Comput Math Organ Theory (2013) 19:370–401
DOI 10.1007/s10588-012-9136-8

SI: BRIMS 2011

Inducing models of behavior from expert task performance in virtual environments

Bradley J. Best

Published online: 29 September 2012

Abstract We developed an end-to-end process for inducing models of behavior from expert task performance through in-depth case study. A subject matter expert (SME) performed navigational and adversarial tasks in a virtual tank combat simulation, using the dTank and Unreal platforms. Using eye tracking and Cognitive Task Analysis, we identified the key goals pursued by and attributes used by the SME, including reliance on an egocentric spatial representation, and on the fly re-representation of terrain in qualitative terms such as "safe" and "risky". We demonstrated methods for automatic extraction of these qualitative higher-order features from combinations of surface features present in the simulation, producing a terrain map that was visually similar to the SME annotated map. The application of decision-tree and instance-based machine learning methods to the transformed task data supported prediction of SME task selection with greater than 95 % accuracy, and SME action selection at a frequency of 10 Hz with greater than 63 % accuracy, with real time constraints placing limits on algorithm selection. A complete processing model is presented for a path driving task, with the induced generative model deviating from the SME chosen path by less than 2 meters on average. The derived attributes also enabled environment portability, with path driving models induced from dTank performance and deployed in Unreal demonstrating equivalent accuracy to those induced and deployed completely within Unreal.

Keywords Virtual environment · Avatar · Spatial representations · Instance-based learning · Cue learning · Hierarchical task network

Our goal in this study was to explore methods for creating a cognitive model of subject matter expert (SME) behavior in a real-time video game by relying as directly

B.J. Best (✉)
Adaptive Cognitive Systems, 909 Harris Avenue St. 202D, Bellingham, WA 98225, USA
e-mail: bjbest@adcogsys.com

as possible on the recorded traces of the expert's behavior, automating as much of the process as possible. Cognitive models are often developed in one of two ways: (a) through extensive task analysis and knowledge engineering to produce rule-based systems, or (b) by inducing performance models from the data, using paradigms known as case-based reasoning, instance-based modeling (Aha et al. 1991) or learning from examples (Simon and Zhu 1988; Simon and Gobet 1996). This second approach is the focus of our work.

One common way to situate a cognitive model is to develop it within a cognitive architecture (Newell 1990). Recently, the ACT-R cognitive architecture (Anderson et al. 2004) has been used as a platform to develop models using an instance-based methodology (Taatgen and Wallach 2002; Gonzalez et al. 2003; Best and Lovett 2006). ACT-R itself is a production-system based framework for developing cognitive models that incorporates a symbolic layer largely concerned with rule-based (procedural) processing, and combines this with a subsymbolic layer that incorporates an activation-based (declarative) memory, and various forms of statistical learning.

An instance-based ACT-R model typically uses some set of productions to simulate the broader scale of cognition, and then performs retrievals from declarative memory to perform a similarity-based lookup for cases similar to the current context that have already been memorized. Equipped with a bootstrapping mechanism (e.g., the ability to randomly guess on initial trials), an instance-based ACT-R model can operate in an environment, learning from its own actions as it performs the task (e.g., Best et al. 2008). Many of these models, however, have focused on discrete tasks, where the instance-based model produces exactly one action when requested to do so and then returns control to the higher-level logic that drives it. It is unclear how suitable these models are for continuous control in real-time environments, since these tasks have the potentially to produce large and computationally cumbersome memories (see Douglass et al. 2009, for an approach to managing large-scale ACT-R memories), and it is also unclear how individual instance-based task models might be sequenced or stitched together. It is unclear how suitable these models are for continuous control in real-time environments, and how individual instance-based task models might be sequenced or stitched together.

Our aim was to evaluate the feasibility of using instance-based modeling to induce a cognitive model from a recorded real-time continuous performance of a subject matter expert (SME), as demonstrated in a virtual environment, with our initial modeling explorations focusing on the ACT-R cognitive architecture. The SME piloted a virtual ground vehicle, a tank with an independent turret and main chassis, and performed a variety of tasks that included combat, seeking cover, scouting terrain, and driving a twisty forested path. We translated the continuous behavior of an SME performing a task in a virtual environment into a set of discrete instances to create the learning examples. These instances could include any of a number of ground truth attributes from the simulation environment, or derived attributes. Furthermore, many instance-based models focus on a particular narrow task (e.g., a typical laboratory task), but identifying the tasks to be performed in a larger scale task (for example, an unstructured combat operation) is a prerequisite to being able to apply an instance-based model to a narrow task that it is suited for. Identifying and selecting an appropriate

task given an unfolding scenario and extracting and determining which instance to use during task performance are significant technical challenges. Addressing both the task selection problem and the task performance problem are critical requirements to applying instance-based models to large-scale modeling of behavior. The remainder of this report will focus on these issues.

1 Task environments

Virtual environments can place severe demands upon computational resources, making it challenging to develop cognitive models capable of acting within real-time constraints (Best and Lebiere 2006). That is, while it is straightforward to simulate real-time performance given a simulation that is allowed to run slower than wall clock time, when coupled to an embedded real-time system, a cognitive model must execute its perceive-act cycle within the constraints of wall clock time, and that constraint can be extremely demanding. In particular, agent behavior in the environment becomes unacceptable using cycle times of more than 200 ms, while less than 100 ms tends to produce smoother behavior, and a response of less than 50 ms is required to achieve targeting behavior (e.g., Best and Lebiere 2006). The actual response time is especially critical since a cognitive entity must address its own "lag" in producing actions—the world it perceives, if dynamic, is not likely to be in the exact same state 50 ms in the future. As this lag between perception and response grows, so too does the error produced by a cognitive system attempting to produce responses in a real-time virtual environment.

There are possibly similarities between the minimum cycle time needed to generate "non-jerky" actions in a virtual real-time task environment and the estimates of cognitive cycle time. Multiple researchers, in attempting to formulate computational theories of cognition, have asserted a fundamental discrete human decision cycle of approximately 50 ms (Newell 1990; Meyer and Kieras 1997; Anderson and Lebiere 1998). This cycle, and in particular its relation to driving tasks, has been explored in detail by Salvucci and colleagues (e.g., Salvucci and Gray 2004; Salvucci 2006; Brumby et al. 2007). Their work has demonstrated that simulating human behavior in real-time tracking tasks such as steering can be accomplished with a high degree of fidelity with an effective discrete cycle of 50 ms (though also requiring some interleaving of tasks if the central cognitive processor performs a variety of tasks).

This cycle time is somewhat independent of the data sampling rate, which needs to be frequent enough to capture snapshots of the performance, but need not necessarily be the same as the cycle time for performance. Some researchers have used sampling rates as rapid as 10 ms for capturing sensor data from driving tasks performed by human participants (e.g., Best et al. 2008). Similarly, Ball and Gluck (2003) report using a simulated task environment for a Predator Uninhabited Air Vehicle (UAV) with a simulation "heartbeat" of 20 ms. At the other end of the scale, Scott and Cummings (2007) report collecting human performance data in a real-time simulated command and control task environment at the rate of 6 cycles per second, or 166.67 ms per cycle. Finally, some researchers have sidestepped the issue of cycle time by only collecting and logging data snapshots coincident with user actions (e.g., Schoelles and Gray 2000). The critical latent issue here is one of pairing actions with percepts when

actions are discrete. If an action only occurs once, a higher cycle rate would result in more "empty" action entries in a log of simulation state paired with participant actions. We will return to this issue later.

2 Paper roadmap

This study pursues a set of interconnected goals, and reports on the progress of intermediate steps in pursuing the overall goal of inducing a cognitive model directly from subject matter expert task performance. The paper is thus organized as follows. First, we detail the method of data collection in the virtual task environment, including a discussion of the tasks used, the subject matter expert, the raw results of data collection, and the results of a cognitive task analysis on these results. Second, we discuss a methodology for applying transformations to the "raw" data to produce data that has similar information content compared to the information identified in the cognitive task analysis. Third, we apply learning methods to both task performance (actions) and task selection (goals) in the transformed task data, analyze and determine the performance of selected algorithms on the task data, examine algorithmic implications for data storage and processing. Fourth, we connect the various pieces and present a complete processing model for a particular task, path driving. Fifth, and finally, we present our conclusions.

3 Method

3.1 Participants and models

SME The subject matter expert (SME) was a habitual heavy video game player and a consistent user of the dTank and Unreal simulation environments over the period of more than a year, including the different phases of this study. The SME trained against the synthetic opponents until able to consistently defeat three simultaneous opponents with near certainty, and could repeatedly drive various obstacle courses without colliding with obstacles, as well as complete various other tasks within the synthetic environment. We estimate that the SME logged several hundred hours in the dTank simulator, as well as several hundred more in the Unreal simulator.

CIBRE The Cognitive Instance-Based Rule Engine (CIBRE) architecture is a lightweight instance-based learning system implemented in the LISP programming language (Best and Gerhart 2011). CIBRE learns from a data set of snapshots in time, or instances, and uses knowledge elicited from those instances to predict a response based on the values of attributes contained in the current context. The instances chosen to base a response on are selected based on similarity to the current context, and the critical calculation is thus a "lookup" in the instance memory of the relevant prior experiences. Each instance contains one or more attributes and at least one response. Particular task models are stored within partitions in CIBRE's instance memory, preventing tasks from interacting.

CIBRE makes one pass through a training set and stores any instance that makes a novel contribution into its instance memory. When engaged in an instance-based

loop, as the model provides responses, the context chunk is compared attribute-by-attribute to all instances in the instance memory (the context chunk can either be an instance from the testing partition or the current state of the environment in which the model is embedded). The similarity of all of the attributes is summed to calculate the similarity for the entire instance. Once the most similar stored instance is found, the model returns the response from that instance as the predicted response for the current context. The response can drive behavior (e.g., a response corresponds to and is translated into a key-press in the dTank behavioral domain) or can simply be a classification (e.g., the response in the validation domain named "mushroom" is whether or not the mushroom is poisonous).

3.2 Materials and procedures

SME tasks and battle simulations We recorded and analyzed real-time performance and behavior of the SME in a virtual environment. The SME participated in four complete battle simulations, and a number of constrained scenarios. In these simulations the SME piloted a virtual ground vehicle, a tank with an independent turret and main chassis. The constrained scenarios that the SME participated in were:

- Aim: Rotate the tank turret to search for an opponent, and fire projectile at the opponent
- Avoid-dense: Drive around the map in 750 seconds without hitting any buildings or woods
- Chase-long: Chase an enemy tank for 750 seconds
- Path: Drive around a grass path lines with trees within 400 seconds
- Seek-cover: There are two blue tanks, one of which will attack. Seek cover in a map containing hills, trees, and towns.
- Avoid: Drive around the map in 750 seconds without hitting any buildings or woods
- Chase-short: Chase an enemy tank for 150 seconds.
- Target. There is a blue officer on the map. Find the target and drive the tank directly to it.
- Shoot: There is a blue officer on the map. Rotate the turret to the target and fire a projectile at it.

The majority of data collected were complete battle simulations in which the SME participated in a 1,000 second simulation against three enemies (two tanks and one officer on foot), providing a maximum of 10,000 discrete samples per battle (battles terminated either at the end of 1,000 seconds, when all opponents were defeated, or when the SME was defeated). The constrained scenarios were run a variable number of times, dependent on the performance of the target models and the length in time of the typical scenario, but no scenario was recorded fewer than five times. One constrained scenario, the Path scenario, and the complete battle scenarios were chosen for further exploration during this study.

Virtual environments We worked with the dTank virtual environment (Ritter 2008; Fig. 1) and the Unreal Tournament game engine for task performance and data collection. The dTank environment provides a "bird's eye" view of a battlefield, including

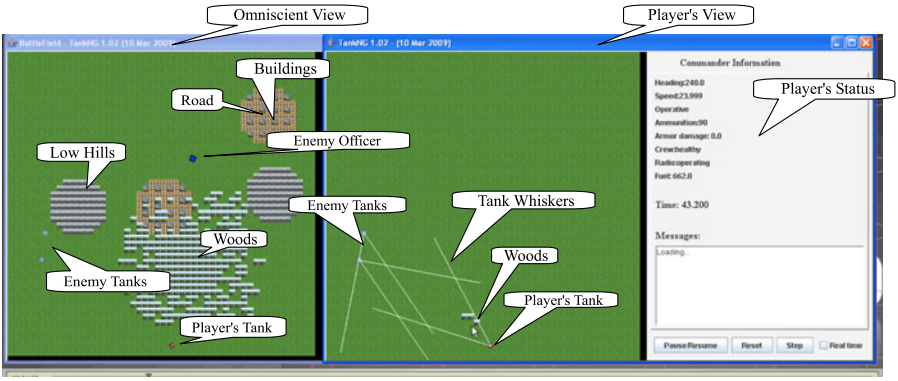

Fig. 1 A screen capture of both the omniscient (*left*) and commander views (*middle*) of a dTank battle. Bubble captions have been added for clarification purposes

an omniscient view for third-party observers, and an actual commander view providing limited visibility for terrain and obstacles through a moveable viewing cone (in the direction of the turret). In the commander view, the player, or in this case, the SME, can only see what is in the "whiskers", or arrows, that are immediately in front of him. The actual map, shown on the left, is the same in the commander view, but is unseen by the player. The Unreal Tournament game engine is a real-time three-dimensional video game engine that provides support for a variety of "first person shooter" games, and includes support for vehicles such as tanks. Our core methodology has been to prototype and develop interaction models using the lightweight dTank virtual environment, and then port the resulting models to the Unreal platform. We will first focus on the dTank environment, and will return to the Unreal environment later.

Instance retrieval rates The dTank environment is not a fast-paced environment, and action cycles of 200 ms are sufficient for producing reasonable behavior. However, we chose to use a minimum cycle time of 50 ms (20 Hz) for two primary reasons. First, we were interested in examining the practical difficulties in performing instance-based retrievals at a rate shown to match many basic findings in cognition. Second, we anticipated integrating models with the Unreal Engine, which has stringent real-time demands, and moves at a much faster pace. We will return to this later, illustrating the factors that limit the speed of instance-based processing.

Collecting and segmenting data from human task performance We collected continuous data in two different ways: (a) we collected behavioral streams from the SME during battle simulations in the dTank domain, and (b) we collected a series of scripted behaviors in which the actions were more constrained.

For both of these situations, the data were automatically converted offline into instances based on a dTank message frequency of 5 Hz. This frequency determines the rate of which responses are collected. Responses in dTank, however, are often "discrete": they occur at exactly one point in time. As a result of this, at a very high sampling frequency, almost no conditions would result in a response. To account for

this, responses that occurred between sample points (during the 200 ms interval) were pushed forward to the next sample point.

These data were collected for two entirely different purposes. Our overall goal is to produce instance-based models of task behavior, but this presupposes the model can appropriately select a unit task to execute. Thus, we aim to solve two separate problems using instance-based decision making. The first problem is task selection, while the second problem is task execution (given a task to execute). We will deal with each of these in turn.

The SME behavior was analyzed into distinct goals and behavioral sequences through a Cognitive Task Analysis (CTA). We segmented the battles into higher-level categories of either goals or behavioral sequences. Doing this repetitively on a broad scale is largely unrealistic; the time to hand-annotate 5,000 lines (per simulation) created a severe bottleneck in the data collection process. Our process, however, involves producing a model that can automatically learn how to perform task selection given a sufficient set of learning data. The remainder of our data collection has focused on task execution, where a lower level task is given as the performance goal to a human participant. Details of one such lower level task—following a tree-lined path—is presented in two different virtual domains later in this paper.

Determining the task structure We asked the SME to play a round of dTank battles, which were recorded for later analysis. During these assessment battles, the SME was only able to see the typical 'commander view' window. However, for the purposes of illustration and analysis, the commander view and the 'omniscient view' were both recorded (see Fig. 1). The omniscient view was obscured from the SME's sight during the battle trials. Each battle was segmented into discrete 'behavioral chunks', where each such behavioral chunk is identified by the elapsed battle time at which it began. This breakdown of each battle into discrete behavioral chunks was then later refined in the expert review and assessment of SME battle performance.

A portion of an example verbal protocol, from the beginning of Battle 1 until the first time the SME fires, appears in Table 1. It corresponds to Fig. 1, a screen capture (with both omniscient and commander views) taken at the onset of behavioral chunk beginning 43200 milliseconds into the battle (the commander view is on the right, and includes the 'turret whiskers' of the two enemies and the human-controlled tank). A video screen capture of the SME battle performance was recorded for later analysis.

Eye-tracking dTank trials We had the SME perform a series of dTank battle trials during which we recorded his eye movements with a ViewPoint EyeTracker by Arrington Research®, using terrain maps drawn from the map annotation activity.

Using instances: the path driving model We developed a model of path driving using the CIBRE instance-based rule engine (Best and Gerhart 2011) to simulate SME behavior. The SME completed the path five times, avoiding obstacles (trees) along the way, and this data was used to train the CIBRE agent. We compared each recorded position of the CIBRE agent against the closest recorded SME position from among the five paths.

Table 1 Protocol summary for the first 61200 milliseconds from Battle1

Time (ms)	Summary
0	BEGIN BATTLE
0	scout perimeter
3600	see Opponent
7000	start for cover, keep scouting
7600	see 2nd Opponent
9000	head for cover
12600	they see me, keep an eye out as seek cover
19400	good cover, wait to be loaded, monitor Opponent position
32400	It's been a while, seek Opponent location
43200	see Opponents again, they're still close together
45000	stay out of sight
54600	nearly loaded; take another peek; they don't see me
60400	loaded, select nearest Opponent as target
61200	FIRE

4 Results

4.1 Determining the task structure through cognitive task analysis

Following the argument that expertise leads to simple yet sophisticated systems of representation, we revisited the video battles and attempted to categorize the dTank SME "behavioral segments" in a way that expressed the SME's battle representation not at the key-press level but at the intention level. For example, the SME would often remain as concealed as possible while continuing to peek out at an opponent he was stalking. This approach led to the identification of 7 discrete Tactic Categories that were then used to classify the SME behavior:

- Fire At Opponent,
- Monitor Opponent Position,
- Scout Perimeter,
- Seek Cover,
- Seek Opponent Location,
- Stay Out Of View, and
- Target Opponent.

These seven categories were determined based on a single rater's perception of the SME behavior and comments during the CTA. The SME reported engaging in multiple battle "tasks" simultaneously. With that in mind, we also explored categorizing each behavioral segment by at least one and as many as three hierarchically ordered Tactic Categories: Primary Tactic, Secondary Tactic and Tertiary Tactic.

Four complete SME battles were coded using the set of 7 Tactic Categories. Each battle consisted of approximately 50 individual behavioral segments, totaling 206 individual segments across all battles. Although the complete set of possible Tactic Category combinations (given a minimum of 1 and a maximum of 3 categories per

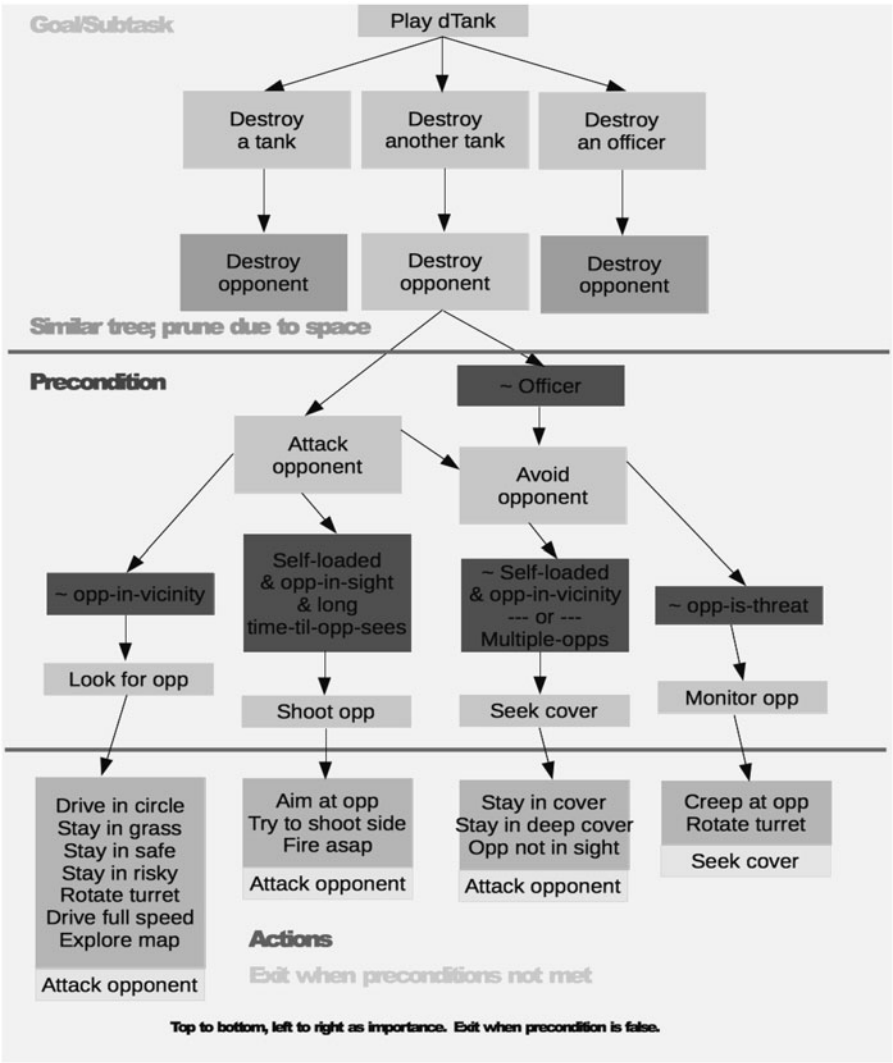

Fig. 2 The task structure used by a subject matter expert (SME) for a dTank battle

chunk) is 259, only 27 actually came up in these 4 battles. Interestingly, only 5 of the 27 unique Tactic combinations were used in all 4 battles; 5 were used in 3 of the 4 battles; 7 were used in only 2 battles, and 10 (a full third) were unique to a single battle. Four battles is a relatively small data set, yet this analysis indicates that this methodology is in fact sufficiently robust to capture both the commonalities and the unique situations that appear across different battles in a meaningful yet succinct manner. The resulting task structure is shown in Fig. 2.

We used this structure, along with the cue/attribute structure described in the subsequent section to annotate instances and determine if it was possible to predict goals from available cues.

4.2 Determining the cue/attribute structure

The virtual tank commander should, given a set of circumstances, produce behavior that approximates that of the SME in the same circumstances. This depends on identifying the context: What is the SME seeing and attending to within the battle environment that elicits that specific behavior? One obvious source of context is the raw information provided by the dTank and Unreal environments. That raw information, including attributes such as Speed, Heading, Opponent Type (Officer versus Tank), and Time, can easily be provided as input for the virtual commander. However, the CTA revealed that the SME filters and interprets environmental features in a way that far exceeds the simple consumption of raw data. Examples of these less direct—but at least equally important (as reported by the SME in the CTA)—attributes include proximity to cover from fire, and time until opponent is loaded. These 'cognitive' cues must be derived from the (simulation environment) ground truth (e.g., Best and Lebiere 2006; Lathrop 2008).

4.3 Spatial representation in virtual environments

An important aspect of context recognition is the ability to recognize specific terrain configurations that might help determine the appropriate course of action. One such attribute that the SME routinely utilized is the proximity to cover. To adequately simulate human behavior, a model must interact with the virtual environment in much the same way that the human does; this may entail some form of SME-like visual parsing of the environment by the synthetic commander. The next sections detail our efforts to understand and quantify the SME's visual parsing of the dTank environment and cues.

This is of particular interest in that large areas of the terrain maps remain underspecified or unknown to the tank commander (whether SME or virtual agent) during battle. This is rather different from the majority of typical training environments where much of what is present is known to the operator or agent. Use of the dTank environment in this project therefore afforded a unique opportunity to investigate such situations that are relatively rare in test environments but relatively common in the real world. We believed that an investigation of the SME's representation of the dTank battle space would be an essential component underlying the development of a virtual tank commander. To that end, we proceeded with two tracks of investigation into the SME's visual representation of the dTank environment: Test Terrain Map Assessment, and Eye-tracking dTank trials.

4.4 SME assessment of test terrain dTank maps

The CTA revealed that the SME was very sensitive to the specific configuration of the terrain both leading up to and during engagements with opponents. This suggested that it might be necessary for the model to make SME-like assessments of the terrain. To enable this, we produced a set of 21 test terrain maps (see Fig. 3) and asked the SME to assess the omniscient view of those terrain maps.

The SME verbally indicated that any areas he had indicated as 'high potential threat' (YELLOW) would also be 'strategically weak areas of engagement during

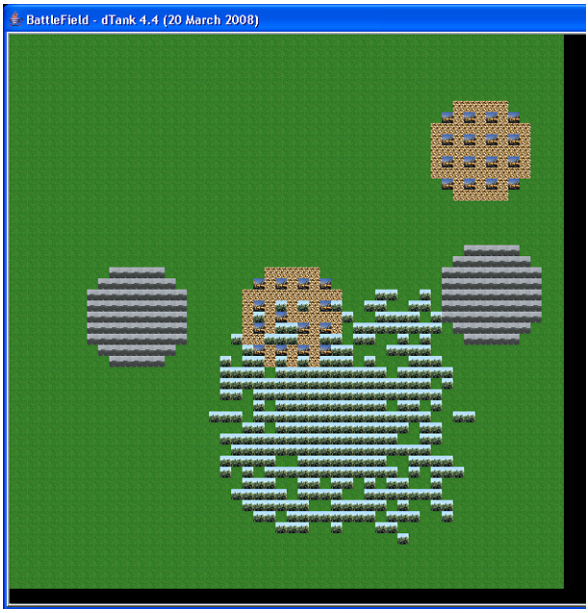

Fig. 3 Example test terrain map

battle' (RED). However, not all RED (strategically weak areas) are also 'high potential threat' (YELLOW) areas. The SME reported that in the absence of a specific battle (including tank position) configuration, the areas of 'low confidence regarding opponent tank location' (MAGENTA) would be redundant with areas of 'high potential threat'. Additional information would be necessary to disambiguate these two categories and to specify the areas of 'high confidence regarding opponent tank location' (BLACK). Figure 4 shows where the SME annotated a map and verbally reported that he was being "extremely careful" with the placement of the 'strategically' strong areas of engagement during battle' (GREEN) to hug the periphery of the low hills area. This is in marked contrast to the placement of the 'strategically strong' areas around the towns. We investigated this information in the eye-tracking analysis of the SME battle behavior.

4.5 Eye-tracking dTank trials

The omniscient view terrain map and the SME's eye movements during one battle appear in Fig. 5a. In this display, saccades, which are defined as eye movements that exceed a qualitative threshold velocity of 0.2 on a scale of 0 to 1 (a parameter of the ViewPoint data analysis software), appear in blue and fixations appear in green. The saccades show a pattern of jumping from the current tank position to relevant information on the information panels. And then right back, while the fixations are most heavily focused on the battlefield elements (self and opponents). Figure 5b displays a superposition of fixations (in pink; the saccade traces removed for ease of interpretation) on the omniscient version of the terrain map. Note that the SME never sees this omniscient view of the terrain map during the battle.

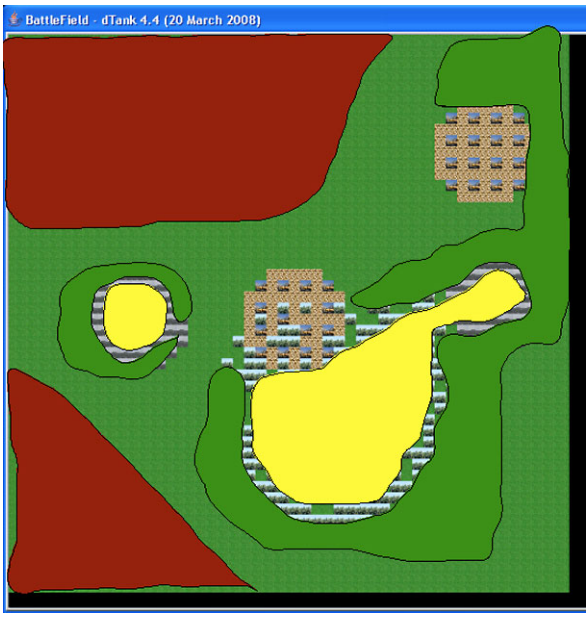

Fig. 4 Map annotated by the SME, where *yellow* is classified as a 'high potential threat', *red* a 'strategically weak area of engagement during battle', and *green* a 'strategically strong area of engagement during battle'

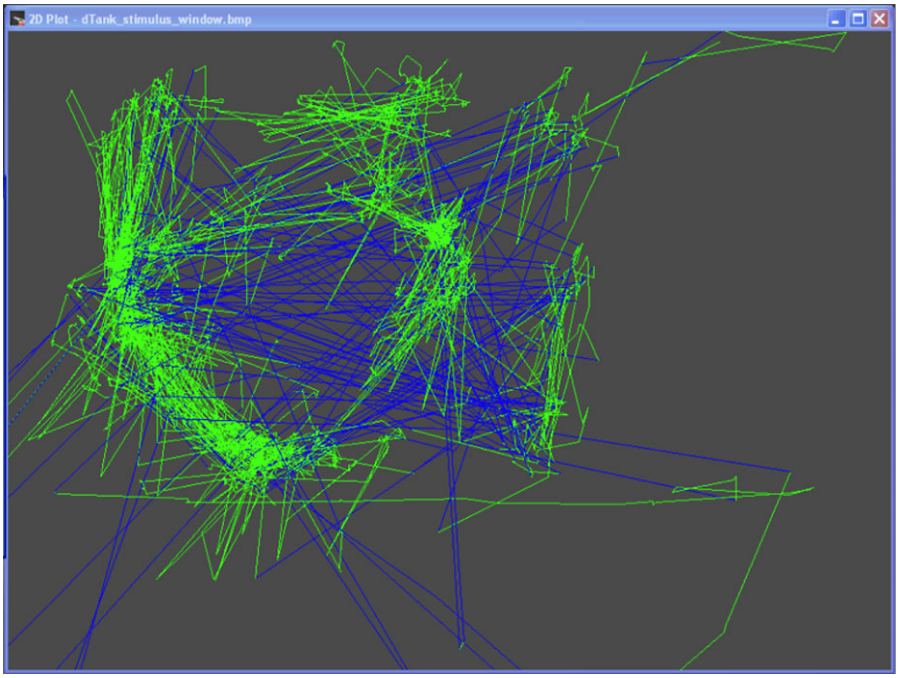

Fig. 5a Example SME eye trace with saccades and fixations displayed

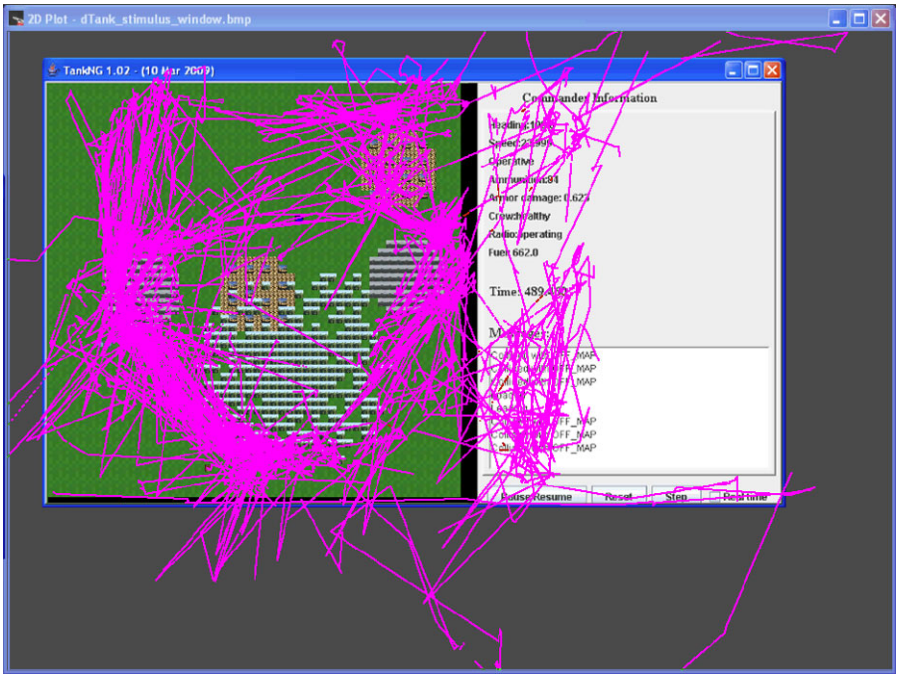

Fig. 5b Example SME eye trace that is superimposed upon the omniscient map, with only the fixations displayed

The SME spends a small amount of time looking at the battle statistics (i.e., heading, speed, time, etc.), while the vast majority of the SME's gaze is directed at the location of the opponents and at the terrain immediately in the vicinity of the SME's tank. The fact that the SME spends much of his time regarding the immediate vicinity is not surprising: that is the only visual information about the terrain that is present to the human commander (see Fig. 1). However, it is very interesting that the SME spends so much of his time gazing at the location of the opponents, even if the opponent is not currently visible.

To quantitatively report how much time was spent looking at the different regions of the terrain map, we generated a 'weather map' version of the eye tracking results. In this display, the terrain map has been segmented into a 10×10 grid, and the total fixation time for each of the 100 terrain areas has been color coded. The total fixation times associated with each color is given in Table 2 ('warmer' colors represent a disproportionate amount of time spent focusing on that map region), while the corresponding frequency map is shown in Fig. 6.

The comparison of the different visual perceptions of the terrain map by the SME reveals that the SME's eye movement during the battle is strongly influenced by the location of the opponents during the battle. This is not surprising—the goal of the battle is not to scout the terrain, but to seek and engage opponents. This analysis further reveals that the SME spends a great deal of time looking at the opponent—sometimes even when there is nothing to see. At this stage of the battle, the SME has

Table 2 Color coding of total fixation times as displayed in the eye tracking 'weather maps'

Color	Total Fixation (ms)
Red	800 <
Orange	200 – 799
Yellow	100 – 199
Green	25 – 99
Aqua	10 – 24
Blue	1 – 9
Gray	< 1

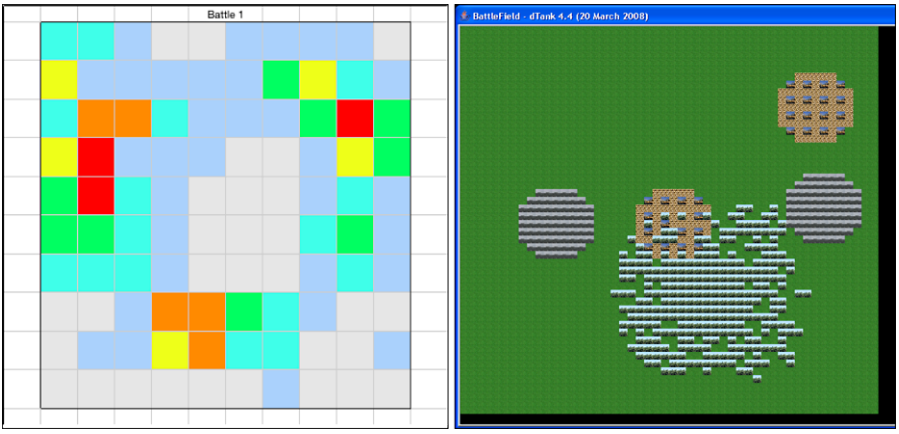

Fig. 6 The total fixation time in the 100 regions of the Battle 1 terrain map, with the terrain map showing on the right

destroyed 2 of the 3 opponents and has located the final opponent tank—it is in the middle of the trees. The vast majority of the time, the SME can see absolutely nothing of the middle of those trees—yet, as displayed in the associated 'weather map', the SME spends an extraordinary amount of time focusing on that region of the map, looking where he thinks the opponent is, even when the opponent is out of view.

The SME's visual fixations are less concerned with studying terrain features, but instead trying to locate and keep track of the opponents. The SME is very aware of the terrain features, and uses them to his advantage during battles, but the knowledge he is maintaining is less spatial and more environmental. In particular, the eye fixations are directed towards particular areas of the map. This leads to the consideration of what

environmental information the SME is detecting, representing and utilizing during the course of a battle, and how to best represent this information in attributes.

4.6 Map annotation

During the Cognitive Task Analysis (CTA), and most directly as a result of using the eye-tracking methodology, it became apparent that the SME was not merely paying attention to and making decisions based on the raw environment information he was getting from the game. Rather than keeping track specifically of where woods, buildings, hills, and grass were located, the SME was maintaining a representation of which areas of the map were safe to be in, which provided a good place to hide and which were dangerous. The key finding is that the SME was directing visual saccades towards areas that had particular properties related to the context of the terrain and mission, but that were not visual properties of the environment. That is, the SME selectively devoted attention to terrain areas that were indistinguishable from other terrain areas just based on visual features; one patch of grass that might attract substantial attention had exactly the same visual features as any other patch of grass, and thus the visual features of the targets did not provide sufficient explanation for the eye-tracking results, and thus it was obvious that the models would be unable to account for this directed attention (the models were responsible for moving the turret and adjusting the visual cone to obtain terrain and opponent information). However, our initial models only used the "ground-truth" or raw environment information that they could parse directly from the environment. Thus, we turn to determining how to interpret and represent the environmental information similar to the SME and account for the eye gaze data.

Using a handful of maps annotated by the SME as reference, we automated and approximated the process. As Fig. 4 depicts, the entire map has not been annotated by the SME; we defined unannotated areas of grass as "risky" and unannotated areas of terrain as "cover". We also broke up the categories of safe (green) and deep-cover (yellow) into three categories—safe, cover, and deep-cover. The resulting five categories give the agent a comprehensive but abstract representation of the entire dTank environment. Grass is coded based on proximity to terrain as safe (within 2 tiles), risky (2 to 6 tiles away), or dangerous (more than 6 tiles away). Similarly, terrain was classified based on proximity to grass as safe (within 1 tile), cover (within 2 tiles), or deep-cover (more than 2 tiles away).

Figure 7 shows a comparison of the map annotated by the SME and the map annotated by our automated system. There are more categories of terrain in the automated version; however, the two maps are qualitatively very similar. Both show the same "dangerous" and "deep-cover" areas, even preserving the slightly fuzzy boundaries between territory considered safe and cover. The main difference to note is the red circle in the upper-right corner of the map that was annotated automatically. The entire town has been colored in red based on the SME comment that he tried to stay away from towns because they offered a false cover. We designate towns as "dangerous" to convey this message to the agent. These "dangerous" areas are exactly the areas the SME often turns the tank turret towards and devotes a majority of eye gaze towards, thus allowing us to account for the eye gaze data within a computational model.

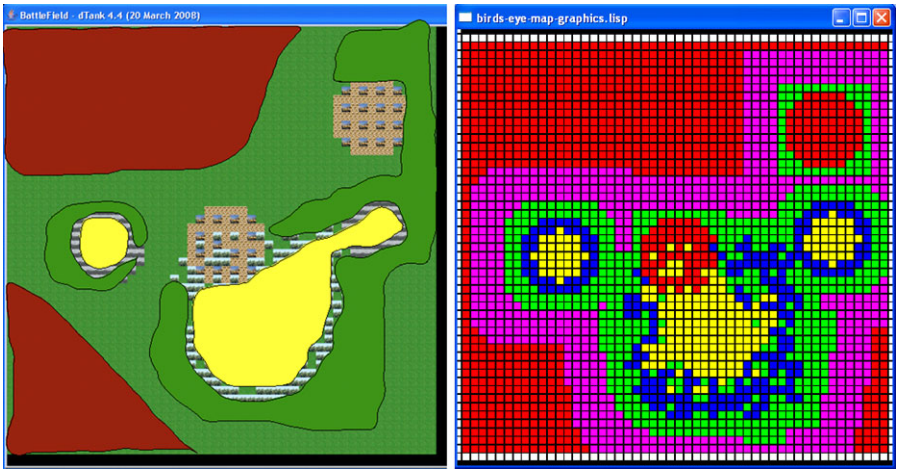

Fig. 7 The map from Fig. 4, as annotated by the SME (*left*) and the same map, annotated automatically (*right*). *Red* (dangerous), *green* (safe), and *yellow* (cover with poor visibility) represent the same type of terrain in both maps. On the right, blue designates terrain that is considered in cover while pink designates risky terrain. In the eye-tracking data, the "risky" terrain receives a vastly disproportionate amount of the attention of the SME

4.7 Egocentric spatial representation

As part of our attribute enumeration, we identified a significant gap in the information conveyed in the attributes as directly extracted from the environment. While the human SME was able to identify objects on the map and navigate around them, the environment attributes did not convey the spatial reasoning that humans take for granted (i.e., the disconnected nature of the attributes prevented any sort of navigation with respect to obstacles). To remedy this, following the method described in Best and Lebiere (2006), we developed an egocentric spatial representation, dividing the area surrounding the tank into bins of varying width, and adding several attributes that provided information as to the identity of and distance to the objects surrounding the tank. We chose to employ finer granularity towards the front of the tank and rougher granularity towards the back in an effort to capture the fact that humans generally encode a greater level of detail about the environment in front of them and pay less attention to what is behind them (see Fig. 8).

The distance to the closest terrain object (woods, hill, wall, etc.) in each bin is returned as an attribute. We also identify the closest and farthest of the objects and provide the angles to those objects as attributes. The addition of these navigational attributes allowed us to successfully create a model that drives a tank through a curvy path surrounded by trees—even if the path is different than any path used in the data collection. We have also implemented the same type of binning using the cognitive attributes discussed in the next section.

The result of this analysis is a real-time extraction layer for dTank that converts the "bird's-eye" view of the map to an egocentric representation on the fly, computing the geometry between the driver's tank and the objects that are either within the tank's view cone, or have been viewed within the last few seconds (we have used 2.5 seconds

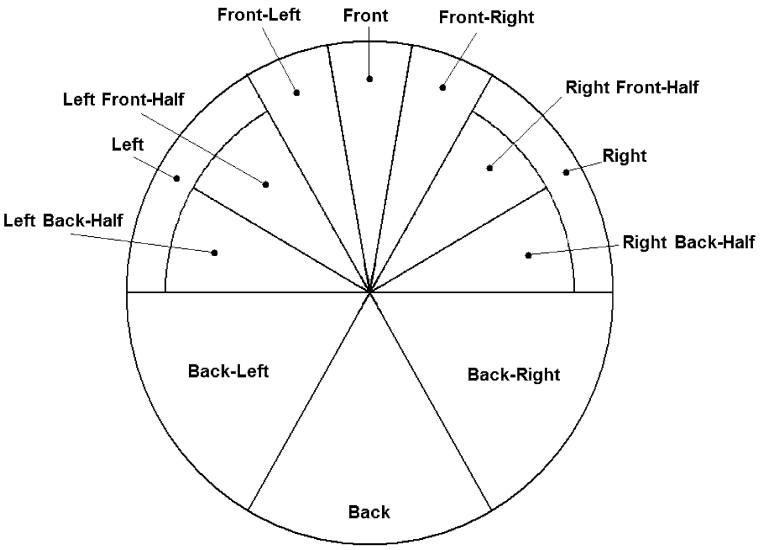

Fig. 8 Spatial bins used with dTank for determining nearby object attributes

as a spatial memory span to enable some persistence, but this is easily a research topic unto itself). Each of the bins in Fig. 8 is represented twice: once for what the object that fills the bin in (a tree, for example), and once for the distance to the nearest object in that direction (18.2 meters, for example).

4.8 Architectures for instance-based performance

Instance-based models base decisions on similarity to stored examples. The greatest problem in instance selection is one of efficiency; as the number of attributes is increased, efficient search for similar instances (a requirement for instance-based reasoning) becomes more difficult, with approaches such as kd-trees losing their advantage at dimensions greater than about 10, while instances can be influenced by even very distant examples in similarity space, requiring calculating similarity across large portions of the stored instances (see Deng and Moore 1995, for a thorough treatment of these issues).

We initially used ACT-R for the instance-based model, but experienced difficulties in achieving satisfactory real-time performance (response times within 200 ms) as the number of attributes and instances we used were scaled upwards. This led us to attempt to develop an in-house architecture capable of performing instance-based processing at a higher rate. In an effort to provide greater performance, we incorporated several key features. In particular, we focused on being able to identify new instances that produced no new learning (redundant), and prune them, preventing growth in instances, and on being able to filter out irrelevant attributes, thereby reducing the dimensionality and resulting search times of the stored instances.

In our experiments, we were able to prune approximately three quarters of all new instances without any loss in accuracy. Through further improvements, including

adaptive cue-weighting, which allows identification of irrelevant attributes and their filtering (the majority of attributes are irrelevant or redundant in many problems), we were able to increase both the speed and the accuracy of instance-based processing well beyond what we were able to achieve using ACT-R (Best and Gerhart 2011). We tested the speed of processing instances using both ACT-R 5.0 and ACT-R 6.0. The ACT-R 5.0 system, because it uses a simpler representation at the code level, is actually substantially faster than ACT-R 6.0 when processing instances (ACT-R 6.0 uses more robust data structures based on the Common Lisp Object System, which are somewhat slower than the struct-based ACT-R 5.0 system), and thus we have chosen to test CIBRE against the faster ACT-R 5.0 implementation (note that this speed differential may be confined to components leveraged in instance-based processing and the ACT-R 6.0 system may be generally faster than ACT-R 5.0—we have not attempted any comprehensive performance comparison of the two implementations).

It is a fair question to ask how many instances an instance-based system that attempts to capture cognitive performance might need to store. Simon and Gilmartin (1973) estimated chess expertise as consisting of 50,000 chunks, where a chunk consisted of ∼7 slots that could hold individual pieces of information, while Gobet (1997) estimated that 100,000 to 150,000 chunks would be required for expertise in a particular domain. Given these estimates, we focused our exploration on instance memories of up to 100,000 instances with 7 slots (attributes). The processing implications of chunks and slots are largely independent and multiplicative, and the two can thus be multiplied to provide an overall system load, so 50,000 chunks with 14 slots would produce nearly identical results. Many of the datasets we worked with here involving SME performance in a virtual environment had greater than 10,000 instances, but also had greater than 100 attributes, resulting in memories of greater than 1,000,000 attributes. Given this, this simulation will explore the processing implications of storing up to 700,000 attributes, demonstrating the implications of storing a typical, not extreme, memory for task performance in a virtual environment.

Figure 9 shows the instance-based processing times of CIBRE compared to the ACT-R 5.0 implementation. The time is the amount of wall clock time required to execute a single retrieval from the instance memory for a given number of instances (known as chunks in the ACT-R framework) with 7 attributes (slots in ACT-R) as the number of instances increases to 100,000.

We highlight the 50 ms decision time level, which we suggest is the minimum threshold for real-time usability. These results suggest that the CIBRE platform is capable of handling real-time instance-based processing for a memory of 100,000 items on a current generation computer, which in this case is a hex-core processor running at 3.6 GHz. Multiplying chunks by slots, CIBRE can process memory partitions of approximately 700,000 attributes and respond within 50 ms, while memories of 1,000,000 attributes, by extrapolation, can still be processed within 100 ms. The ACT-R architecture, on the other hand, is less well-suited for real-time environments, and the out-of-date (but much faster) 5.0 implementation crosses the critical 50 ms threshold at approximately 30,000 instances (chunks), handling approximately 200,000 attributes within 50 ms, while the 6.0 implementation is substantially slower, and even the faster ACT-R 5.0 system is unable to process an instance memory with 1,000,000 attributes within 200 ms.

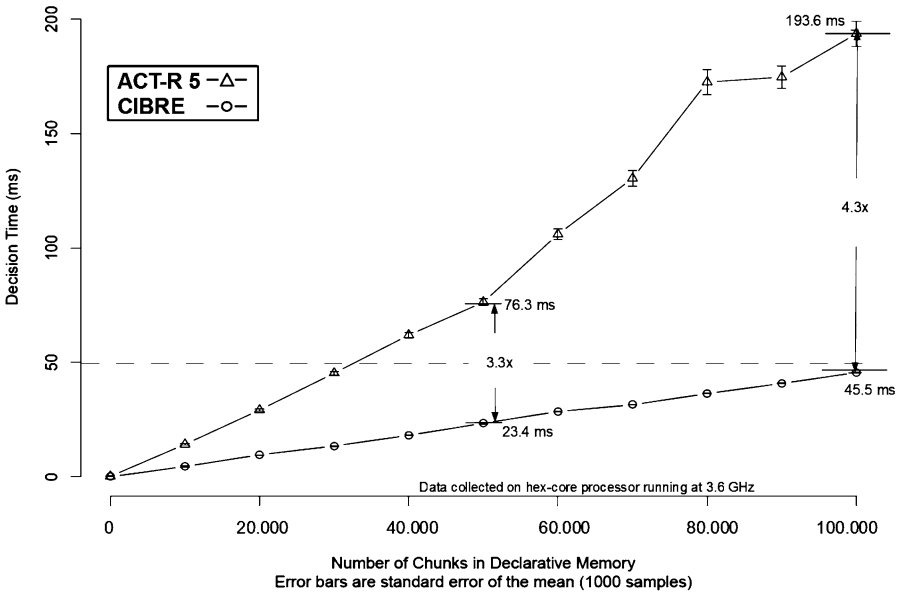

Fig. 9 Comparison of wall clock instance-based processing times for CIBRE and ACT-R against the critical 50 ms real-time processing threshold

The simple reason for the speed difference is in the underlying processing that must be done: the CIBRE system uses a tight inner loop and smaller structures to hold the same information. It is also able to use datatypes to optimize its performance (even though it runs in the same Lisp as ACT-R), and need not calculate or maintain the more extensive data structures of ACT-R. As the number of chunks increases, however, the larger memory footprint of the ACT-R chunks also start to interact, further slowing the system through memory management interactions, while the CIBRE performance is linear out to the limits of our exploration. As the non-linearity of ACT-R performance in the previous graph shows, this focus on careful memory management becomes more important as memories become larger.

It is important to note that in every problem set we have encountered, we have been able to both prune the number of instances and the number of attributes automatically using the CIBRE system, and the times presented factor this extra processing in for the CIBRE architecture, while the ACT-R comparisons do not involve any extra processing that might be called for in instance-based processing (e.g., advanced similarity functions). Focusing on a worst-case analysis, even if the gains due to pruning were not realizable for some domain-specific reason, the CIBRE system consistently responds to instance-based decision queries more than three times faster than ACT-R 5.0 (and more than ten times faster than ACT-R 6.0), allowing processing of an expert sized memory of 50,000 to 100,00 chunks in real-time. However, we are consistently able to maintain accuracy on test domains while pruning 80 % of the data (Best and Gerhart 2011), resulting in an overall processing speed approximately 15× faster than ACT-R 5.0 when performing induction on the same dataset, representing

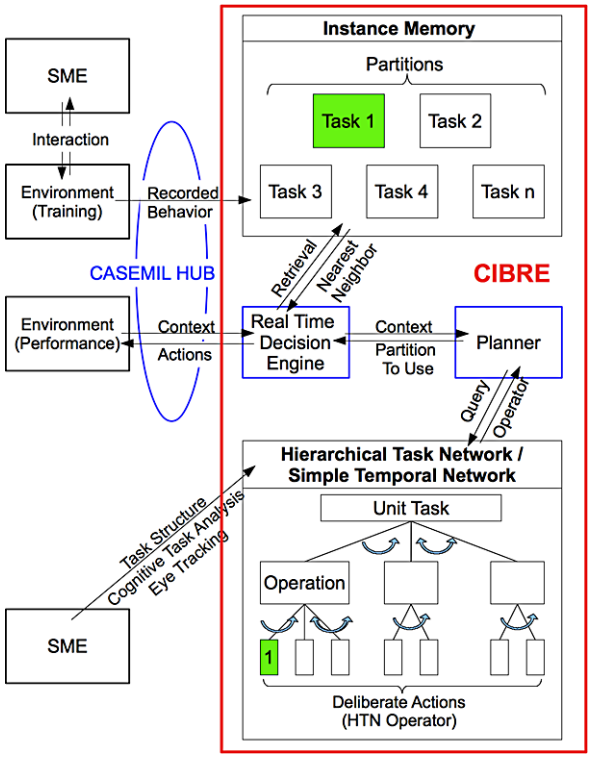

Fig. 10 Task selection and instance-based task execution flow of control within CIBRE

more than an order of magnitude in speedup. This enables storing and processing large quantities of instances in real-time, one of our critical problem constraints.

We tested the representations described in the previous section using the instance-based rule engine and toolkit we developed, CIBRE (Best and Gerhart 2011), for both the entire battle (task selection) and several sub-task (task execution) paradigms. A diagram of the entire system developed appears in Fig. 10. The system uses a Hierarchical Task Network Planner for high-level control (specifying the interrelationship of tasks and switching between them) and an instance-based processor for task execution. The goal of this design is to support multiple instance-based models within the same framework (shown as "partitions" within the instance memory in Fig. 10), allowing a broad task (such as the general dTank battle) to be modeled using a divide-and-conquer approach, with multiple focused instance-based models providing the low-level task performance detail.

In the context of the current task environments, the SME's performance is stored within the Instance Memory through their direct action with the task environment (and logging through the CASEMIL hub). The Planner within CIBRE evaluates the context (task environment variables and state) against its stored knowledge of goal selection, and selects an appropriate task. Given a task to work on and the current context (state variables and their transformations), the Real Time Decision Engine then queries the Instance Memory for the appropriate action. This action is determined by blending its prior relevant experiences, where relevance is determined through

comparing similarity between stored experiences and the current context along those dimensions that have been determined to be important. The selected action is then passed back out to the environment through the CASEMIL hub thereby producing behavior in the virtual task environment.

4.9 Using cues to select goals and perform tasks

Applying generative models on a broad scale requires identifying tasks to be executed (task selection) and performing tasks that have been selected (task execution). We evaluated two machine-learning approaches, comparing the use of instance-based models of task selection and execution to a decision-tree approach. The choice of these two formalisms was motivated by their contrasting properties in the context of large, noisy human performance data sets characterized by the number of attributes, m, and the number of cases in the training set (instances), n, and where n is typically at least an order of magnitude larger than m. Learning new material in instance-based systems is extremely rapid because it consists mainly of storing new instances, with time complexity $O(mn)$, but the time required to produce decisions from those stored instances is also a linear factor of the size of the knowledge base, or complexity $O(mn)$. Decision trees, on the other hand, exhibit the opposite tendency, and while the learning phase (tree construction) can be extremely laborious, with a time complexity of $O(mn^2 \log n)$, the decision making phase can be executed with extreme efficiency, with time complexity of $O(m)$. Thus, with $n \approx 10$ m, the learning time of the instance-based approach should be strictly less than the learning time for decision tree induction, while the execution time at task performance of the instance-based approach can be expected to be approximately two orders of magnitude greater than the decision-tree approach.

Given the cues and goals we identified, we evaluated whether the cues comprised a sufficient set of predictors for the goals and actions derived from the SME performance (assuming that they were, in fact, predictable) by testing the prediction performance of these two algorithms. One challenge in this was that the SME often identified multiple goals they were pursuing simultaneously (for example, staying hidden while monitoring an enemy tank; See Fig. 2 for a more complete goal decomposition).

We used the four previously recorded "battles" as a source for test datasets, breaking the data into train and test segments. We explored several splits, including splitting all instances in the entire games randomly into either the training or testing partition (labeled "All Battles" in Tables 3, 4, and 5), or by choosing one entire battle as a training set, and then using it to predict the remainder of the training sets (labeled "Game[n]" in Tables 3, 4, and 5). The training instances were then selected from the from the training partition by either choosing 1500 random instances, or by using all available instances in the training partition. We also explored whether using all of the available attributes or using a hand-selected subset of the attributes provided better accuracy as well as its impact on run time. Because accuracy is significantly influenced by chance, we also calculated chance accuracy on these datasets, shown in Table 3. Thus, Table 3 shows the probability of guessing the action taken (e.g., throttle change) and the primary goal identified by the SME, based on whether the

Table 3 Goal prediction accuracy using a hand-selected set of cues for either 1500 training instances, or using all available training instances. Individual game predictions (Game 1–Game 4) derived from only using samples from other games

	# Instances	Testing Set	Game 1	Game 2	Game 3	Game 4
Chance		19.44 %	19.93 %	19.79 %	22.93 %	24.48 %
C4.5 Primary Goal	1500	89.83 %	33.99 %	12.39 %	40.31 %	14.22 %
IB Primary Goal	1500	94.93 %	44.06 %	38.43 %	38.25 %	10.66 %
C4.5 Primary Goal	All	95.71 %	34.74 %	17.85 %	47.72 %	17.18 %
IB Primary Goal	All	97.82 %	45.46 %	36.79 %	39.61 %	12.20 %

training data used is either a random half of the cases contained in all battles or one of the individual battles.

We now turn to methods for predicting actions and goals from these datasets, using the instance-based methods described previously in contrast to the C4.5 decision tree learning algorithm. The C4.5 classification algorithm (Quinlan 1993) has seen widespread adoption and application to problems in machine learning and classification, and generally shows an advantage of several orders of magnitude over instance-based methods in terms of processing speed when applied to the same data (e.g., Best et al. 2008), reflecting our decision time complexity predictions. Table 3 shows the C4.5 derived classifier was able to correctly identify primary goals an overwhelming majority of the time, compared to the probability of correctly guessing . In particular, when using a all available training cases of behavior from all of the battles with a hand-selected set of attributes, the C4.5 classifier was able to predict the Primary Goal 95.71 % of the time. The decision tree classifier, however, underperformed the accuracy of the instance-based classifier, which achieved an accuracy of 97.82 % when also using all available cases of training behavior with the hand-selected attributes.

We also tested whether or not we could predict the action taken using both the C4.5 classifier and the instance-based algorithm with the hand-selected cues. When using all available training cases, a randomly selected half of the total cases, to predict the other half of the cases, the C4.5 classifier predicted the Action Taken with an accuracy of 58.59 % while the instance-based prediction achieved an accuracy of 61.46 %. However, prediction of actions taken in individual games using only cases taken from other games is near chance.

The instance-based algorithm we have employed here (Best and Gerhart 2011) is capable of selecting meaningful attributes, and it is thus possible to compare the performance of the instance-based system using automatic selection of cues against the hand-selection of cues. Table 5 reports the performance of the instance-based system on both goal selection and action selection using all available cues derived from the CTA. In all cases, the instance-based performance using all available cues exceeded the accuracy of the instance-based system using hand-selected cues, with the system correctly predicting each individual action in the split-half testing set 64.82 % of the time. More significantly, when trained on subsets of battles, the automatic cue selection model was able to predict each individual action for games that were not sampled in the training set nearly as well, with accuracies of 61.63 %, 79.41 %, 50.58 %, and

Table 4 Action Taken prediction accuracy using a hand-selected set of cues for either 1500 training instances, or using all available training instances. Individual game predictions (Game 1–Game 4) derived from only using samples from other games

	# Instances	Testing Set	Game 1	Game 2	Game 3	Game 4
Chance		19.44 %	19.93 %	19.79 %	22.93 %	24.48 %
C4.5 Action Taken	1500	48.03 %	20.38 %	12.02 %	15.77 %	14.61 %
IB Action Taken	1500	59.99 %	24.75 %	16.39 %	21.48 %	23.26 %
C4.5 Action Taken	All	58.59 %	15.68 %	10.38 %	16.64 %	15.40 %
IB Action Taken	All	64.16 %	27.14 %	10.92 %	14.40 %	18.83 %

Table 5 Goal and Action Taken prediction accuracy using automatically selected cues for either 1500 training instances, or using all available training instances. Individual game predictions (Game 1–Game 4) derived from only using samples from other games

	# Instances	Testing Set	Game 1	Game 2	Game 3	Game 4
IB Primary Goal	1500	96.77 %	26.32 %	50.27 %	31.21 %	22.24 %
IB Primary Goal	All	98.36 %	26.16 %	42.99 %	32.16 %	22.23 %
IB Action Taken	1500	61.12 %	60.72 %	74.49 %	51.07 %	46.45 %
IB Action Taken	All	64.82 %	61.63 %	79.41 %	50.58 %	43.68 %

43.68 % for Games 1-4. Thus, automatic cue selection led to much greater generalization across datasets.

The instance-based system, when used with a full set of cues, showed a large accuracy advantage over alternatives we explored, especially in generalizing to games that were previously unseen, making it a clear best choice for modeling continuous control in task execution. When using the full set of attributes for prediction of goal selection and action selection, we ran into difficulty using the C4.5 algorithm, and our implementation was unable to return results within any practical amount of time. This scaling issue is predicted by the computational complexity analysis at the beginning of this section. We note, however, that the subset of accuracies returned from model runs that did complete was strictly less than those returned using a hand-selected set of attributes with the C4.5 algorithm. That is, accuracy was negatively impacted when using C4.5 and greater numbers of attributes, so not only did the decision tree build more slowly when attributes were added, but its accuracy also decreased.

The C4.5 algorithm performed well in terms of goal selection accuracy using the hand-selected cues, providing accuracy within $\sim$2 % of the instance-based model. Given its superior runtime performance (two orders of magnitude faster), it is practically expedient to use a C4.5 derived classifier for task selection, while relying on the instance-based method for task execution.

The significance of this result is that, given the set of cues and attributes we derived, we were able to accurately predict the appropriate goal or task for a model or agent to perform in a free-form battle scenario (where appropriate indicates correspondence with the SME's choices) using a decision-tree classifier. While this is a somewhat naturalistic case study, it also serves as an existence proof: Given a large-

scale scenario, we have identified a method that can be efficiently applied to the problem of task selection, doing so with a high level of accuracy when compared to the decisions of a human SME. This system has been implemented within the CIBRE framework, allowing CIBRE to learn a decision tree capable of driving task selection based on the current (and possibly shifting) context. While providing a level of control similar to that exhibited by a finite state machine, in this case the logic of task switching is learned directly from the annotated data derived from the SME performance.

The decision tree approach produces a ruleset for making decisions while an instance-based approach relies on similarity-based matching from memory. The relative advantages and disadvantages between them come into focus in the context of identifying goals and identifying actions. Decision trees are at their weakest when faced with learning data that are either noisy, or are characterized by decision surfaces that are non-orthogonal to the attribute axes. While the first issue can be addressed by thoroughly cleaning and scrubbing the data to remove noise, and the second issue can be addressed using methods such as Principal Components Analysis (PCA) to align the decision surfaces with the attribute axes (neither of which were done here), this must be done carefully, and is challenging to automate. The instance-based methods, on the other hand, are extremely tolerant of both noise and irregular decision surfaces, allowing for better performance without resorting to ad-hoc transformations of the data. The action data were both noisy and highly non-linear, making the instance-based method a clear choice for an automatic system. The sacrifice, however, is computational processing demands, and thus we have chosen to use the decision tree approach on the task selection data, where the lower noise and orthogonality of the decision surfaces allowed the decision tree approach to perform at nearly as high a rate of accuracy as the instance-based system, but in much less time.

4.10 Using instances: the path driving model

Turning to the problem of task execution, we developed a model of path driving using the CIBRE instance-based rule engine (Best and Gerhart 2011) to simulate SME behavior. The SME completed the path five times, avoiding obstacles (trees) along the way, and this data was used to train the CIBRE agent. We then ran the CIBRE model on the same task (the model is deterministic, with the possible exception of latency differences caused by the underlying hardware and software, and thus only one run was used for calculation, though several runs were conducted to confirm its repeatability). We compared each recorded position of the CIBRE agent against the closest recorded SME position from among the five paths. A plot of both of these positions overlaid on the map is presented in Fig. 11. CIBRE's average deviation from the closest SME path was 1.95 meters with a standard deviation of 1.81 meters and a maximum distance of 13.95 meters. Given that the path is generally about 20 meters wide, expanding to 60 meters wide in some places, this represents close agreement between the training data and CIBRE's behavior.

Using the methodology described by Best and Lebiere (2006), and Best et al. (2010), we then transferred the dTank model to a second virtual environment, UT2004, to validate behavior in the new environment. We also collected instances

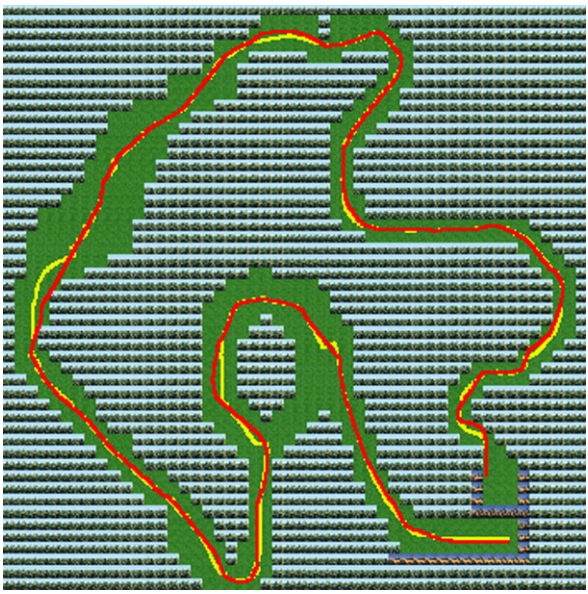

Fig. 11 CIBRE (*red*) vs SME (*yellow*) path tracings in dTank

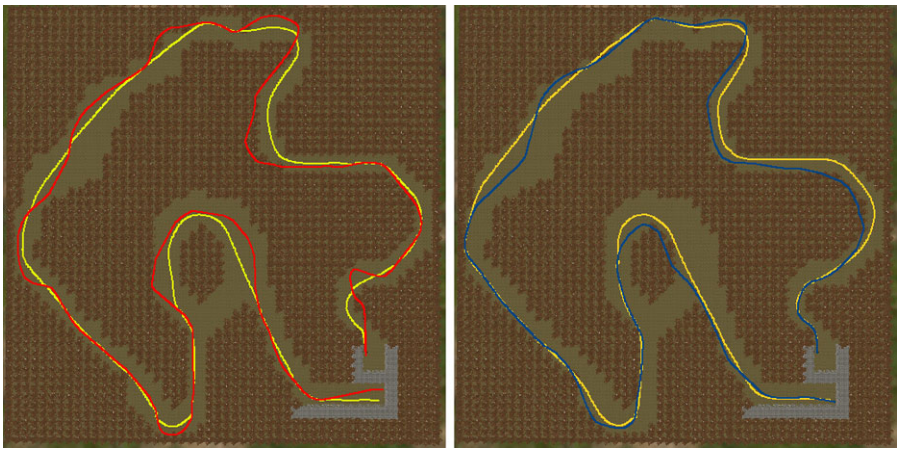

Fig. 12 *Left*: CIBRE model ported from dTank (*red*) vs. closest SME (*yellow*) path. *Right*: CIBRE model trained in UT2004 (*blue*) vs. the closest SME (*yellow*)

of our SME driving the path in the new environment and trained a new cognitive agent. The results from both models are shown in Fig. 12 while the actual view of the CIBRE agent in the UT2004 environment during task performance is shown in Fig. 13. The image on the left shows the path as driven by CIBRE in UT2004 trained on data collected in dTank (in red). The image on the right shows the path as driven by CIBRE trained on data collected in UT2004 (in blue). The SME performance in UT2004 is on both panels (in yellow). The same analysis was performed on both models. The differences between the ported dTank model and the closest SME path in UT2004 (mean 6.73, standard deviation 6.56, maximum 29.93) were similar to the

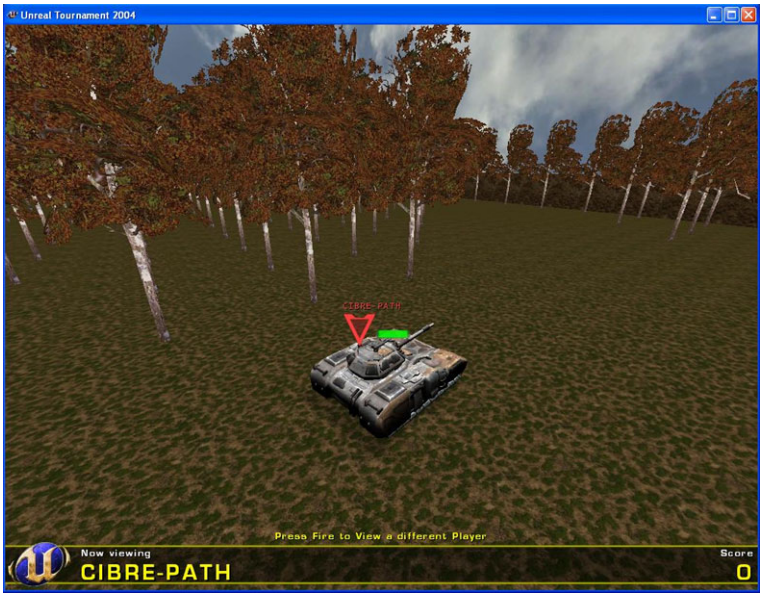

Fig. 13 A CIBRE model following the tree-lined path

differences between the UT2004 model and the closest SME path in UT2004 (mean 7.20, standard deviation 6.79, maximum 30.68). While the differences are larger than those reported for the dTank model in dTank, they are still within the confines of the path, indicating some success in porting cognitive models between virtual environments using abstract egocentric spatial representations.

Our analysis has shown that while the cognitive model does not follow the SME's behavior exactly, it does a very good job of approximating it, even when ported to a new environment, as long as the same information can be extracted from both environments. The use of an egocentric spatial representation makes this transformation straightforward: the model acts on a model-centric representation that is identical across the virtual environments.

5 Conclusions

The analysis of the SME behavior told an interesting story: contrary to our original hypothesis, the SME's visual representation of the virtual environment was deceptively simple; sophisticated, but simple. This result is not surprising: experts in a given field are often distinguished from novices in that they are able to recognize and interact with larger groupings—or "chunks"—of information than novices are able to do. The eye-tracking data, in particular, was confusing when interpreted in terms of what was visible on the screen, and was only clearly interpretable when viewed in terms of strategic direction of attention to areas that might contain threats. Thus, the eye-tracking data revealed spatial strategies and information foraging relative to the current game context rather than visual search strategies.

Careful analysis of the SME's discussion of his battle processes revealed that the SME was taking discrete dTank information, such as the current time and the presence of trees, and turning them into meaningful battle cues, such as 'opponent will be loaded soon' and 'the Opponent can see—and possibly fire—at me'. On the basis of this insight, we developed a series of environmental cues or attributes that are computationally derived in real-time directly from the discrete dTank information that is available to the SME during a battle. We extracted a total of 113 attributes, and the simple path model uses approximately 10,000 instances, for a total of more than 1,000,000 total attributes (counting across all of the instances). We were unable to generate real-time (within 200 ms) predictions for such large numbers of instances and attributes using ACT-R. We have found that execution of many of the tactics requires only a subset of these attributes. For example, seeking cover requires knowledge of the terrain with respect to the tank whereas firing at an enemy tank requires knowledge of that enemy's location. However, determining that is a labor-intensive process, and trial and error and hand-coding are not scalable strategies. To address these issues, we developed a Cognitive Instance-Based Rule Engine (CIBRE), a platform for real-time performance of spatial tasks in virtual environments, with an emphasis on computational efficiency, deep integration of rule-based and instance-based processing, and pruning of instances and attributes as a means of reducing computational load We have focused on developing automatic methods for reducing the number of attributes used through adaptive cue-learning, and have developed methods that match or exceed the accuracy performance of other similar systems reported in the machine learning literature (Best and Gerhart 2011).

The analysis we presented on chunk processing rates is also, unfortunately, a best-case for the ACT-R system. Not only did we use the faster ACT-R 5.0 implementation, but the attributes themselves that we used for the testing presented in this paper were minimal—all were a single digit. The actual attribute values we worked with in the virtual environments were often much larger than single digits, and thus the memory load on the Lisp subsystem could be expected to be much greater. Our actual experience suggests this is true and was, in fact, the primary reason for developing CIBRE: We were unable to get consistent rapid instance-based responses from large memories when using ACT-R and while the system might keep up for small segments of the performance, it would inevitably stumble and fall behind. This issue, in particular, was documented in regards to the models presented in Best and Lebiere (2006), along with several practical remedies that were necessary to achieve satisfactory performance (such as flushing network buffers when processing fell behind), but those remedies are still relevant since the instance-based systems of today are just now approaching the processing speeds of the rule-based systems of a decade ago.

We do not take this as a weakness of the ACT-R system. Rather, it has many things to do, and along with the capability, for example, for predicting blood oxygen levels in brain regions, come corresponding data structures and memory demands. Instance-based systems all suffer from the curse of dimensionality, and they hang on the edge of computational intractability. If the target is real-time interaction this calls for ruthlessly Spartan programming, which is certainly at odds with the goals for a complete account of cognition in the form of a cognitive architecture.

The conclusion from the architectural explorations we conducted is clear: instance-based methods can easily overwhelm computational resources in task data

extracted from virtual environments. Optimizing code can only achieve modest gains relative to existing systems—the real problem is simply the dimensionality of the underlying data, which cannot be searched any more effectively by cleverly programmed algorithms than by a simple serial search once approximately ten or more attributes are present in any instance (Deng and Moore 1995). To overcome this "curse of dimensionality", the only options are to either reduce the number of attributes, or to reduce the number of instances used. Within the ACT-R architecture, attribute selection is managed by hand-coding the retrieval that specifies the relevant slots in an instance. This binary selection of attributes (they are either used or not) is limited in terms of accuracy (Best and Gerhart 2011), and does not scale well to large projects, or those that might initially include large numbers of attributes, as is often the case in virtual environments. An automated means for selecting and weighting attributes is an absolute requirement for a scalable method for these domains (and the ACT-R theory currently precludes weighting of attributes).

Using instance-based methods, but hand-selecting a minimal set of attributes, it is still possible to obtain reasonable real-time performance using the ACT-R instance-based loop with the dTank virtual environment, which only requires about a 200 ms cycle time for reasonable interaction, as long as the task data is also constrained to be relatively "light" in terms of instances, and the total number of instances times attributes is kept under approximately 500,000. However, this is insufficient for the Unreal Engine interaction, which is a much faster-paced system, and would not support more complex tasks under even the lightweight dTank environment. While this conclusion is qualitative, based on our explorations, we would advise using special purpose instance-based systems such as CIBRE, which are an order of magnitude faster (or more) than the ACT-R system, in preference to ACT-R for fast-paced three-dimensional environments such as flight simulators and first-person shooters like Unreal.

The Cognitive Task Analysis provided a great deal of insight into what the SME is attending to within the environment and how the SME uses that information to perform the overall task of playing dTank. This activity led to the production of both Attributes and Tactical Categories that were specific to the dTank domain and the tasks performed by the SME. These Attributes and Tactical Categories comprise the features and responses that collectively make up the instances upon which the instance-based learning system relies, and provided the grist for the sifting process of determining which attributes were relevant to any particular task. The attribute set itself is a super-set of the features used for each individual subtask—the method of automatic attribute weighting used in CIBRE allows the instance-learning system to determine which attributes matter for which tasks, which greatly simplifies the software engineering aspect of model building.

The tactical categories themselves correspond to tasks to be executed during the battle phase. We demonstrated a decision-tree based system capable of learning from the SME data and producing task selection decisions that had a high level of agreement with the SME. This system was embedded within the CIBRE framework to address the challenge of task selection, allowing the system to choose a particular task to focus on based on the current context. We note, however, that the decision tree induction algorithm was not well-suited to learning the continuous control actions

required for performance in the domain (likely due to noise in the data and irregular decision surfaces), and performed poorly when predicting actions (an area where the instance-based method excels for our domains). Rather, the algorithm succeeded best at the more symbolic categorization task of identifying the goal the SME would have selected. In this case, however, the decision tree, which can be represented directly as a much faster ruleset, (when compared to an instance-based method) is clearly preferable. It would be entirely feasible to use the instance-based system for both tasks, but this would double the processing time of the system, and thus we have adopted a hybrid approach to maximize real-time responsiveness while maintaining accuracy.

The entire range of attributes, determined from the CTA on the complete battle task, supported task execution on a variety of micro-tasks. That is, the unconstrained and naturalistic task helped provide a superset of attributes that covered those that were necessary to perform various individual subtasks. One such task, a path driving task, was investigated in detail. Using this task, we demonstrated that the induced instance-based model was capable of performing at a level similar to that of the SME. Further, and perhaps more importantly, we demonstrated that the use of an egocentric spatial representation allowed for a level of environment agnosticity: The dTank model of path driving was successfully deployed in the Unreal Tournament fully immersive three-dimensional environment, performing the same task without any model modifications at similar levels of accuracy.

One issue with the data collected is that discrete actions such as key presses only occur at one specific point in time, and thus the sampling rate relative to the base rate for responses will produce some number of instances with no action (samples that are taken between discrete actions). This is not a quirk of dTank or Unreal; rather it will be true of any real-time task environment that takes discrete inputs such as key presses or mouse clicks and is sampled using a set frequency. Our primary means for dealing with this was to have the model determine a desired action state rather than a specific discrete action. Thus, a model might target a certain speed, rather than reproduce a throttle movement, and only produce an action when the current action state mismatched the desired action state. The action at the level of instances is therefore to set the vehicle state, rather than to adjust the state. While this difference is subtle, it is also profound. In a sequence of 100 instances, there might be one throttle adjustment among them, but every instance will have a throttle state associated with them. If discrete actions were used as the basis for actions in the instances, the sheer number of instances containing no action would overwhelm the rest, and result in a model that produced no action whatsoever.

Another issue with the data collected in the entire battle phase was that there was a large portion of time in which the appropriate behavior was to do nothing. These "empty" sequences of behavior were difficult to reproduce, and caused an overestimation of how much behavioral data we had actually collected. These periods were often related to the task structure itself. For example, many tasks approached by the SME were started with a task orientation phase, where the SME scanned the display for several seconds without producing any action. This requires either incorporating the task time delta into the instances, which is generally problematic because it can result in the instance-system learning to attend to timestamps (rather than meaningful attributes), or finding a way to account for the task structure outside of the instance

system. For these reasons, we had the SME perform a series of pre-determined "vignettes" consisting of high-level actions identified during the CTA.

Approximately one third of the data we collected were pre-segmented behavioral sequences during which the SME performed a simpler task several times (e.g., following an opponent, aiming and shooting, following the tree-lined path, etc.). These data streams were much more compact, and were easier to replicate in real-time. Most likely, the increased repetitions of behavioral sequences actually resulted in more useable data per model, despite having many times fewer instances than the models created from entire simulations. The improved real-time replication may also be explained by the fact that the behavioral streams were in separate models, so there was no interference between goals (i.e., given a particular situation the appropriate action might depend on the goal to be achieved.

In the end, we were able to induce a model of task selection and a model of task performance that both approximated human behavior directly from the data. However, this was only possible through the combination of 'cognitive attributes', such as safe and risky territory, and an egocentric spatial representation—our attempts at leveraging ground truth directly for both task selection and task performance were consistently unsuccessful. Instead, the Cognitive Task Analysis was an essential tool to developing a spatial representation that used features of the environment the expert was using. Thus, knowledge engineering was replaced, by necessity, with attribute engineering.

While we have induced models of task selection and task execution, for the free-form battle scenario, we have not yet combined these individual models within a framework for capturing the entire task performance. And, in fact, the naïve approach of simply placing all of those instances in a single task memory does not result in a system capable of reproducing the highly structured and variable behavior produced by the SME. This approach merges too many instances that are specific to particular subgoals, and results in a system that produces a poor approximation to human behavior. Rather, we have focused on complete modeling of more constrained tasks, and have demonstrated models of individual tasks up to the complexity of path driving, where the path driving starts at a landmark (a virtual building) and ends at a corresponding landmark (also a virtual building). This task does have components, and the model must engage the throttle to start the tank moving, and bring the tank to a stop when it reaches its destination, but this is approaching the limit of task structure that we have been able to learn within a single instance memory.

The ability of CIBRE to reproduce human behavior based off of a recording of an SME behavioral trace, producing a generative model of behavior by observing a demonstration, has far reaching implications. This type of learning architecture could easily be used in a virtual training environment due its ability to learn and model human behavior based off of recorded data—most of the learning and induction is automated and is part of the core algorithm, so extending it to new domains is much simpler than conventional knowledge engineering.

We believe the most important contribution of this work is in detailing a methodology for recording and analyzing an expert performance in a virtual domain and translating that directly into a performance model. There are, of course, many steps left to take in this research, including increasing the breadth of the models so they

include many smaller tasks and switch between them. We have identified a means of selecting tasks, but have not yet approached the stitching together of those tasks using a planning framework into a comprehensive model of overall behavior. We are currently applying this methodology in concert with the CIBRE architecture to work with flight simulators and hope to report on that in the near future.

Acknowledgements We would like to thank Nathan Gerhart, Caitlin Furjanic, and Missy Schreiner for assistance with various phases of this project. In addition, anonymous reviewers provided comments that were invaluable in improving the manuscript. The work reported here was partially funded through Phase I and II SBIRs provided by the Naval Aviation Engineering Directorate (NAVAIR), and partially through a Phase I SBIR provided by the National Science Foundation (NSF).

References

Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6:37–66

Anderson JR, Lebiere C (1998) The atomic components of thought. Erlbaum, Mahwah

Anderson JR, Bothell D, Byrne MD, Douglass S, Lebiere C, Qin Y (2004) An integrated theory of the mind. Psychol Rev 111(4):1036–1060

Ball JT, Gluck KA (2003) Interfacing ACT-R 5.0 to an uninhabited air vehicle (UAV) synthetic task environment (STE). In: Proceedings of the 2003 ACT-R workshop

Best BJ, Gerhart N (2011) Learning from examples using CIBRE: a cognitive Instance-based rule engine. Poster presented at the 20th conference on behavior representation in modeling and simulation, Sundance, UT

Best BJ, Lebiere C (2006) Cognitive agents interacting in real and virtual worlds. In: Sun R (ed) Cognition and multi-agent interaction: from cognitive modeling to social simulation. Cambridge University Press, New York, pp 186–218

Best BJ, Lovett M (2006) Inducing a cognitive model from examples provided by an optimal algorithm. In: Proceedings of the seventh international conference on cognitive modeling, Trieste, Italy, pp 56–61

Best BJ, Dixon KR, Speed A (2008) Modeling an unstructured driving domain: a comparison of two cognitive frameworks. In: Proceedings of the 2008 conference on behavioral representation in modeling and simulation

Best BJ, Gerhart N, Leibere C (2010) Extracting the ontological structure of OpenCyc for reuse and portability of cognitive models. In: Proceedings of the 19th conference on behavioral representation in modeling and simulation

Brumby DP, Salvucci DD, Howes A (2007) Dialing while driving? A bounded rational analysis of concurrent multi-task behavior. In: Proceedings of the 8th international conference on cognitive modeling. Ann Arbor, Michigan, USA

Deng K, Moore AW (1995) Multiresolution instance based learning. In: Proceedings of IJCAI, 1995. Morgan Kaufmann, San Mateo

Douglass S, Ball J, Rodgers S (2009) Large declarative memories in ACT-R. In: Proceedings of the 9th international conference of cognitive modeling (paper 234), Manchester, UK

Gobet F (1997) A pattern-recognition theory of search in expert problem solving. Think Reasoning 3:291–313

Gonzalez C, Lerch FJ, Lebiere C (2003) Instance-based learning in real-time dynamic decision making. Cogn Sci 27(4):591–635

Lathrop SD (2008) Extending cognitive architectures with spatial and visual imagery mechanisms. Computer science and engineering, The University of Michigan, Ann Arbor

Meyer DE, Kieras DE (1997) A computational theory of executive cognitive processes and multiple-task performance: part 1. Basic mechanisms. Psychol Rev 104:3–65

Newell A (1990) Unified theories of cognition. Harvard University Press, Cambridge

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo

Ritter FE (2008) dTank: A lightweight synthetic environment for teaching and theoretical research in the human behavioral synthetic research environments (HB-SRE) symposium. In: Proceedings of the 17th conference on behavior representation in modeling and simulation, 08-BRIMS-034. University of Central Florida, Orlando

Salvucci DD (2006) Modeling driver behavior in a cognitive architecture. Hum Factors 48:362–380

Salvucci DD, Gray R (2004) A two-point visual control model of steering. Perception 33:1233–1248

Schoelles MJ, Gray WD (2000) Argus prime: modeling emergent microstrategies in a complex simulated task environment. In: Taatgen N, Aasman J (eds) Proceedings of the third international conference on cognitive modeling. Universal Press, Veenendal, pp 260–270

Scott SD, Cummings ML (2007) An experimental platform for investigating decision and collaboration technologies in time-sensitive mission control operations. Technical report HAL2007-04 prepared for phantom works, boeing by, Massachusetts Institute of Technology

Simon HA, Gilmartin KJ (1973) A simulation of memory for chess positions. Cogn Psychol 5:29–46

Simon HA, Gobet F (1996) Templates in chess memory: a mechanism for recalling several boards. Cogn Sci 31:1–40

Simon HA, Zhu X (1988) Learning mathematics from examples and by doing. In: Models of thought, vol. II. Yale University Press, New Haven, London, pp 167–184

Taatgen NA, Wallach D (2002) Whether skill acquisition is rule or instance based is determined by the structure of the task. Cogn Sci Q 2(2):163–204

Bradley J. Best is a Principal Scientist at Adaptive Cognitive Systems, LLC, in Boulder, CO., where he focuses on cognitive modeling of adaptive behavior in complex environments, especially those that have significant spatial and temporal aspects. His current research interests include integrating perception with decision making in robotic and virtual agents and the development of methods for analyzing, understanding and visualizing model behavior in these environments.