Heuristics for Designing the Control of a UAV Fleet With Model Checking

Christopher A. Bohn∗

Department of Systems and Software Engineering
Air Force Institute of Technology
Wright-Patterson AFB OH 45385, USA
E-mail: christopher.bohn@afit.edu

Summary. We describe a pursuer-evader game played on a grid in which the pursuers can move faster than the evaders, but the pursuers cannot determine an evader’s location except when a pursuer occupies the same grid cell as that evader. The pursuers’ object is to locate all evaders, while the evader’s object is to prevent collocation with any pursuer indefinitely. The game is loosely based on autonomous unmanned aerial vehicles (UAVs) with a limited field-of-view attempting to locate enemy vehicles on the ground, where the idea is to control a fleet of UAVs to meet the search objective. The requirement that the pursuers move without knowing the evaders’ locations necessitates a model of the game that does not explicitly model the evaders. This has the positive benefit that the model is independent of the number of evaders (indeed, the number of evaders need not be known); however, this has the negative side-effect that the time and memory requirements to determine a pursuer-winning strategy is exponential in the size of the grid. We report significant improvements in the available heuristics to abstract the model further and reduce the time and memory needed.

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. 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

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

We previously [5] described how and why we modeled this game as a system of concurrent finite automata, and the use of symbolic model checking to extract pursuer-winning search strategies for games involving single- and multiple-pursuers, games with rectilinear and hexagonal grids, games with and without terrain features, and games with varying pursuer-sensor footprints. We further outlined the state-space explosion problem essential to our approach and suggested heuristics that may be suitable to cope with this problem.

Here we present the results of our investigation into these heuristics. In Section 2, we reiterate the technique of using model checking to discover pursuer-winning search strategies. In Section 3, we describe our heuristics and demonstrate their utility. In Section 4, we establish necessary pursuer qualities for a pursuer-winning search strategy to exist. Finally, in Section 5 we consider directions for future work.

2 Background

We begin by describing model checking, an automatic technique to verify properties of systems composed of concurrent finite automata. After examining model checking, we review the model of the pursuer-evader game and how model checking can be used to discover pursuer-winning search strategies.

2.1 Model Checking

Model checking is a software engineering technique to establish or refute the correctness of a finite-state concurrent system relative to a formal specification expressed using a temporal logic. Originally, model checking involved the explicit representation of an automaton’s states, which placed a considerable constraint on the size of models that could be checked. With the advent of symbolic model checking, checking models with greater state spaces was possible. Symbolic model checking differs from explicit-state model checking in that the models are represented by reduced, ordered binary decision diagrams, which are canonical representations of boolean formulas. Examples of symbolic model checkers are SMV [2] and its re-implementation, NuSMV [1]; Spin [3] is an example explicit-state model checker. Should a model fail to satisfy its specification, SMV, NuSMV, and Spin all provide computation traces that serve as witnesses to the falsehood of the specification; these counterexamples are often used to identify and correct errors the model.