Recovering Visibility and Dodging Obstacles in Pursuit-Evasion Games

Ahmed Abdelkader
Department of Computer Science, University of Maryland at College Park

Introduction

- Visibility-based pursuit-evasion:
  - Motion: holonomic, max speed per player
  - Visibility: omnidirectional, optional range
  - Traditionally, the game ends when the pursuer loses sight of the evader.
- Applications:
  - Surveillance, monitoring, FPS and racing games, etc.

Discretization and Strategy Matrix

- Represent the map as a grid of pixels (white: clear, black: obstacle).
- Assume the players take turns, with the evader moving first.
- For each pair of positions (p, e), $S[p, e] = 1$ iff the evader can win.
- Easily accommodates different motion and sensing models.
- Solve for the optimal strategy by backward induction.
- $\mathcal{N}(x)$ denotes the neighboring locations player x can reach in 1 turn.

The Classical (Primal) Game

- Is it possible to keep the evader in sight? How?
- Initialize with visibility queries then solve the recurrence:

$$S[p,e,i] = \begin{cases} \neg v(p,e) & \text{if } i = 0, \\ \bigvee_{e' \in \mathcal{N}(e)} \bigwedge_{p' \in \mathcal{N}(p)} S[p',e',i-1] & \text{otherwise.} \end{cases}$$

The Visibility Induction Loop

Input : An initialized strategy matrix S.

begin
  $S' \leftarrow 0$;
  $iter \leftarrow 0$;
  while $S' \neq S$ **do**
    $S' \leftarrow S$;
    foreach $p \in w \times h$ **do**
      foreach $e \in w \times h$ **do**
        foreach $e' \in \mathcal{N}(e)$ **do**
          $isExit \leftarrow True$;
          foreach $p' \in \mathcal{N}(p)$ **do**
            if $S'[p', e'] = 0$ **then**
              $isExit \leftarrow False$;
          if $isExit = True$ **then**
            $S[p, e] \leftarrow 1$;
            break;
    $iter \leftarrow iter + 1$;
  return S;

The Dual Game

- Is it possible to stay out of the pursuer's sight? How?
- If a pursuer fails to keep an evader in sight, it may be able to recover.
- We get a new recurrence as the logical negation of the previous one:

$$S[p,e,i] = \begin{cases} \neg v(p,e) & \text{if } i = 0, \\ \bigwedge_{e' \in \mathcal{N}(e)} \bigvee_{p' \in \mathcal{N}(p)} \neg S[p',e',i-1] & \text{otherwise.} \end{cases}$$

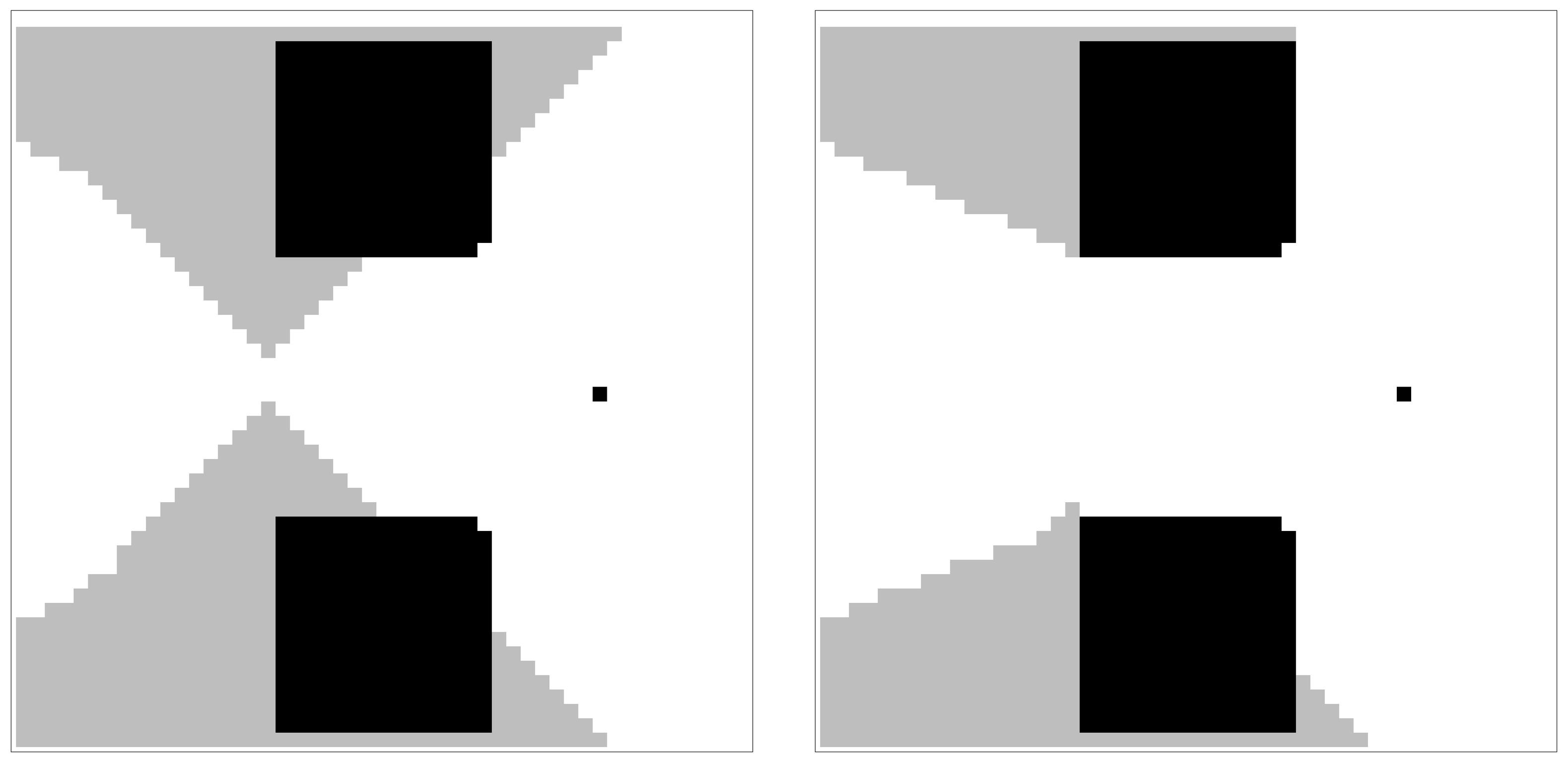
The Dual Induction Loop

Input : A strategy matrix S, grid map M, tolerance d.

begin
  $iter \leftarrow 0$;
  while $iter < d$ *and* $S' \neq S$ **do**
    $S' \leftarrow S$;
    foreach $p \in w \times h$ **do**
      foreach $e \in w \times h$ **do**
        $hasExit \leftarrow False$;
        foreach $e' \in \mathcal{N}(e)$ **do**
          $isExit \leftarrow True$;
          foreach $p' \in \mathcal{N}(p)$ **do**
            if $M.vis(p', e')$ *or* $S'[p', e'] = 0$ **then**
              $isExit \leftarrow False$;
          if $isExit = True$ **then**
            $hasExit \leftarrow True$;
        if $hasExit = False$ **then**
          $S[p, e] \leftarrow 0$;
    $iter \leftarrow iter + 1$;
  return S;

Recovering Visibility

- Pursuer view in the case with $d = 0$ (left) vs. $d = 5$ (right).



Dodging Obstacles

- Assume obstacle trajectories are known for a duration T.
- Time-varying visibility has to be incorporated into the recurrence:

$$S[p,e,t] = \begin{cases} \neg v_t(p,e) & \text{if } t = T, \\ \neg v_t(p,e) \vee \bigvee_{e' \in \mathcal{N}(e)} \bigwedge_{p' \in \mathcal{N}(p)} S[p',e',t+1] & \text{otherwise.} \end{cases}$$

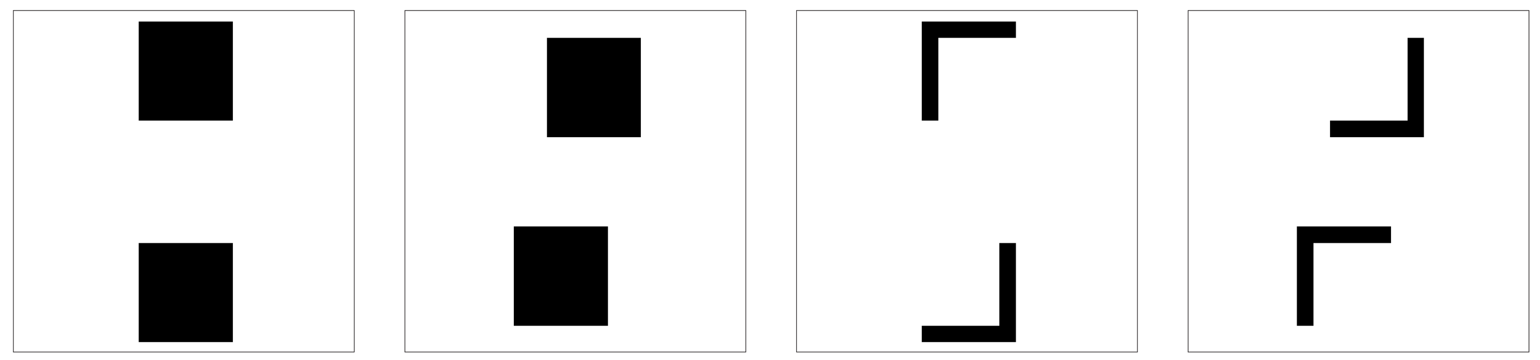
The Dynamic Induction Loop

Input : A sequence of maps $\{M_t\}$, $t = 1 \ldots T$.

begin
  $S \leftarrow 0$;
  $t \leftarrow T$;
  while $t > 0$ **do**
    $S' \leftarrow S$;
    foreach $p \in w \times h$ **do**
      foreach $e \in w \times h$ **do**
        foreach $e' \in \mathcal{N}(e)$ **do**
          $isExit \leftarrow True$;
          foreach $p' \in \mathcal{N}(p)$ **do**
            if $M_t.vis(p, e)$ *and* $S'[p', e'] = 0$ **then**
              $isExit \leftarrow False$;
          if $isExit = True$ **then**
            $S[p, e] \leftarrow 1$;
            break;
    $t \leftarrow t - 1$;
  return S;

Maintain optimal strategies after a single update in the map

- Move obstacles by add/remove. Use primal/dual induction to update S.



Conclusions & Future Work

- Optimal strategies allowing visibility to be recovered within d turns
- Optimal strategies in dynamic environments
- Future directions
  - State space reduction for improved running times
  - Generate optimal strategies for more than two players

akader@cs.umd.edu