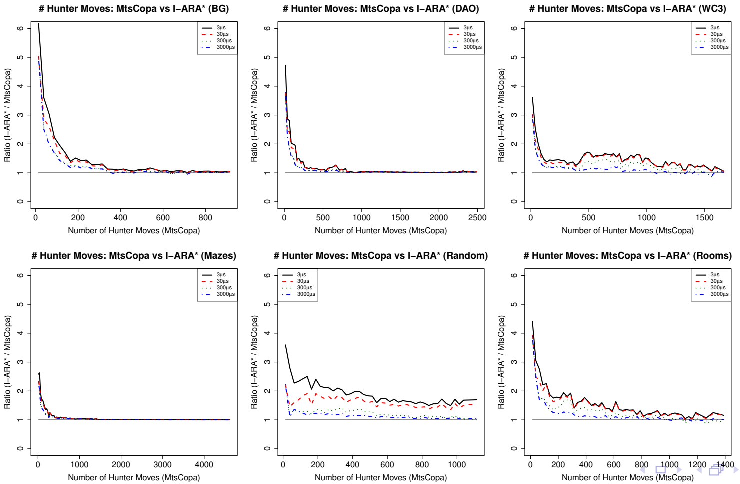
Evaluating MtsCopa

- 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)
  - recent, state-of-the-art MTS solver
  - incremental search, reusing data from previous searches
- 3.47GHz machine, Red Hat Enterprise

Online search time



Solution length (hunter moves)



SRC in GPPC-14

Entry	Averaged Query Time over All Test Paths (μs)			Preprocessing Requirements	
	Slowest Move in Path	First 20 Moves of path	Full Path Extraction	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

Joint evaluation Copa vs SRC vs MRC

Setup:

- Core i7-3770 CPU @ 3.40GHz
- Copa, SRC, and MRC experiments were run on that machine
- HL experiments were scaled

Comparison on game maps

	$\|V\|$	DB Size (MB)				
		Copa	SRC +cut	SRC +dfs	SRC +input	MRC +cut
Dragon Age: Origins (27 maps)						
Med	5341	1	< 1	1	2	< 1
Avg	30740	12	6	8	53	5
Max	99630	75	35	44	349	31
StarCraft (11 maps)						
Med	273500	183	83	126	956	69
Avg	288200	351	172	222	983	148
Max	493700	934	630	660	2947	549

- Game maps were used in GPPC'12

Comparison on game maps

	$\|V\|$	First-Move Query time (ns)				
		Copa	SRC +cut	SRC +dfs	SRC +input	MRC +cut
Dragon Age: Origins (27 maps)						
Med	5341	81	20	31	38	30
Avg	30740	156	25	36	54	34
Max	99630	316	67	78	138	95
StarCraft (11 maps)						
Med	273500	334	66	77	133	95
Avg	288200	358	66	82	132	105
Max	493700	436	108	118	208	197

- Game maps were used in GPPC'12

Comparison vs Hub Labels

	$\|V\|$	Size (MB) Copa	Size (MB) HL	Size (MB) SRC	Query Time (ns) Copa	Query Time (ns) HL	Query Time (ns) SRC
BAY-d	321270	317	90	160	527	488	62
BAY-t		248	65	117	469	371	52
COL-d	435666	586	138	268	677	564	68
COL-t		503	90	192	571	390	58
NY-d	264346	363	99	252	617	621	75
NY-t		342	66	217	528	425	67
ost100d	137375	n/m	62	49	n/m	598	58
FrozenSea	754195	n/m	429	753	n/m	814	104

- n/m = not measured
- HL exploits that games maps are symmetric, SRC does not
- Roads graphs originate from DIMACS Challenge [DGJ09]
- Game maps originate from `http://movingai.com/benchmarks/`

Summary of features

- Very fast response for individual moves
- No first-move lag
- Fast computation of full paths
- Well suited for fixed environments with frequent path invalidation
- Preprocessing time
  - acceptable in many cases, potentially challenging on large maps
  - trivial to run iterations in parallel – linear speed-up
- Memory: acceptable in many cases, potential bottleneck depending on map size and topology

Future work

- New compression ideas
  - Exploiting the fact that many graphs are undirected
  - Efficiently storing borders between clusters
- Lower-memory databases that require a small graph search
- Decentralized vs distributed multi-agent pathfinding
- Collision avoidance in multi-agent moving target search

References I

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.

J. Baier, A Botea, D. Harabor, and C. Hernández.
A fast algorithm for catching a prey quickly in known and partially known game maps.
Computational Intelligence and AI in Games, IEEE Transactions on, PP(99), 2014.

Adi Botea and Daniel Harabor.
Path planning with compressed all-pairs shortest paths data.
In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*. AAAI Press, 2013.

Adi Botea.
Ultra-fast optimal pathfinding without runtime search.
In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'11)*, pages 122–127. AAAI Press, 2011.

Adi Botea.
Fast, optimal pathfinding with compressed path databases.
In *Proceedings of the Symposium on Combinatorial Search (SoCS'12)*, 2012.

References II

Adi Botea, Ben Strasser, and Daniel Harabor.
Complexity results for compressing optimal paths.
In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press, 2015.

Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors.
The Shortest Path Problem: Ninth DIMACS Implementation Challenge, volume 74 of *DIMACS Book*.
American Mathematical Society, 2009.

Daniel Delling, Andrew V. Goldberg, Andreas Nowatzyk, and Renato F. Werneck.
PHAST: Hardware-accelerated shortest path trees.
Journal of Parallel and Distributed Computing, 73(7):940–952, 2013.

Edsger W. Dijkstra.
A note on two problems in connexion with graphs.
Numerische Mathematik, 1:269–271, 1959.

Peter E. Hart, Nils Nilsson, and Bertram Raphael.
A formal basis for the heuristic determination of minimum cost paths.
IEEE Transactions on Systems Science and Cybernetics, 4:100–107, 1968.

Ben Strasser, Adi Botea, and Daniel Harabor.
Compressing optimal paths with run length encoding.
J. Artif. Intell. Res. (JAIR), 54:593–629, 2015.

References III

Ben Strasser, Daniel Harabor, and Adi Botea.
Fast first-move queries through run-length encoding.
In *Proceedings of the 5th International Symposium on Combinatorial Search (SoCS'14)*. AAAI Press, 2014.

Nathan 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 6th International Symposium on Combinatorial Search (SoCS'15)*. AAAI Press, 2015.