Simulating Terrorist Threat in The Hats Simulator

Clayton T. Morrison[1], *Paul R. Cohen*[1], *Gary W. King*[2], *Joshua Moody*[1] *and Andrew Hannon*[2]

[1] USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292, USA
{clayton,cohen,moody}@isi.edu

[2] University of Massachusetts
140 Governors Drive
Amherst, MA 01003, USA
{gwking,hannon}@isi.edu

Keywords: Effects Based Nodal Analysis, Multiple Competing Hypotheses, Terrorism, Simulation

Abstract

The Hats Simulator is a lightweight proxy for many intelligence analysis problems, and thus a test environment for analysts' tools. It is a virtual world in which many agents engage in individual and collective activities. Most agents are benign, some intend harm. Agent activities are planned by a generative planner. Playing against the simulator, the job of the analyst is to find harmful agents before they carry out attacks. The simulator maintains information about all agents. However, information is hidden from the analyst and some is expensive. After each game, the analyst is assessed a set of scores including the cost of acquiring information about agents, the cost of falsely accusing benign agents, and the cost of failing to detect harmful agents. The simulator is implemented and currently manages the activities of up to a hundred thousand agents.

1. Introduction

The Hats Simulator was designed originally to meet the needs of academic researchers who want to contribute technology to Homeland Security efforts but lack access to domain experts and classified problems. Most academic researchers do not have security clearances and cannot work on real data, yet they want to develop tools to help analysts. In any case, real data sets are expensive: They cost a lot to develop from scratch or by "sanitizing" classified data. They also are domain-specific, yet much of the domain expertise is classified. Because data sets are expensive, many that have been made available to researchers are relatively small and the patterns to be detected within them are fixed, few, and known, so working with these data sets is a bit like solving a single "Where's Waldo" puzzle. Sometimes there also is the problem that real data sets model "signal" (terrorist activities) not "noise" (everything else) yet extracting signal from noise is a great challenge. Data sets in general are static, whereas data become available to analysts over time. It would be helpful to have a data *feed*, something that generates data as events happen. To validate analysts' tools, it would be helpful to have a generator of terrorist and non-terrorist activities. The generator should be parameterized for experimental purposes (e.g., varying the distinctiveness of terrorist activities, to make them more or less easily recognizable); and it should come up with novel activities, requiring analysts and their tools to both recognize known patterns and reason about suspicious patterns.

Hats is home to hundreds of thousands of agents (hats) which travel to meetings. Some hats are covert terrorists and a very few hats are known terrorists. All hats are governed by plans generated by a planner. Terrorist plans end in the destruction of landmarks. The object of a game in the Hats simulator is to find terrorist task forces before they carry out their attacks. One pays for information about hats, and also for false arrests and destroyed landmarks. At the end of a game, one is given a score, which is the sum of these costs. The goal is to play Hats rationally, that is, to catch terrorist groups with the least combined cost of information, false arrests, and destroyed landmarks. Thus Hats serves as a test bed not only for analysts' tools but also for new theories of rational intelligence analysis. Hats encourages players to ask only for the information they need, and to not accuse hats or issue alerts without justification.

The Hats simulator is very lightweight: Agents have few attributes and engage in few elementary behaviors; however, the number of agents is enormous, and plans can involve simultaneously many agents and a great many instances of behaviors. The emphasis in Hats is not domain knowledge but managing enormous numbers of hypotheses based on scant, often inaccurate information. By simplifying agents and their elementary behaviors, we de-emphasize the domain knowledge required to identify terrorist threats and emphasize covertness, complex group behaviors over time, and the frighteningly low signal to noise ratio.

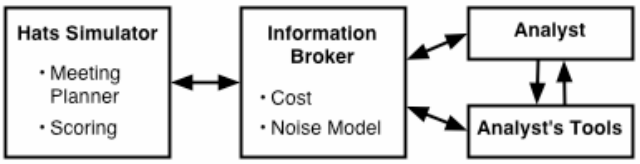

Figure 1 – Information Broker Interface to the Hats Simulator

The Hats environment consists of the core simulator and an information broker. The information broker is responsible for handling requests for information about the state of the simulator and thus forms the interface between the simulator and the analyst and her tools (see Figure 1). Some information has a cost, and the quality of information returned is a function of the "algorithmic dollars" spent. Analysts may also take actions: they may raise beacon alerts in an attempt to anticipate an attack on a beacon, and they may arrest agents believed to be planning an attack. Together, information requests and actions form the basis of scoring analyst performance in identifying terrorist threats and preventing terrorist attacks. Scoring is assessed automatically and serves as the basis for analytic comparison between different analysts and tools. The simulator is implemented and manages the activities of up to a hundred thousand agents.

The following sections outline the Hats domain, including how we generate populations of hats and how the planner schedules hat meetings. We describe the information request framework, the actions the analyst may take, and scoring. We conclude with a discussion of the future of the Hats Simulator.

2. The Hats Domain

The Hats Simulator models a "society in a box" consisting of many very simple agents, hereafter referred to as *hats*. Hats get its name from the classic spaghetti western, in which heroes and villains are identifiable by the colors of their hats. The Hats society also has its heroes and villains, but the challenge is to identify which color hat they should be wearing, based on how they behave. Some hats are known terrorists; others are *covert* and must be identified and distinguished from the *benign* hats in the society.

Hats is staged in a two-dimensional grid on which hats move around, go to meetings and trade capabilities. The grid consists of two kinds of locations: those that have no value, and high-valued locations called *beacons* that terrorists would like to attack. All beacons have a set of attributes, or *vulnerabilities*, corresponding to the *capabilities* which hats carry. To destroy a beacon, a task force of terrorist hats must possess capabilities that match the beacon's vulnerabilities, as a key matches a lock. In general, these capabilities are not unique to terrorists, so one cannot identify terrorist hats only on the basis of their capabilities.

The Hats society is structured by organizations. All hats belong to at least two organizations and some hats belong to many. Terrorist organizations host only known and covert terrorist hats. Benign organizations, on the other hand, may contain any kind of hat, including known and covert terrorists.

2.1 Population Generation

Hats populations may be built by hand or generated by the Hats Simulator. Because the constitution of a population affects the difficulty of identifying covert terrorists, population generation is parameterized. There are four sets of population parameters. The first set specifies the total number of known terrorists, covert terrorists and benign hats in the population. Another set defines the number of benign and terrorist organizations. Not all organizations have the same number of members, so a third set of parameters assigns the relative numbers of hats that are members of each organization, represented as a ratio among organizations. For example, the ratio 2:1:1 means that the first organization has twice as many members as the other two. Finally, hats may be members of two or more organizations. An overlap parameter determines the percentage of hats in each organization that are members of two or more *other* organizations. Since hat behaviors are governed by their organization membership, as we will see in the next section, organization overlap affects how difficult it is to identify covert terrorist hats. To generate populations with hundreds of thousands of hats and thousands of organizations, we use a randomized algorithm that estimates organization overlap percentage and membership ratios while matching the total number of organizations and hats in the population. When the population is generated, each hat is assigned a *native* capability that they will carry throughout the duration of the simulation, and a set of *traded* capabilities that are temporary, expiring after some number of ticks (e.g., within 40 ticks). Hats are also assigned random locations in the Hats grid world.

2.2 Meeting Generation

Hats act individually and collectively, but always planfully. In fact, the actions of hats are planned by a generative planner. Benign hats congregate at locations including beacons. Terrorist hats meet, acquire capabilities, form task forces, and attack beacons. The purpose of the planner is to construct an elaborate "shell game" in which capabilities are passed among hats in a potentially long sequence of meetings, culminating in a final meeting at a target. By moving capabilities among hats, the planner masks its intentions. Rather than directing half a dozen hats with capabilities required for an attack to march purposefully up to a beacon, instead hats with required capabilities pass them on to other hats, and eventually a capable task force appears at the beacon.

Each organization has a generative planner that plans tasks for its members. Hats that are currently participating in a task are *reserved*; hats not currently part of a task are *free*. At each tick, each organization has a chance of begin-

ning a new task. When a new task is started, the Hats meeting planner creates a *task force*, a subset of hats selected from the free hats of the organization. The size of a task-force is controlled by a parameter. The planner next selects a *target location* in the Hats world. With some probability, that location may be a beacon, otherwise a random location is selected. If a beacon is selected as the target, the goal of the task is to bring to that location the set of *required capabilities* that match the vulnerabilities of the beacon. If the location is not a beacon, a random set of required capabilities is selected as the set to bring to the location.

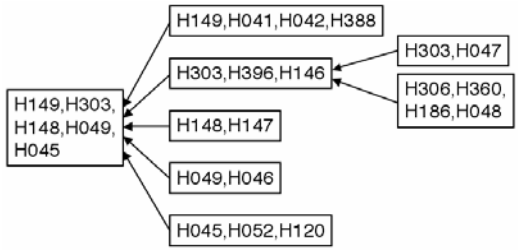

Figure 2 – Example of a generated meeting tree. Each box represents a meeting and contains a list of participating hats. Arrows indicate the temporal order of meetings.

Task force members may or may not already possess the required capabilities; usually they don't. The planner creates a set of meetings designed to ensure that the task force acquires all of the required capabilities before it reaches the target location. This is accomplished by constructing a *meeting tree* that specifies meetings and their temporal order. Figure 2 shows an example meeting tree, where boxes represent planned meetings among hats and arrows represent the planned temporal partial order of meetings. The tree is "inverted" in the sense that the arrows point from leaves inward toward the root of the tree. Parent meetings, where arrows originate, are executed first. When all of the parent meetings of a child meeting have completed, then the child meeting happens. This ensures that none of the hats that participate in the child meeting are busy in other meetings. Meeting execution means that the hats participating in the meeting begin moving toward the meeting location. The final, root meeting takes place at the task target location and includes all of the task force hats. The locations of the other meetings in the tree are selected randomly.

Initially, the meeting tree is skeletal, containing meetings whose only participants are the task force members themselves. From the organization's remaining free hats, the planner selects a second group of *resource* hats that carry required capabilities not currently carried by the task force. Resource hats are randomly assigned to existing meetings, and trades of required capabilities are scheduled to take place during the meeting. The planner finishes tree construction by adding decoy meetings, spurious trades and additional free hats not already involved in moving required capabilities to the goal. A constraint maintained throughout tree construction is that at least one hat from a parent meeting will go on to meet in a child meeting. These hats will either be task force members, resource hats carrying required capabilities to trade in the next meeting, or will be decoy hats arriving from decoy meetings.

Completed meeting trees are added to a queue of pending tasks. At each tick, the simulator engine searches the task queue for meetings with no currently executing parent meetings. These meetings are then assigned to a queue of currently executing meetings and the participant hats are incrementally moved toward the meeting location. When all of the participants have arrived at the meeting location, the meeting itself lasts for two ticks, after which all hats not participating in more meetings are set "free" and become available to participate in new meetings.

Meeting trees typically have a depth of 2 to 7. The frequency of new tasks depends on both the probability of starting a new task as well as the number of hats in each organization.

3. The Information Broker

We are currently developing a human interface to Hats to enable human analysts to play the Hats game. As an analyst playing the game, your job is to protect the Hats society from terrorist attacks. You need to identify terrorist task forces before they attack beacons, but you also need to avoid falsely accusing innocent hats. The only way to do this successfully is to gather information about hats, identify meetings, track capability trades and form hypotheses about the intentions of groups of hats. The *information broker* provides information about the state of the Hats world. The information broker will respond to questions such as *Where is* Hat_{27} *right now?* It will also provide information by subscription to analysts' tools, which in turn make information broker requests. For example, a tool might process requests like, *Identify everyone* Hat_{27} *meets in the next 100 ticks*, or, *Tell me if* Hat_{27} *approaches a beacon with capabilities* c_1, c_7 *or* c_{29}.

Some information is free, but information about states of the simulator that change over time is costly. The quality of the information obtained is determined by the amount paid. The following two sections describe the two central components of the request framework: the cost of information and noise. Together, these components make the Hats simulator an experimental environment for studying the economics of information value in the context of intelligence analysis.

3.1 The Cost of Information

Some information from the broker is free. This includes information about the population (who the known terrorists are), the simulator world (world-map dimensions), and some event bookkeeping (locations of attacks, a list of currently arrested hats). Other types of information require payment and the amount paid sets a base probability that is used to determine the accuracy of the information. Use of this base probability is explained in the next section. In the current implementation, increas-

ing accuracy requires exponentially more "algorithmic dollars." The function in Equation 1 maps payment to probability.

$$probability = 1 - \frac{1}{\log_2\left(\frac{payment}{5} + 2\right)} \quad (1)$$

The same function is applied to every payment-based request. This particular function was chosen because of its desirable rate of exponential growth, but other functions may be used.

3.2 Making Requested Information Noisy

Modeling noise is a topic suitable for an entire research program. There are many issues to consider, including whether one can request the same information multiple times, and if so, how information quality changes; whether one can get accurate information about events that occurred in the past; whether one can tell which information sources are reliable by asking several times the same question; and so on. We have started simply, imposing the constraint that the analyst may request a particular piece of information only once. This means that they must select the level of payment for the information at the time of the request and there is no going back once the request is made. Information that updates from one tick to the next, such as the current location of a hat, may be asked again at the next tick. However, information that is fixed in time, such as when or where a meeting took place, can be requested only once. This model allows us to avoid, for now, the thorny issue of how to "noise up" multiple requests for the same information.

Using the payment function described in the previous section, a payment amount is mapped to a "base" probability *p*. With probability *p*, the information requested is returned in its entirety; with probability 1-*p* it is subject to noise.

There are eight basic information requests that may be made: the hats at the location, if any, participants in a meeting, capabilities carried by a hat, capability trades, meeting times, hat death time, meeting locations and hat locations. The first five requests return lists of things, such as hats, capabilities, times, etc. The latter three return single elements. How we add noise to responses to these requests depends on the type of thing requested as well as whether they involve single elements or lists of elements.

For locations and times, adding noise is treated as sampling from a normal distribution, where the mean is the location or time of the requested item, and the variance is a function of the size of the Hats world (for locations) or the amount of time since beginning the simulation (for times). Tests ensure that noisy locations are not off the Hats world map and that noisy times are not reported as having happened in the future. Adding noise to reports about a hat or capability requires sampling from the original set of hat and capability ids defined for the scenario.

Information about lists of elements is made noisy in two stages. First, the list itself is modified by discarding or adding elements. Then, with probability 1-*p*, each element of the resulting list is replaced by an element sampled uniformly from the relevant domain (e.g., replacing a true hat id by one selected at random from among all hat ids).

Information that is requested about events or entities that do not exist are also subject to noise. If noise is not applied, then a query accurately responds that the requested information does not exist. If, however, the answer is to be made noisy, then random information of the same type requested is returned.

3.3 Exporting Data

The Hats Simulator and Information Broker are designed to provide an online data feed and allow for interaction between the analyst and simulation. However, we also have implemented facilities to export batch data from the Information Broker. Hats data can be exported as perfect information (ground truth) or noisy data sets. Application of noise works differently because there is no analog of online requests with payment levels. Noise is applied to exported data in three ways: exclusion of perfect information, inclusion of false information and corruption of perfect information. The level and type of noise is parameterized. Exported Hats data has been used in several projects, including an EAGLE Program mini-TIE and controlled experiments with social network analysis tools.

4. Actions

In addition to requesting information, the analyst playing the Hats game can also change a beacon's alert level and arrest hats. Both actions affect an analyst's performance score (discussed in Section 5).

4.1 Raising Alerts

We may not be able to stop an attack, but if we know it is coming, we can prepare and minimize loss. This is the inspiration behind modeling alerts. Each beacon can be in one of three alert levels: off (default), low or high. These correspond to the conditions of no threat, a chance of an attack, and attack likely. The analyst decides which the level of each beacon alert, but the Hats Simulator keeps track of alert states over time and whether an actual attack occurs while the state is elevated. The simulator keeps statistics including counts of hits (occurrences of attacks during elevated alerts) and false positives (elevated alerts that begin and end with no beacon attack occurring). The goal of the analyst is to minimize the time beacon alerts are elevated. High alerts are more costly than low ones. On the other hand, if an attack does occur on a beacon, a high alert is better than a low alert, and a low alert is better than none.

4.2 Arresting Hats

Analysts can also issue arrest warrants for hats in order to prevent beacon attacks. Arrests are successful only when the targeted hat is currently a member of a terrorist task force. Attempted arrests under any other conditions, including hats that are terrorists but not currently part of a terrorist task force, result in a *false arrest* (a false positive). Under this model, a hat can be a terrorist but not be guilty of any crime. Unless terrorist hats are engaged in *ongoing* terrorist activities, their arrest incurs penalties. While this is a simple model, it places realistic constraints on the analyst's choice of actions.

Successful arrests do not guarantee saving beacons. A beacon is only attacked when some subset of members from a terrorist task force successfully carry the capabilities matching the target beacon's vulnerabilities to a final meeting at on that beacon. It is possible to successfully arrest a terrorist task force member but the other terrorist taskforce members still have the capabilities required to attack the beacon. However, if the analyst successfully arrests a terrorist task force member carrying required capabilities that no other task force member has, then the final meeting of the task force will take place but it will not be attacked. This is counted as a beacon *save*.

5. Scoring Analyst Performance

The Hats Simulator and Information Broker together provide an environment for testing analyst tools. The object of the game is to identify terrorist task forces before they attack beacons. Three kinds of costs are accrued:

1 The cost of acquiring and processing information about a hat. This is the "government in the bedroom" or intrusiveness cost.
2 The cost of falsely arresting benign hats.
3 The cost of harm done by terrorists.

The skill of analysts and the value of analysis tools can be measured in terms of these costs, and the Hats environment tracks them automatically as analysts play. At the end of a game, a final report is generated that includes the following four categories:

1 Costs: the total amount of "algorithmic dollars" spent on information.
2 Beacon Attacks: including the total number of attacks that succeeded and the total number of attacks that were stopped by successful arrests
3 Arrests: the number of successful arrests and the number of false arrests (false positives)
4 Beacon Alerts: the number of low and high alert hits and false positives.

6. Conclusion

Intelligence analysts tell us that Hats has many attributes of "the real thing." Some say in the same breath that Hats ought to have other attributes, for instance, telephone communications, rapid transportation of hats around the board, different kinds of beacons, and so on. We resist these efforts to make Hats more "realistic" because for us, the purpose of Hats is to provide an enormously difficult detection problem without the overhead of building rich (and probably classified) models of real domains. No doubt Hats will change over time, but we will strive to keep it simple. The other goal that guides our development of Hats is what we might call the "missing science" of intelligence analysis. To the best of our knowledge, in the current climate, analysts penalize misses more than false positives. This sort of utility function has consequences – raised national alert levels, lines at airports, and so on. Hats is intended to be a simulated world in which analysts can experiment with different utility functions. It is a laboratory in which scientific models of intelligence gathering, filtering, and use – models based on utility and information theory – can be tested and compared.

To meet these goals, we will continue development of Hats along these lines: (1) increasing the scale and efficiency of the simulator to accommodate hundreds of thousands of hats running in reasonable time to conduct experiments and play in real-time; (2) building WebHats, a web-based interface to Hats, enabling any researcher with access to the web to make immediate use of Hats as a data source; (3) providing league tables of analyst/tool performance scores from playing the Hats game, promoting public competition to better intelligence analysis technology; and (4) developing a user-friendly interface to Hats, including more complex information querying and visual aids so that human analysts can play the Hats game more naturally.

Acknowledgments

Paul Cohen and Niall Admas conceived of the Hats Simulator at Imperial College in the summer of 2002. Professor Cohen implemented the first version of Hats. Bob Schrag at IET contributed useful ideas and built a simulator similar to Hats for use in the DARPA EELD and AFRL EAGLE program. Work on this project was funded by the Air Force Research Laboratory, account number 53-4540-0588.