Any-angle path planning

Jiri Dutkevic

Sources

htm

Alex Nash
University of Southern California
http://idm-lab.org/bib/abstracts/Nash12.html

htm

Mihail Pivtoraiko, Maxim Likhachev, Sven Koenig
http://www.seas.upenn.edu/~mihailp/aaai12_trl/index.html

Tools

web

Visual Understanding Environment
http://vue.tufts.edu/

Outline

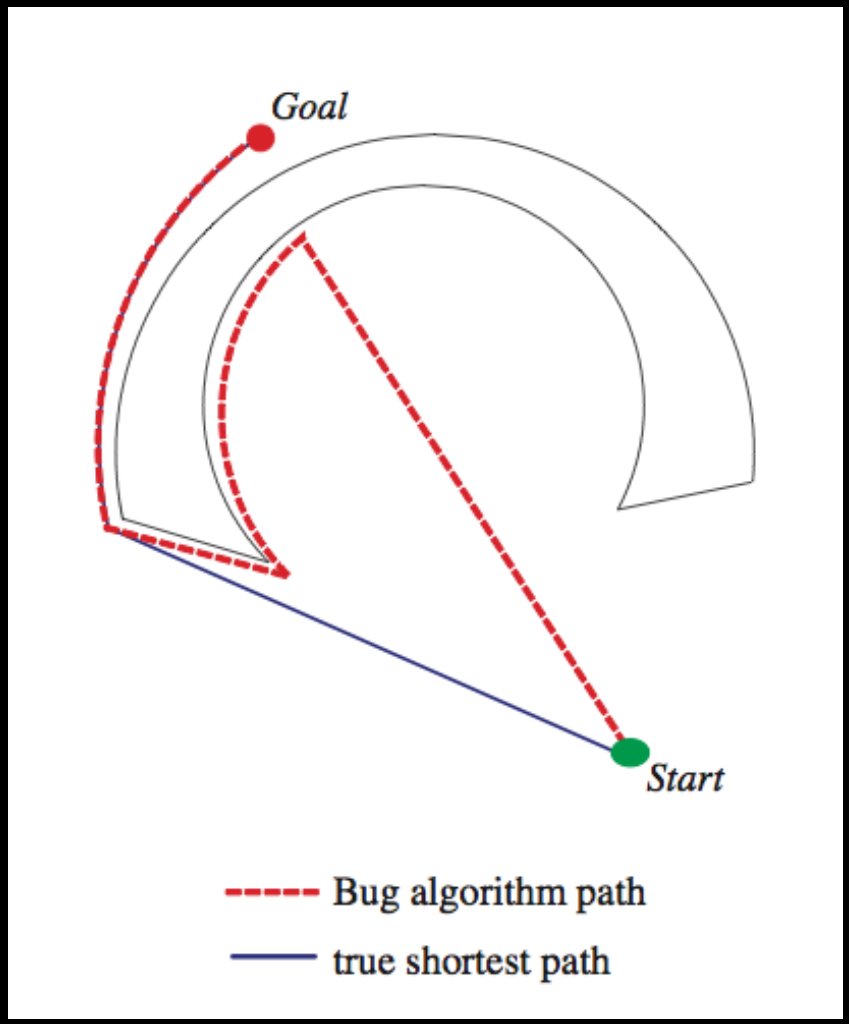
what

why

how

path planning

robotics & games



Generate graph problem

skeletonization

visibility graph

waypoint graph

cell decomposition

regular grids

circle based waypoint graph

navigation meshes

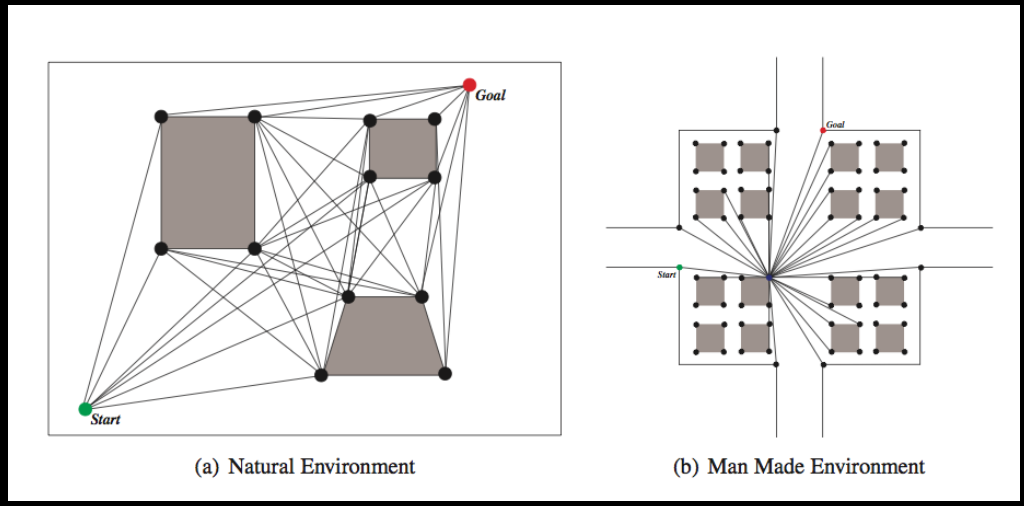
any-angle path planning

skeletonization

visibility graph

waypoint graph

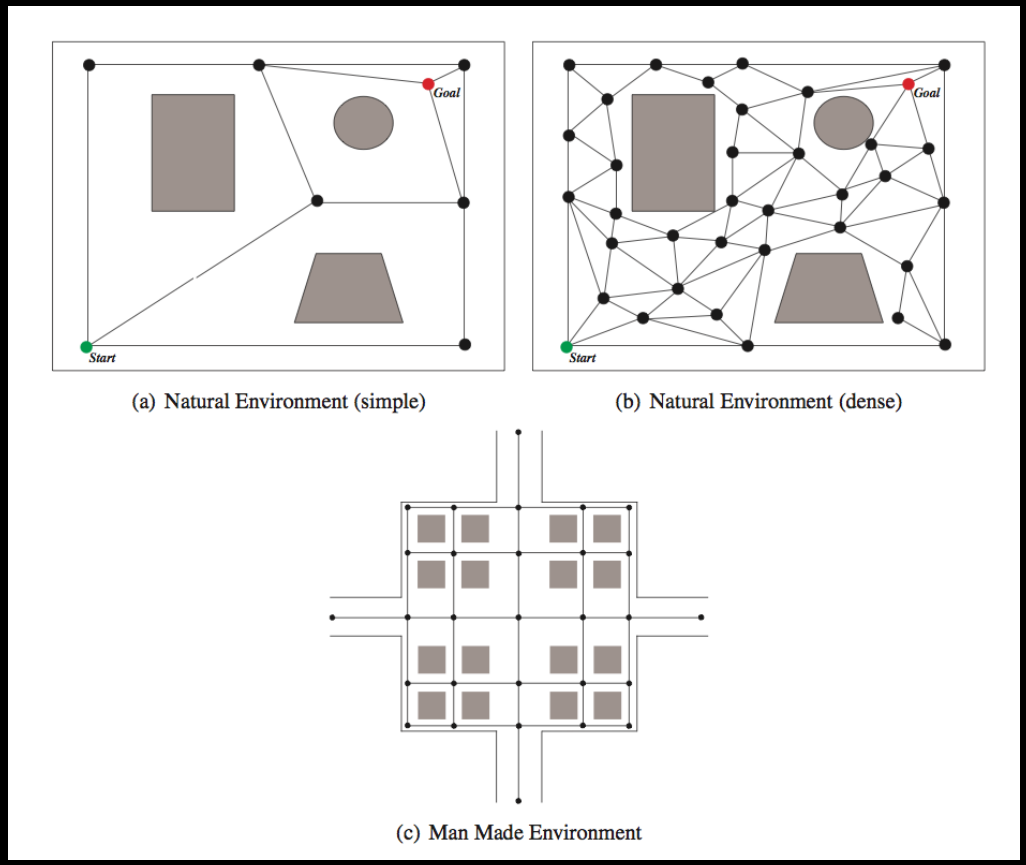
visibility graph



(a) Natural Environment

(b) Man Made Environment

waypoint graph



(a) Natural Environment (simple)

(b) Natural Environment (dense)

(c) Man Made Environment

skeletonization

visibility graph

waypoint graph

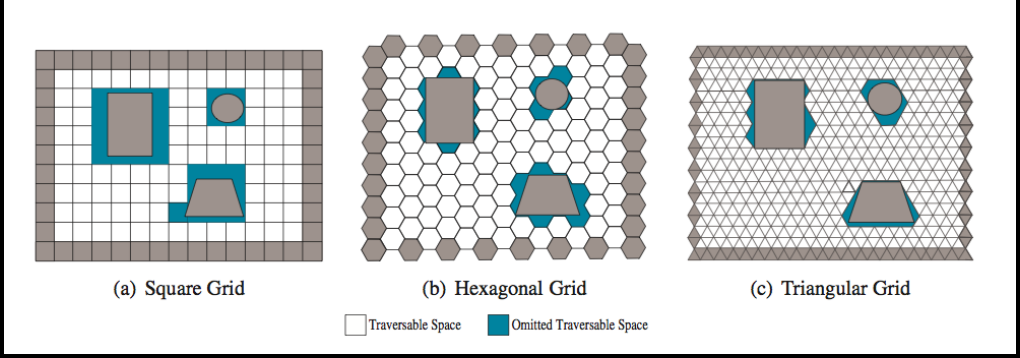
cell decomposition

regular grids

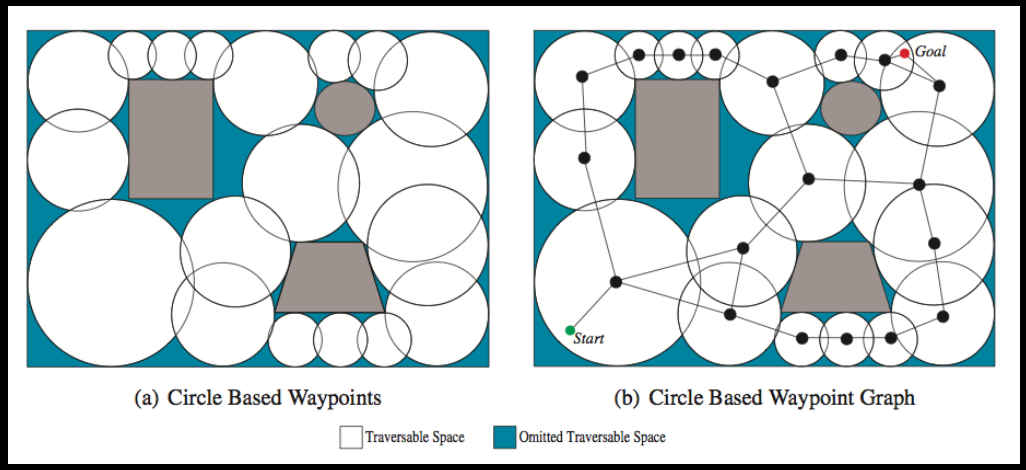
circle based waypoint graph

navigation meshes

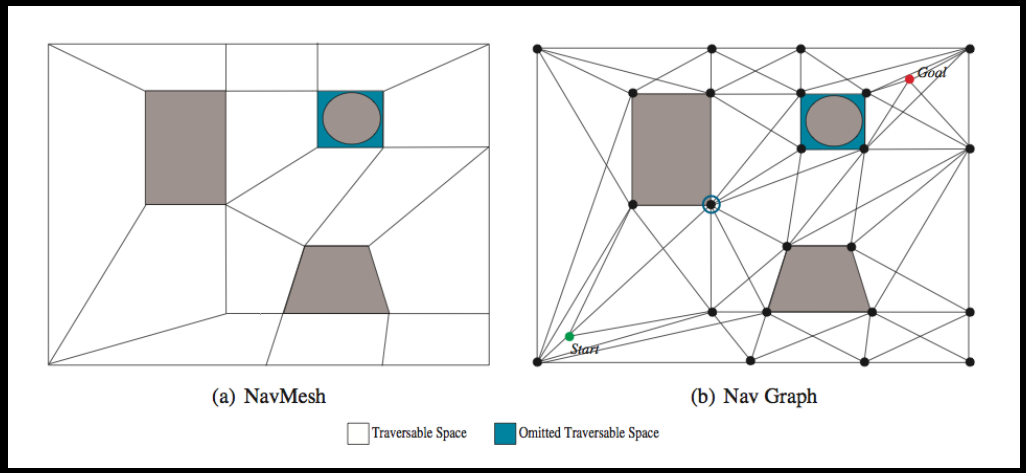
regular grids



circle based waypoint graph



navigation meshes



cell decomposition

regular grids

circle based waypoint graph

navigation meshes

Generate graph problem

skeletonization

visibility graph

waypoint graph

cell decomposition

regular grids

circle based waypoint graph

navigation meshes

Find path problem

desired properties

base algorithms

improved algorithms

desired properties

- simplicity
- efficiency
- generality
- completeness
- correctness
- optimality

base algorithms

single shot

A*

incremental

differential A*

any-angle

A* with Post Smoothing

A*

```
1  Main()
2      g(s_start) := 0;
3      parent(s_start) := s_start;
4      open := ∅;
5      open.Insert(s_start), g(s_start) + h(s_start));
6      closed := ∅;
7      while open ≠ ∅ do
8          s := open.Pop();
9          if s = s_goal then
10             return "path found";
11         closed := closed ∪ {s};
12         /* The following line is executed only by AP Theta*.  */;
13         [UpdateBounds(s)];
14         foreach s' ∈ nghbr_vis(s) do
15             if s' ∉ closed then
16                 if s' ∉ open then
17                     parent(s') := NULL;
18                     g(s') := ∞;
19                 UpdateVertex(s, s');
20     return "no path found";
21 UpdateVertex(s,s')
22     g_old := g(s');
23     ComputeCost(s, s');
24     if g(s') < g_old then
25         if s' ∈ open then
26             open.Remove(s');
27         open.Insert(s', g(s') + h(s'));
28 ComputeCost(s,s')
29     /* Path 1 */
30     if g(s) + c(s, s') < g(s') then
31         parent(s') := s;
32         g(s') := g(s) + c(s, s');
```

Heuristic

Straight Line

2D Grid Path

3D Grid Path

2D Grid Path

```
33 PostSmoothPath([s_0, ..., s_n])
34     k := 0;
35     t_k := s_0;
36     foreach i := 1 ... n − 1 do
37         if NOT LineOfSight(t_k, s_{i+1}) then
38             k := k + 1;
39             t_k := s_i;
40     k := k + 1;
41     t_k := s_n;
42     return [t_0, ..., t_k];
```

3D Grid Path

```
92 h(s)
93     Δ_x := |s.x − s_goal.x|;
94     Δ_y := |s.y − s_goal.y|;
95     Δ_z := |s.z − s_goal.z|;
96     largest := Max(Δ_x, Δ_y, Δ_z);
97     middle := Middle(Δ_x, Δ_y, Δ_z);
98     smallest := Min(Δ_x, Δ_y, Δ_z);
99     return √3 · smallest + √2 · (middle − smallest) + (largest − middle);
```

base algorithms

single shot

A*

incremental

differential A*

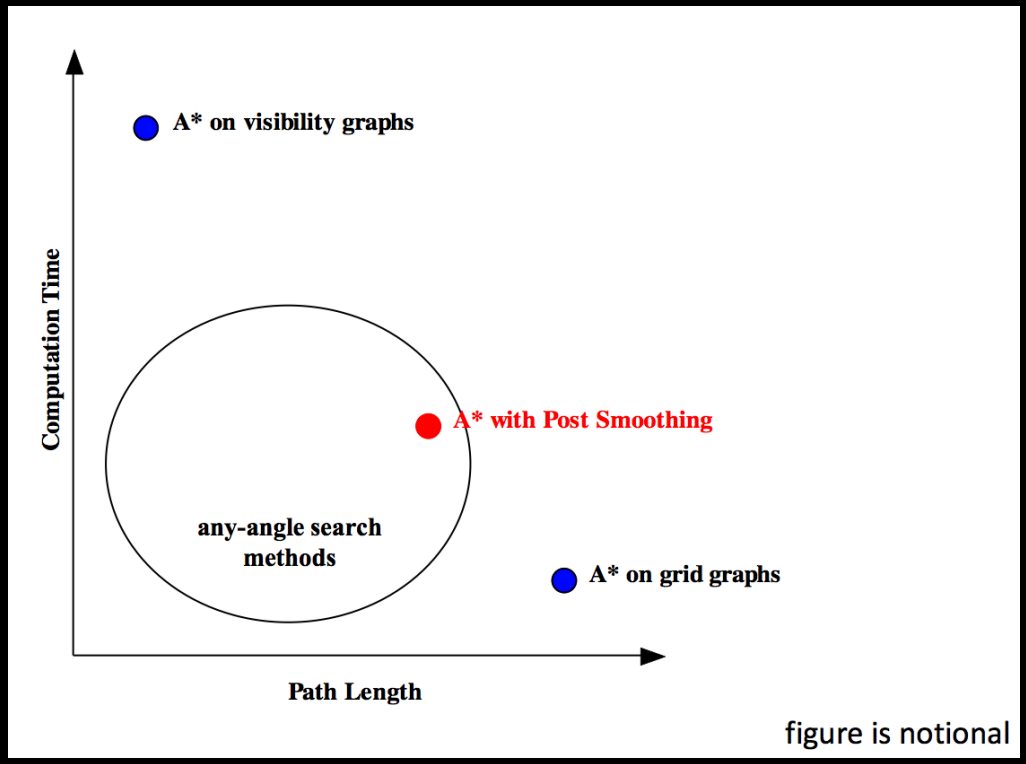
any-angle

A* with Post Smoothing

A* PS

```
33 PostSmoothPath([s_0, ..., s_n])
34     k := 0;
35     t_k := s_0;
36     foreach i := 1 ... n − 1 do
37         if NOT LineOfSight(t_k, s_{i+1}) then
38             k := k + 1;
39             t_k := s_i;
40     k := k + 1;
41     t_k := s_n;
42     return [t_0, ..., t_k];
```

Comparison


A* on visibility graphs
Computation Time
A* with Post Smoothing
any-angle search methods
A* on grid graphs
Path Length
figure is notional

improved algorithms

known 2D environment

Basic Theta*

Angle Propagation Theta*

known 3D environment

Lazy Theta*

Lazy Theta*-R

Lazy Theta*-P

unknown 2D environment

Basic Theta*

Incremental Phi*

known 2D environment

Basic Theta*

Angle Propagation Theta*

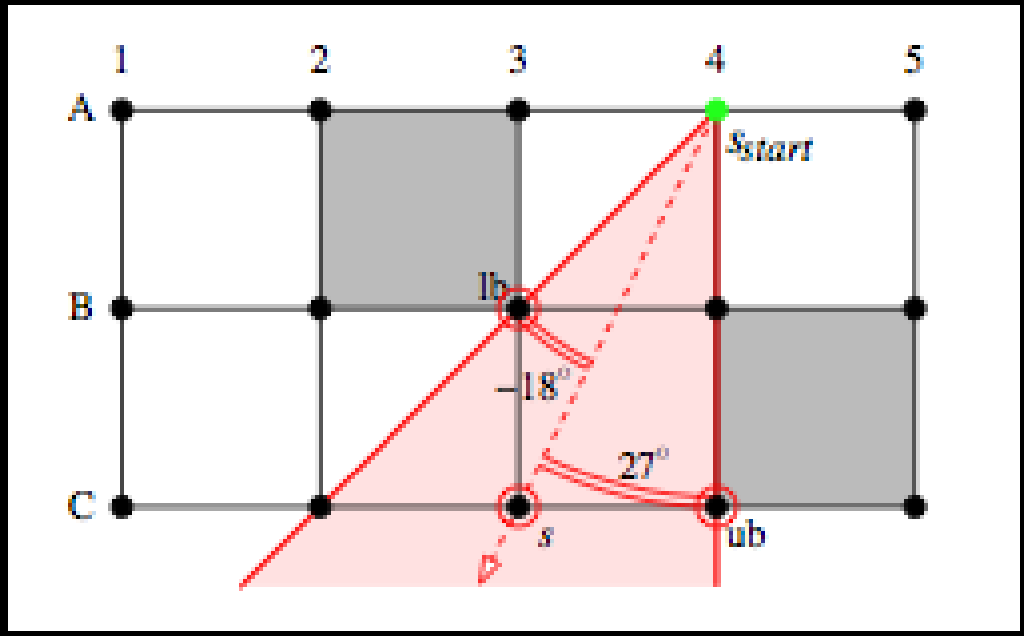
Basic Theta*

```
43 ComputeCost(s,s')
44     if LineOfSight(parent(s), s') then
45         /* Path 2 */
46         if g(parent(s)) + c(parent(s), s') < g(s') then
47             parent(s') := parent(s);
48             g(s') := g(parent(s)) + c(parent(s), s');
49     else
50         /* Path 1 */
51         if g(s) + c(s, s') < g(s') then
52             parent(s') := s;
53             g(s') := g(s) + c(s, s');
```

desired properties

- simplicity
- efficiency
- generality
- completeness
- correctness
- optimality

AP Theta*



known 3D environment

Lazy Theta*

Lazy Theta*-R

Lazy Theta*-P

Lazy Theta*

```
100 Main()
101     g(s_start) := 0;
102     parent(s_start) := s_start;
103     open := ∅;
104     open.Insert(s_start, g(s_start) + h(s_start));
105     closed := ∅;
106     while open ≠ ∅ do
107         s := open.Pop();
108         SetVertex(s);
109         if s = s_goal then
110             return "path found";
111         closed := closed ∪ {s};
112         foreach s' ∈ nghbr_vis(s) do
113             if s' ∉ closed then
114                 if s' ∉ open then
115                     parent(s') := NULL;
116                     g(s') := ∞;
117                 UpdateVertex(s, s');
118     return "no path found";
119 SetVertex(s)
120     if NOT LineOfSight(parent(s), s) then
121         /* Path 1*/
122         parent(s) := argmin_{s'∈nghbr_vis(s)∩closed}(g(s') + c(s', s));
123         g(s) := min_{s'∈nghbr_vis(s)∩closed}(g(s') + c(s', s));
124 ComputeCost(s, s')
125     /* Path 2 */
126     if g(parent(s)) + c(parent(s), s') < g(s') then
127         parent(s') := parent(s);
128         g(s') := g(parent(s)) + c(parent(s), s');
```

Lazy Theta*-R

```
129 SetVertex(s)
130     if NOT LineOfSight(parent(s), s) then
131         /* Path 1*/
132         parent(s) := argmin_{s' ∈ nghbr_vis(s) ∩ closed} (g(s') + c(s', s));
133         g(s) := min_{s' ∈ nghbr_vis(s) ∩ closed} (g(s') + c(s', s));
134         open.Insert(s, g(s) + h(s));
135         Goto Line 107;
```

Lazy Theta*-P

```
136 SetVertex(s)
137     /* Path 2 */
138     if LineOfSight(parent(parent(s)), s) then
139         parent(s) := parent(parent(s));
140         g(s) := g(parent(s)) + c(parent(s), s);
```

known 3D environment

Lazy Theta*

Lazy Theta*-R

Lazy Theta*-P

unknown 2D environment

Basic Theta*

Incremental Phi*

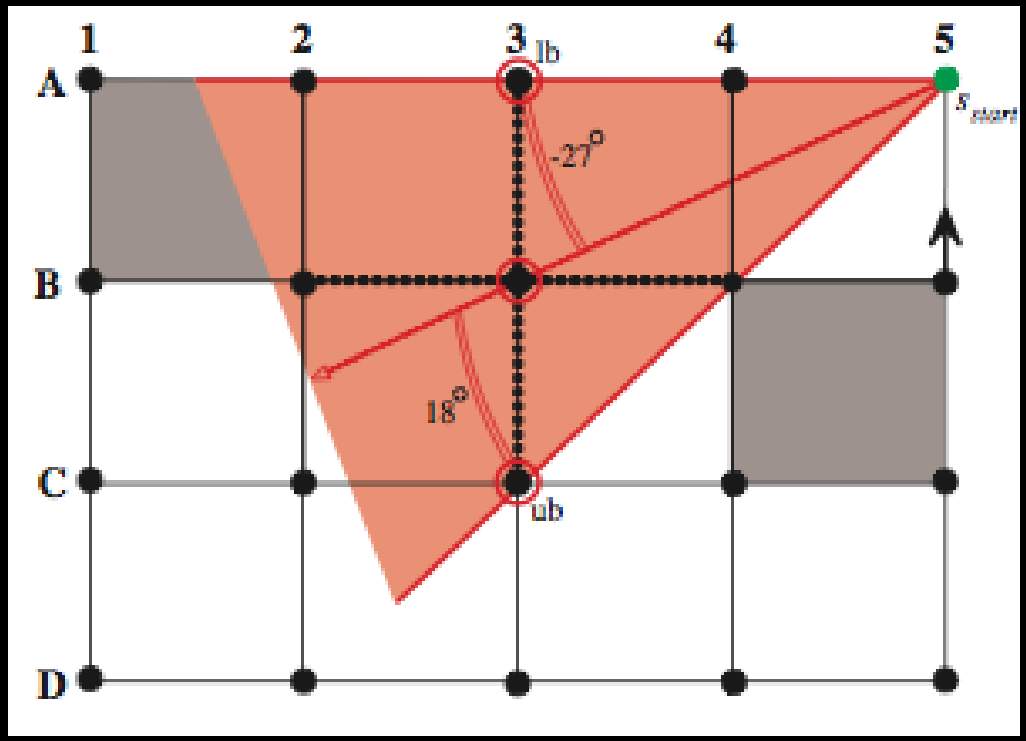
Basic Theta*

```
141 Main()
142     Initialize();
143     ComputeShortestPath();
144     if g(s_goal) ≠ ∞ then
145         return "path found";
146     else
147         return "no path found";

148 Initialize()
149     open := ∅;
150     closed := ∅;
151     InitializeVertex(s_start);
152     InitializeVertex(s_goal);
153     g(s_start) := 0;
154     parent(s_start) := s_start;
155     local(s_start) := s_start;
156     open.Insert(s_start, g(s_start) + h(s_start));

157 InitializeVertex(s)
158     g(s) := ∞;
159     parent(s) := NULL;
160     local(s) := NULL;
161     [lb(s) := −∞];
162     [ub(s) := ∞];

163 ComputeShortestPath()
164     while open.TopKey() < g(s_goal) + h(s_goal) do
165         s := open.Pop();
166         closed := closed ∪ {s};
167         foreach s' ∈ nghbr_vis(s) do
168             if s' ∉ closed then
169                 if s' ∉ open then
170                     InitializeVertex(s');
171                 UpdateVertex(s, s');

172 ComputeCost(s, s')
173     if LineOfSight(parent(s), s') then
174         /* Path 2 */
175         if g(parent(s)) + c(parent(s), s') < g(s') then
176             parent(s') := parent(s);
177             g(s') := g(parent(s)) + c(parent(s), s');
178             local(s') := s;
179     else
180         /* Path 1 */
181         if g(s) + c(s, s') < g(s') then
182             parent(s') := s;
183             g(s') := g(s) + c(s, s');
184             local(s') := s;
```

Phi*

```
185 ComputeCost(s, s')
186     if ∠(s', parent(s)) is not a multiple of 45° and LineOfSight(parent(s), s') and
        Φ(s, parent(s), s') ∈ [lb(s), ub(s)] then
187         /* Path 2 */
188         if g(parent(s)) + c(parent(s), s') < g(s') then
189             parent(s') := parent(s);
190             g(s') := g(parent(s)) + c(parent(s), s');
191             local(s') := s;
192             l := min_{s''∈nghbr4(s')} Φ(s', parent(s), s'');
193             h := max_{s''∈nghbr4(s')} Φ(s', parent(s), s'');
194             δ = Φ(s, parent(s), s');
195             lb(s') := max(l, lb(s) − δ);
196             ub(s') := min(h, ub(s) − δ);
197     else
198         /* Path 1 */
199         if g(s) + c(s, s') < g(s') then
200             parent(s') := s;
201             g(s') := g(s) + c(s, s');
202             local(s') := s;
203             lb(s') := −45°;
204             ub(s') := 45°;
```

Angle Propagation

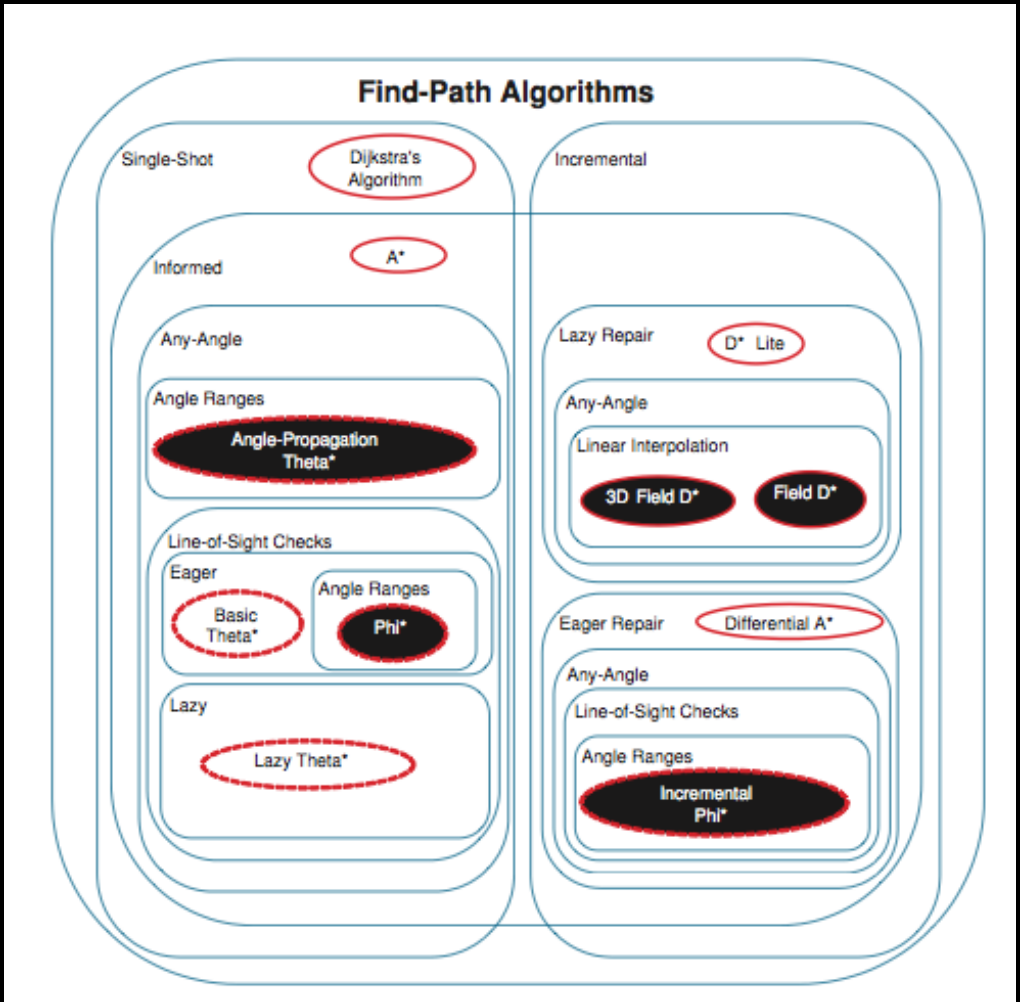


Incremental Phi*

```
205 Main()
206     Initialize();
207     while true do
208         ComputeShortestPath();
209         Wait for grid cells to become blocked;
210         foreach newly blocked grid cell c do
211             Update blockage status of grid cell c to blocked;
212             foreach s' ∈ corners(c) do
213                 if (s' ∈ closed or s' ∈ open) and s' ≠ s_start then
214                     ClearSubtree(s');

215 ClearSubtree(s)
216     under := over := ∅;
217     under.Enqueue(s);
218     while under ≠ ∅ do
219         u := under.Dequeue();
220         over.Enqueue(u);
221         InitializeVertex(u);
222         if u ∈ open then
223             open.Remove(u);
224         if u ∈ closed then
225             closed := closed \ {u};
226         foreach s' ∈ nghbr_8(u) do
227             if local(s') = u then
228                 under.Enqueue(s');
229     while over ≠ ∅ do
230         v := over.Dequeue();
231         foreach s' ∈ nghbr_vis(v) do
232             if s' ∈ closed then
233                 UpdateVertex(s', v);
```

Conslusion

Algorithms


Find-Path Algorithms
Single-Shot
Dijkstra's Algorithm
Incremental
Informed
A*
Any-Angle
Angle Ranges
Angle-Propagation Theta*
Line-of-Sight Checks
Eager
Basic Theta*
Angle Ranges
Phi*
Lazy
Lazy Theta*
Lazy Repair
D* Lite
Any-Angle
Linear Interpolation
3D Field D*
Field D*
Eager Repair
Differential A*
Any-Angle
Line-of-Sight Checks
Angle Ranges
Incremental Phi*

desired properties

- simplicity
- efficiency
- generality
- completeness
- correctness
- optimality

Sources

htm

Alex Nash
University of Southern California
http://idm-lab.org/bib/abstracts/Nash12.html

htm

Mihail Pivtoraiko, Maxim Likhachev, Sven Koenig
http://www.seas.upenn.edu/~mihailp/aaai12_trl/index.html