Acquiring Commonsense Through Simulation

Andrea Lyn Lockerd
MIT Media Lab
20 Ames St. E15-313
617-253-0597
alockerd@media.mit.edu

ABSTRACT
In this paper, I present simulation as an alternative solution to the problem of commonsense knowledge acquisition. This position stems from the fact that commonsense is essentially redundant shared experience over time, and posits that reality computer games could be a forum in which computers could gain this shared experience from a human user. The implementation and initial results of a learning program, ThoughtStreams, is shown to have some initial success in predicting commonsensical future events based on past experiences in a simple simulation game environment.

Keywords
Commonsense, Knowledge Discovery

INTRODUCTION
For years there has been much discussion in the AI community about what it would take for a machine to have commonsense. Marvin Minsky discusses many aspects of commonsense, but perhaps his biggest message is that if something is only understood in one way then it is hardly understood at all, and that common sense reasoning is more than just facts but includes knowledge about how to think, negative expertise, analogies, and self-reflection [5]. John McCarthy is of the formalist camp, and has dealt for years with the problem of expressing common sense knowledge and common sense reasoning in terms of formal logic [3].

There have also been a few attempts to represent commonsense knowledge to a computer. In the 1980s, Doug Lenat set off on the venture to encode commonsense knowledge into a database, a project called CYC, wherein each commonsense fact is written in a formal language, arranged in an ontology, and tagged with context information [1]. This project has been an enormous knowledge engineering feat, and is still underway. Open Mind Common Sense is a similar endeavor, while the main goal is still to build a database of commonsense knowledge the representation and acquisition is quite different [7]. Facts in OMCS are written in natural language (English), and the vision is to have people be able to teach the system about common sense in the form of – giving knowledge pieces, clarification, organization, validation, repair. Thus, unlike CYC, all of the work is left to an inference engine to make sense of and organize the knowledge instead of the person entering the knowledge. Another effort to encode commonsense knowledge sits perhaps between these two examples. Eric Muller's ThoughtTreasure is a database of commonsense facts that are represented in natural language but also organized in a hierarchy.

Each of these approaches of having people encode facts about the world in a representation accessible to a machine runs into the problem of context. Defining what context was considered when the statement was entered and having to keep adding and adding exceptions and clarifications to statements when there's a conflict. CYC, for example, uses a 12-dimensional context space in which statements can be situated, but again the burden of adequately describing a context is left to the knowledge engineer.

Is building up big databases of commonsense the only approach? This paper presents an alternative position, acquiring commonsense through simulation.

There are a number of different opinions about what should and shouldn't be included in common sense, and what it means for a computer to "use" common sense, but one way to think about common sense is shared knowledge or implicit knowledge. For instance, in conversation, what goes unsaid is implicitly understood to be shared knowledge, or common sense. In the physical world, "things fall down not up", is implicitly understood to be shared knowledge given that we all share the same physical world. This idea extends into smaller circles of shared knowledge, where the smallest circle of common sense could be 2 people or a person and a computer. Any time a group of people shares an experience; they can talk about or refer to that experience assuming that the others share the common ground.

Building on this idea that common sense is what goes unsaid, how do we acquire all of this implicit knowledge? Even though we cannot remember not knowing most of what is considered common sense, at some point we all had to learn these things either by being told explicitly by a parent or teacher or through observation and experience. One way a computer might be able to go through a similar learning process is through simulation or a virtual world. This approach takes advantage of the fact that there is a world in which a user can exhibit some commonsense behavior and the computer can sense and effect. This is related to and motivated by the ideas of Programming by Example [2]. Letting the user and the agent interact in this virtual space turns into an interactive learning session for

the agent, allowing the computer to learn in a common sense way: learning what is common by watching multiple interactions, learning about a new context or situation by recognizing action/environment similarities and differences, making predictions and seeing how they pan out.

This paper describes a learning agent built to observe a simulated reality and glean some commonsense knowledge. The following sections will go through the motivating scenario, the particular approach and implementation, and then a discussion of the resulting system.

MOTIVATION

The following example is one originally posed by Lenat and widely discussed in the commonsense community. When you hear or see the sentence "*Fred told the waiter he wanted some chips.*" What is some of the commonsense that lets you *understand* this sentence:

The word "he" means Fred—and not the waiter.

This event took place in a restaurant.

Fred was a customer dining there.

Fred and the waiter were a few feet apart.

The waiter was at work there, waiting on Fred at that time.

Both Fred and the waiter are live human beings.

Fred was speaking words to the waiter.

Both of them speak the same language.

Both were old enough to talk.

The waiter is old enough to work. Fred is hungry.

He wants and expects that in a few minutes the waiter will bring him a typical portion—which Fred will start eating soon after he gets them.

We can also assume that Fred assumes that the waiter also assumes all those things.

Now the question is how to get a computer to bring all of these things to bear and *understand* this sentence? Lenat and the various database approaches propose that all of this knowledge be encoded in a database for an inference engine to use in its deliberation. But consider the simulation approach. If the machine had seen a few different examples of going to a restaurant, it might be able to determine at least some of these facts just through looking at what is common between the current situation and its past experiences. It is not hard to imagine that it would be able to say the following having seen a few restaurant interactions in the past:

This event took place in a restaurant.

Fred was a customer dining there.

Fred and the waiter were a few feet apart.

He wants and expects that in a few minutes the waiter will bring him a typical portion—which Fred will start eating soon after he gets them

APPROACH

To try this idea I built a small game world of a mall that allows a user to show the computer common behaviors in a few different stores and restaurants. This is a simulation of having a game environment like the SIMs [8]. A reality computer game like the SIMs encapsulates commonsense knowledge on two levels. On one level there is the commonsense knowledge that went into building the game: dividing up the world into places, actors, and objects and enabling interactions between the three. On another level there is the commonsense that is exhibited by someone playing the game and giving the computer examples of commonsensical sequences of interactions in the world.

A defining element in the simulation approach to commonsense acquisition is the ability to notice differences and similarities. This is motivated by what Minsky calls our 'world of differences'. He notes that our perceptual faculties are tuned to react to changes in time [4] Steven Pinker also relates this idea in terms of language acquisition [6]. He explains how children hate synonyms, that no two words are exactly the same; when there's a difference there's a reason and kids are good at finding it.

Another important aspect of this approach is experimental learning. This is the process of predicting outcomes and comparing results to see if your memories are reliable, and then changing them or forgetting them if they are not.

IMPLEMENTATION

The system can be divided into three parts: the game, the game log output, and ThoughtStreams the commonsense acquisition program. This game is meant to represent a very slim version of a reality game like The SIMs, and what type of reasoning and knowledge acquisition this sort of platform might afford. The hope is that this method of common sense reasoning would generalize to any system or game world where the computer could sense a world view over time, and had some measure of similarity.

The Game

The game starts at a Storefront, and the user has various options of stores and restaurants they can enter (Starbucks, The Gap, etc) and each store has some objects and actors and interactions that are available with these objects and actors. As the user is going through the game a log of what happens is output to a file.

The Game Log

As previously mentioned, a computer game is a nice paradigm because the world is divided up by the designer into: places, actors, objects, and interactions/effects (actor/actor & actor/object), and the events of the game can be output to a file over time.

In this implementation, a LogEvent encapsulates the information in a single line of the game log: Place, ActorsInView, ObjectsInView, and Interactions/Effects. Actors, objects and places have a unique string name used to refer to them, and interactions have a unique string name and optional arguments (EX: eatdrink<object>, buy<object>, enter<place>, leave<place>, sayhi<actor>,

saythanks<actor>, readmenu, sitattable). The LogEvent also contains the functionality of calculating similarity and difference between two LogEvents.

ThoughtStreams

ThoughtStreams is an application that runs separate from the game and watches the game log. The goal of the program is to discover common sense about the 'world' that is exhibited through people playing the game. It does this through building long-term memories of common experiences and using these to form expectations about future events.

ThoughtStreams is comprised of the interaction of multiple threads of activity and their various interactions with the data structures for short-term memory (STM) and long-term memory (LTM). In short, there is a *Monitor* thread that packages a new line from the game log into a log event and stores it in STM. If the current event triggers a long-term memory the *Monitor thread* starts a *Remember* thread, and if the current event has some similarity to a recent event in short term memory the *Monitor* thread starts a *MakeMemory* thread. The *MakeMemory* thread packages the similar prior events that lead up to the two similar instances into a LTM structure. The *Remember* thread is triggered by these prior events and is then able to predict future events. Finally, there is a *Ponder* thread that tries to improve long term memory by removing pieces of a LTM that have shown particularly poor prediction reliability.

Monitor

This is the main thread of the program that watches the game log file and creates a LogEvent for each new entry and stores it in an array called short term memory (STM). Another LogEvent is created that encapsulates the difference between this LogEvent and the previous one and is stored in an array called delta short-term memory (ΔSTM). For each new event a FindSimilarDiff thread is started.

Find Similar Difference

Giving the difference between the current event and the last one, the FindSimilarDiff thread searches through ΔSTM and LTM for similar differences. If this one is similar to a LTM then a Remember thread is started. Otherwise it looks through ΔSTM, and if a similar difference is found then a new thread is started to make a LTM of this event.

Make Memory

The MakeMemory thread takes the 2 times in ΔSTM (now and then) where there's been a similar difference and tries to generalize this event into a long-term memory. A long-term memory contains the similar difference that caused this memory to be created, and a list of the prior events that were similar in both cases.

Remember

A Remember thread is started when the FindSimilarDiff thread sees that the current difference in the world matches one of the prior events for a LTM. When a LTM is triggered, the reference similar difference for this LTM is predicted to happen in the future. This thread makes the prediction and waits and then checks to see if the predicted difference was seen. The particular prior event that triggered this LTM is then credited with being right or wrong.

Ponder

The Ponder thread is the final thread of activity and is a meta reasoning facility. The main goal of this thread is to try to improve long-term memory. Currently there is one type of improvement partially implemented, but one could imagine a number of meta reasoning algorithms: putting LTMs together that get activated simultaneously, generalizing on particular slots of a LTM (i.e. combining the LTM for eatdrink<chocolate> and eatdrink<vanilla> to eatdrink<object in view>), etc. The current meta reasoning implementation has to do with "cleaning-up" the list of prior events that trigger a LTM. Since statistics are kept on how well each of the priors predicts, the Ponder thread will be able to go through and prune away priors that consistently fail to predict the LTM's event.

RESULTS

This is the first implementation of ThoughtStreams, which is certainly a work in progress, but the initial results are promising. This section details an example taken from an actual run of the game, inferences the system made, and a discussion of how this performance could be improved.

Table 1, gives the elements of STM and ΔSTM from run of the game. The interactions in this example include: go into Starbucks, say hi to the cashier, read the menu, order something, sit at a table, drink it, get up, and then leave. go into the gap, but decide to leave right away.

DISCUSSION

In this particular slice of the game the system made 23 predictions, 11 of which ended up being correct. This slice of the simulation was randomly selected from near the end of a 500-event run of the game. A more elaborate analysis is necessary to determine true failure rates of the ThoughtStreams inferences.

Some examples of successful predictions:

- At time, 2, it expected to see 'readmenu<>' when it saw 'enter<Starbucks>'.
- And at time, 8, it expected to see 'eatdrink<X>' when it saw 'order<latte>'.
- My favorite *successful* example though, is that the system noticed that I forgot to buy my latte! At time, 12, when it saw eatdrink<latte> it expected to later see 'buy<X>'…but did not.

The inference failures fall into two basic categories. In a few of the failures, the remember facility needed to wait a little longer to see the given effect. Currently it is hard coded to wait for 10 events and then give up, but this is certainly an aspect of the design to revisit. The second failure type is more complicated and has to do with resolution of generalization. The problem is deciding how to distinguish the really false predictions from the 'kinda'

	STM					ΔSTM			
T	Place	Actors	Ojbects	Effect		ΔPlace	ΔActors	ΔObjects	ΔEffects
1	'Storefro nt'					'Storefront'			
2	'Storefront'			'enter<Starbucks>'					'enter<Starbucks>'
3	'Starbucks'	Cashier	Table, Menu			'Starbucks'	Cashier	Table, Menu	
4	'Starbucks'	Cashier	Table, Menu	'sayHi<Cashier>					'sayHi<Cashier>
5	'Starbucks'	Cashier	Table, M enu	'Cashier says hi!'					'Cashier says hi!'
6	'Starbucks'	Cashier	Table, Menu	'readmenu<>'					'readmenu<>'
7	'Starbucks'	Cashier	Table, Menu	'menu says latte, mocha, coffee'					'menu says latte, mocha, coffee'
8	'Starbucks'	Cashier	Table, Menu	'orde r<latte>'					'order<latte>'
9	'Starbucks'	Cashier	Table, Menu, latte					latte	
10	'Starbucks'	Cashier	Table, Menu, latte	'sitattable<>'					'sitattable<>
11	'Starbucks'	Cashier	Table, Menu, latte	'now you're sitting'					'now you're sitting'
12	'Star bucks'	Cashier	Table, Menu, latte	'eatdrink<latte>'					'eatdrink<latte>'
13	'Starbucks'	Cashier	Table, Menu					~latte	
14	'Starbucks'	Cashier	Table, Menu	'getup<>'					'getup<>'
15	'Starbucks'	Cashier	Table, Menu	'now you're not sitting'					'now yo u're not sitting'
16	'Starbucks'	Cashier	Table, Menu	'leave<Starbucks>'					'leave<Starbucks>'
17	'Storefront'					'Storefront'	~Cashier	~Table, ~Menu	
18	'Storefront'			'enter<The Gap>'					'enter<The Gap>'
19	'The Gap'	Attendant, Cashier		'Attendan t says welcome to the gap'		'The Gap'	Attendant, Cashier		'Attendant says welcome to the gap'
20	'The Gap'	Attendant, Cashier		'leave<The Gap>'					'leave<The Gap>'
21	'Storefront'					'Storefront'	~Attendant, ~Cashier		

Table 1: 20 event window of game log

false predictions. For example, if the prediction said – I expect to see the actors <ashier and attendant> disappear – and later its sees a change where these actors do disappear … but also the place changes. This should probably still be considered a case of successful prediction, or a 'kinda false prediction'.

Another conclusion I have drawn from analyzing the resulting predictions, is that predictions that consistently co-occur *and* are right and wrong in the same cases should then be collapsed into one memory. Moreover, I believe that this mechanism can be used to solve the second failure problem mentioned above. The root of the problem is that currently the system is trying to talk about too many things at the same time. It's making predictions about Place/Actor/Object/Effects concurrently. So, I propose that if each of these types is predicted separately, the individual predictions can be judged and the most consistent co-occurring changes would then be gathered together in a single memory.

FUTURE WORK

I have two directions I am interested in taking this work. I plan to explore both over the next few months.

1. An obvious next step is to keep improving ThoughtStreams and work towards testing it in a more elaborate simulation environment. Two options for this include: modifying the SIMs to produce a log file with more details about all of the interactions in the world, or using the UnrealTournament platform to build our own reality game. This richer context will lead to more proof of this approach and cross context learning. It can also be deployed to a wider user population to increase the amount of data collection.
2. A more ambitious direction that I'd like to take this work is thinking about how both the knowledge

acquired in a simulated environment and the learning process itself can be leveraged outside of that environment. If this process of learning could generalize to any system where there is a world view over time and a facility for calculating similarity and difference of events, this could be a really useful architecture in the realm of context-aware computing applications, allowing us to create systems with adaptive context formation and context switching.

CONCLUSIONS

There have been are three major attempts to give common sense knowledge to a computer: CYC, OpenMind, and ThoughtTreasure. A major limitation with all of these is the lack of context around the learning/input of the knowledge. In this paper, I have presented simulation as an alternative to the database solution of commonsense knowledge acquisition. This position stems from the fact that commonsense is essentially redundant shared experience over time, and posits that reality computer games could be a forum in which computers could gain this shared experience from a human user. The learning program, ThoughtStreams, was built and shown to have some initial success in predicting commonsensical future events based on past experiences in a simple simulation game environment. In a 20 event window of the program's output, taken after about 500 events of the game, ThoughtStreams was able to generate 23 commonsense statements 11 of which were correct. The correct elements were obtained as expected through the aggregation of redundant and complementary events. Failures fell into two categories, not having enough (or the right amount of) memory time, and over redundant or simultaneous analysis of the multiple elements (place, actor, object, effects). Both of these failure modes are believed to be reduced or corrected in future work which will include an expansion of the system to more elaborate simulation environments. In addition to contributing to the commonsense AI problem, a generalization of this work should make contributions toward understanding context aware computing in terms of context formation and context switching.

REFERENCES

1. D. Lenat, *CYC: A Large-Scale Investment in Knowledge Infrastructure*, Communications of the ACM, Nov 1995, Vol. 38, No. 11.
2. H. Lieberman, *Your Wish is my Command: Programming by Example*, Morgan Kaufmann, San Francisco, 2001.
3. J. McCarthy, *Artificial Intelligence, Logic and Formalizing Common Sense, P*hilosophical Logic and Artificial Intelligence, edited by Richmond Thomason, Dordrecht ; Kluwer Academic, 1989.
4. M. Minsky, *The Emotion Machine*, book draft, Ch. 6.
5. M. Minsky, *Commonsense-Based Interfaces*, Communications of the ACM, Aug 200, Vol. 43, No. 8.
6. S. Pinker, *The Language Instinct*, HarperCollins, NY, 1995.
7. P. Singh, et. al., Open Mind Common Sense: Knowledge Acquisition from the General Public, AAAI, 2002.
8. The SIMs, www.thesims.com.