Recovering Visibility and Dodging Obstacles in Pursuit-Evasion Games

Ahmed Abdelkader
Department of Computer Science
University of Maryland
College Park, Maryland 20742
Email: akader@cs.umd.edu

***Abstract*—Pursuit-evasion games encompass a wide range of planning problems with a variety of constraints on the motion of agents. We study the visibility-based variant where a pursuer is required to keep an evader in sight, while the evader is assumed to attempt to hide as soon as possible. This is particularly relevant in the context of video games where non-player characters of varying skill levels frequently chase after and attack the player.**

In this paper, we show that a simple dual formulation of the problem can be integrated into the traditional model to derive optimal strategies that tolerate interruptions in visibility resulting from motion among obstacles. Furthermore, using the enhanced model we propose a competitive procedure to maintain the optimal strategies in a dynamic environment where obstacles can change both shape and location. We prove the correctness of our algorithms and present results for different maps.

I. Introduction

Pursuit-evasion games have received considerable attention in both the AI planning and robotics communities which resulted in a wealth of results. In the visibility-based variant, the problem of deciding whether the evader possesses an escape strategy is known to be NP-hard [10]. Analytical solutions to the problem for limited obstacle geometries have been derived by appealing to differential game theory [2]. Other variants of the problem has been studied under complete information [7], imperfect information [13] and partially-observable spaces [12]. More realistic models of agents with a limited range of vision have also been considered [4].

The solution method we are interested in is based on backward induction. Starting from terminal states, where the outcome of the game is known, the outcome for earlier states can be determined recursively by considering the actions available at each state. It may be viewed as a discrete analogue to integrating a system of differential equations from a set of initial conditions. This intuitive recursive formulation easily lends itself to dynamic programming which provides an efficient solution to a variety of problems.

In the same spirit, backward induction can also be used for optimization problems like cost-to-go Bellman formulations to path planning problems, e.g., [17]. To bound the number of states that need to be explored, sampling approaches are typically used as in Rapidly-exploring Random Trees (RRTs) [9] or adaptively refined meshes [16]. More traditionally, uniform grids continue to be a standard tool to model domains for planning problems. Recent works on any-angle path planning, e.g., [3], have made it possible to overcome the unrealistic trajectories generated by such grid techniques.

Alternatively, precomputation has been considered to stay close to optimality at the cost of higher storage. Scalable precomputed search trees (SPST) is one recent example where RRT type trees are computed to provide uniform coverage of a domain [8]. Similar approaches utilizing roadmaps have been reported for the problem of pursuit-evasion [14]. Roadmap based techniques have been applied to the visibility-based variant as well, e.g., [6].

In this paper, we develop an enhanced model for visibility-based pursuit-evasion that allows us to compute optimal strategies for two interesting scenarios particularly relevant to video games. In the first scenario, a *tolerance parameter* is specified to allow interruptions in visibility of bounded duration. In the second scenario, the map is allowed to change, i.e., obstacles can change both shape and location. We reuse the algorithmic framework we presented earlier for computing a strategy matrix by backward induction [1]. *To the best of our knowledge, these are the first algorithms to compute optimal pursuit-evasion strategies in these scenarios and only heuristic-based or suboptimal limited-depth approaches were known.* This enables the design of more intelligent computer players and also helps with level design to assess the difficulty of different layouts and choose entry points for respawning.

The rest of the paper is organized as follows. In Section 2, we define the visibility-based pursuit-evasion game and recall the solution method we will be using in this study. Section III introduces the dual formulation, which is key to the remainder of the paper, and studies the first scenario where we relax the hard visibility constraints with a tolerance parameter. Then in Sections IV and V, we continue to demonstrate how similar techniques can be used to extend this solution method to dynamic environments where obstacles can change both shape and location. Finally, we conclude in Section VI.

II. The Visibility-Based Pursuit-Evasion Game

The pursuit-evasion game studied here can be defined as follows. We are given two agents: a pursuer (p) and an evader (e) at known initial positions in an environment with obstacles that block both motion and visibility. The pursuer is required to keep the evader in sight, while the evader is assumed to attempt to break the pursuer's line of sight in the shortest amount of time possible. Both players have complete information about the other's location and move at bounded speeds. Hence, the first natural question is to decide for a given environment, initial positions, and maximum speeds of both players, whether the evader has an escape strategy.

The solution method we presented earlier [1] uses a grid map discretization of the environment and assumes both players take turns to move between cells of this grid, which bears similarity to cop-robber games on graphs [5]. Per the description of the game, the game state can be completely determined by the locations of both players, denoted by the ordered pair (p, e), and which of the two players moves next.

This method is summarized in Algorithm 1. Given a grid map of dimensions $w \times h$, computation is performed on a $(w \times h) \times (w \times h)$ boolean matrix that stores for each pair of locations whether or not the evader has an escape strategy. Starting at terminal states, which are pairs (p, e) where e is not visible to p, the game can be decided for these states (Line 3). This is implemented by a simple procedure, $M.vis(p, e)$, that checks whether the line connecting e to p passes through any of the obstacles. To decide the game for earlier states, standard backward induction is performed (Line 4) as described in more details in procedure InductionLoop. Note that we set $S[p, e] = 1$ iff the pursuer starting at position p cannot keep the evader starting at position e in sight, with the evader moving first.

Algorithm 1: Decides the game for a given map.

```
Input  : A (w × h) grid map of the environment M.
Output: The strategy matrix S.
1 begin
2     InitVisibility($M$, $S$);
3     InductionLoop($S$);
4     return $S$
```

Function InitVisibility(M, S)

```
Input  : A grid map $M$ and a strategy matrix $S$.
Output: The initialized strategy matrix $S$.
1 begin
2     $S \leftarrow 0$;
3     foreach $p \in w \times h$ do
4         foreach $e \in w \times h$ do
5             if $\neg M.vis(p, e)$ then
6                 $S[p, e] \leftarrow 1$;
7     return $S$
```

Procedure InductionLoop(S)

```
Input  : A strategy matrix $S$.
Data: A secondary $(w \times h) \times (w \times h)$ binary matrix $S'$.
1 begin
2     $S' \leftarrow 0$;
3     $iter \leftarrow 0$;
4     while $S' \neq S$ do
5         $S' \leftarrow S$;
6         foreach $p \in w \times h$ do
7             foreach $e \in w \times h$ do
8                 foreach $e' \in \mathcal{N}(e)$ do
9                     isExit ← True;
10                    foreach $p' \in \mathcal{N}(p)$ do
11                        if $S'[p', e'] = 0$ then
12                            isExit ← False;
13                    if isExit = True then
14                        $S[p, e] \leftarrow 1$;
15                        break;
16        $iter \leftarrow iter + 1$;
17    return $S$;
```

The function InductionLoop repeatedly evaluates the escape condition for each game state, which may only be available after adjacent states have been determined. Once an escape strategy is found, there is no need to process the state again. The escape conditions can be expressed as the recurrence relation of the form:

$$S[p, e] = \bigvee_{e' \in \mathcal{N}(e)} \bigwedge_{p' \in \mathcal{N}(p)} S[p', e']. \quad (1)$$

We do not attempt to optimize the implementations here to keep the presentation as simple as possible. More elaborate optimizations along with their theoretical analysis were discussed in [1]. We use $\mathcal{N}$ to denote the neighborhood of locations the player can reach in a single turn, which implicitly depends on its speed. Letting N be the size of the map, i.e., $w \times h$, and κ be the largest size of a neighborhood $\mathcal{N}$, we recall the following result established in [1]:

Theorem 1. (Visibility Induction [1]) *Algorithm 1 decides the discretized game for a general environment in $\mathcal{O}(\kappa^2 N^3)$.*

In the next section, we formulate the dual game and describe the dual induction loop which is key to the algorithms in Sections III and IV.

III. Recovering Lost Visibility

In order to tolerate visibility interruptions, we do not terminate the game and declare that the pursuer has lost as soon as line of sight visibility is broken. Instead, we introduce a parameter d that controls how long we allow the evader to remain out of the evader's sight in one streak. The pursuer would then seek strategies that can recover visibility to the evader if that is possible to achieve within $d-1$ steps, and the evader only wins if it is able to hide for at least d consecutive steps. The optimal strategy for the evader is still to find the fastest way to *win*. As such, the evader does not favor intermediate visibility interruptions if they do not lead to a sooner victory.

We introduce the *dual game* to model the situation after visibility is lost. In this phase, the evader attempts to remain out of the pursuer's sight as long as possible, while the pursuer attempts to recover visibility to the evader as soon as possible. Note that in the original game the evader's objective was to minimize the visibility time, while in this phase it is to maximize the occlusion time. Similarly, in the original game, the pursuer's objective was to maximize the visibility time, while in this phase it is to minimize the occlusion time. In a sense, the agents exchange their roles but the dynamics stay the same.

For this reason, we refer to this situation as the dual game. We obtain a corresponding recurrence relation for recovering visibility as the logical negation of the escape conditions in Equation 1:

$$S[p,e] = \bigwedge_{e' \in \mathcal{N}(e)} \bigvee_{p' \in \mathcal{N}(p)} \neg S[p',e']. \tag{2}$$

Observe that the dual game is only defined for pairs of player positions that are not mutually visible. These are exactly the pairs that defined the terminal states for the original game. It is clear that in order to allow the game to proceed as long as visibility can be recovered within d steps, we need to exclude those pairs from the terminal states. With that, all that is needed is to run a *dual induction* on the non-visible pairs to get the *relaxed* terminal states. Then, running the original induction backwards from the restricted set of terminal states yields the desired strategies.

These steps are summarized in Algorithm 2. After initializing the matrix by marking all pairs that are not mutually visible, the procedure DualInductionLoop is invoked. Each iteration in this procedure bears strong similarity to the original induction loop. However, the result is that for certain pairs initialized as terminal, with the evader winning and the pursuer losing, this decision is simply undone (Line 16).

Algorithm 2: Tolerating interruptions in visibility.

```
   Input  : A strategy matrix $S$, grid map $M$, tolerance $d$.
1  begin
2      InitVisibility($M$, $S$);
3      DualInductionLoop($S$, $M$, $d$);
4      InductionLoop($S$);
5      return $S$;
```

Using the updated terminal states, Algorithm 2 computes the pursuit-evasion strategies for all pairs of initial positions to maintain visibility as long as possible, while tolerating interruptions in visibility within d steps. As we are essentially reusing the induction loop studied in [1], we get the same bound on the running time. In addition, the same optimizations can be applied to speed up the computation.

The correctness of the DualInductionLoop procedure is established in the next lemma.

Lemma 2. *The DualInductionLoop correctly computes strategies to recover visibility in less than d steps, if any.*

Proof. For the base case, when $iter = 0$, $S[p,e] = 0$ iff (p,e) are mutually visible, as initialized by InitVisibility. Then, when $iter = i$, $S[p,e]$ is assigned 0 (Line 16) iff e does not have a neighbor e' such that no neighbor p' of p satisfies

Procedure DualInductionLoop(S, M, d)

```
    Input  : A strategy matrix $S$, grid map $M$, tolerance $d$.
    Data: A secondary $(w \times h) \times (w \times h)$ binary matrix $S'$.
1   begin
2       $iter \leftarrow 0$;
3       while $iter < d$ and $S' \neq S$ do
4           $S' \leftarrow S$;
5           foreach $p \in w \times h$ do
6               foreach $e \in w \times h$ do
7                   hasExit ← False;
8                   foreach $e' \in \mathcal{N}(e)$ do
9                       isExit ← True;
10                      foreach $p' \in \mathcal{N}(p)$ do
11                          if $M.vis(p', e')$ or $S'[p', e'] = 0$
                             then
12                              isExit ← False;
13                      if isExit = True then
14                          hasExit ← True;
15                  if hasExit = False then
16                      $S[p, e] \leftarrow 0$;
17          $iter \leftarrow iter + 1$;
18      return $S$;
```

$M.vis(p',e')$ or $S'[p',e'] = 0$. If $M.vis(p',e')$ is true, then p' has direct visibility to e'. Otherwise, if $S'[p',e'] = 0$, then by the induction hypothesis, p' has a strategy that guarantees visibility to e' is recovered within $i-1$ steps. □

Following with an invocation of the original induction loop in procedure InductionLoop, the next theorem proves the correctness of the whole scheme, which relaxes the result in Theorem 1 to scenarios with less strict visibility requirements.

Theorem 3. *Algorithm 2 decides the discretized game for a general environment in $\mathcal{O}(\kappa^2 N^3)$, tolerating arbitrary interruptions in visibility of $d = \mathcal{O}(N)$ steps.*

Proof. By invoking the DualInductionLoop (Line 2), $S[p,e] = 1$ iff e has a strategy to hide out of sight for at least d steps as established in Lemma 2. Then, the InductionLoop is invoked (Line 3) starting at $iter = 0$ with S as returned from DualInductionLoop. For $iter = i$ in the InductionLoop, $S[p,e]$ is assigned 1 (Line 14) iff e has a neighbor e' such that for all neighbors p' of p we have that $S'[p',e'] = 1$. By the induction hypothesis, it follows that for any such p', it must be the case that by $iter = i - 1$, e' has found an escape strategy to stay out of p''s sight for at least d steps.

The bound on the running time follows by Theorem 1. Observe that for large values of d, the overhead of running the DualInductionLoop cannot be greater than the worst case for running InductionLoop itself. Hence, the total running time has the same bound. □

Figure 1 shows a pursuer with all evader locations it cannot keep in sight colored in gray. We compare the traditional scenario of zero tolerance against allowing broken visibility for 5 turns. Beyond deciding which initial conditions enable each player to win, the computed strategy matrix can be used for trajectory planning as discussed in [1]. Figure 2 shows a basic example of successful tracking although visibility was initially broken.

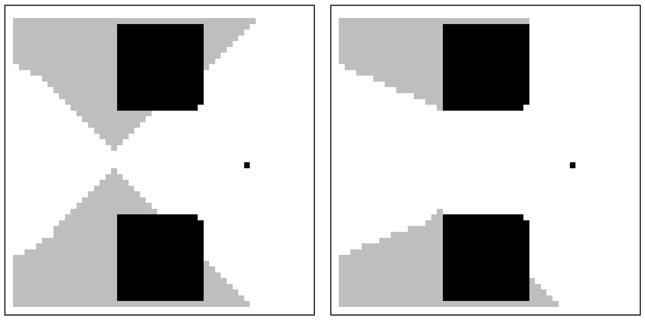
Fig. 1. Pursuer view in the case with $d = 0$ (left) vs. $d = 5$ (right).

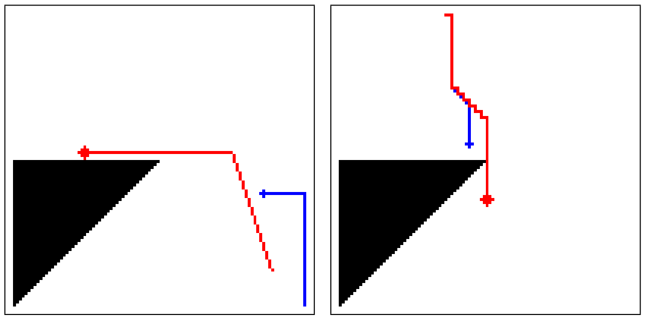
Fig. 2. A pursuer recovering visibility around an infinite corner.

IV. Moving Obstacles by Add/Remove

The interplay between the classical game and its dual as seen in Algorithm 2 yields new insights into computed strategies as encoded in the matrix S. In the original game, mutual visibility is established and the evader attempts to hide altering an entry in the matrix from 0 to 1. In the dual game, visibility is broken and the pursuer attempts to recover it altering an entry in the matrix from 1 to 0.

Using this enhanced understanding, we study the visibility-based pursuit-evasion game in a dynamic environment where obstacles can change both shape and location. Naturally, in a scenario like that, initially established visibility can get broken by the changes in the environment, rather than the actions of the agents. It follows that the agents need to update their strategies to match the current environment. To keep the presentation simple, we do not tolerate interruptions in visibility in this section. However, the same method from Section III can be applied to relax visibility tests.

In this section, we present a procedure to maintain the optimality of precomputed strategies that can offer considerable savings compared to recomputing a strategy matrix from scratch. We use a slightly modified version of the dual induction as listed in the ConservativeDualInductionLoop procedure. This ensures that dual updates do not enable pursuers to chase after evaders they do not see directly.

```
Procedure ConservativeDualInductionLoop($S$, $M$)
Input : A strategy matrix $S$, grid map $M$.
Data: A secondary $(w \times h) \times (w \times h)$ binary matrix $S'$.
begin
    $iter \leftarrow 0$;
    while $S' \neq S$ do
        $S' \leftarrow S$;
        foreach $p \in w \times h$ do
            foreach $e \in w \times h$ do
                if $\neg M.vis(p, e)$ then
                    continue;
                $hasExit \leftarrow False$;
                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
                        $hasExit \leftarrow True$;
                if $hasExit = False$ then
                    $S[p, e] \leftarrow 0$;
        $iter \leftarrow iter + 1$;
    return $S$;
```

We use a simple `diff` model to capture the motion of obstacles. We keep track of all grid cells that witness a change in occupancy. It is clear that any change in the environment resulting from a change in the shape or location of obstacles can be expressed as introducing new obstacles at a subset of grid cells and removing existing obstacles from another subset.

To remove obstacles, we first need to establish line-of-sight visibility only between those pairs of positions that were blocked by the removed obstacles. Eventually, some of these pairs may terminate with the evader finding an escape strategy. This means we need to run the original induction loop to find such strategies, if any. The updated strategies propagate to other pairs that may use the newly found routes to improve their outcomes. Adding obstacles is slightly trickier as the added obstacles block both visibility and mobility. For example, an added obstacle may not necessarily help an evader if it does not provide a shorter escape trajectory and instead requires that the evader move around it to reach a more secure exit while a faster pursuer is getting closer which makes it harder for the evader to win.

Both adding and removing obstacles, can be performed in one shot as shown in Algorithm 3. The algorithm simply updates line-of-sight visibility to the limited set of player positions dictated by the updates. Once these updates are established, new strategies are computed and propagated by consecutive invocation of the induction procedures.

Algorithm 3: Updates strategies by a diff of the grid map.

```
Input : A strategy matrix $S$, grid map $M$, map diff ($M^+, M^-$).
1  begin
2      $M \leftarrow M + M^+ - M^-$;
3      foreach $p \in w \times h$ do
4          foreach $e \in w \times h$ do
5              if $M.vis(p,e)$ and $\neg M^-.vis(p,e)$ then
6                  $S[p,e] \leftarrow 0$;
7              else if $\neg M^+.vis(p,e)$ then
8                  $S[p,e] \leftarrow 1$;
9      InductionLoop($S$);
10     ConservativeDualInductionLoop($S$, $M$);
11     return $S$;
```

Note that some of the decisions applied by the first invocation will need to be corrected by the second one. In fact, the order of invocation does not matter in the correctness of the result. Depending on the required updates, it can be more efficient to start with one type of induction or the other. The order shown here proved to be faster in our experiments moving obstacles by small offsets. For larger shifts, it is more efficient to compute a new matrix from scratch.

Figure 3 shows an initial map with two square obstacles. In Figure 4, the two obstacles have moved diagonally in two opposite directions. Figures 5 and 6 show the difference in occupancy between the initial and final maps. Locations that are no longer occupied by obstacles are denoted by M^- and those that receive new obstacles are denoted by M^+.

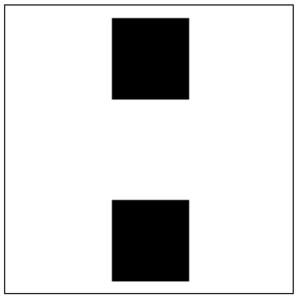
Fig. 3. Initial map M.

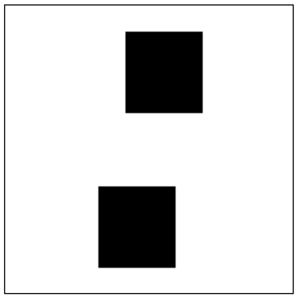
Fig. 4. Updated map M'.

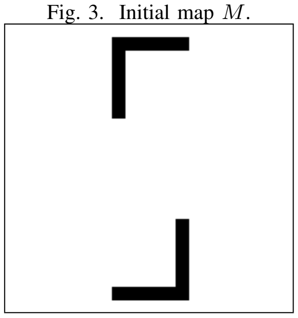
Fig. 5. Removed diff M^-.

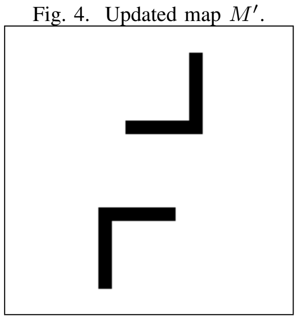
Fig. 6. Added diff M^+.

The correctness of Algorithm 3 is established in the next theorem.

Theorem 4. *Algorithm 3 correctly updates the strategy matrix in a discretized game given a diff map of the environment.*

Proof. We argue that the returned strategy matrix is *correct*, i.e., for any pair (p, e) in the returned matrix, $S[p, e] = 1$ iff the evader has an escape strategy.

Keeping in mind that the input strategy matrix S was correct and that the visibility constraints of the updated map were enforced (Lines 2-8), it follows that after invoking InductionLoop (Line 9) any pair (p, e) where the evader has an escape strategy will have $S[p, e] = 1$. By the assumption that e has an escape strategy, there will eventually be neighbors $e' \in \mathcal{N}(e)$ with $S[p', e'] = 1 \forall p' \in \mathcal{N}(p)$ that satisfy the escape conditions for e, which InductionLoop detects correctly.

It remains to show that for all pairs where the evader does not have an escape strategy, $S[p, e] = 0$. This is achieved by the invocation of ConservativeDualInductionLoop (Line 10). Similar to the preceding argument, by the assumption that p can keep e in sight indefinitely, there will eventually be a neighbor $p' \in \mathcal{N}(p)$ for each $e' \in \mathcal{N}(e)$ with $S[p', e'] = 0$ that satisfies the recovery conditions for p, which ConservativeDualInductionLoop detects correctly. Otherwise, if e does have an escape strategy, the recovery conditions for p must fail eventually. □

V. Continuously Moving Obstacles

Unlike the case in the previous section where obstacles move unexpectedly, it might be the case that their motion trajectories can be estimated in advance. In that case, line-of-sight visibility between pairs of locations becomes a function of time, and the players need to plan their motions taking this into account.

In our discrete setting, assuming a time horizon of T steps, we only need access to $vis(p, e)$ at each step t. This can be encoded as a sequences of matrices $\{M_t\}$ with $t = 1 \dots T$. This can be computed efficiently for obstacles with nice shapes as in [15]. Working backwards from the last step T, we can easily identify terminal states either directly by a visibility test. Given these terminal states, we run backward induction on t. We say that an evader wins at time t if line-of-sight visibility is broken at t or if the evader is guaranteed an exit at a later time step. Introducing a step index to capture the dependence on time, the recurrence relation for this case can be written as:

$$S[p, e, t] = \neg v(p, e, t) \vee \bigvee_{e' \in \mathcal{N}(e)} \bigwedge_{p' \in \mathcal{N}(p)} S[p', e', t+1]. \quad (3)$$

Algorithm 4 implements the induction for this case. Next, we establish its correctness.

Theorem 5. *Algorithm 4 decides the discretized game for a sequence of maps $\{M_t\}$, with $t = 1 \dots T$, in $\mathcal{O}(\kappa^2 N^2 T)$.*

Proof. For the base case, at $t = T$, we have that $S' = 0$. It follows that $S[p, e, T] = 1$ only if $\exists e' \in \mathcal{N}(e)$ such that $\forall p' \in \mathcal{N}(p)$ we have $\neg M_T.vis(p', e')$ and the condition in (Line 11) is never satisfied for e'.

Then, at iteration $t = i$, if $S[p, e, t]$ is set to 1, it must be the case that all pursuer actions p' failed the test in (Line 11) for at least one evader action e', i.e., either visibility is already broken and $M_i.vis(p, e)$ is false or $S[p', e', j + 1] = 1$, which by the induction hypothesis means that an escape strategy for e at a later step is available through e'.

Observing that the algorithm performs exactly T iterations, the bound on the running time follows. □

Algorithm 4: Decides the game for a dynamic map.

```
   Input : A sequence of maps $\{M_t\}$, $t = 1 \dots T$.
   Data: A secondary $(w \times h) \times (w \times h)$ binary matrix $S'$.
1  begin
2      $S \leftarrow 0$;
3      $t \leftarrow T$;
4      while $t > 0$ do
5          $S' \leftarrow S$;
6          foreach $p \in w \times h$ do
7              foreach $e \in w \times h$ do
8                  foreach $e' \in \mathcal{N}(e)$ do
9                      isExit ← True;
10                     foreach $p' \in \mathcal{N}(p)$ do
11                         if $M_t.vis(p, e)$ and $S'[p', e'] = 0$
                            then
12                             isExit ← False;
13                     if isExit = True then
14                         $S[p, e] \leftarrow 1$;
15                         break;
16         $t \leftarrow t - 1$;
17     return $S$;
```

Assuming the environment does not change after the time horizon T, we may wish to let the game proceed on this fixed situation. This can easily be accommodated by replacing (Line 2) in Algorithm 4 with an invocation of Algorithm 1 on M_T. Looking at the proof for Theorem 5, this would only change the base case in an obvious way.

It is also possible to tolerate limited interruptions in visibility in this case as well. However, this requires the use of counters rather than boolean values in the strategy matrices. By incrementing the counter for each step the evader stays out of the pursuer's sight, we can detect when it completes d steps or when the counter should be reset. A similar technique was applied in [1] to compute the fastest escape trajectory and the corresponding optimal pursuit trajectory, where the original recurrence relation is written as a min-max over such counters, rather than an or-and of booleans.

VI. Conclusion and Future Work

We presented a novel dual formulation to the standard visibility-based pursuit-evasion game that allows an easy way to relax the visibility constraints. To the best of our knowledge, this is the first algorithm to compute optimal pursuit-evasion strategies that accommodate recovering visibility once it is lost. Combined with the original formulation, we derived a competitive update procedure to maintain the optimality of the computed strategies in dynamic environments where obstacles change both shape and location. We proved the correctness of our algorithm and presented basic experimental results for simple maps to demonstrate the contribution.

To make the discretized model more practical, it would be interesting to consider state space reduction such that only few game states are represented explicitly. For a fixed initial position, the approach in [11] seems promising.

References

[1] Ahmed Abdelkader and Hazem El-Alfy. Visibility induction for discretized pursuit-evasion games. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.

[2] Sourabh Bhattacharya and Seth Hutchinson. On the existence of nash equilibrium for a two player pursuit-evasion game with visibility constraints. In *Algorithmic Foundation of Robotics VIII*, pages 251–265. Springer, 2010.

[3] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, pages 533–579, 2010.

[4] Brian P Gerkey, Sebastian Thrun, and Geoff Gordon. Visibility-based pursuit-evasion with limited field of view. *The International Journal of Robotics Research*, 25(4):299–315, 2006.

[5] Geňa Hahn and Gary MacGillivray. A note on k-cop, l-robber games on graphs. *Discrete mathematics*, 306(19):2492–2497, 2006.

[6] Volkan Isler, Dengfeng Sun, and Shankar Sastry. Roadmap based pursuit-evasion and collision avoidance. In *Robotics: Science and Systems*, volume 1, pages 257–264, 2005.

[7] Kyle Klein and Subhash Suri. Complete information pursuit evasion in polygonal environments. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[8] Manfred Lau and James J. Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, pages 299–308, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[9] Steven M Lavalle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*. Citeseer, 2000.

[10] Rafael Murrieta-Cid, Raul Monroy, Seth Hutchinson, and Jean-Paul Laumond. A complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2657–2664. IEEE, 2008.

[11] Valentin Polishchuk, Esther M Arkin, Alon Efrat, Christian Knauer, Joseph SB Mitchell, Guenter Rote, Lena Schlipf, and Topi Talvitie. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry*, 7(2):77–100, 2016.

[12] Eric Raboin, Ugur Kuter, and Dana Nau. Generating strategies for multi-agent pursuit-evasion games in partially observable euclidean space. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1201–1202. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[13] Eric Raboin, Dana Nau, Ugur Kuter, Satyandra K Gupta, and Petr Svec. Strategy generation in multi-agent imperfect-information pursuit games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 947–954. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[14] Samuel Rodriguez, Jory Denny, Takis Zourntos, and Nancy M Amato. Toward simulating realistic pursuit-evasion using a roadmap-based approach. In *Motion in Games*, pages 82–93. Springer, 2010.

[15] Y-HR Tsai, L-T Cheng, Stanley Osher, Paul Burchard, and Guillermo Sapiro. Visibility and its dynamics in a pde based implicit framework. *Journal of Computational Physics*, 199(1):260–290, 2004.

[16] Dmitry S Yershov and Emilio Frazzoli. Asymptotically optimal feedback planning: Fmm meets adaptive mesh refinement. In *Algorithmic Foundations of Robotics XI*, pages 695–710. Springer, 2015.

[17] Dmitry S Yershov and Steven M LaValle. Simplicial Dijkstra and A∗ algorithms: From graphs to continuous spaces. *Advanced Robotics*, 26(17):2065–2085, 2012.