Online Graph Pruning for Pathfinding on Grid Maps

Daniel Harabor and Alban Grastien, AAAI 2011
Presented by James Walker

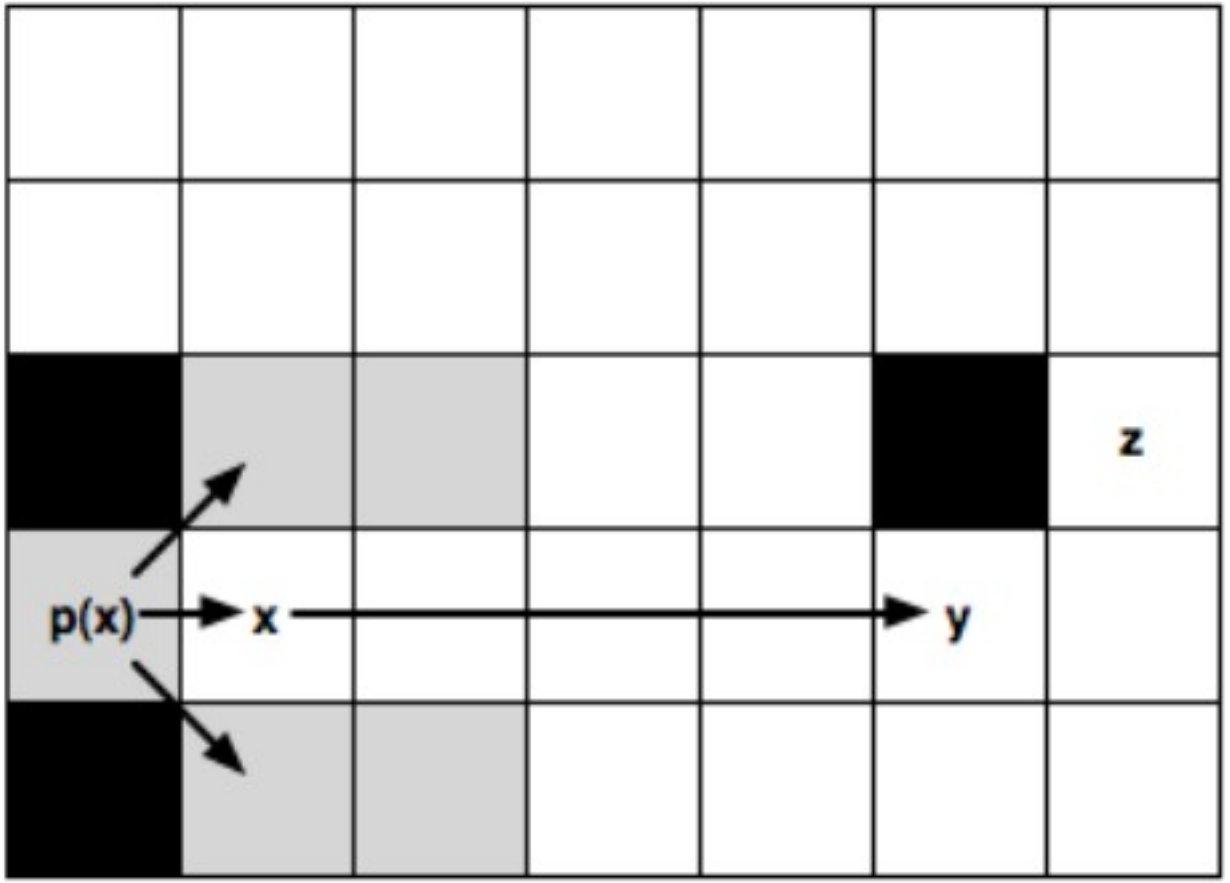
Synopsis

- An algorithm for improving A* performance on uniform-cost grid search spaces
- Works by only expanding nodes called “jump points”
- In benchmarks, improves A* performance dramatically and outperforms other A* optimizations including hierarchical pathfinding

Characteristics

- Search performance dramatically improved
- Guaranteed optimality
- No preprocessing required
- Fairly simple to implement
- Orthogonal to existing techniques (so it can be combined with other optimizations)
- Essentially no drawbacks!!

Example



When moving in a straight line toward y, x dominates all shaded nodes so there's no need to expand them.

Example Cont.

- Repeat until another jump node is encountered
- If blocked by obstacles, conclude that searching in this direction is fruitless and generate nothing

Pruning Rules

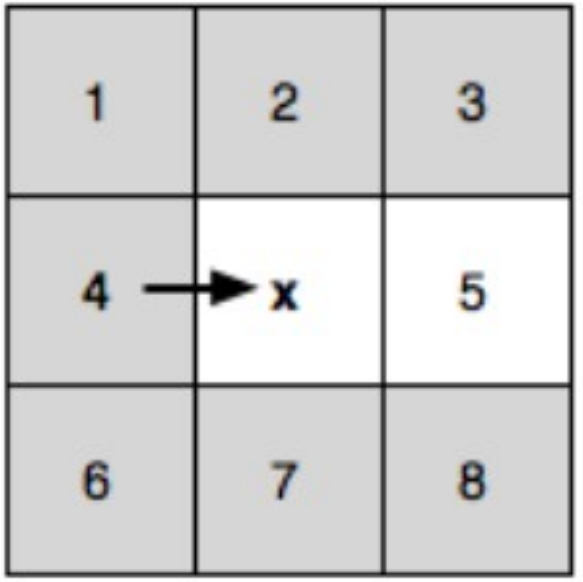
- Goal: For the current node, identify which of its neighbors don't need to be expanded to reach the goal optimally.
- 2 cases: straight move & diagonal move

Straight move pruning

- Prune any node n ε neighors(x) which satisfies the following dominance constraint:

  len(<p(x), …, n> \ x) <= len(<p(x), x, n>)

- Example: Prune all neighbors except 5

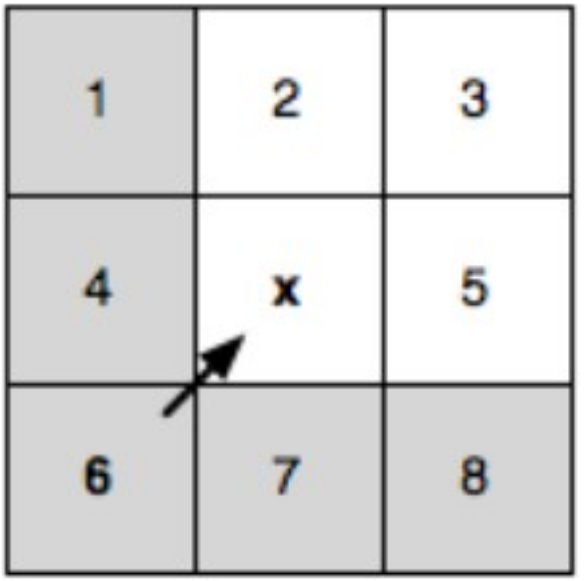


Diagonal move pruning

- Almost identical to previous case, except strict dominance is required:

  len(<p(x), …, n> \ x) < len(<p(x), x, n>)

- Example: Prune all neighbors except 2,3,5

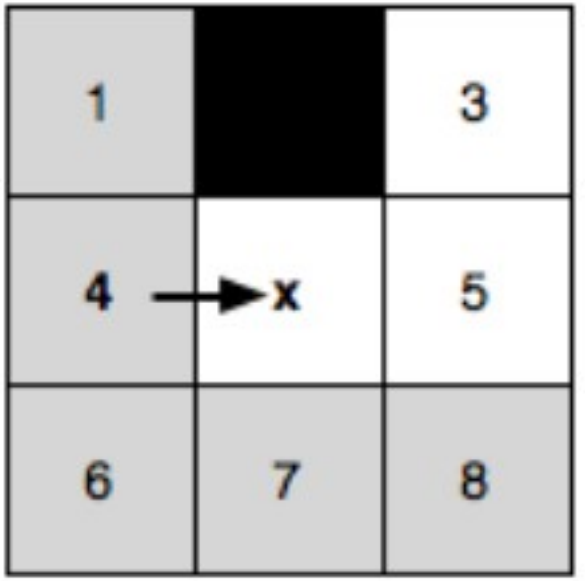


Forced evaluation

- Sometimes an obstacle can force us to evaluate a node we would otherwise prune:

  len(<p(x), x, n>) < len(<p(x), …, n> \ x)

- Example: Evaluation of 3 is forced.



Jump Point Formal Definition

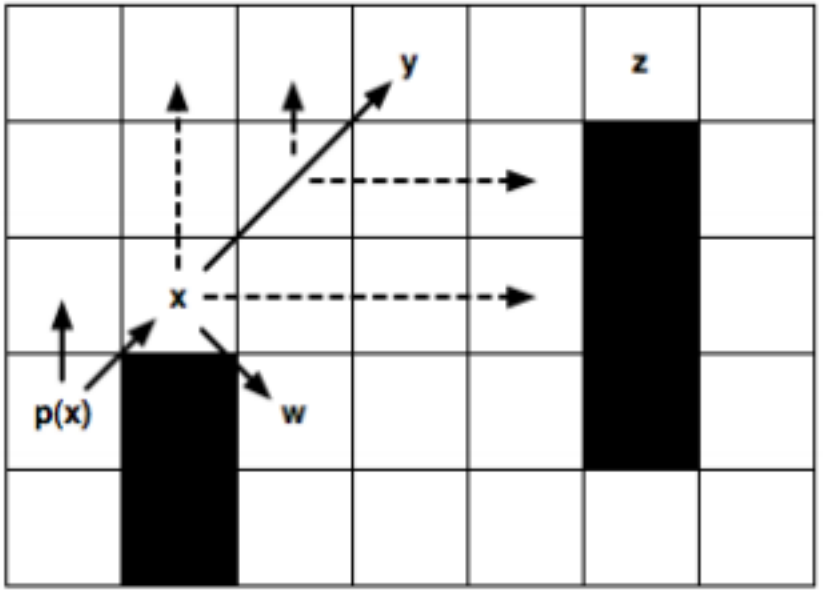
- Node y is a jump point from x, heading in direction d, if y minimizes the value k such that $y = x + kd$ and one of the following conditions holds:

  1. y is the goal

  2. y has at least one neighbor with forced evaluation

  3. d is a diagonal move and there exists a node $z = y + k_i d_i$ which lies $k_i \varepsilon N$ steps in direction $d \varepsilon \{d_1, d_2\}$ s.t. z is a jump point from y by condition 1 or 2.

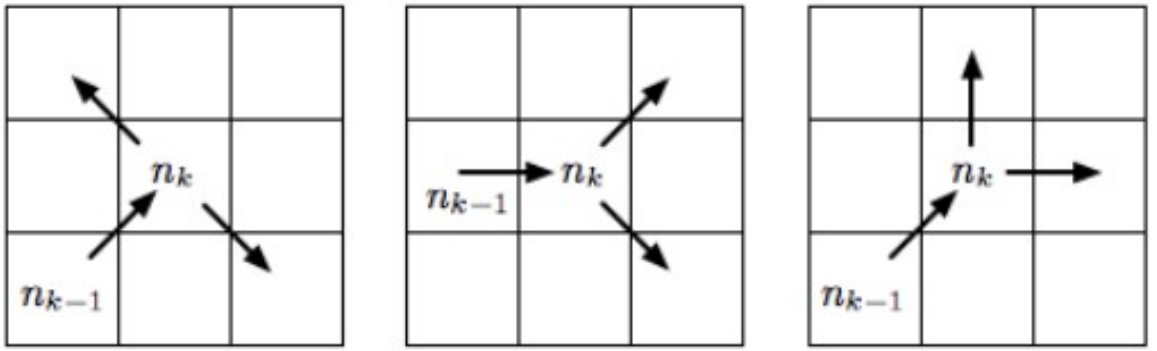
Condition 3 Example



- Intuitively, it's simple: If you're on a diagonal and you turn to a non-diagonal to the next jump point, you yourself are a jump point.

Proof of Optimality

- For each optimal length path in a grid map there exists an equivalent length path which can be found by only expanding jump point nodes.
- Turning points:



- Note that straight-to-straight turning points are trivially suboptimal.

Proof of Optimality Cont.

- **Diagonal-first path:** A path π is diagonal-first if it contains no straight-to-diagonal turning point <n_k-1, n_k, n_k+1> which could be replaced by a diagonal-to-straight turning point <n_k-1, n'_k, n_k+1> s.t. that the length of π remains unchanged.

- **Lemma 1:** Each turning point along an optimal diagonal-first path π' is also a jump point (see paper for proof of lemma).

Proof of Optimality Cont.

- Let π be an arbitrary optimal path and π' a diagonal-first symmetric equivalent. Every turning point in π' is expanded optimally when searching with jump point pruning.
- Divide π' into segments π'_i, where all moves in each π'_i are in the same direction. Every node at beginning and end of π'_i is a turning point.

Proof of Optimality Cont.

- Because each π'_i consists of single-direction moves (straight or diagonal), jump points will travel from the start to end nodes of π'_i w/o necessarily expanding all intermediate nodes; this part is guaranteed to be optimal by the pruning rules.
- **Invoke Lemma 1.** Each start and end node of π'_i is a turning point, thus also a jump point, and is expanded.

Proof of Optimality Concluded

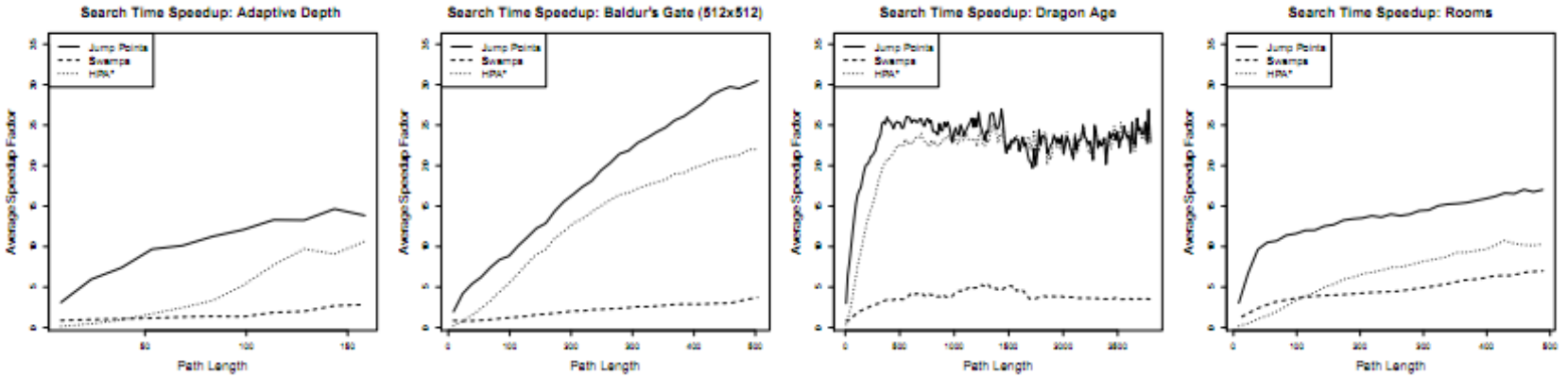
- Because all endpoint nodes of each segment π'_i are expanded optimally, and all straight-line paths between endpoints are optimal, the entire path is optimal.

Experimental Results

Nodes Expanded

	A. Depth	B. Gate	D.Age	Rooms
Jump Points	**20.37**	**215.36**	**35.95**	**13.41**
Swamps	1.89	2.44	2.99	4.70
HPA*	4.14	9.37	9.63	5.11

Computation Time



Summary

- An algorithm for improving A* performance on uniform-cost grid search spaces
- Works by only expanding nodes called “jump points”
- Search performance dramatically improved
- Guaranteed optimality
- No preprocessing required
- Orthogonal to existing techniques (so it can be combined with other optimizations)

Online Graph Pruning for Pathfinding on Grid Maps

Daniel Harabor and Alban Grastien, AAAI 2011
Presented by James Walker

Thank you.
Any questions?