Visibility Induction for Discretized Pursuit-Evasion Games

Ahmed Abdelkader and Hazem El-Alfy

Engineering Mathematics and Physics Department
Faculty of Engineering, Alexandria University
Alexandria, Egypt

Introduction

- Visibility-based pursuit-evasion:
  - Motion: holomonic, bounded speed per player
  - Visibility: omnidirectional, optional range
  - The game ends when the pursuer loses sight of the evader
- Applications:
  - Surveillance, monitoring, home care, etc.
- Is it possible to keep a given evader in sight? How?

Recurrence Relation

$$\mathbf{Bad(p,e,0)=1} \quad \forall \mathbf{(p,e)} \text{ s.t. } \mathbf{p} \text{ does not see } \mathbf{e}$$

$$\mathbf{Bad(p,e,i+1)=1} \quad \forall \mathbf{(p,e)}\ \exists \mathbf{e'} \in \mathcal{N}\mathbf{(e)} \text{ s.t. } \forall \mathbf{p'} \in \mathcal{N}\mathbf{(p)}\ \mathbf{Bad(p',e',i)=1}$$

The Visibility Induction Algorithm

Input : A map of the environment.
Output: The **Bad** function encoded as a bit matrix.
Data: Two $\mathbf{N} \times \mathbf{N}$ binary matrices $\mathbf{M}$ and $\mathbf{M'}$.

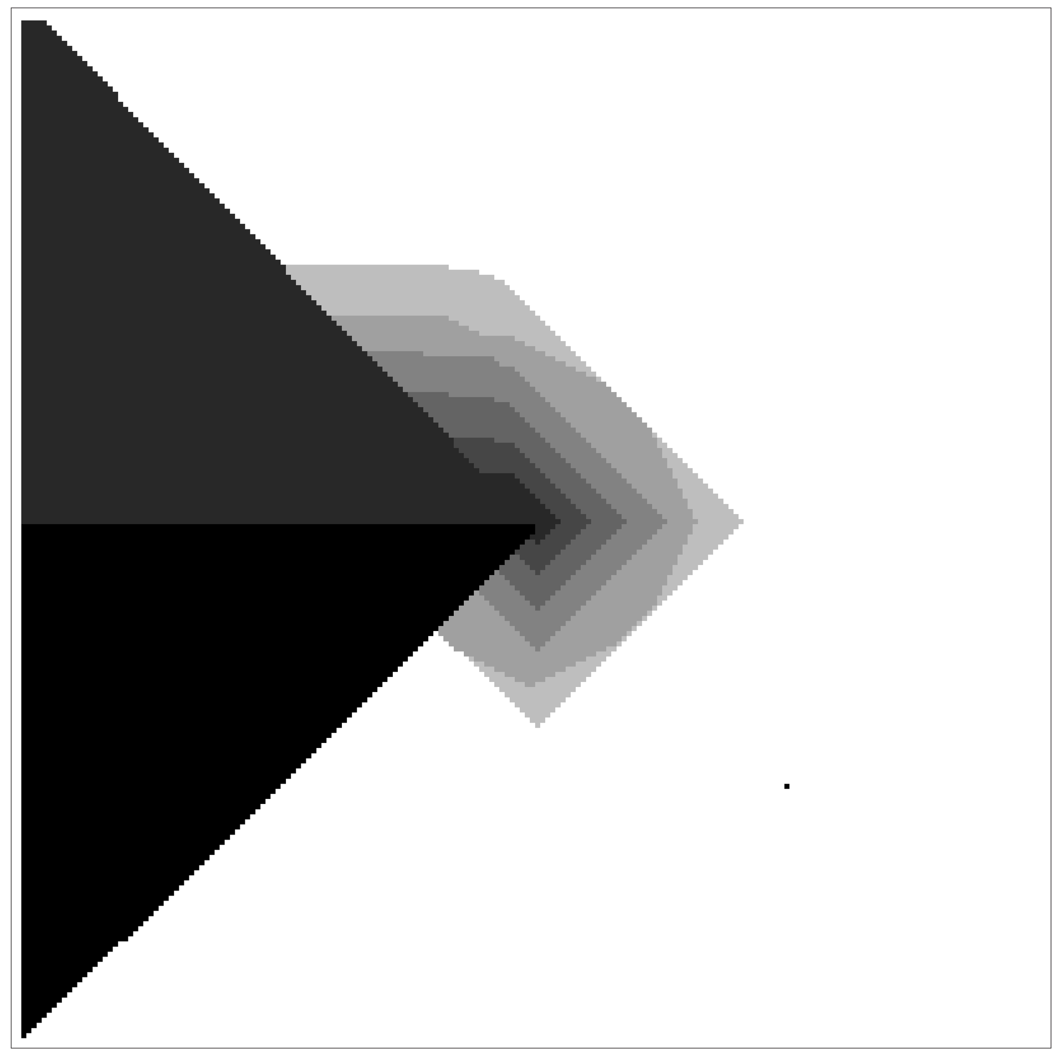
```
begin
  Partition the map into a uniform grid of N cells.
  Initialize M and M' to 0.
  foreach (p, e) ∈ grid × grid do
    if e not visible to p then
      M[p, e] = 1
  while M' ≠ M do
    M' = M
    foreach (p, e) ∈ grid × grid do
      if ∃e' ∈ N(e) s.t.∀p' ∈ N(p) M'[p', e'] = 1 then
        M[p, e] = 1
  return M
```

Optimizations

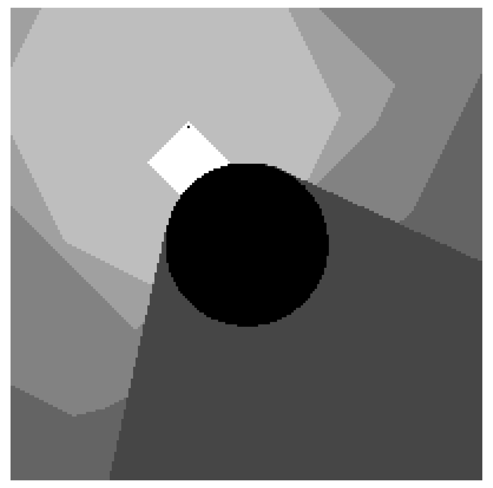
- Memory Savings:
  - 1-bit per entry, no auxiliary matrix.
- Parallelization:
  - $\mathbf{M}$ updates are embarrassingly parallel.
- Caching:
  - Synchronized Neighborhoods

$$\neg\mathbf{Bad(p,e,i)} \wedge \mathbf{Bad(p,e,i+1)} \implies \exists \mathbf{(p^*,e^*)} \in \mathcal{N}\mathbf{(p)} \times \mathcal{N}\mathbf{(e)} \text{ s.t. } \neg\mathbf{Bad(p^*,e^*,i-1)} \wedge \mathbf{Bad(p^*,e^*,i)}$$

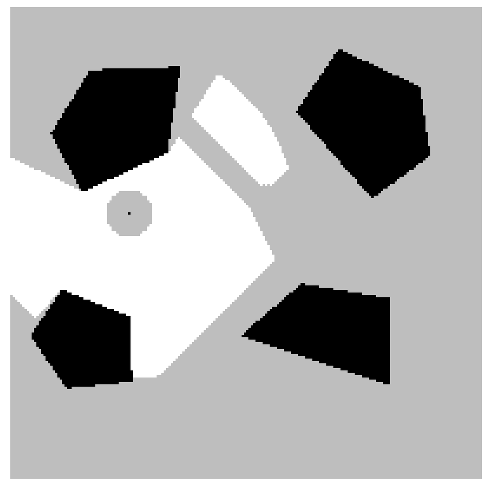
Decision Maps for Different Layouts and Constraints



Infinite Corner (P)

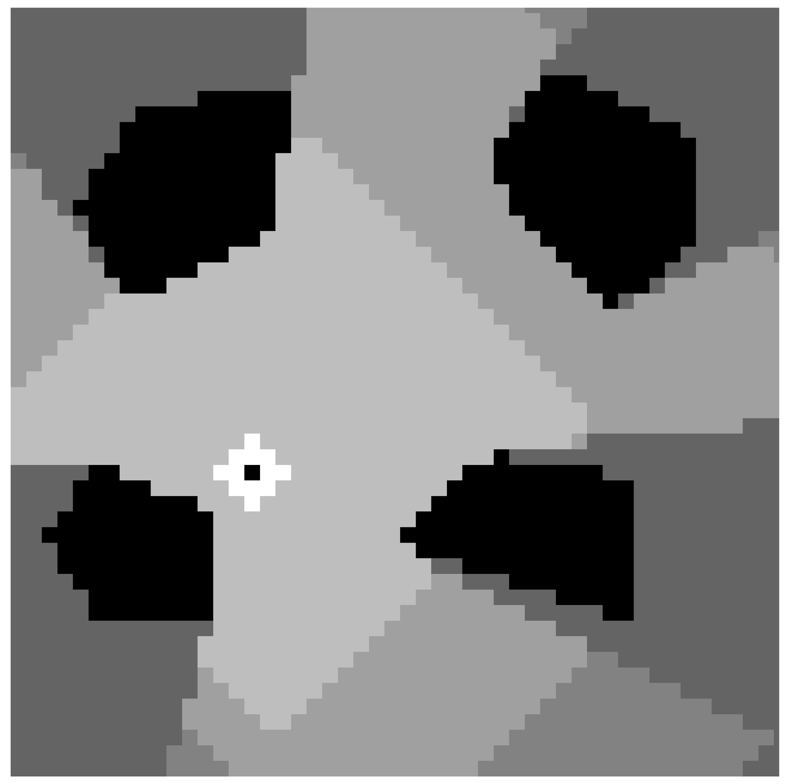


Circular Obstacle (E)

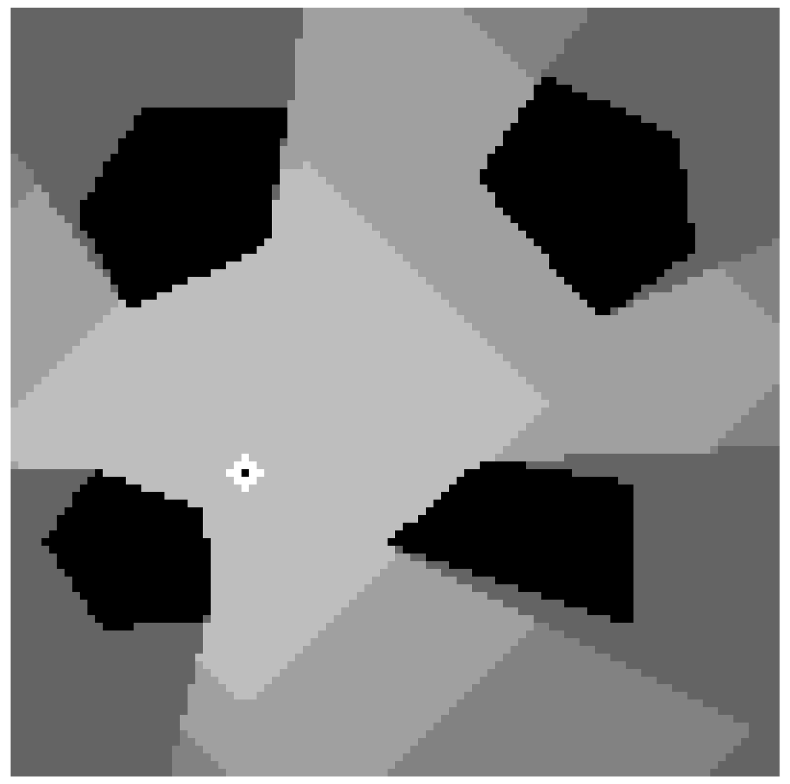


Limited Range (E)

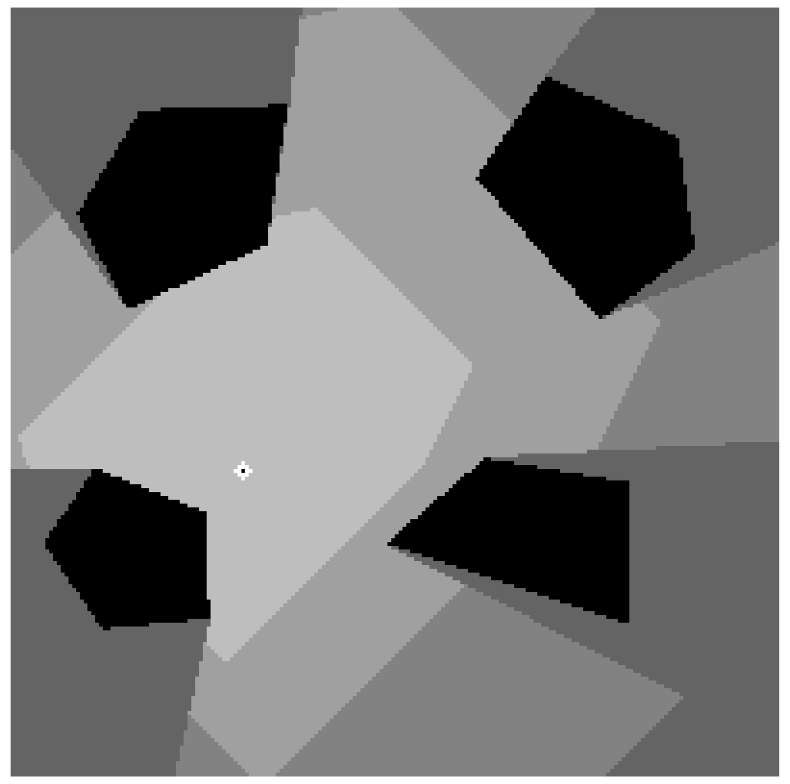
Decision Maps Using Different Resolutions



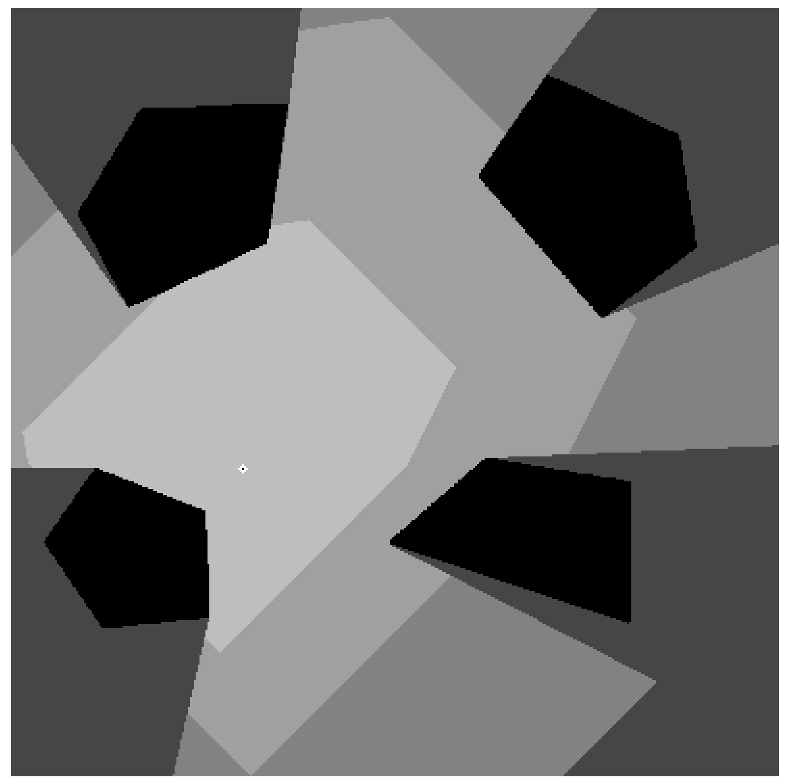
50 × 50



100 × 100

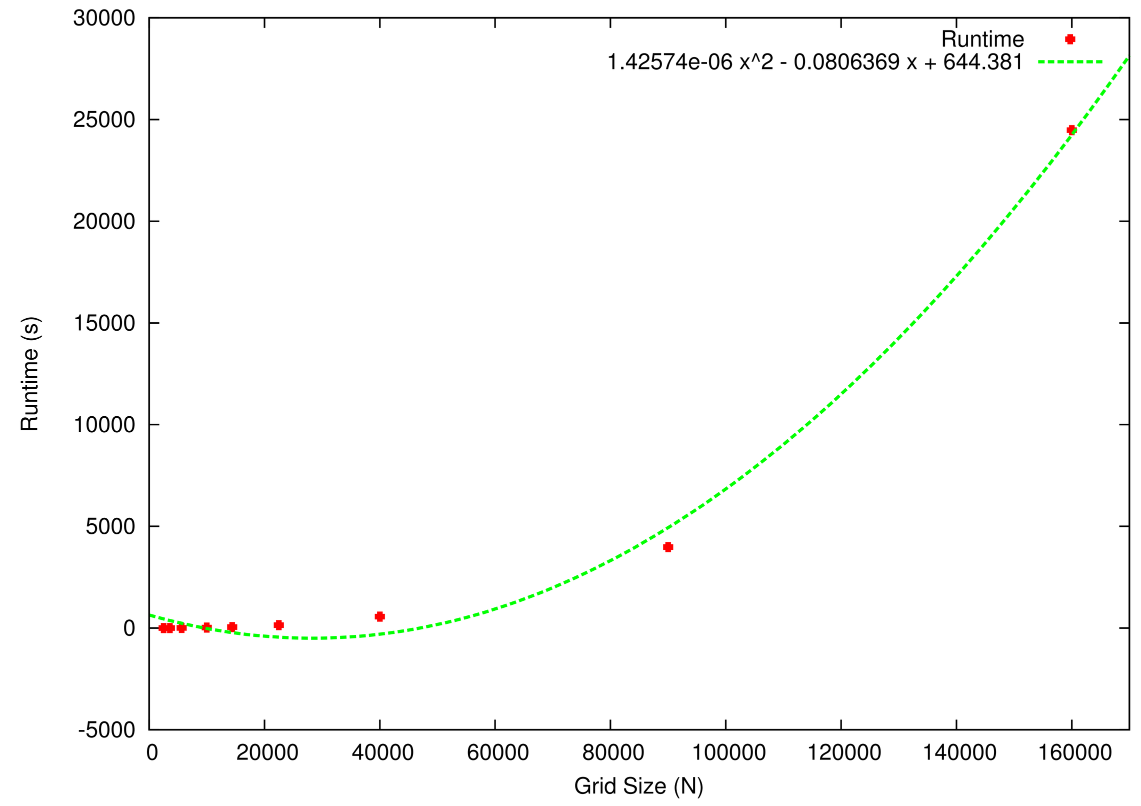


200 × 200



400 × 400

Average Runtimes



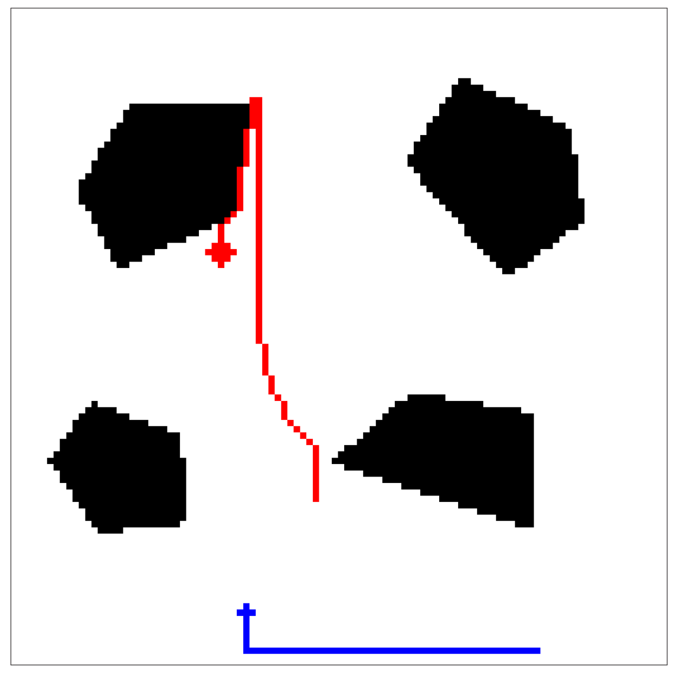
Grid size	Runtime
50 × 50	00:00:01
60 × 60	00:00:02
75 × 75	00:00:06
100 × 100	00:00:23
120 × 120	00:00:52
150 × 150	00:02:27
200 × 200	00:09:24
300 × 300	01:06:18
400 × 400	06:47:57

Generic trajectory planning for winners

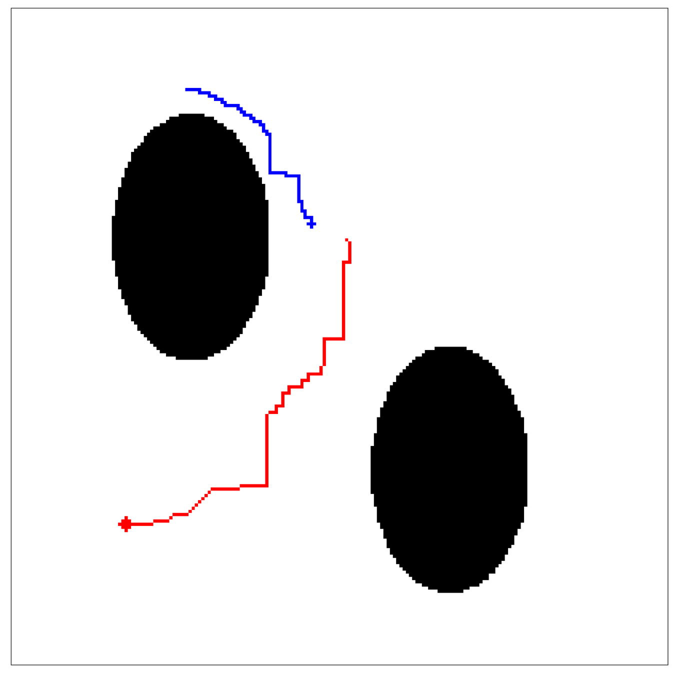
Input: $\mathbf{Bad(.,.)}$, current state $\mathbf{(player, opponent)}$.

```
begin
  N* = {}
  foreach n ∈ N(player) do
    if ¬Lose(n, n') ∀n' ∈ N(opponent) then
      N* = N* ∪ n
  Move to any neighbor in N*.
```

Simulations



Arrogant Pursuer



Oval Obstacles

Conclusions & Future Work

- Decide all games for moderately sized maps
- Arbitrarily shaped obstacles
- Extensions to trajectory planning
  - Optimal escape trajectories
- Future directions
  - Blind interruptions
  - More players

{abdelkader, helalfy}@alexu.edu.eg