Designing the Control of a UAV Fleet with Model Checking

Technical Report OSU-CISRC-1/04-TR06

Christopher A. Bohn*
Paolo A.G. Sivilotti
Bruce W. Weide

Department of Computer and Information Science
The Ohio State University

{bohn,paolo,weide}@cis.ohio-state.edu

Abstract

We model the problem of unmanned aerial vehicles searching for moving targets as a pursuer-evader game in which the pursuers have a speed advantage over the evaders but are incapable of determining an evader's location unless a pursuer occupies the same location as that evader. By treating the players as nondeterministic finite automata, we can model the game and use it as the input for a model checker. By specifying that there is no way to guarantee the pursuer can locate the evader, the model checker will either confirm that this is the case, or it will provide a counterexample showing one search pattern for the pursuer that will guarantee the evader must eventually be caught. As an ongoing effort to reduce the time to find pursuer-winning strategies, we also present heuristics to limit the state space of the game models.

1 Introduction

The challenge of an airborne system locating an object on the ground is a common problem for many applications, such as tracking, search and rescue, and destroying enemy targets during hostilities. If the target is not facilitating the search, or is even attempting to foil it by moving to avoid detection, the difficulty of the search effort is greater than when the target aids the search. In recent years, interest in using unmanned aerial vehicles (UAVs) for "the dull, the dirty, and the dangerous" combat and non-combat scenarios has grown, as evidenced by the US Department of Defense's (DoD) investment in UAV development, procurement, and operations: $3 billion in the 1990s, $1 billion from 2000–2002,

*The views expressed in this article are those of the authors and do not necessarily reflect the official policy of the Air Force, the Department of Defense or the US Government.

and a projected $10 billion by 2010 [Office of the Secretary of Defense, 2002]. Their application to the search problem is limited by the size of their sensor footprint – indeed, the DoD has expressed concern over "the soda-straw field of view used by current UAVs that negatively affects their ability to provide broad SA [situational awareness] for themselves, much less for others in a formation" [Office of the Secretary of Defense, 2002].

Where other researchers are addressing the problems of locating stationary targets [Baum, 2002, Oğraş, 2002, Zhang and Ordóñez, 2003] or of locating moving targets with some probability of success [Kanchanavally et al., 2004], our research is intended to address a technical hurdle for locating moving targets with certainty.

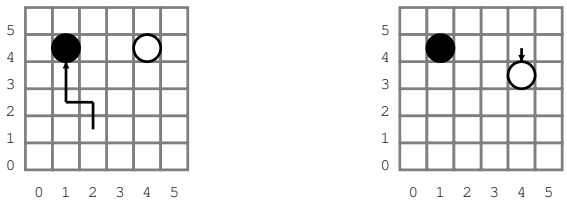
We have abstracted this problem of controlling a fleet of UAVs to meet some search objective into a pursuer-evader game played on a finite grid. The pursuers can move faster than the evaders, but the pursuers cannot ascertain the evaders' locations except by the collocation of a pursuer and evader. Further, not only can the evaders determine the pursuers' past and current locations, they have an oracle providing them with the pursuers' future moves. The pursuers' objective is to locate all evaders eventually, while the evaders' objective is to prevent indefinitely collocation with any pursuer [Bohn et al., 2003, Bohn and Sivilotti, 2004]. Part of our contribution is to represent the game as a concurrent system of finite automata and to use symbolic model checking [Clarke et al., 1999] to determine whether all evaders can eventually be located within the model; if so, we can use the model checker's output (a sequence of states that serve as a witness to the satisfaction/dissatisfaction of the model's specification) to generate a search pattern for the UAVs to follow. Using this approach, we are able to address the problem of whether and how the UAVs can be *guaranteed* to locate *all* of a set of *moving* targets.

We demonstrate that model checking can be used to suggest strategies by which a single UAV can locate targets. Moreover, model checking can be used to coordinate the actions of a fleet of UAVs. We also exhibit heuristics that can reduce the size of the models enough to obtain these strategies fast enough for on-the-fly planning.

In this chapter, we present the model of our pursuer-evader game (Section 2), how that model can be used to generate pursuer-winning search strategies when they exist (Section 3), our efforts to reduce the time required to generate search strategies (Section 4), and the directions we intend to take our research (Section 5).

2 Modeling the Game

We begin by describing the model of the pursuer-evader game. After examining the finite automata we use to model the game, we present sufficient conditions to guarantee that the pursuers will locate all evaders.



(a) Pursuer's turn

(b) Evader's turn

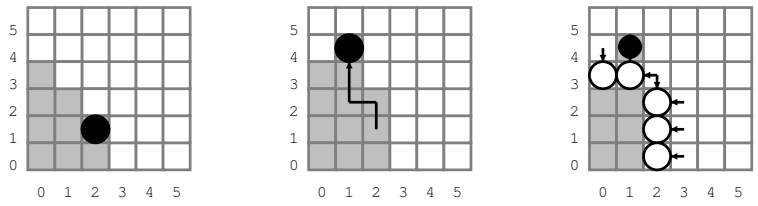
Figure 1: Examples of movements by the pursuer and the evader. Solid circle is the pursuer; hollow circle is the evader.

2.1 The Model

In its simplest form, the game is played on a rectilinear grid with one pursuer and one evader taking turns to move. There are no obstacles to either the pursuer or evader, so each can occupy any location and can transition from any location to any adjacent location. Consider Figure 1. In Figure 1(a), the pursuer moves three spaces north and one west on a 6×6 board; this is followed by the evader moving south one space in Figure 1(b). There are four basic variations on the game's simplest form, based on the definition of "adjacent location". In all four variants, the players can move in the four cardinal directions (north, south, east, west); the variations are whether the pursuer, the evader, both, or neither can move diagonally (northwest, northeast, southwest, southeast). These simplifications facilitate an analysis of the required pursuer speed.

An obvious (but, it turns out, inappropriate; see Section 3.2) model for the game is to treat the pursuers and evaders as nondeterministic finite automata (NFAs) with each player's state defined only by its current grid cell. Instead, we define a model that does not explicitly include evaders. Instead, each grid cell has a single boolean state variable CLEARED that indicates whether it is impossible for an undetected evader to occupy that cell. CLEARED is `true` if and only if no undetected evader can occupy that cell, and CLEARED is `false` if it is *possible* for an undetected evader to occupy that cell. Trivially, cells occupied by pursuers are CLEARED – either there's no evader occupying that cell, or it has been detected. A cell that is not CLEARED becomes CLEARED when and only when a pursuer occupies it. A CLEARED cell ceases to be CLEARED when and only when it is adjacent to an unCLEARED cell during the evaders' turn to move; if all its neighboring cells are CLEARED then it remains CLEARED.

Compare Figure 2 with Figure 1. In this hypothetical scenario, the pursuer has CLEARED a region of the southwest corner of the grid, as shown by the shaded portion of Figure 2(a) and can conclude the evader must be outside that region. As in Figure 1(a), the pursuer moves four spaces north and west in Figure 2(b), increasing the CLEARED region by three cells. Since the pursuer



(a) Before pursuer moves

(b) Pursuer's turn

(c) Evader's turn

Figure 2: Examples of changes in the possible locations for the evader. Evader is known to be in unshaded region.

does not know where the evader is located, the CLEARED region must shrink in accordance with the union of all possible moves by the evader. A move by the evader south from the northeastern-most corner would not cause the evader to enter a previously-CLEARED cell, but Figure 2(c) shows there are six ways the evader *could* move from an unCLEARED cell into a CLEARED cell, and the five CLEARED cells that could now be occupied by the evader may no longer be considered CLEARED.

By modeling the evaders' possible locations instead of their actual locations, the model does not need to be modified when additional evaders are incorporated. The same model is used whether there are one, ten, or (importantly in many practical situations) an unknown number of evaders. Henceforth, we talk of "the evaders" without saying how many there are.

2.2 Sufficient Pursuer Qualities for Simple Game Variants

We wish to characterize the necessary and sufficient pursuer qualities for pursuer victories in the myriad variations of the game. While we have proven sufficient conditions for a pursuer win in general, we have yet to establish the necessary conditions except for specific game instances.

2.2.1 One Pursuer

We have established the sufficient pursuer qualities to guarantee a pursuer win for the simpler forms of the game. When the game is played with one pursuer and with evaders that move 1 space/turn and do not move diagonally on a rectilinear grid with the shorter dimension being n cells, we have shown the existence of a search pattern that will guarantee the pursuer's victory if the pursuer can move n spaces per turn. If the evaders can move diagonally, then such a pursuer-winning strategy exists if the pursuer can move $n+1$ spaces per turn [Bohn and Sivilotti, 2004].

The formal proofs can be found in [Bohn and Sivilotti, 2004]; however we

shall provide here an informal operational argument. Consider the case in which the evaders do not move diagonally and the pursuer can move n spaces/turn. As shown in Figure 3, the pursuer can cause the number of CLEARED cells in a column to increase, up to the point at which all cells in that column are CLEARED. The pursuer can then shift into the initial position for the next column (while preserving the CLEARED cells in the previous column). By repeatedly applying this process, eventually every cell in each column will be CLEARED.

We can demonstrate that a pursuer speed of $n-1$ spaces/turn is insufficient in general. Consider a 2×2 grid, and for simplicity's sake assume the pursuer does not move diagonally either. The pursuer is not guaranteed to find even a single evader if the pursuer's speed is $n - 1 = 1$ spaces/turn. The evaders can avoid detection indefinitely as follows: after each move by the pursuer, move to (or stay in) the cell diagonally opposite the pursuer. Because the pursuer cannot move 2 spaces/turn, it cannot move to a diagonally opposite corner in a single move; because the evaders can always view the pursuer, they can always move to the diagonally opposite corner.

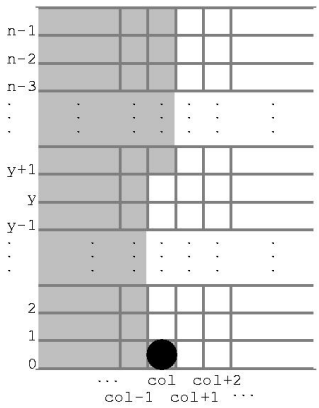
Further, we believe (but have not proved) that for all values of $n \geq 2$, $n - 1$ spaces/turn is an insufficient speed to assure the pursuer's victory. This is based on the observation that the evaders have less freedom of movement in the boundary cells (and even less still in the corner cells), and so evaders have increasing freedom of movement as the ratio of non-border cells to border cells increases. And so we believe that if the pursuer is not certain to locate the evaders at $n-1$ spaces/turn when all cells are corner cells, then the pursuer will not be able to locate the evaders at $n-1$ spaces/turn when there are non-border cells.

2.2.2 Multiple Pursuers

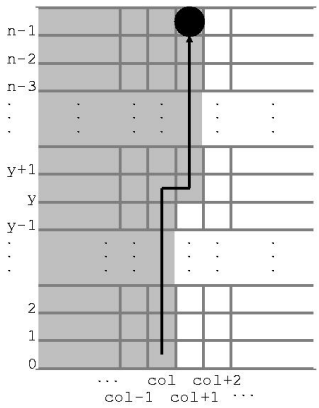
We have also turned our attention to characterizing the necessary and sufficient pursuer qualities for a fleet of pursuers on a grid with shorter dimension n. Clearly, if at least one pursuer moves n spaces/turn when the evaders do not move diagonally, the pursuers can win – but we can do better. Through model checking, we have shown that there are configurations in which all but one pursuer can remain stationary, and the single moving pursuer can detect all evaders even when its speed is less than n spaces/turn. More generally, if all pursuers move at least 1 space/turn (the same speed as the evaders), then a pursuer-winning strategy exists for k pursuers if:

$$\sum_{p \text{ is a pursuer}} p.speed \geq n - k + 1 \tag{1}$$

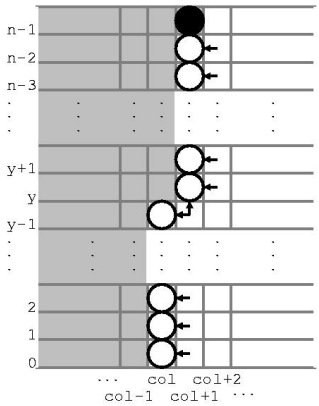
It may not be practical – or even possible – to characterize analytically the necessary pursuer qualities for games in which there are arbitrary restrictions on players' movements. Instead, it may be more practical to use the model checker, as described in Section 3, to determine whether the proposed pursuer qualities are sufficient. As an example of the difficulty in characterizing the necessary and sufficient conditions of a game variant in general, our analysis



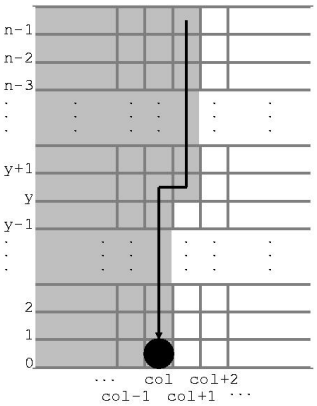
(a) Initial conditions: cells $y+1..n-1$ are CLEARED in this column



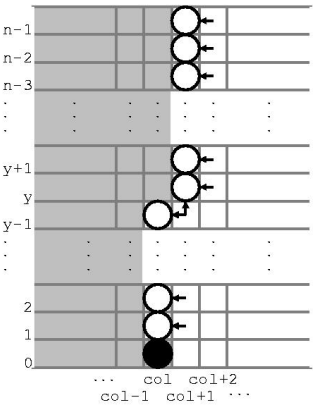
(b) Pursuer moves n spaces



(c) Evaders move, causing cells on the frontier to be unCLEARED



(d) Pursuer moves n spaces



(e) Evaders move, causing cells on the frontier to be unCLEARED; cells $y..n-1$ are CLEARED in this column

Figure 3: Execution of the "clear-cell" algorithm when the evaders cannot move diagonally.

of even the "simple" game variants was aided by model checking: informed by tool-generated search strategies for specific game instances of a game variant, we reanalyzed the model of that game variant and showed the sufficiency of weaker pursuer characteristics than were indicated by our previous analysis. In short, we learned something from the model checker.

3 Generating Search Strategies

Now that we have covered the model of the pursuer-evader game, we shall explain how we obtain pursuer-winning search strategies by using model checking. We first offer an overview of symbolic model checking. We follow this with the manner in which we use a symbolic model checker to obtain a sequence of computation states that correspond to a series of moves the pursuers can make to detect every evader.

3.1 Symbolic Model Checking

Model checking is a technique for verifying that finite state concurrent systems satisfy certain properties expressed in a temporal logic. Typical examples of its use are in verifying properties of complex digital circuit designs and communication protocols [Clarke et al., 1999].

Symbolic model checking has three features that are particularly desirable. First is that for certain logics, such as Computational Tree Logic (CTL), a model can be checked in time linear in the number of the model's states; more precisely, checking a specification f expressed as a predicate in CTL requires $\mathcal{O}\left((|S| + |R|) \cdot |f|\right)$ time, where S is the set of states and R is the set of transitions between states in the model.

The second feature of symbolic model checking is that the model's states are not represented explicitly; rather, the model checker uses binary decision diagrams. Because of this, symbolic model checkers use less memory than explicit-state model checkers for a given model (or, for the same memory footprint, symbolic model checkers can check larger models). When first introduced, symbolic model checking could verify models with up to $10^{20} (\sim 2^{66})$ states; subsequent refinements have permitted the checking of models with up to $10^{120} (\sim 2^{400})$ states [Clarke et al., 1999].

The third feature is that if the specification being checked is not satisfied by the model, the model checker can provide a specific counterexample showing why the property does not hold. We make use of this last capability by modeling the pursuer-evader game as a nondeterministic finite automaton and instructing the model checker to prove that no matter how the pursuer moves, some evader can successfully avoid the pursuer. If the model checker indicates this property is true, then for the given game conditions there is no guarantee the pursuer and an evader will be collocated. On the other hand, if the model checker indicates the property is false, then as a counterexample, it will indicate the model's state

changes that, when properly interpreted, correspond to a sequence of moves the pursuer can make to guarantee it will locate all the evaders.

3.2 Model Checking the Game

Had we modeled all players as NFAs with state defined only by the occupied grid cell, then we would check whether the evaders can always avoid detection. The model checker would promptly offer the counterexample of the pursuer moving immediately to the evaders' locations; this is not a useful result.

Instead, we model each grid cell as CLEARED or unCLEARED, and the property to check against this model can be described using CTL:

$$\mathsf{AG} \bigvee_{\substack{0 \le r < n \\ 0 \le c < m}} \neg \text{CLEARED}(r, c) \tag{2}$$

That is, along every computation path ("A") beginning at the initial state, every state ("G") includes at least one unCLEARED cell. If the specification is not satisfied, then the counterexample can be used to generate a search pattern that will cause every cell to be CLEARED. When all cells are CLEARED, there can be no undetected evaders.

Example models coded in the model description language for the SMV[1] model checker can be found in [Bohn et al., 2003].

As reported in Section 2.2, we have determined mathematically the sufficient pursuer speed to guarantee a pursuer victory for the simple forms of the game on an arbitrarily-sized grid. As a sanity test, we note that the results of model checking specific instances of these simple forms match the analytical results: when the pursuer speed is set to what we believe to be the minimally-sufficient speed, the pursuer has a winning search strategy; when the pursuer speed is less, the pursuer does not have a winning search strategy. Interestingly, the winning search strategies suggested by the model checker do not correspond to those used in the proofs of our analysis. So, there are winning strategies other than that of Figure 3.

Clearly, a single pursuer traveling over a rectilinear grid with no obstacles is not a faithful model of the real world, though it is useful for investigating some of the issues in using a model checker for our purpose. For this reason, we have demonstrated that we can also use the model checker with a hexagonal grid, with multiple pursuers, when the pursuer's sensor footprint is not simply the cell it currently occupies, or when there is terrain (modeled as cells the evaders cannot enter and cell boundaries the evaders cannot cross).

For example, consider the terrain in Figure 4, and assume the evaders cannot occupy the central lake, cannot traverse the sheer western wall of the crevice, and can traverse the oblique easter wall of the crevice in only one direction. If we model this terrain on a 4×6 grid, then by Equation 1, we know the pursuers can detect all evaders if the sum of the k pursuers' speeds is at least $3-$

[1] http://www.cs.cmu.edu/∼modelcheck/smv.html

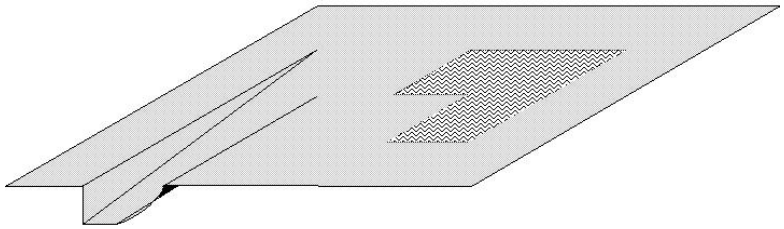


Figure 4: Example of terrain with crevice and lake.

k spaces/turn. Because of the restrictions on the evaders' movements, however, it is possible for a single pursuer to locate all evaders when the pursuer's speed is only 3 spaces/turn.

4 Performance

In this section we consider the performance of the technique described in Section 3.2: how much time is required to obtain a winning search strategy? After demonstrating why the method is intractable, we present three heuristics, two of which we have implemented.

4.1 Model Checking Complexity

As stated in Section 3.1, CTL has the desirable characteristic that CTL properties can be model checked in time linear in the product of the size of the model and the size of the specification [Clarke et al., 1999]. Unfortunately, the model we have described has a number of states that is exponential in the size of the grid. Specifically, for an $m \times n$ grid with k pursuers, the fastest of which moves s spaces/turn, the number of states is:

$$\underbrace{(mn)^k}_{\substack{\text{pursuers'} \\ \text{locations}}} \cdot \underbrace{2^{mn}}_{\substack{\text{cells} \\ \text{CLEARED?}}} \cdot \underbrace{(s+1)}_{\substack{\text{“clock”} \\ \text{artifact}}} \tag{3}$$

Since the number of states is exponential in the number of grid cells, the execution time is exponential in the number of grid cells. We cannot attain a faster time without running the risk of failing to find a legitimate pursuer-winning strategy. From [Reif, 1984], we can conclude that even if we were to use a different temporal logic, the execution time would still be exponential in the number of grid cells.

Table 4.1 shows the effect on the execution time and memory displacement to run SMV on a 933MHz Pentium III system for different-sized grids with one pursuer and movement only in the four cardinal directions. The attentive reader will notice that the number of states listed in the table for each configuration

Table 1: Time needed to obtain a pursuer search strategy.

Board Dimensions	Number of Locations	Pursuer Speed (moves/turn)	Number of States	Reachable States	Time
2×2	4	2	$2^{8.6}$	$2^{6.6}$	$0.01s$
2×4	8	2	$2^{13.6}$	$2^{11.1}$	$0.04s$
3×3	9	3	$2^{15.2}$	$2^{13.3}$	$0.08s$
3×5	15	3	$2^{21.9}$	$2^{17.6}$	$0.66s$
4×4	16	4	$2^{23.3}$	$2^{19.4}$	$5.39s$
4×6	24	4	$2^{31.9}$	$2^{23.3}$	$7m\ 26s$
5×5	25	5	$2^{33.2}$	$2^{25.7}$	$2h\ 4m\ 3s$
5×7	35	5	$2^{43.7}$	??	$> 24h$
6×6	36	6	$2^{45.0}$	??	$> 24h$

is exactly twice the number of states described by Equation 3; this is due to an extra boolean variable necessitated by our choice of model checker. The number of *reachable* states reported by SMV is of interest because of an optimization in SMV that reduces the execution time to a function of the number of reachable states. The last two rows of Table 4.1 are incomplete, not because checking a model with $2^{43.7}$ states is impossible, but because checking a model with $2^{43.7}$ states required more time than we were willing to allot for our initial experiments.

4.2 Heuristics to Reduce the State Space

We are considering heuristics that reduce the size of the state space, albeit at the cost of potentially excluding pursuer-winning search patterns. We consider this to be an acceptable trade-off. The benefit is that we can dramatically reduce the execution time. If the model checker indicates it cannot find a pursuer-winning search strategy, then one may or may not exist for the conditions checked; however, if such a search strategy is found, then we know it is a valid search strategy.

4.2.1 The "Sweep" Heuristic

The first heuristic we present is to break the problem of clearing the grid into the smaller problem of clearing one column and ending up positioned to clear the next column, without permitting any undetected evaders to pass into previously-CLEARED columns; see Figure 6. This corresponds to the algorithm in the proof sketch of Section 2.2.1. If it is ever possible for the evader to enter the westernmost region, then the technique of clearing columns will not compose. However, if it is possible to accomplish this feat, repeated applications of this

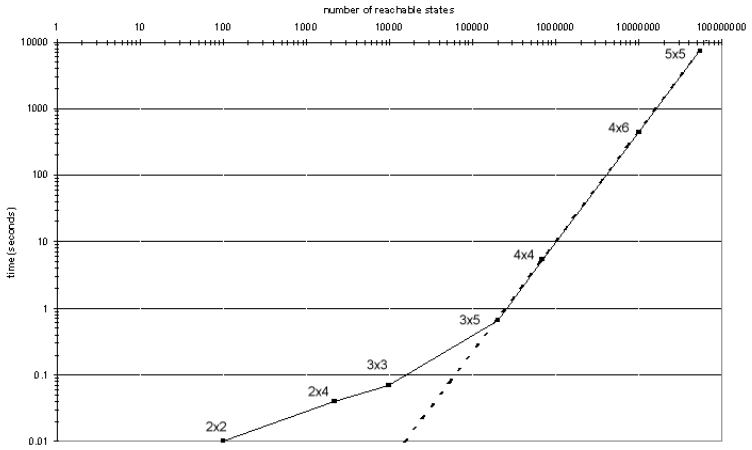


Figure 5: Execution time vs. number of reachable states

"clear-column" procedure can be composed to clear the whole grid by sweeping from one side of the grid to the other. Now, if we only need to model $w \times n$ cells explicitly (where $w \ll m$), then the number of states is:

$$(wn)^k \cdot 2^{wn+1} \cdot (s+1) \tag{4}$$

and the property to check is:

$$\neg \mathsf{E}\left[\textsc{cleared}(west)\mathsf{U}\left(\begin{array}{l}\bigwedge_{0\le r<n}\textsc{cleared}(r,0)\wedge\textsc{cleared}(west)\wedge\\ clock=0\wedge\text{position}(pursuer)=(0,1)\end{array}\right)\right] \tag{5}$$

That is, there does not exist a computation path ("E") in which the western region remains CLEARED until ("U") all cells in the first explicitly-modeled column are CLEARED along with the western region, and the pursuer is positioned in the southern cell of the second explicitly-modeled column at the start of a turn. If the specification is not satisfied, then the counterexample can be used to generate a search pattern that will cause the first modeled column to be CLEARED and the pursuer positioned to clear the next column at the beginning of its turn. Repeated application of this search pattern will eventually sweep the entire grid.

We have, in fact, used such models with a model checker. Using this heuristic, we obtained a pursuer-winning strategy for one pursuer moving 6 spaces/turn on a "$\infty \times 6$" grid ($w = 2$) in 76 seconds on a 933 MHz Pentium III [Bohn et al., 2003]. Contrast this with the performance for the 6×6 grid in Table 4.1.

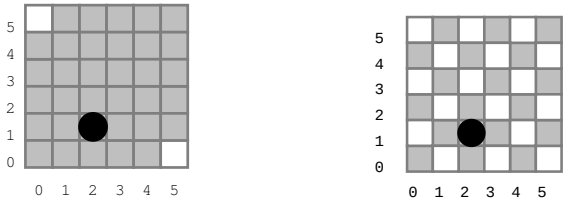
4.2.2 The "Cleared-Bars" Heuristic

Besides composing subsolutions, we also consider changes to the manner in which we model the game. For example, many possible states do not bring the

(a) Before column is cleared

(b) After column is cleared

Figure 6: Abstraction of grid unbounded along the horizontal axis.



(a) All cells CLEARED except two at opposite corners.

(b) Alternating CLEARED and unCLEARED cells.

Figure 7: Examples of undesirable states.

pursuer any closer to winning the game. One such state is the "7-10 split" in Figure 7(a). Unless the pursuer can move diagonally 8 spaces/turn (or cardinally 14 spaces/turn), then more cells will become unCLEARED in the next turn than the pursuer could make CLEARED. We believe that the resulting state could have been achieved without passing through this state first. Similarly, the "checkerboard" in Figure 7(b) is not a useful state, and it not even reachable without diagonal moves.

So instead of considering whether each cell is CLEARED, we instead can define sets of contiguous CLEARED cells. For example, under the belief that if a pursuer-winning strategy exists, one exists that "grows" the CLEARED area as a set of contiguous bars, we can define the endpoints of CLEARED cells in each row (or column) and require that the CLEARED cells in each row be contiguous from one endpoint to the other (Figure 8(a)). With this model, the number of states is:

$$\underbrace{(mn)^{2k}}_{\substack{\text{pursuers'}\\ \text{locations}}} \cdot \underbrace{(m+1)^{2n}}_{\substack{\text{endpoints}\\ \text{of bars}}} \cdot \underbrace{(s+1)}_{\substack{\text{"clock"}\\ \text{artifact}}} \tag{6}$$

The first term is raised to the power of $2k$ instead of k because there are conditions in which the pursuers' current and last locations are needed to update the bars correctly. The middle term is $m + 1$ instead of m to provide for "endpoints" when there are no CLEARED cells in a given row.

Table 4.2.2 contrasts the execution time without a heuristic to that with this heuristic.

Table 2: Time benefit of "Cleared-Bars" Heuristic.

Board Dimensions	Pursuer Speed (moves/turn)	Time Cleared-Cells	Time Cleared-Bars
4×4	4	$5.39s$	$1.69s$
5×5	5	$2h\ 4m\ 3s$	$32.27s$
6×6	6	$> 24h$	$32m\ 56s$

4.2.3 The "Cleared-Regions" Heuristic

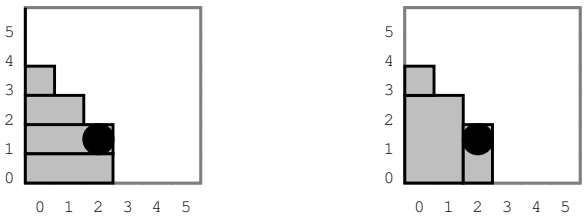
Alternatively, we might instead define the CLEARED regions geometrically by possibly-overlapping convex polygons. Figure 8(b) shows how the CLEARED area in Figure 2(a) can be described using three rectangles. While this will dramatically increase the complexity of the model description, we believe it will also dramatically decrease the number of states in the model. Consider, for example, a game with k pursuers moving s spaces/turn, with r rectangles describing the CLEARED regions. The size of the state space would be:

$$\underbrace{(mn)^k}_{\substack{\text{pursuers'}\\ \text{locations}}} \cdot \underbrace{(mn)^{2r}}_{\substack{\text{diagonal}\\ \text{corners of}\\ \text{rectangles}}} \cdot \underbrace{(s+1)}_{\substack{\text{"clock"}\\ \text{artifact}}} \tag{7}$$

Under the belief that if a pursuer-winning strategy exists, one exists when r is a small constant, this heuristic should offer substantial runtime improvement.

5 Conclusions and Future Work

We have shown how we can model the problem of UAVs searching for ground targets as a pursuer-evader game, and we have shown how we can use model checking to generate pursuer-winning search strategies when they exist. After considering the complication that the execution time is exponential in the size of the grid, we presented three heuristics to reduce the time to obtain search strategies, and the two that have been implemented have provided significant reductions in the execution time, and the third promises an even greater savings.



(a) Using CLEARED bars

(b) Using CLEARED regions.

Figure 8: Alternate ways to describe the configuration of Figure 2(a).

We plan to add additional heuristics to our arsenal. Once we have the state space growth managed, our work will address improving the fidelity of the model, such as by increasing the size of the grids, by permitting concurrent movement of the pursuers and evaders, and by placing restrictions on the pursuers' maneuverability.

Acknowledgments

The authors gratefully thank the US Air Force and the Air Force Institute of Technology for their direct support of the primary author. This work was supported by the AFRL/VA and AFOSR Collaborative Center of Control Science (Grant F33615-01-2-3154).

References

[Baum, 2002] Baum, M. (2002). A search-theoretic approach to cooperative control for uninhabited air vehicles. Master's thesis, The Ohio State University.

[Bohn and Sivilotti, 2004] Bohn, C. A. and Sivilotti, P. A. G. (2004). Upper bounds for pursuer speed in rectilinear grids. Technical Report OSU–CISRC–1/04–TR01, The Ohio State University.

[Bohn et al., 2003] Bohn, C. A., Sivilotti, P. A. G., and Weide, B. W. (2003). Using model checking to find a hidden evader. In *Proceedings, Workshop on Agent/Swarm Programming*, pages 1–7.

[Clarke et al., 1999] Clarke, Jr., E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. The MIT Press.

[Kanchanavally et al., 2004] Kanchanavally, S., Ordóñez, R., and Layne, J. (2004). Mobile target tracking by networked uninhabited autonomous vehicles via hospitability maps. Submitted to *American Control Conference*.

[Office of the Secretary of Defense, 2002] Office of the Secretary of Defense (2002). Unmanned aerial vehicles roadmap: 2002–2027.

[Oğraş, 2002] Oğraş, Ü. Y. (2002). Cooperative search strategies for multi-vehicle teams. Master's thesis, The Ohio State University.

[Reif, 1984] Reif, J. H. (1984). The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):207–301.

[Zhang and Ordóñez, 2003] Zhang, C. and Ordóñez, R. (2003). Decentralized adaptive coordination and control of uninhabited autonomous vehicles via surrogate optimization. In *Proceedings, American Control Conference*, pages 2205–2210.