The Hats Simulator

Submitted For Track 2: Emerging Applications, Technology, and Issues

Paul R. Cohen and **Clayton T. Morrison**
Center for Research on Unexpected Events (CRUE)
USC Information Sciences Institute

Author Contact Information

Postal address (both authors):
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, California 90292
e-mail: {cohen,clayton}@isi.edu
Telephone:
Paul R. Cohen: (310) 448-9342
Clayton T. Morrison: (301) 448-8430
FAX (both authors): (310) 823-6714

Application Domain(s)

Homeland Security, multi-agent simulation, data mining

AI techniques employed and issues addressed

Large-scale multi-agent simulation; generative planning; statistical analysis

Application Status

Deployed Application

Abstract

The Hats Simulator is designed to be 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. The activities are planned by a generative planner, so the simulator is continually inventing new ways for agents to achieve their objectives. Playing against the simulator, the job of the analyst is to find harmful agents before they carry out their plans. 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 ten thousand agents.

The Hats Simulator

Paul R. Cohen and **Clayton T. Morrison**
Center for Research on Unexpected Events (CRUE)
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, California 90292
{cohen,clayton}@isi.edu

Submitted For Track 2: Emerging Applications, Technology, and Issues

Abstract

The Hats Simulator is designed to be 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. The activities are planned by a generative planner, so the simulator is continually inventing new ways for agents to achieve their objectives. Playing against the simulator, the job of the analyst is to find harmful agents before they carry out their plans. 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 ten thousand agents.

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 as such); 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 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 against the Hats simulator is to find terrorist task forces before they carry out their plans. 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 testbed 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.

The Hats Simulator 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 a beacon attack, and they may arrest agents believed to be planning an attack. Together, infor-

mation requests and actions form the basis of scoring analyst performance in identifying terrorist threats. Scoring is assessed automatically and serves as the basis for analytic comparison between different analysts and tools. The simulator is implemented, manages the activities of up to ten thousand agents, and is a resource to a growing community of researchers.

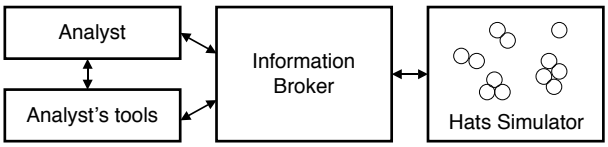

Figure 1: Information Broker Interface to Hats Simulator

The following sections outline the Hats domain, including how we generate populations of hats and how the planner schedules meetings for hats to attend. 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.

The Hats Domain

The Hats Simulator is a virtual world in which agents move around, go to meetings, acquire capabilities, do business, and, for a small subpopulation of agents, do harm. Agents move on a two-dimensional board which has only two kinds of locations: Beacons are high-value places that terrorist agents would like to destroy, other locations have low value. All beacons have a set of attributes, or vulnerabilities, corresponding to the capabilities that agents carry. To destroy a beacon, a task force of agents must be in possession of a set of capabilities that match the beacon's vulnerabilities, as a key matches a lock. In general, these sets of capabilities are not unique to terrorists, so one cannot identify a terrorist task force from its constituent capabilities, alone.

Henceforth, agents are called hats[1] and identified as benign and terrorist; overt and covert are subcategories of terrorist hats. In general, benign hats outnumber terrorists by orders of magnitude.

Some agents are known, a priori, to intend harm – they are "known terrorists" – others are covert. This is modeled easily by assigning each agent a true and an advertised hat class:

	True Hat Class	Advertised Hat Class
Benign	Benign	Unknown
Known Terrorist	Terrorist	Terrorist
Covert Terrorist	Terrorist	Unknown

The Hats Simulator knows the true class of each hat, and it plans agents' activities accordingly, but analysts must infer hat class from how agents behave. While agents that advertise terrorist hats are "known terrorists," a very small fraction of agents that advertise an unknown class are also terrorists. They are the ones to worry about.

[1]The "hats" name is an allusion to the classic spaghetti western, in which the villain and hero are identifiable by the color of their hat.

Organizations and Population Generation

Hats populations consist of known terrorist hats, covert terrorist hats and benign hats. All hats are members of at least one organization; some belong to many. There are two types of organizations. Terrorist organizations are made up of only known and covert terrorists. Benign organizations, however, may contain any kind of hat – that is, while known and covert terrorists must be members of at least one terrorist organization, they may also be members of benign organizations.

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. The `organization-overlap` parameter, a real number between 0 and 1, determines the percentage of hats in each organization that are members of other organizations. For example, if `organization-overlap` is 0.4, then 40% of the members of each organization are also members of other organizations, but the remaining 60% are only members of their native organization. The number of organizations an overlapping hat may belong to is determined by an exponential random number (thus, overlapping 3 organizations is rare, 4 is very rare, 5 is extremely rare, etc., ...). The population generator manages overlap so that the `organization-overlap` percentage is as close as possible to its parametric value.

The total numbers of known terrorist, covert terrorist and benign hats in the population are determined by the `num-terrorists`, `num-coverts` and `num-benigns` parameters, respectively. Known and covert terrorists must be members of at least one terrorist organization and may also be members of benign organizations. Benign hats, on the other hand, may only be members of benign organizations. Not all organizations have the same number of members. The variable `covert-org-members-ratio` represents the ratio of covert terrorist hats assigned to each terrorist organization and `benign-org-members-ratio` represents the ratio of benign hats to each benign organization.

Assignments of hats to organizations (respecting the parameters for organization-overlap, organization members ratios, and the numbers of hat types) takes place before any actual hats are created. Once assignments have been determined, the hats themselves are generated and given their organization assignments. At this time, each hat is also assigned a native capability, which the hat will carry throughout the simulation, and a set of "traded" capabilities which are temporary, expiring after some number of ticks (e.g., within 40 ticks). Hats are also assigned random locations in the Hats world game board.

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 for commerce and pleasure at

locations including beacons. Terrorist hats meet, acquire capabilities, form task forces, and attack beacons. Several hats might plan to visit a beacon, and might collectively have the capabilities to destroy the beacon, yet are benign. Or, one covert terrorist might plan to visit three known terrorists in succession, acquiring from each a capability that threatens a beacon; and yet might remain dormant, approaching no beacon, for a time.

Each organization has a generative meeting planner associated with it that plans tasks for its members. A *task* is a set of meetings planned to deliver a set of capabilities to some goal location in the Hats World. Hats that participate in a task are *reserved*. Hats not part of a task are *free*. At each tick each organization has a chance of beginning a new task according to the probability specified by the `p-start-new-task` parameter. When a new task is started, the Hats meeting planner selects a subset of hats from the free hats of the organization. This subset of hats is called a *taskforce*. The size of the taskforce is determined by the `num-in-meetings` parameter. The meeting planner also selects a coordinate in the Hats World game board as the *target location* of the task. With probability specified by the `p-beacon-meeting` parameter, the planner will select a beacon location as the task target. Otherwise a random Hats World coordinate is selected. If a beacon is the task target, then the set of vulnerabilities of the beacon determines the set of capabilities the taskforce must bring to the target. If the target is not a beacon, then a random set of capabilities is selected – the size of the set of random capabilities is determined by the `num-requirements` parameter. The set of capabilities the taskforce must bring to the task target is referred to as the taskforce's *required capabilities*. The taskforce members may or may not already possess the required capabilities.

In fact, if the taskforce members generally do not have all these capabilities, then the meeting planner can construct an elaborate "shell game" in which capabilities are passed among hats at a long sequence of meetings, culminating in the fatal meeting at the target. By moving capabilities among hats, the planner can mask its intentions. It certainly is not the case that, say, half a dozen hats with required and known capabilities march purposefully up to a beacon. Instead, the hats with the required capabilities pass them on to other hats, and eventually a capable task force appears at the beacon.

Once the taskforce, target location, and required capabilities have been chosen, the meeting planner creates a set of meetings designed to ensure that the taskforce acquires all of the required capabilities before going to the target location. The meeting planner accomplishes this by constructing a *meeting tree*. Figure 2 shows an example meeting tree, where the contents of each box represent the hats participating in a meeting. The tree is "inverted" in the sense that the root is the last meeting, with branches from the root representing parent meetings that take place prior to the target meeting – Figure 2 depicts the temporal ordering of meetings by directed arrows. At this point, the meeting planner incrementally fills-out the meeting tree, starting with the final meeting. The final, root meeting takes place at the target location and involves all of the taskforce hats. The parent meetings of the final meeting each have one taskforce member. The locations of all other meetings added to the meeting tree are selected randomly.

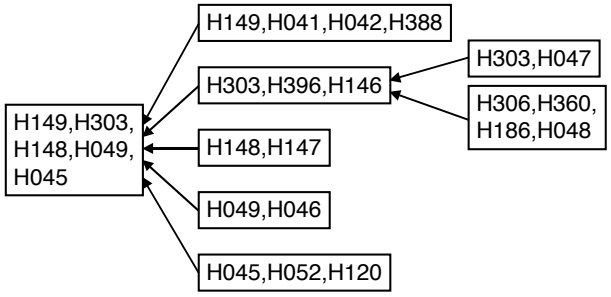

Figure 2: Example meeting tree

The meeting planner selects a second set of hats (from the organization's free hats) that carry required capabilities that the taskforce does not currently carry; these hats are called *resource hats*. Each of the resource hats are randomly assigned to taskforce members. Meetings between resource hats and taskforce members are called *resource meetings*. Resource meetings are added to the meeting tree as follows. The planner traverses a branch of the meeting tree which a taskforce member originates from (initially, these are just the direct parents of the final, root meeting). With probability `p-required-resource-meeting-origin`, the meeting planner adds a new meeting as a parent of the current meeting, which initially contains only the taskforce member. The planner traverses to that meeting and checks the probability again. With probability $1-$`p-required-resource-meeting-origin`, the current meeting becomes a resource meeting between the resource hat and the taskforce member. In a resource meeting, capability trades are planned to transfer the required capabilities to the taskforce members. This process is repeated until all of the resource hats have been assigned to taskforce members.

At this point, the meeting tree has all of the necessary meetings with trades to ensure that the taskforce will arrive at the task target with all of the required capabilities. The meeting planner then fills out the tree with additional meetings, participants, and capability trades. The additional meetings and trades are referred to as "decoys" because they are not directly involved in the task completion. The parameter `p-produce-decoy-meeting` is used to determine whether a decoy meeting should be added to a leaf meeting of the current meeting tree.

Once a meeting tree has been completely filled-out, it is added to a queue of current tasks and it will start to be executed at the next step of the simulation. During execution, the current leaves of each meeting tree are added to the `currently-executing-meetings` list and the Hats engine starts moving currently executing meeting participants toward their meeting locations. Once all of the meeting participants have arrived at a meeting location, the meet-

ing lasts for two ticks, after which all hats not participating in more meetings are set "free" (and thus available to participate in new planned tasks). All other hats still reserved for meetings then begin moving to their next meeting.

The meeting trees created by this meeting planner typically have a depth ranging from 2 to 5. The frequency of tasks planned depends on both `p-start-new-task` and the number of hats in each organization (which comprise the resources available to the planner).

The Hats Simulator is designed to accommodate any meeting planner that adheres to a planner API. We are developing the API and anticipate using other planners. For example, a variation on the above planner would plan tasks that relate meetings as directed acyclic graphs (DAGs) as opposed to trees. This allows taskforce members to meet with one another repeatedly before the final meeting. We are also exploring other meeting topologies in conjunction with researchers in social network theory.

The Information Broker

Think of the Hats Simulator as a society in a box and your job, as an analyst, is to protect the society against terrorist taskforces. Specifically, you need to identify terrorist task forces as such before they damage beacons. To do so, you require information about the hats in the box. Information is acquired from an *Information Broker*, as shown previously in figure 1. The Information Broker will respond to questions from you, such as, *Where is Hat_{27} now?* and it will also provide information by subscription to analysts' tools (which in turn make requests for information). For example, a tool might issue a request like, *Identify everyone Hat_{27} meets in the next 100 steps*, or, *Tell me if Hat_{27} approaches a beacon with capabilities c_1, c_7 or c_{29}*.

Information comes at a price. Some 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 to the request framework: the cost of information and noise. Together, these components make the Hats simulator an experimental environment in which to study the economics of the value of information in the task of identifying malevolent behavior in the Hats domain.

The cost of information

Three kinds of information are available from the Information Broker for free: (1) information about the population assumed to be available to the user (e.g., who the known terrorists are), (2) information about the Hats simulated world (e.g., the world-map dimensions, the list of beacons and their names, and the list of all of the capabilities that exist), and (3) some event bookkeeping (an event history, list of currently arrested hats, etc.). Information types 1 and 2 are determined when the simulation is initialized and do not change over time; type 3 is updated at each step of the simulation.

For Information Broker requests that require payment, the amount paid (a real number) will determine a base probability, which in turn determines the accuracy of the requested information. In the current implementation, increased accuracy requires exponentially more "algorithmic dollars." The payment function, shown in Equation 1, maps payment to probability.

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

The same function is applied to every payment-based request.

Noise model

The development of a suitable noise model and the schemes for how noise is applied to requested information is, itself, an entire field of study. We list here three approaches, in increasing order of complexity:

1. The analyst may only request a particular piece of information once and must choose the level of payment for (and therefore quality of) the information at the time of request. No additional requests may be made. The analyst must decide at the time of request the value of that piece of information.
2. The analyst may request information multiple times. However, in order to receive information beyond previous request(s), the analyst must pay more than previous requests (according to the payment scale). Repeated requests at or below the same level will return precisely the same information, but paying more returns less noisy versions of the original request.
3. The analyst may request information multiple times, paying varying amounts. This approximates the existence of multiple information sources (for example, acquiring information from multiple witnesses of an event). Such multiple information sources might be made explicit, introducing the potential of modeling sources of trust relationships.

Many other schemes are possible, but these provide some indication of the wide variety of approaches to modeling noise.

The current implementation of the information broker employs the first scheme. The payment the analyst specifies determines the base probability p of whether, and to what degree, the information requested will be noisy: with probability p, the information requested is returned in its entirety, otherwise the noise model is applied.

Although the basic noise application scheme is simple, there still is a variety of types of information each of which requires a different noise model variant. The table in Figure 3 summarizes how different types of requested information are made noisy. Following the noise application scheme, analysts may only request each piece of information *once*. Some information, such as the capabilities currently carried by a hat (`ib-hat-capabilities`), is updated at each tick, so the analyst may request that information once each tick. Other information does not update, such as information about the members of a meeting that took place (`ib-meeting-participants`) – here the analyst is allowed only *one* request of this information. The column labeled "Request Frequency" shows the frequency with which an analyst may request information.

Request Return Type		IB Request	Request Frequency	Mean List Length … if information Exists	Mean List Length … if information ¬Exists	For Each Element: … if information Exists	For Each Element: … if information ¬Exists
List	hats	ib-location-contents	Each tick	length of Original list	2	If *Noisy*: Select Random, Else… Select from Original Without Replacement	NIL
		ib-meeting-participants	1				
	capabilities	ib-hat-capabilities	Each tick				
	trades	ib-meeting-trades	1		3		
	times	ib-meeting-times	Each tick	3	3	Select without replacement from most recent; NIL if none left	NIL
Element	time	ib-hat-obituary	Each tick			If *Noisy*: Exponential random with mean 1; If 0, then NIL	
	location	ib-hat-location	Each tick			If *Noisy*: Random location	
		ib-meeting-location	1				

Figure 3: Noise model

The table is split into two groups based on whether the requested information is a single element (bottom portion of the table) or a list of elements (top portion of the table).

Lists Noise is applied to lists in two stages: first, noise affects the length of the list to be returned, and then noise is applied to each element of the list. The two main columns on the right-hand side of the list portion of the table indicate how noise is applied to list-length and to each element; in either case, noise is applied differently depending on whether or not the request is for information about entities that exist or events that occurred – true, non-noisy information about entities that do not exist or events that did not occur is returned as `NIL`.

List length is determined by sampling a random value from a normal distribution with a standard deviation of 1.0 and a variable mean.[2] The "Mean List Length" column describes how the mean for the normal sampling distribution is set. For example, if the analyst requests the current contents of a Hats world location (using `ib-location-contents`), and there are in fact 3 hats at that location, then the length of the potential return information (3) determines the mean; subsequently, the noisy length of the list of hats that will be returned as a result of the request will be a random number selected from a normal distribution with mean 3, standard deviation 1. If, on the other hand, no hats exist at that location, then the mean of the normal distribution is 2 (as specified in Figure 3). These means have been chosen because they resulted in reasonable values during experimentation. If the analyst requests information involving a list and the selected random value rounds to 0 or lower, then the return value will be an empty list (or `NIL`).

Next, assignments are made for each element slot in the list to be returned. For each element, the noise model again uses the base probability p to determine whether the element slot will be noisy. If it is to be noisy, an element of the requested information type is uniformly randomly selected (with replacement) from the set of *all* elements of that type. For example, a random hat would be selected from *all* existing hats. In the case of trades, a noisy trade consists of two randomly chosen hats and one randomly chosen capability. With probability $1 - p$ the element will *not* be noisy. In this case, the element will be uniformly selected, *without* replacement, from the list of elements that would be returned if the information was uncorrupted; if the request is for information that does not exist, then that element of the list will be empty.

Elements The elements portion of the table describes noise applied to information consisting of single elements. Random locations are selected when noise is applied to location information. A random location is chosen by selecting two random numbers, one for each coordinate component (x, y). The random numbers are selected from a standard normal distribution (mean 0, standard deviation 1.0). The value selected is then multiplied by the entire range of the x or y axis of the Hats World game board and divided by 10. This heuristic returns reasonable distances relative to the size of the game board dimensions. The adjusted value is then added to the true coordinate component. If the adjusted coordinates exceed the boarders of the game board, the amount exceeded is "reflected". For example, if a hat is at x-coordinate 3 and the adjustment is -5, then rather than returns an x-value of -2, the value is "reflected" to x = 2. If, on the other hand, the Game World maximum x size is 10 and the adjusted value is 12, then the value is "reflected" to x = 8.

Actions

In addition to requesting information, the analyst playing the Hats game can also change a beacon's alert level and arrest hats. These actions affect the analyst's performance score (described in the next section).

Beacon Alerts

Each beacon can be set, by the analyst, to be in one of three alert levels: `off` (default), `low` or `high`, indicating no threat of an impending attack, a chance of an attack, and a likely attack, respectively. The Hats Simulator keeps track of beacon alert levels, including the amount of time a beacon alert is elevated (`low` or `high`) and whether actual attacks actually occur during elevated alerts. These statistics include counts of "hits" and "false positives," where "hits" $\equiv$ occurances of an attack while alert is elevated (above `off`), and "false-positives" $\equiv$ elevated alerts that begin and end with no beacon attack occurring. These scores are kept for both `low` and `high` alert levels. In general, the goal is to minimize the time beacon alerts are elevated, and `high` alerts are deemed "more costly" than `low` alerts. On the other hand, if an attack *does* occur on a beacon, it is generally better to have a higher alert level.

Arresting Hats

Analysts can also issue an arrest warrant for hats in order to prevent beacon attacks. A successful arrest results when the

[2] The sampled value is rounded to make it a valid list length.

arrested hat is currently a member of terrorist taskforce. Arrests of any other hats, including hats that are terrorists but not currently part of a terrorist taskforce, result in arrest failure and are equivalent to a *false arrest* (a false positive). This is an important aspect of the semantics of "being a terrorist" in the Hats model: one can be a terrorist but not be guilty of any crime. Under this interpretation, "being a terrorist" is a matter of having a propensity to engage in terrorist acts. A terrorist act in the Hats domain is participating in an attack on a beacon. Thus, terrorist hats must be engaged in an ongoing terrorist activity to be successfully arrested. According to this model, if a hat previously committed a terrorist act but is *not currently* part of a terrorist taskforce, it cannot be successfully arrested.

Successful arrests do not guarantee saving beacons. As noted, a beacon is only attacked when some subset of members from a taskforce carry the requisite capabilities that match the target beacon's vulnerabilities engage in a final meeting on said beacon. Thus, it is possible to successfully arrest a terrorist taskforce member but the other terrorist taskforce members still have the requisite capabilities to attack the beacon. If, on the other hand, the analyst successfully arrests a terrorist taskforce member carrying required capabilities that no other taskforce member carries, then the taskforce meeting will take place on the beacon, but it will not be attacked. This is counted as a "beacon save."

In the present version of Hats, the successful arrest of a hat does not remove it from the game – the hat will still behave as if it had not been arrested. It will still move toward goals and go to meetings. However, it will not be able to trade any of its capabilities nor contribute to enabling a beacon attack – it will be as though the hat were not present.

Currently, the statistics on beacon alert "hits," "false positives," "successful arrests," and "false arrests" are not combined into a uniform cost model. They are simply reported as additional measures of comparative player performance.

Scoring Analyst Performance

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

- The cost of acquiring and processing information about a hat. This is the government in the bedroom or intrusiveness cost.
- The cost of falsely identifying benign hats as terrorist
- The cost of harm done by terrorists

The skill of analysts and the value of analysts tools can be measured in terms of these costs, and these are assessed automatically by the Hats simulator as the analyst plays the Hats game. The final report generated by the Hats Simulator after terminating a simulation run is divided up into four categories, as described in the following list:

- Costs: the total amount of "algorithmic dollars" spent on information from the Information Broker.
- Beacon Attacks: including the total number of terrorist attacks that succeeded and the total number of attacks that were stopped by successful arrests.
- Arrests: the number of successful arrests and the number of false-arrests (false-positives)
- Beacon Alerts: the number of low and high hits (the number of raised alerts during which an attack occurred), and the number of low and high false-positives (the number of raised alerts during which no attack occurred).

Discussion

We are told by intelligence analysts 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 with low domain knowledge overhead. No doubt Hats will change over time, but we will strive to keep it simple. Big, complex, covert, but 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 theory and information – can be tested and compared.

To meet these goals, our ongoing development of Hats includes the following: (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.

History and Acknowledgments

The Hats Simulator was conceived of by Paul Cohen and Niall Adams at Imperial College in the summer of 2002. Cohen implemented the first version of Hats, and David Westbrook, Clayton Morrison, Andrew Hannon and Michiharu Oshima have subsequently developed major portions of the simulator. Thanks also are due to Gary King for help. Bob Schrag at IET contributed useful ideas and built a simulator similar to Hats for DARPA's Evidence Extraction and Link Discovery (EELD) program. Work on this project was funded by EELD.