LEARNING AND ABSTRACTION IN SIMULATION

Sarah E. Goldin and Philip Klahr

The Rand Corporation
Santa Monica, California 90406

ABSTRACT

Complex simulation programs typically require large amounts of computation to produce highly detailed output difficult for users to understand. Building abstracted simulation systems that simplify both computation and output can make simulation both more economical and more intelligible. Ve describe an approach to abstracted Simula* tion that uses a scenario network to represent typical sequences of events in the simulation domain. Abstract simulation output is generated by probabilistically determined event-to-event transitions within the network. A learning process determines probabilities and builds up more abstract "chunked" events based on the actual frequency of event sequences in runs of the detailed simulator. The approach is generalizable across domains, and fulfills many of the goals of abstracted simulation: reducing computation, saving resources, filtering information and providing aggregated, intelligible output.

I INTRODUCTION

Artificial Intelligence techniques developed over the last decade have enabled us to create complex simulations in many domains. Such simulations typically include many rules that govern interactions between system entities and engage in intensive computation in order to generate their output. The complexity of these simulations can pose problems, however, in situations where the user is time-limited, or cannot afford the necessary computation, or is interested only in rough results. Furthermore, the output of such simulations, which often consists of an agonizingly detailed trace of system events, can be difficult to understand at a global or intuitive level. These two considerations, economy of resources (time, cycles) and intelligibility, argue for the development of abstracted simulation systems. simulations of reduced complexity that ignore certain interactions or collapse over some dimensions relevant in the detailed simulation. Abstracting a detailed simulation can simplify both computation *and* output, providing a general picture of simulation events with a minimum expenditure of resources.

Our interest in abstracted simulation was stimulated by our work on the ROSS simulation environment and our implementation of strategic air warfare simulator [5]. The detailed simulator includes objects such as bombers, fighters, radars, targets and command centers. These objects interact according to heuristic rules to produce •vents such as radar detections, bombings, fighter assignments, dogfights, and so on. Although ROSS has a graphics output capability that improves intelligibility, its main output is a lengthy event history. Furthermore, its speed decreases as increasing numbers of objects are included in the simulation. Hence, we have turned to abstracted simulation as a means of fulfilling several goals: 1) Filtering and aggregating output to improve intelligibility; 2) Allowing "quick and dirty" approximate simulations to save resources; 3) Allowing the user to choose a level of detail appropriate to his or her needs. In addition, we have found that abstracted simulation provides an arena for exploring a number of interesting AI issues involving learning and knowledge representation.

II A SCENARIO-BASED SCHEME FOR ABSTRACTED SIMULATION

Many alternative approaches to abstracted simulation are possible: 1) abstraction over time, e.g. by coalescing micro-events into more aggregated macro-events; 2) abstraction over instances, e.g. by replacing individual objects with generic class objects; 3) abstraction over space, e.g. by combining spatial areas or schematizing spatially defined entities (such as radar ranges or routes); 4) abstraction over procedures, e.g. by simplifying computations or caching average results as a substitute for computation [6]. In this research, we have focused on alternatives (1) and (2). The abstracted simulation model (ASM) generates event output by following a probabilistically- determined path through a scenario network that represents prototypic event sequences within the domain. The event network represents multiple levels of temporal aggregation; through interaction with the detailed simulation model (DSM), it builds abstract events by chunking together lower level events that tend to form recurring sequences in runs of the DSM. The user can then choose a level of abstraction and run the abstracted simulator as an independent system.

The abstracted simulation process includes two phases, the learning phase, in which the ASM uses DSM output to generate an initial scenario network and build up new macroevents, and the event generation phase, in which probabilistic event-to-event transitions are used to produce an event trace similar to the DSM output. Each of these phases is described in more detail below.

III LEARNING PHASE

Building up an abstracted simulation model begins with user specification of elementary simulation events. The set of elementary events derives from the output events produced by the detailed simulation model. In the ROSS air battle simulator, we used the event messages sent to the event reporting module to extract an elementary event set, e.g. "Radar4 detects bomber3", "Fighter9 detects bomber8". During the learning phase, these elementary events are connected by links indicating temporal succession to create elementary event networks.

Once the elementary events have been specified, they must be grouped according to the roles involved in each event. The actors in an event description determine its roles. For instance, the roles associated with "Radar4 detects bomber3" are RADAR and BOMBER. Typically, the event trace of a detailed simulation will indicate clusters of roles that interact (participate in common events) during some portion of the simulation history. For example, bombers interact with radars, with fighters and with targets, radars interact with command-centers and with bombers, and so on. We refer to these role clusters as interaction sets. At the conclusion of the learning phase, the ASM scenario network will include a subnetwork corresponding to each interaction set, comprised of events that share the same roles.

In the learning phase, the elementary event srt is modified in response to input from the DSM. Links between elementary events are created when those events occur in succession in the DSM input. Probabilities for each event-to-event transition are built up by monitoring the frequency of DSM events. Each event in the network keeps track of how often instances of that event occur in DSM runs; each event-to-event link keeps track of how often the two events it connects occur in succession. The ratio of link frequency to first-event frequency defines the conditional probability of transiting to the second event from the first.

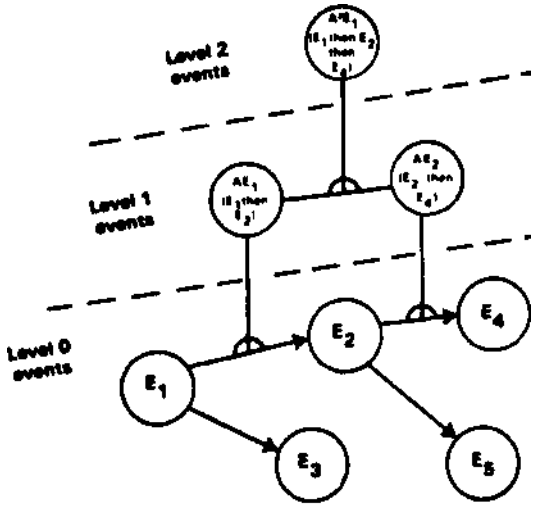

Fig. 1--Chunking to form abstract events

When several events occur together with sufficient frequency, the ASM chunks them to create an abstract event that represents that event sequence as a single entity (Fig. 1). This chunking depends on the fact that each link can also be viewed as a higher order event (namely, "event 1 then event 2"). Abstract events are created by linking together two pre-existing links. This process of aggregation can extend upward, generating more and more general descriptions of domain events.

Each interaction set is considered separately during the learning process. Links are created and incremented within each interaction set subnetwork, but not between subnetworks. However, the ASM monitors for the first event occurrence relevant to each subnetwork. When a first occurrence is detected, the ASM attaches a constraint action to the preceding event, indicating that the preceding event activates the new subnetwork. In the air battle domain (and, we suspect, in other domains as well), interaction sets operate in cascade. Bomber/radar interactions trigger radar/command-center interactions, which trigger command-center/fighter interactions, and so on. The constraint actions associated with a triggering event represent these inter-subnet dependencies.

Figure 2 shows a portion of the subnetwork structure build by the current version of the ASM, including event transitions, abstract events, and constraint actions. The set of subnetworks created during the learning phase provide the basis for event generation.

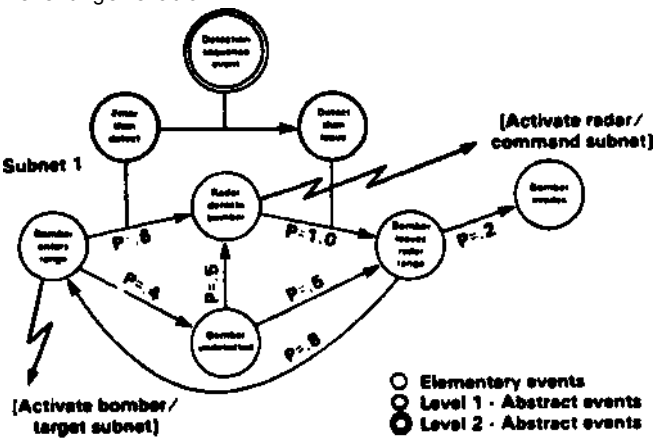

Fig. 2--Part of the abstracted scenario for air battle simulator

IV EVENT GENERATION PHASE

In the event-generation phase, the ASM acts like a Markov model, moving from eventestate to ovent-state based on the transition probabilities built up during learning. Event generation begins with the activation of the first subnetwork and the starting of a simulation clock. Activating a subnetwork involves setting its current state to a predefined start state and scheduling its first transition. Each subnetwork makes a transition

every x clock ticks, where x is a subnet-specific parameter. The transition procedure probabilistically selects the next event state from the candidate set specified by the network. Sometimes, no linked event will be chosen, and the subnet will remain in the same event-state until the next move. As each subnet transits to a new event-state, it can output the name of that event, thus producing an event history similar to that of the DSM.

The different subnetworks usually generate events independently. However, contingencies between subnets require the execution of constraint acts associated with triggering events. For example, the event "Radar reports detection to command-center", a part of SUBNET2, should not he generated until after the SUBNET1 event "Radar detects bomber". During the learning phase, the constraint action (ACTIVATE SUBNET2) will be associated with the "detection" event. If the subnet transits to that event, its constraint actions will be executed, allowing the "report" event to be generated by transitions in SUBNET2.

The event-generation phase of ASM allows the user to select several output modes. In the sampling mode, the user selects one or two roles to follow through the network. Only events involving the sampled roles are reported. Thus, this mode fulfills an information filtering function. Alternatively, the user may select the statistics mode, specifying statistics of interest (e.g. number of bombs dropped) and the number of each role should be considered. The ASM treats the probabilities associated with inter-event transitions as proportional factors, multiplying probabilities through the network until the event relevant to the statistic is reached.

All discussion so far has focused at the elementary event level. However, the same procedures apply to generating events at higher levels of abstraction, since the structure of event objects and links *is* the uniform. In general, the higher the level of abstraction chosen, the fewer the events in the network, and the faster the ASM runs.

V EVALUATION AND CONCLUSIONS

A. Implementation Status

Both the detailed and the abstracted simulation models are implemented in the ROSS language |7], an object-oriented, message-passing language based on Kahn's DIRECTOR (4). The learning mode has been implemented, using elementary events from a first generation detailed air battle model. The event-generation mode is in the process of implementation. Meanwhile, a second generation DSM is being created. Its more complex event structure will better test the power of the abstracted simulation approach described here.

!• Limitations of the ASM Approach

Although the current approach seems promising, it does have some limitations. First, no methods currently exist to guide the selection of elementary event messages for the original DSM. We have not defined "event" in any formal way, but have relied on intuition to structure the simulation output into useful and manageable units. Further experience with this approach may enable us to translate these intuitions into formal procedures for event specification. Second, ASM learning capabilities are limited to chunking together existing event concepts; it cannot discover new concepts. However, much of human learning consists of similar chunking processes |3]. Third, the ASM is not as flexibly parameterized as would be desirable. The detailed model allows the user to change both object parameters and rules of behavior [5]. The ASM trades flexibility and modifiability for intelligibility and speed.

C• Advantages of the ASM Approach

The ASM approach offers many advantages. It can be generalized to simulation in practially any domain. Any detailed simulation model can provide a set of elementary events on which the abstracted simulation can be based. Furthermore, this approach does fulfill many of the goals *of* abstracted simulation. It reduces computation, saves resources, allows information filtering, and generates aggregated, intelligible output. Finally, this prototype-based scheme in many ways it parallels the workings of human learning and memory. Based on world experience, humans build up prototypic event sequences that represent likely occurrences, i.e., scripts or schemata |8). These event sequence representations can be considered at multiple levels of aggregation, e.g. scenes, episodes, etc. (1) When asked to predict the events that will occur in a specific situation, humans will retrieve typical events in the order they are likely to occur |2). The ASM is not intended as a serious model of human memory and mental simulation. However, it may have heuristic value in investigating these areas, as well as practical value in improving simulation effectiveness.

REFERENCES

|1) Abbott. V. and Hack, J.B."The Representation of Scripts xn Memory". Yale University Cognitive Science Technical Report #$, 1980

|2] Goldin, St "Script* and Mental Simulation". Paper In preparation

(3) Hayes-Roth, I. "Evolution of Cognitive Structure* and Processes." Psychological Review. 1977, 4, 26O-27*\

IM Kahn. KM. "Director Guide", A! Heex> 4S2B. Massachusetts Institute of Technology 41 Laboratory, Cambridge. HA. If79.

|5| Klahr, P. and Faught. VS. "Knowledge-baaed Simulation " InProc—dint*. first Annual National Conference "on Artificial IntelUsance. Stanford University, 1960.

|6) Unit. D , Hayes-Roth. F. and Klahr, P. "Cognitive Econoaiy in Artificial Intelligence Syateae." In Proceedinss UCAI»7e, 531-536

(7| McArthur, 0. and Sovitrel, H. NAn Object-oriented Language for Constructing Simulations." In Proceedings UCAI-tl.

(0) Schank, R. and Abelton. R. Scripts. Plans. Go,«U and Understanding Lawrence Erlbeue Associates, 1977.