Autonomous Learning of Commonsense Simulations

Benjamin Johnston and Mary-Anne Williams

University of Technology, Sydney
Ultimo, Sydney, New South Wales, Australia
johnston@it.uts.edu.au

Abstract

Parameter-driven simulations are an effective and efficient method for reasoning about a wide range of commonsense scenarios that can complement the use of logical formalizations. The advantage of simulation is its simplified knowledge elicitation process: rather than building complex logical formulae, simulations are constructed by simply selecting numerical values and graphical structures. In this paper, we propose the application of machine learning techniques to allow an embodied autonomous agent to automatically construct appropriate simulations from its real-world experience. The automation of learning can dramatically reduce the cost of knowledge elicitation, and therefore result in models of commonsense with breadth and depth not possible with traditional engineering of logical formalizations.

Introduction

Comirit is an open-ended hybrid architecture for commonsense reasoning. We have previously described (Johnston and Williams 2008) how Comirit is a generalization of the method of analytic tableaux; combining rich 3D simulation with formal logic. In this paper we extend the architecture so that it supports *autonomous learning* in addition to deduction. We demonstrate the system by implementing stochastic search and an auto-associative network in the framework: this enables our experimental system to autonomously acquire and maintain reliable knowledge of novel objects and their behaviors.

At Commonsense 2007 (Johnston and Williams 2007), we argued that simulations are a potentially rich resource of commonsense knowledge and are an expressive and efficient mechanism for commonsense reasoning. The potential for breadth and depth is clearly evident when one considers the advances in modern animations and computer games: modern games offer realistic open-ended 'sandbox' environments for unlimited experimentation and interaction. We showed that a generic graph-based representation can be used to rapidly create similarly rich and realistic simulations with minimal software development, and thereby produce the possibility of extracting and directly reasoning with the knowledge that would otherwise be only implicitly represented in simulation. Generic graph-based representations further simplify the knowledge elicitation process to a task that can be performed as a routine software development process without the expense (or concern for a shortage) of skilled logicians or philosophers.

While simulation may be seen as a powerful heuristic for deducing possible and likely future states from current conditions, a weakness of simulation is that it must follow the 'arrow-of-time'. That is, it is impossible to simulate a complex situation in *reverse* to deduce likely causes or precursors of a situation: one cannot simulate spilled milk in reverse to discover a likely cause—it is far easier to simulate the outcome from a particular cause such as dropping an open milk carton onto the floor. When greater deductive power is required than that of forward-running simulations, it is therefore necessary to augment simulation with other mechanisms.

We proposed (Johnston and Williams 2008) the integration of simulation and logical deduction as a way of combining the efficiency and richness of simulation with the power of logic. Our framework generalized the method of analytic tableaux to allow both logical terms and simulation objects within a single search structure. A single unifying principle based on the idea of searching through spaces of possible worlds enabled these disparate mechanisms to be harmoniously combined in a single system.

While our framework offers a mechanism for commonsense reasoning, we acknowledge that an effective system includes not only a reasoner, but also a comprehensive commonsense knowledge-base. The 20-year Cyc project (Panton *et al.* 2006) to create a commonsense knowledge-base serves as a clear demonstration that such engineering can be exorbitantly expensive. We therefore wondered, "is it possible to automate knowledge acquisition?" Our graph-based simulations simplify the engineering process, so we initially expected that powerful semi-automatic engineering tools could permit rapid knowledge elicitation. In designing these tools, it soon became apparent that the graph representation could allow *fully autonomous* learning and generalization of aspects of the behavior of novel objects.

The purpose of this paper is therefore to explain how learning may be incorporated into our commonsense reasoning framework. We view cumulative learning as an iterative process of hypotheses generation and selection: this perspective can be elegantly incorporated into a tableaux reasoning framework by ranking branches of the search tree and allowing for factoring of sub-problems. The purpose of these modifications is to allow branches of a tableau to contain hypotheses, and for the search algorithm to focus only on those branches with the best hypothesis.

This paper begins with a brief overview of the existing framework: the underlying representation used by simulation, and the hybrid reasoning strategy. We then explain how the framework is extended to allow learning, and conclude with a concrete exploration into how stochastic hill-climbing and auto-associative networks may be incorporated into this framework to allow a system to autonomously learn simulations of novel objects.

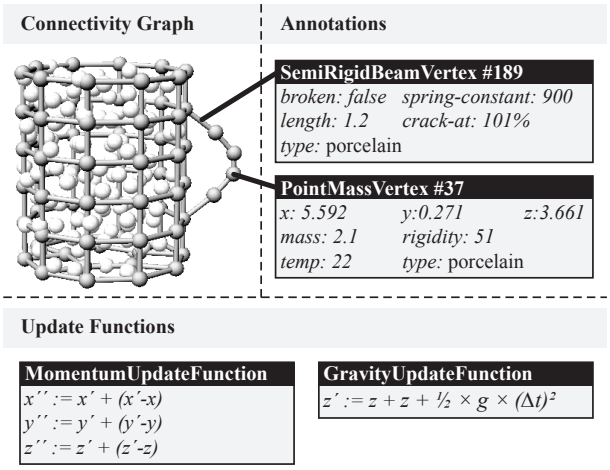

Figure 1: *Examples of the components of a simulation*

Simulation

In the Comirit framework, simulations are used as the underlying mechanism and representation for large scale commonsense knowledge. Not all knowledge can be represented efficiently in simulations (*e.g.*, 'What is the name of the Queen of England?'), but simulation works well in problems governed by simple laws (such as physics) and so simulation is used in the framework wherever possible.

Comirit simulations are a sophisticated generalization of an early proposal by Gardin and Meltzer (1989); extended to support 3D environments and non-physical domains. Comirit simulations are constructed from a graph-based representation. The fundamental structure of a problem is first approximated by a graph. The graph is then annotated with frame-like structures, and simulation proceeds by the iterative update of the annotations by update functions.

The formal details of simulation are not needed to understand this paper, so we will use illustrative examples. Readers interested in the formal details should refer to our previous publication (Johnston and Williams 2007).

A Comirit simulation consists of the following parts:

1. a system clock that increments by finite intervals
2. a (relatively) static graph representation that models the underlying structure of a problem domain,
3. a set of highly dynamic annotations that record the state of a simulation, and
4. a static set of computable functions or constraints that update the annotations with each iteration of the system clock and thereby drive the computation of the simulation.

This representation is intentionally generic. We claim that it can be used to represent simulations from any rule-driven problem domain including physical, social, legal, economic and purely abstract realms. Our research has emphasized physical reasoning and naïve physics, so we will illustrate simulation and learning through examples based upon 3D simulations of physical models.

Consider a simple domestic robot facing a physical reasoning problem: *given a mug filled with coffee, is it 'safe' to perform fast movements to carry the mug?* In the Comirit framework, the robot considers the problem by internal simulations of the scenario; testing whether a simulated mug is damaged by fast movement, or if such motion causes damage to the environment by spilling coffee.

The generic graph structure is used to represent the underlying structure of the problem. For example, the mug of coffee can be approximated as a mesh of point masses connected by semi-rigid beams. A visualization of such a graph appears in Figure 1. Note that both the spheres and beams are vertices of the underlying graph, their connectivity is recorded by edges in the graph.

Each vertex of the simulation graph (*i.e.*, each point mass *and* each semi-rigid connecting bar) is annotated with a set of frame-like attributes such as the current 3D position, local mass distribution, rigidity, physical state, temperature and whether the local structure has been broken (due to over-stressing). Examples of particular annotations and values also appear in Figure 1. Note that these annotations represent only *local* properties of the simulation. The total mass of the mug of coffee is equal to the *sum* of all of the `mass` attributes.

Simulation proceeds by the iterative update of annotation values. Newton's laws of motion are applied to each of the point masses, and Hooke's law (describing the behavior of a spring) is applied to the connecting beams. Figure 1 illustrates update functions for the laws of momentum and gravity. Note that these functions have only short-term and local effects.

The combined effect of iteratively computing local updates on the annotations is emergent behavior that closely resembles the actual behavior of real world scenarios. Laws of physics are simple at the microscopic scale. The macroscopic shape of an object, its centre of mass, its rotational inertia and its viscosity or brittleness vastly complicate the physical laws of motion of large bodies, but these simply emerge from iteration of simple laws at the microscopic scale. Indeed, this method of simulation may be seen as a variation on the Euler method of numerical integration. Simulation effectively performs numerical integration over the fundamental laws of physics that are expressed as stepwise differential equations in the update functions.

If we run a mug of coffee simulation we may result in an outcome such as that depicted in Figure 2. Symbolic results are reported by simple routines that inspect the state of the simulation to determine if, for example, any semi-rigid bars have broken (in the case of a '`broken`' symbol), or if any liquid is no longer contained by the mug (in the case of a '`mess`' symbol). This outcome would imply that the robot should not use fast movements with the mug.

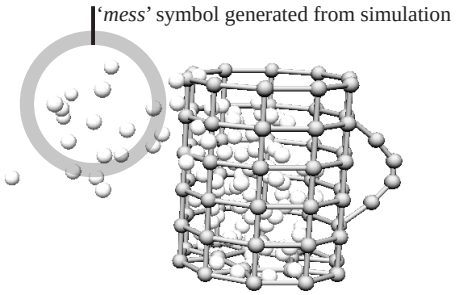

Figure 2: *Simulating a 'fast move' on a mug of coffee*

Hybrid Architecture

Simulation is a powerful and efficient tool for commonsense reasoning, but it only supports a 'forward chaining' inference mode: it is therefore an incomplete solution for general purpose commonsense reasoning. We integrate simulation with logical deduction in a hybrid architecture in order to combine the strengths and complement the weaknesses of each mechanism. That is, we use the deductive power of a general-purpose logical reasoner to make up for the inflexibility of simulation.

In combining simulation and logic, blackboard architectures, tuple spaces and agent architectures serve as obvious choices for implementation: they have a long history of application to problems of integration in intelligent systems. Unfortunately, our experience is that the conceptual mismatch between simulation and logic is such that application of these integration techniques eventually results in systems that are unworkably complex and difficult to maintain. Instead, a clear and unifying abstraction is required to harmonize the semantics of the reasoning mechanisms. We claim that this can be achieved by our hybrid architecture that performs logical deduction with the method of analytic tableaux, and interprets both simulation and logical deduction as operations over spaces of worlds.

The method of analytic tableaux (Hähnle 2001) is an efficient method of mechanizing logical theorem proving. Analytic tableaux have been successfully applied to large problems on the semantic web, and there is a vast body of literature on their efficient implementation (*ibid.*). The method constructs trees (tableaux) through the syntactic decomposition of logical expressions, and then eliminates branches of the tree that contain contradictions among the decomposed atomic formulae. Each branch of the resultant tableau may be seen as a partial, disjunction-free description of a model for the input formulae. The crucial insight is that if a tableau algorithm is given knowledge of the world and a query as logical input, then the conjuction of the atomic formulae in a branch of the resultant tree represents a *space of worlds that satisfy the query*.

The tableau method *and* simulation can thereby be unified through this common abstraction. The tableau algorithm generates spaces of worlds, and simulation expands upon knowledge of spaces of worlds (*i.e.*, forward chains to future states based on current states). In our framework, we perform commonsense reasoning by generalizing the tableau so that it can contain non-logical terms such as simulations, functions and data-structures in addition to the standard logical terms of traditional tableaux. Deduction proceeds by application of both tableau rules that expand, fork or close branches of the tree, and simulations that can expand and close branches of the tree.

The full details of this method appear in our earlier publications, but we will review the principles here by way of a simplified example. Consider the following scenario:

> *A household robot needs to move an object across a table. Its actuators can perform a soft or a hard movement. It is unsafe to move any object 'quickly'. What commands may be sent to the actuators?*

For the convenience of our example calculations, let us assume that the mass of the object is 1kg, the soft force is 1N, the hard force is 2N, the object is simply pushed for 1s and unsafe speeds are 1.5ms^{-1} or higher. Furthermore, we assume the following highly simplified and abstracted simulation[1]:

```
function simulate(Mass, Force, Time):
    set Speed := Time * Force / Mass
    return {speed = Speed}
```

We can then convert the scenario to a logical form:

$$time{=}1 \wedge mass{=}1 \wedge safe \wedge (command{=}hard\text{-}force \vee command{=}soft\text{-}force)$$

With this logical form, we may then apply the method of analytic tableau and simulation to find models that satisfy

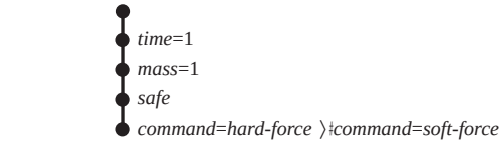

Figure 3a: *First, each conjunct in the original query is expanded into separate nodes.*

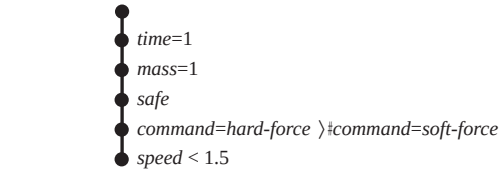

Figure 3b: *The term 'safe' is expanded per its definition.*

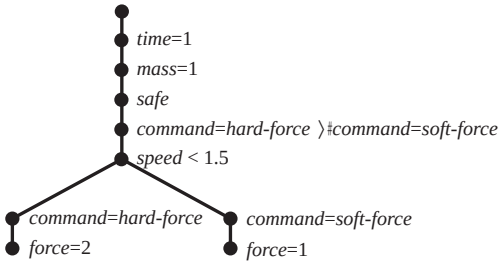

Figure 3c: *The tableau is forked into two branches: one branch for each disjunct in the fifth node. 'Hard-force' and 'soft-force' are then expanded per their definitions. Logical deduction has now 'stalled': no more logical rules apply.*

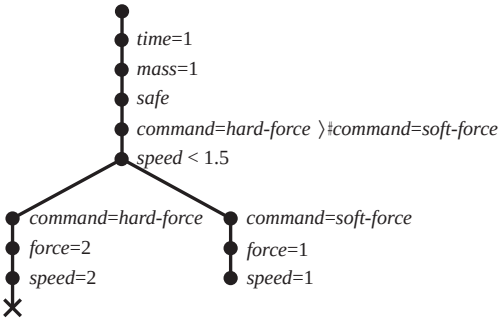

Figure 3d: *Simulation is now invoked. As a result, the left branch becomes inconsistent (speed=2 and speed<1.5). The right branch remains open and therefore describes a scenario satisfying the original query (i.e., the robot can safely use soft-force).*

[1]Note that while highly simplified, this algorithm has similar constraints to real simulations: the inputs are numerical and fully specified, and the algorithm can only be used in the 'forward' direction.

the formula. The algorithm proceeds in the steps illustrated in Figures 3a–3d.

When the algorithm terminates there is a tree with only one open branch. Reading atomic formulae along that remaining branch, we see that it describes a world in which the *command=soft-force* action is applied and the object moves safely at $1ms^{-1}$.

Thus, we have used both simulation and logical deduction in a single mechanism to solve a (simplified) commonsense reasoning problem. Details such as data structures, methods for prioritizing computation, search strategies and output variables have been omitted from this example for clarity, however these can be found in our earlier publication (Johnston and Williams 2008).

Integrating Learning

Commonsense reasoning requires more than a hybrid method of deduction: it depends on the availability of rich and accurate knowledge of the world. In contrast to logical methods that depend on highly skilled logicians painstakingly encoding their intuitions into formal axiomatizations, a small number of fundamental laws are first implemented in a simulation, and then descriptions of the world can be readily added to a simulation in a simple two stage process:

1. Structuring the underlying simulation graph to match the observed structure of the situation to be simulated (such as creating a 3D wire-frame model),
2. Configuring the annotations on the graph so that the simulations closely predict reality.

In our own experiences with constructing simulations, we observed that these tasks involved little mental effort but were a tedious process of careful tuning of parameters to match reality (*e.g.*, trial-and-error to determine an appropriate spring constant to simulate a rubber ball). We begun creating tools to support this process, but quickly realized that tedious and undemanding tasks are ideal candidates for full automation. Consider the following:

1. The update functions in a simulation are static: they are easily implemented manually, and are rarely changed. For example, once the laws of Newtonian physics have been implemented, they can be used in simulations of almost all mechanical systems. While we currently view this as outside the realm of feasible automation, we do not consider this effort to be substantial.
2. The underlying static graph can often be directly observed from the environment. In the case of physical simulations, a wire-frame approximation can be automatically assembled from the 3D volumes reconstructed from moving camera images, stereoscopic vision, LASER sensing, 3D scanning, LIDAR, Z-cameras, direct physical contact or other scanning/imaging techniques.
3. The annotations that guide the dynamic behaviour of simulations are typically numeric so their values may be learnt with standard machine learning techniques (*i.e.*, by search for parameters that minimize the error between simulation and observed reality).

For example, if we have a household robot that uses simulation as an underlying representation, then the robot can acquire knowledge of a novel object (say a mug of coffee) by building a wireframe model from a robust 3D shape reconstruction algorithm, and then searching for annotations that match the observed behaviour of the object. Observations can be used to test hypotheses about annotation values by simulating the behaviour of the novel object and comparing them to the observation. In our framework, the robot automatically and continuously learns object and annotations in a background 'process' by constantly simulating from recent observations, and testing that expectations match the current observation. This is illustrated in Figure 4.

The relationship between input and output in a simulation is difficult to compute analytically (indeed, if there were simple analytical solutions, it is unlikely that simulation would be applied to the problem in the first place). Annotations must be computed by numerical optimization or machine learning algorithms. In particular, we intend to use a greedy search to find these values (any other generate-and-test algorithm can be used, as appropriate to the problem: genetic programming, beam search or simulated annealing).

How then, can optimization be incorporated into our unifying abstraction of search over spaces of possible worlds? Optimization requires comparison of separate spaces of worlds, and therefore involves comparing separate parts of the search space or separate branches of a tableau. Unfortunately, the standard tableau algorithm only permits manipulation or inspection of a single branch. We avoid this problem by introducing an (incomplete) ordering over branches, and then modifying the tableau algorithm so that it searches for minimal models. A branch in a tableau is no longer considered 'open' simply if it is consistent—it is open if it is either *consistent and unordered*, or else *it has an ordering and is minimal among all other consistent and ordered branches*. That is, the algorithm considers all unordered branches, and only one ordered branch. (Note also that if the minimal ordered branch is found inconsistent, the next most minimal branch is then considered again.)

We *could* define the ordering outside of the tableau. For example: "Branch *a* is smaller than branch *b* if the error of the first hypothesis in *a* is smaller than that of the first hypothesis in *b*. If the first hypothesis in *a* and *b* are equal, then the ordering is determined by the second hypothesis (and so on)". However, such ordering rules are inconvenient because they are defined outside the tableau.

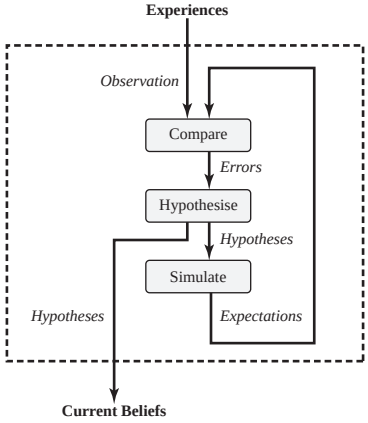

Figure 4: *Background learning process*

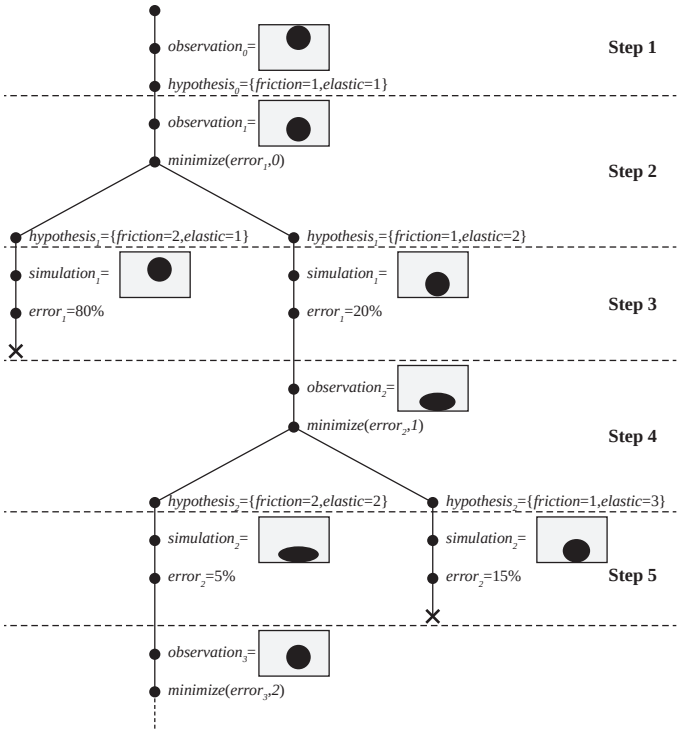

Figure 5: *An example of learning in a tableau*

Instead, we define an ordering with symbols stored within the tableau. We hold the set of propositions **`minimize`**(*VariableName*, *Priority*) as tautologically true in the standard tableau algorithm, but use them in evaluating the order of the branch. The *Priority* is a value from a totally-ordered set (such as the integers) and indicates the order in which the values of variables are sorted: branches are first sorted by the highest priority variable, then equal values are sorted using the next highest priority variable, and so on.

Consider our household robot example again. If the robot encounters a novel object it will need to find suitable parameters to simulate and therefore reason about the object. If there is another agent or a designated 'teacher' demonstrating how to handle the object, it has a ready stream of observations for learning. If the object is simply sitting alone, it may need to apply its most conservative and gentle action to the object to gather some initial data. The robot can then use a stochastic hill-climbing strategy on its observations to learn about the object:

1. Initially, a default hypothesis about the values of annotations is assigned to the novel object.
2. When a new observation arrives, the robot generates a set of alternate hypotheses (random perturbations, in the case of a stochastic hill-climbing strategy) as a disjunction in the tableau: this disjunction produces new branches in the tableau.
3. The alternate hypotheses are each simulated from the prior observation in order to generate predictions for the current observation. The error between expectation and reality is computed.
4. The best hypothesis is implicitly chosen by the tableau algorithm, due to its preference for minimally ordered branches. The algorithm continues again with step 2.

Note also that because our tableaux can contain logic, simulations and functions, the system may use logical constraints or ad-hoc 'helper functions' even when searching for values in a simulation (*e.g.*, a constraint such as $mass > 0$, or a custom hypothesis generator that samples the problem space in order to produce better hypotheses).

An example of learning the behaviour of a falling ball by hypothesis search in a tableau appears in Figure 5:

Step 1: The tableau initially contains the first observation of a ball and the initial hypothesis generated (many other control objects, meshes, functions and other data will be in the tableau, but these are not shown for simplicity).

Step 2: The system observes movement in the ball. It generates new hypotheses, seeking to find a hypothesis with minimal error.

Step 3: The system simulates from $hypothesis_0$. The result of the simulation is compared with $observation_1$ to determine the error in the hypothesis. The right branch has smaller error so the left branch is no longer open.

Step 4: As with Step 2, the system observes more movement and generates new hypotheses, further refining the current hypothesis.

Step 5: The system then simulates as with Step 3, but this time the left branch is minimal. In the following steps, the algorithm continues yet again with more new observations and further hypothesizing…

Experimental Results

We tested an implementation of this technique in a simple virtual environment: boxes and balls of varying sizes, masses and elasticity (some were rigid, others quite elastic), were pushed by variable forces for variable periods of time[2]. A virtual 2D 'camera' observed the interactions, and the Comirit learning algorithm was used to construct models from observations, generate hypotheses and simulations, and compare observation with simulations. The accuracy (as tested across many experiments) was surprisingly high:

1. A single observation pair is sufficient to learn annotations for simulating with 94% pixel-by-pixel accuracy.
2. Four observation pairs bring this accuracy up to 97.5% accuracy.
3. Further observations result in small, incremental improvements, approximately halving the error with each doubling of the number of new observations.

In our subsequent experiment, we used a broader concept of 'hypothesis'. Rather than learning the isolated annotations of individual objects, the hypothesis space was a self-organizing map (SOM) with feature vectors that include observable appearance and hidden parameters. The system retrieves an initial hypothesis for the annotations of a new object by searching the self-organizing map with a partial vector describing only the object's observable appearance. When the system observes errors between observation and simulated expectation, a new and complete feature vector is

[2] *i.e.*, Two observable object parameters (*shape*, *size*), two hidden parameters to be learnt (*mass*, *elasticity*) and two known observation parameters (*force*, *duration*)

generated and this is updated into the self-organising map.

We hoped to demonstrate an ability to generalize knowledge, so we added structure to our learning problem:

1. Color was inversely correlated with elasticity (we observe a similar effect in real life when shiny metallic objects in the real world are usually rigid).
2. Heavy (dense) objects were generally inelastic (as we often observe in real life).
3. Round objects were generally inelastic (we also observe similar correlations between shape and behavior in real life: a mug is generally rigid so that it may safely hold hot liquids).

To our surprise, not only did the system build a self-organizing map that captured the problem structure (and therefore allowed it to correctly generalize its learning about previously unseen 'heavy boxes'), but the self-organizing map improved the speed at which the algorithm learnt. This improvement is due to the map offering better hypotheses during early learning by generalizing from similar cases. The ability of the framework to rapidly learn from very few observations, meanwhile prevented it from being committed to its SOM hypothesis when it encountered 'outlier' objects.

In this latter trial we used a less aggressive hill-climbing strategy to allow greater exploration of the search space and better test the advantage of a SOM. With this weaker strategy, learning on a single object requires up to 8 observations to achieve 90% accuracy. A randomly initialized SOM has less than 5% accuracy, but after observing 14 objects once each, it achieved 60% accuracy, and then reached 90% accuracy after just six sets of observations and the whole map converged after 15 sets of observations at 94% accuracy.

We consider these trials as early demonstrations of the soundness of the basic concept. In future, we will identify and adapt a robust 3D reconstruction technology, and run these experiments on *real world* objects within *real world* settings. We have skimmed over many of the details of auto-associative learning with self-organizing maps because we expect to make dramatic changes when we fine-tune the technique to real world problems. We are, however, extremely encouraged by the success of our relatively naïve implementation. In future, we plan to extend the use of SOMs to assist in constructing the graph-based models (*i.e.*, by extrapolating the obscured shape of the object from observed shape) and for allowing rich shape-based and affordance-based indexing, retrieval and similarity testing.

Performance Considerations

While placing an ordering on branches enables the tableau algorithm to emphasise optimal consistent branches and discontinue search on suboptimal branches, the suboptimal branches cannot be discarded from memory. This is because an optimal branch may later be found inconsistent, and therefore cause a previously suboptimal branch to become the most optimal *consistent* branch that remains.

In many cases, however, it may be known that this cannot happen. For example, it may be known that any inconsistency will apply to all ordered branches, or it may be known that ordered branches will never contain inconsistency. In this case it is possible to introduce Prolog-style 'cuts' into the tableau: special terms to indicate that non-optimal branches may be dropped. Of course, care must be taken to ensure that cuts are genuinely free of unintended side-effects—that they are 'green cuts', to use the terminology of Prolog.

Another concern is that robots will need to simultaneously learn while engaged in action. If action and learning occurs in the same tableau, there is a need to prevent branching in decision-making from causing the same learning problem to repeat in multiple branches of the tableau. This can be solved by careful factoring: continuous online learning is performed in a separate tableau, but the contents of that tableau are implicitly read in *logical conjunction* with the contents of the primary action and decision making tableau.

Such optimizations have straightforward implementation, however we will provide full details in future publication.

Conclusion

Parameter-driven simulations are not only an effective mechanism for rich commonsense reasoning, but they lend themselves to rapid and autonomous acquisition. We have shown how models of learning can be elegantly incorporated into the Comirit framework (and potentially other tableau-based systems). The extended framework thus simulateously combines the effectiveness and efficiency of simulation with the ability for autonomous knowledge acquisition, and with the full power and generality of logical formalisms.

To date, we have demonstrated the system on simple (but useful and plausible) learning problems. Early experimental results are extremely encouraging: the system learns rapidly and with very few observations.

As a long term goal, we plan to have Comirit autonomously acquire from observation, even the fundamental laws of a simulation and the mechanisms for model building. In particular, we hope that interaction in a complex environment (such as the 3D world) may be interpreted as a problem of determining a hierarchical non-linear flow of 'entities' within an environment. To whatever degree such automation is possible, our framework can accommodate any learning that can be expressed as an optimization problem, and yet allow for ongoing integration with symbolic and logical formalizations.

References

Gardin, G. and Meltzer, B. (1989) 'Analogical representations of naïve physics', *Artificial Intelligence*, vol. 38, no. 2, pp. 139–159.

Hähnle, R. (2001) 'Tableaux and Related Methods', *Handbook of Automated Reasoning*, vol. I, pp. 100–178, Elsevier Science.

Johnston, B. and Williams, M-A. (2007) 'A generic framework for approximate simulation in commonsense reasoning systems', *Proceedings of COMMONSENSE 2007*, pp. 71–76.

Johnston, B. and Williams, M-A. (2008) 'Comirit: Commonsense reasoning by integrating simulation and logic', *Proceedings of AGI-2008*, pp. 200–211.

Panton, K., Matuszek, C., Lenat, D., Schneider, D., Witbrock, M., Siegel, N. and Shepard, B. (2006) 'Common Sense Reasoning – From Cyc to Intelligent Assistant', In Yang Cai and Julio Abascal (eds.), *Ambient Intelligence in Everyday Life*, pp. 1-31, LNAI 3864, Springer, 2006.