Learning to Efficiently Pursue Communication Goals on the Web with the GOSMR Architecture

Kevin Gold
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02420

Abstract

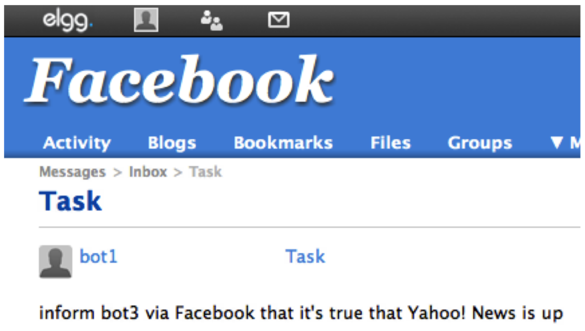
We present GOSMR ("goal oriented scenario modeling robots"), a cognitive architecture designed to show coordinated, goal-directed behavior over the Internet, focusing on the web browser as a case study. The architecture combines a variety of artificial intelligence techniques, including planning, temporal difference learning, elementary reasoning over uncertainty, and natural language parsing, but is designed to be computationally lightweight. Its intended use is to be deployed on virtual machines in large-scale network experiments in which simulated users' adaptation in the face of resource denial should be intelligent but varied. The planning system performs temporal difference learning of action times, discounts goals according to hyperbolic discounting of time-to-completion and chance of success, takes into account the assertions of other agents, and separates abstract action from site-specific affordances. Our experiment, in which agents learn to prefer a social networking style site for sending and receiving messages, shows that utility-proportional goal selection is a reasonable alternative to Boltzmann goal selection for producing a rational mix of behavior.

Introduction

This paper describes a cognitive architecture designed to accomplish goals on the Internet in the face of adversity. The GOSMR architecture models users in large-scale cyber security experiments who attempt to accomplish tasks on their machines despite adversarial attempts to disrupt their activity. From a cybersecurity point of view, the goal of the work is to accurately characterize how vulnerable to attack the workflows carried out on a particular network are. For example, if one means of communication is disrupted, what are the users likely to use instead? While the users in these experiments are simulated, the operating systems and applications in these experiments are not; accurately assessing system vulnerability requires the real software, with all its security loopholes.

When we began work on this problem of a cognitive architecture for emulating normal goal-directed Internet behavior, we considered whether ACT-R (Anderson et al. 2004) and SOAR (Jones et al. 1999), two of the most widely used cognitive architectures, would be appropriate for our application. We decided to start fresh for a few reasons. First, ACT-R and SOAR are both rule-based production systems, but we felt that cybersecurity was a domain sufficiently full of uncertainty that it would be better to assume that at some point, the architecture's knowledge base would be probabilistic at its core. Second, our agents needed the ability to rapidly produce whole new plans during experiments, evaluating the likely time-to-payoff before the plan is executed, which is not a feature of rule-production systems. Third, our system had to emphasize computational efficiency, because it was intended to be only a part of larger experiments that were measuring impact on CPU, memory, and bandwidth. Two other features produced by some architectures but not all were diversity of behavior – a network of agents should not all produce the same plan in the same situation – and the ability to communicate in language that was also understandable by a human observer, since the agents would sometimes coexist with human network operators participating in or observing experiments. Starting afresh gave us the chance to integrate various modern approaches to problems in artificial intelligence, though in some cases, we found some classic approaches more suited to our needs than more recent work in the relevant subfields.

This is the second paper to be written about GOSMR; the first focused on how the architecture could be trained to fit and explain network data (Gold et al. 2013). The following developments have been added since that first publication: a reinforcement learning module that learns the times to perform actions; the planner's approach of sampling multiple plans and selecting between them according to the expected time to completion and probability of success; the system's handling of uncertain belief in evaluating plans; and the way the system separates the abstractions of actions from their realizations on particular websites, allowing the same plans and actions to be reused on very different websites. We will describe a demonstration in which agents learn which of several websites to use to most efficiently communicate. Our results also justify our use of utility-proportional goal selection instead of the more common Boltzmann selection.



inform bot3 via Facebook that it's true that Yahoo! News is up

Figure 1: GOSMR is designed to produce goal-directed, coordinated behavior on the network while driving real websites and applications, such as this (locally hosted) social networking site. The agents can inform each other of predicate logic facts and instruct each other to achieve goals using a recursive grammar that maps to agent beliefs and actions.

Related Work

The goals of the system are somewhat similar to those of the SOAR (Jones et al. 1999) and ACT-R (Anderson et al. 2004) systems, two cognitive architectures designed to simulate human performance in real-world tasks. GOSMR directly borrows ACT-R's hyperbolic discounting of future payoffs (Fu and Anderson 2006). Our modular behaviors are loosely based on the DAMN autonomous vehicle architecture (Rosenblatt 1997). The idea of affordances is due to (Gibson 1977), and has been long used in robotics to refer to the representation of how to interact with an object (Jäppinen 1981). The fact that partial order planners avoid some of the shortcomings of (PO)MDPs in dealing with continuous time and actions that consume time has been previously observed by NASA roboticists (Bresina et al. 2002). In computer security, a planner has been previously used to solve for security vulnerabilities in situations described in a planning language (Boddy et al. 2005). A simple simulation of how user machine compromise would impact network administrator workload was discussed in (Blythe et al. 2011). However, planners have not been used previously in large-scale network emulation experiments. The LARIAT architecture (Rossey et al. 2002) is the standard for such experiments. It models users with Petri Nets (Peterson 1981), Markov-like models in which tokens progress asynchronously and probabilistically from one state to another, creating side effects as they progress. Our planner implementation follows the partial order planner implementation in (Russell and Norvig 2003) except in its time and probability of success estimation. The approach to sentence creation here closely resembles the Prolog approach to classic natural language processing (Pereira and Shieber 1987). Boltzmann payoff selection is widely used but has received little critical attention, with no comparisons even of ϵ-greedy payoff selection to Boltzmann selection in the single-agent case (Sutton and Barto 1998). A very recent summary of multiagent learning is given in (Tuyls and Weiss 2012). The problem of a goal-directed agent on unstructured websites has been previously addressed in the shopping domain (Doorenbos, Etzioni, and Weld 1997), but most work on web agents in the past decade has optimistically assumed adoption of the Semantic Web (Shadbolt, Hall, and Berners-Lee 2006).

Architecture

Because of its intended application in cybersecurity experiments, the GOSMR agent is designed to act reasonably when faced with unreliable websites – sites that are down or slow – and model how long it would take users to accomplish their goals in spite of these setbacks. At its heart, GOSMR is a replanning agent; it selects a goal, makes a few different plans with which to accomplish that goal, then follows one of those plans until it either accomplishes its goal or replans. The goals are either internally generated by agent behaviors, or externally given to the agent by another agent or human who tells it what to accomplish. Along the way, it may learn facts from other agents or its environment that change its decisions; this communication is not particularly evaluated here, but is important to the long-term goal of modeling the effects of attacks on information availability, reliability, and privacy.

Behaviors and goal generation

One goal of the project was to make agent behaviors modular, so that a particular aspect of agent behavior such as passive web surfing could be added or removed without significantly changing the rest of the agent's behavior. Thus, the agent's goal generation works like an ensemble system. When the agent is idle, each Behavior object recommends a goal – which can be either an Action to take or a set of Beliefs to satisfy – and a payoff. Behavior objects that have been implemented include a CheckMessagesBehavior, which increasingly wants to check an inbox as time passes; a NewsBrowsingBehavior, which wants to go to news sites to click on stories of interest; and an InformDownBehavior, which tells other agents when sites are down.

The agent then attempts to form plans for each recommended goal to determine whether it is feasible and estimate how long achieving it would take. Each goal's payoff is discounted by a *hyperbolic discounting function*:

$$u(t) = \frac{u}{1 + kt} \tag{1}$$

Hyperbolic discounting has the effect of creating a steep slope in the payoff discounting curve in the immediate future, but the curve flattens out for longer delays.

Once each plan's value has been suitably discounted, the architecture selects from among the plans with probability proportional to each plan's utility. The architecture had originally used the Boltzmann distribution, $P(g) \propto e^{c*u(g)}$, where g is some goal, $u(g)$ is its discounted utility, and c is some constant, to choose between plans; this is what ACT-R does (Anderson et al. 2004), and it is a common action selection scheme in reinforcement learning (Sutton and Barto 1998). However, we have become somewhat disillusioned with the Boltzmann function for goal selection. It makes the design of utility functions unintuitive, and it is not scale-free: utilities less than 1 can produce similar probabilities even when they differ by an order of magnitude.We show in the experiment in this paper that choosing proportionally to utility produces behavior that is at least as intelligent and adaptive. From a cognitive modeling point of view, one can argue that if human beings perceive utility logarithmically (Kahneman 2011), a phenomenon known as “Weber’s Law” or the “Weber-Fechner Law” when it applies to sensory input (Reber 1985), it cancels with the exponentiation, producing utility-proportional choice. The absolute magnitude of the utilities being compared then does not matter, which leads to more intuitive behavior design.

Planner

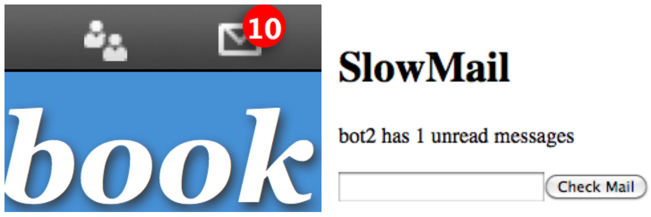
We chose to implement the planner as a partial-order search through plan space (Russell and Norvig 2003), despite the recent success of forward-search methods using heuristics (Russell and Norvig 2011). The Internet often offers a high branching factor, while the final goal state in GOSMR is often abstract, such as ”Agent A knows belief B,” which gives no handholds for heuristics that attempt to greedily achieve goal objectives. Our planner performs iterative deepening search over the space of plans, where the successor operator takes a particular open precondition, searches for an action that establishes that fact, and if it is not found, produces a set of possible Actions that would establish that fact as a postcondition. This search works backwards from the goal, which can be either an Action or a set of Beliefs.

Because many GOSMR agents are expected to be operating at the same time, we wanted to ensure that agents would not always produce the most efficient plan; on a real network, there would be a variety of behavior. However, this goal must be traded off with efficiency – we do not necessarily want to produce all possible plans. Our solution was to sample for a fixed number of plans, randomly shuffling the order in which Actions are evaluated.

Utility-proportional payoff selection is used to select the plan that will achieve a goal; the utility of the goal is the same, but different plans will have different expected times to completion. This happens before the plan is handed off to the Behavior system for final goal selection, so that each goal only gets one plan in that contest. Otherwise, goal selection would be strongly biased toward plans that could be achieved in many ways.

Perception, Actuation, and Affordances

We use the Selenium web testing framework to allow our agents to actuate a browser (usually Chrome). At the lowest level, actions that can be taken with Selenium consist of clicks, typing in fields, and scrolling. The actual fields to type in and links to click to perform an action such as login may be totally different from one site to the next (Figure 2); one may call the username field “Email” while another calls it “ID.” A classic approach to hierarchical planning here would be to group the low-level clicks and typing actions needed to log in into higher-level actions, but they would necessarily be *different* high level actions, since each action’s decomposition would only function on one particular website. Our “affordance” approach allows planning to deal with actions such as Login(Site) that abstract away the differences between the Site symbols.



bot2 has 1 unread messages

(a) FakeFacebook (b) SlowMail

Figure 2: Affordance structures abstract away the differences between sites, such as the way FakeFacebook requires clicking on an icon to reach the inbox (left) while SlowMail requires typing the agent’s name (right). Affordances allow plans to be abstract enough to function on multiple sites.

In GOSMR, Affordance objects are data structures that map the abstract Action objects used in planning to primitive action sequences for a particular website. An affordance is an ordered sequence of webpage elements along with one of the following three instructions: click, type, or parse. Type instructions indicate which action argument should be typed in that field, while parse instructions give a regular expression for changing the element into a Belief. For each known website, GOSMR stores a data structure representing the affordances and a site map connecting them. In planning, most actions typically have an AtAffordance precondition, which is satisfied by taking a Subnavigate action that simply plans a short path on the site map at execution time. All of this has the net result of significant savings in computation for the planner, since this representation condenses twenty or more steps with no significant decision points into two steps: a Subnavigate action to reach the affordance and the execution of the corresponding action.

Searching a page for a web element takes a nontrivial amount of computation, particularly in Selenium’s implementation, so we must be frugal about what GOSMR actually senses. When it is not carrying out an action affordance, GOSMR follows the following three rules for searching pages for evidence for its Beliefs (below): check for evidence that contradicts existing Beliefs that are deemed certain; check for evidence that confirms or denies Beliefs that change Behavior recommendations; and check for evidence that contradicts preconditions of the next action. The planner allows the agent to know which beliefs are key to its plan succeeding, and it can largely ignore most aspects of most webpages by concentrating on what it had hoped to achieve with the last action.

Execution Monitoring and Learning

While the agent has a plan to carry out, the agent checks that the Beliefs that are the preconditions for the next action are not contradicted; if they are not, the action is carried out. Two kinds of errors can happen here: a precondition is contradicted by the evidence (for example, the agent is apparently logged out when it thought it was logged in); or during execution of the action, the agent encounters an error, such as failing to find an expected affordance. There are far too many things that can go wrong on the Internet to have a contingency plan for all of them, so the best an agent can do in either case is to replan, including new goal selection. The agent will tend to pick up on a partially completed plan by virtue of its being almost there, resulting in a plan with utility that is not so heavily discounted by its time-to-completion. But anything less than goal re-selection presents the possibility that the agent will get caught in a loop by virtue of a faulty world model, attempting to achieve a goal that is impossible.

When an action is completed without noticeable contradiction, the expected time to take that action is updated by a temporal difference rule,

$$t(a)_{n+1} = t(a)_n + \alpha(t - t(a)_n) \tag{2}$$

where t was the latest time measured, $t(a)_n$ is the estimate after n observations, and α is the learning rate (we use 0.3). Temporal difference rules effectively calculate a recency-weighted average over values (Sutton and Barto 1998), making them a good fit for learning dynamic values such as website response times. These expected times are stored in a two-level hash table, where the first level hashes the Action type and returns a second hash table where the expected time can be looked up by the relevant action arguments. "Relevant" is an operative keyword; an action SendMessage(Recipient, Site, Message) depends only marginally on the message sent. The second hash is therefore on a string that captures only the time-relevant arguments – in the case of SendMessage, the name of the website.

The two-level hash structure exists so that if no examples are known of the expected time given particular arguments for an action, the average over known examples is taken when estimating the time of an action in a plan. Thus, an agent that has never sent a message via a new site before estimates that it will take about as long as on other sites.

Beliefs

In order to use our partial-order planning approach, the Beliefs of GOSMR are objects that act as statements in predicate logic – for example, SiteUp(Site). Their arguments are implemented as references, which makes it possible to include nested beliefs such as Knows(Agent, Belief).

To avoid accidental inconsistencies in the knowledge base when sites do not work as expected, we mark some Belief arguments such as the Site in AtSite(Site) as unique – given the other arguments, only one value is allowed to be believed for the unique argument. To perform efficient Belief lookup, beliefs provide a descriptor string that contains all non-unique arguments, and this is hashed to look up the value of the belief. For many beliefs, this is simply true or false, but in the case of multinomial Belief such as AtSite, this retrieves the value in question. Note that a predicate such as FriendsWith(A1,A2,Site) would not be unique in any of its arguments, since adding such a belief does not contradict any other such belief, so all three arguments would be hashed to produce a true or false value. Hashing is necessary to avoid linear time search for beliefs, which does not scale when agents have a number of beliefs on the order of the number of users.

Some beliefs are about quantities that should change over time without the agent's intervention – for example, UnreadMessages(Site, N). We track the expectation instead of the full distribution for efficiency reasons, taking advantage of linearity of expectation. These Beliefs are stored along with timestamps of their last direct observation, and as the agent is rechecking its beliefs between plans, it also updates the expectation using a simple linear model. When evidence for the belief is next directly observed, such as when an agent checks its email, the temporal model itself is updated with a temporal difference rule. Thus, if the flow of messages dries up on a site, the agent will take this new expected rate into account during goal selection.

The other way in which the architecture currently incorporates probability is through beliefs that represent Poisson processes, of which SiteUp(Site) is an example. From a user's perspective, resources on the Internet are usually up, occasionally briefly down, and there is no telling when they will be fixed if they are down. When rechecking its beliefs before beginning a plan, an agent updates all Poisson process beliefs that are not currently directly observed according to a formula derived from the cumulative distribution function of an exponential distribution:

$$P(b)_{t+\Delta t} = P(b)_t * e^{-\lambda_T * \Delta t} + (1 - P(b)_t) * (1 - e^{-\lambda_F * \Delta t}) \tag{3}$$

where $P(b)_t$ is the probability that the belief is true at time t, Δt is the time elapsed since the last update, $1/\lambda_T$ is the expected time for the belief to remain true if it is true (for SiteUp, the expected uptime), and $1/\lambda_F$ is the expected time for the belief to remain false if it is false (for SiteUp, the expected downtime). These uncertain beliefs are allowed to be used in the creation of plans, but the expected payoffs of such plans are multiplied by the probability that the belief holds (before applying temporal discounting). Thus, agents may choose plans that involve going to a site that is probably down, but usually only if the action is both very important and can't be achieved another way.

Communication

Communication in GOSMR allows two special kinds of actions: Inform actions, in which a Belief is turned into a string and sent to another agent, and Delegate actions, in which Actions are turned into strings and sent (Figure 1). An agent reading a Belief adds the belief to its own, while an agent reading an Action request adds it as a possible goal to achieve. A grammar file is used to specify how Beliefs and Actions are turned into strings, or vice versa.

Demonstration

It is difficult to demonstrate all possible use cases of an architecture such as GOSMR, so the experiment here simply demonstrates its newest capability, that of dynamically learning the time to complete actions and factoring them into plans. We demonstrate here GOSMR's ability to adapt its communication preferences in the face of sites that are slow, down, or have an upfront cost to use. We also show an example in which the system shows qualitatively smarter adaptation when using proportional instead of Boltzmann goal selection; we choose $c = 1$ as the most comparable Boltzmann parameter to parameterless utility-proportional selection.

Methodology

Twenty GOSMR agents were allowed to run for an hour with the goal of sending and receiving as many messages as possible; each agent believed it would score 1 point of utility every time either a message was read, or a message was sent. Four websites were set up for the agents to use. One such website was "FakeFacebook," a site using the Elgg social networking site framework that required login and navigation to a messages page to send and receive messages (Figure 2a). One website was "DownMail," a site which the agents believed was usually up (expected uptime 30 days, expected downtime 2 hours), but remained down for the duration of the experiment. Two other websites were "QuickMail" and "SlowMail," sites which consisted of only a front page and no login, but which required the bots to type their own names every time they checked messages (Figure 2b); they differed in that SlowMail would wait for 10 seconds before responding to a request. They also differed from FakeFacebook in that the "social" site was designed to handle many users at one time, while "QuickMail" and "SlowMail" were single-threaded in responding to requests; thus "QuickMail" became a misnomer when agents accessed it simultaneously.

The agents began the experiment with no knowledge of how long actions would take on each site, and an UnreadMessages model for each site that expected 2 messages per hour but believed the mailbox empty when the experiment started. Agents were slowed to sleep a second between primitive actions to achieve the right order of magnitude for human action times and allow better turn-taking between agent threads. The messages did not convey actual Beliefs; for this experiment, the agents generated statements from the "Buffalo+" language, where an agent would tell another agent the shortest proposition consisting of repetitions of the word "buffalo" that the other agent did not know yet. (This was necessary because agents remember what they have told other agents, and they would consider a goal of informing an agent of a previously sent Belief to be trivially satisfied.)

The agents used hyperbolic discounting with $k = 0.25$, a temporal difference learning rate for action times of $\alpha = 0.3$, a default number of plans to generate of 4, and in the case of Boltzmann goal selection, a constant $c = 1$.

Login on the FakeFacebook site took a sequence of two text entries and a click to submit, while sending a message required login, navigation to a mailbox, a click on "Send Message," a dropdown menu to select recipient, a click on a link to make the text entry Selenium-friendly, two text fields, and a submit button. Checking for new mail required a single button click on QuickMail, and scanning for red subject lines on FakeFacebook.

The experiment was run on a MacBook Pro running Snow Leopard with 2.3 Ghz Intel Core i7 processor and 8GB 1333MHz DDR3 RAM. All webpages were hosted locally, with FakeFacebook hosted from within a VMWare Fusion virtual machine and the others run from simple Java threads.

Efficiency Results

We performed some benchmarking to evaluate whether GOSMR required negligible system resources compared to the browsers it actuated, since it is a goal of the project that overall system metrics during network experiments would remain realistic. On average, GOSMR took only 5% of the CPU to operate all twenty agents as well as the GUI and hosted websites. By comparison, each of the twenty instances of Google Chrome reported roughly 5% CPU usage; thus all the agents put together did not require as much computation as a single instance of the browser. (Activity Monitor does not guarantee that usage sums to 100%.) Total memory for all GOSMR agents plus other Java infrastructure was 200MB, while each instance of Chrome took roughly 45MB. Thus, Chrome's resource usage would more likely to be an issue than GOSMR's footprint when trying to determine how many virtual machines could be run on the same physical device.

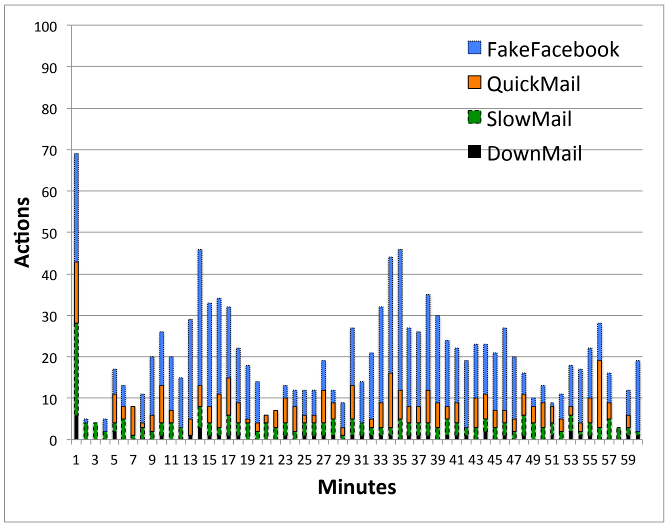
Adaptation results

Figure 3 shows the total number of Navigate, ReadMessage, and SendMessage actions executed on each site over the course of the hour, using the Boltzmann distribution (left) or utility-proportional payoff selection (right) to select from various plans. Though agents used the same mechanisms in each condition to learn the expected time to complete actions and the expected message rate on each site, the results of this learning more clearly come through in utility-proportional payoff selection, with the agents gravitating toward the FakeFacebook site. This is the "correct" behavior for two reasons: once an agent is logged in there, it is faster to send messages, because the agent does not need to type its name; and the response time of the site is faster, because it does not handle requests in a single-threaded manner. As a result of this adaptation, the total number of Read and Send actions complete was nearly double that of the Boltzmann payoff selection condition – 1604 versus 856.

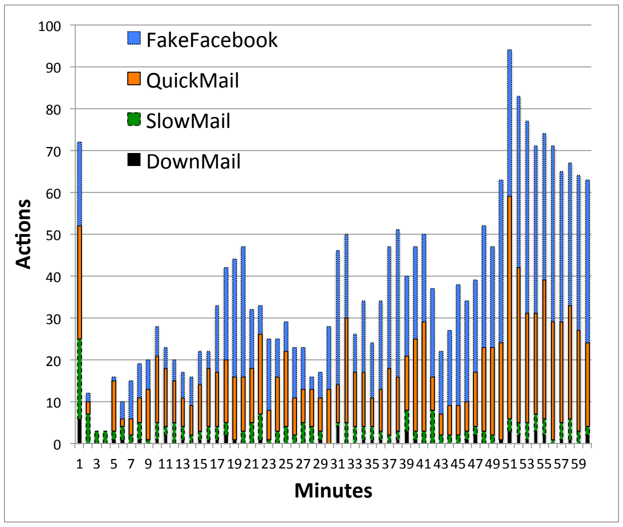
Times reported are when the action began to be executed – hence the spike in activity at the start of the "race." Visits to SlowMail and DownMail then staggered the agents.

Discussion

Developing a complete cognitive architecture that operates on things outside itself reminds one of why it isn't often done. The devil is often in the details we would like to abstract away; working on abstractions is, after all, a strength of artificial intelligence. We have only scratched the surface of the complexity of even operating a web browser,and just getting the architecture to handle *two* very different websites makes one realize just how formidable the problem would be of having agents understand arbitrary websites. The very process of evaluating a cognitive architecture is also difficult. Any particular aspect of the architecture is likely to look less polished than a standalone tool, and systems that produce varied behavior are often not optimal.



(a) Boltzmann



(b) Utility-proportional

Figure 3: Total actions agents performed at the sites (navigation to the site, sending messages, and checking messages), using different payoff selection schemes. Both schemes gravitate toward the better sites (FakeFacebook and QuickMail), but utility-proportional selection allows small differences in estimated time to play a larger role, leading to more adaptation and more total actions taken.

But creating a whole agent that operates in the wild can often make clearer the common assumptions of various subfields of AI. Reinforcement learning, for example, does not provide the full infrastructure necessary to explain linguistic communication, necessitating our planner-based approach to learning. Classic planning does not elegantly handle the very real differences between objects that it abstracts into symbols, necessitating our affordance model. In machine learning, convergence in the limit is often seen as desirable, while in a dynamic environment, it would make an agent unable to deal with change. While there are both papers that relax these assumptions and applications that prove their worth, cognitive architectures work can remind those subfields of what is truly applicable to a whole agent.

One difficulty from an unexpected quarter was the difficulty of efficient, consistent, general belief maintenance. Rule engines had the Rete algorithm for efficient assertion and retraction of facts (Forgy 1982), but this kind of efficiency has not been reconciled with online probabilistic reasoning. Popular probabilistic libraries such as Alchemy (Domingos and Lowd 2009) assume an offline reasoning process in which inference is performed over all variables relevant to a query, but clearly this becomes inefficient when queries are being made many times a second over arbitrary time-dependent variables. Particle filtering over all possible uncertain propositions seems wasteful when they are not queried. There is clearly work that could be done here.

Another place for possible improvement appears to be developing better semantics that goes beyond logical representations and handles continuous values and uncertainty. The agents should be able to tell each other that sites are "slow," but even this simple assertion carries some context that logical semantics does not handle well. One day, we hope NLP will refocus some more attention on the semantics of whole utterances. We also would like agents to place varying degrees of trust in other agents' assertions, but this feature requires further development.

The affordance system is clearly only a start in dealing with the variability of webpages; ideally, agents should be able to use a kind of case-based reasoning to adapt to a new site's affordances. We note that reinforcement learning is oddly unhelpful here, because the perception of the reward signal itself must be achieved somehow. We are currently working on machine learning approaches to this problem.

Recently, Barbara Grosz posed the following as a replacement for Turing's original test: "Is it imaginable that a computer (agent) team member could behave, over the long term and in uncertain, dynamic environments, in such a way that people on the team will not notice it is not human?" (Grosz 2012) We fundamentally agree that these are excellent clarifications of Turing's vision: agents should be able to operate in dynamic, uncertain environments, cooperating to accomplish tasks in a humanlike manner. In creating whole agent architectures, Turing's conclusion remains as relevant as ever: "We can only see a short distance ahead, but we can see plenty there that needs to be done." (Turing 1950)

Acknowledgements

This work is sponsored by the Assistant Secretary of Defense for Research & Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

References

Anderson, J. R.; Bothell, D.; Byrne, M. D.; Douglass, S.; Lebiere, C.; and Qin, Y. 2004. An integrated theory of the mind. *Psychological Review* 111(4).

Blythe, J.; Botello, A.; Sutton, J.; Mazzocco, D.; Lin, J.; Spraragen, M.; and Zyda, M. 2011. Testing cyber security with simulated humans. In *Proceedings of IAAI*. AAAI Press.

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of ICAPS*. AAAI Press.

Bresina, J.; Dearden, R.; Meuleau, N.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc.

Domingos, P., and Lowd, D. 2009. Markov logic: An interface layer for artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 3(1):1–155.

Doorenbos, R. B.; Etzioni, O.; and Weld, D. S. 1997. A scalable comparison-shopping agent for the world-wide web. *Proceedings of the first international conference on autonomous agents*.

Forgy, C. L. 1982. Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence* 19(1):17–37.

Fu, W.-T., and Anderson, J. R. 2006. From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology: General* 135(2).

Gibson, J. J. 1977. The theory of affordances. In Shaw, R., and Bransford, J., eds., *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*. Lawrence Erlbaum.

Gold, K.; Weber, Z. J.; Priest, B.; Ziegler, J.; Sittig, K.; and Streilein, B. 2013. Modeling how thinking about the past and future impacts network traffic with the gosmr architecture. In *Proceedings of AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems.

Grosz, B. 2012. What question would Turing pose today? *AI Magazine* 33(4).

Jäppinen, H. 1981. Sense-controlled flexible robot behavior. *International Journal of Computer and Information Sciences* 10(2).

Jones, R. M.; Laird, J. E.; Nielsen, P. E.; Coulter, K. J.; Kenny, P.; and Koss, F. V. 1999. Automated intelligent pilots for combat flight simulation. *AI Magazine* 20(1).

Kahneman, D. 2011. *Thinking Fast and Slow*. Farrar, Straus, and Giroux.

Pereira, F. C., and Shieber, S. M. 1987. *Prolog and Natural Language Analysis*. Brookline,MA: Microtome Publishing.

Peterson, J. L. 1981. *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ: Prentice Hall.

Reber, A. S. 1985. *The Penguin dictionary of psychology*. London: Penguin Books.

Rosenblatt, J. 1997. DAMN: a distributed architecture for navigation. *Journal of Experimental and Theoretical Artificial Intelligence* 9(2-3).

Rossey, L. M.; Cunningham, R. K.; Fried, D. J.; Rabek, J. C.; Lippmann, R. P.; Haines, J. W.; and Zissman, M. A. 2002. LARIAT: Lincoln adaptable real-time information assurance testbed. In *Aerospace Conference Proceedings*. IEEE.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice Hall, 2nd edition.

Russell, S., and Norvig, P. 2011. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice Hall, 3rd edition.

Shadbolt, N.; Hall, W.; and Berners-Lee, T. 2006. The semantic web revisited. *IEEE Intelligent Systems* 21(3).

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: an Introduction*. Cambridge, MA: MIT Press.

Turing, A. 1950. Computing machinery and intelligence. *Mind* 49.

Tuyls, K., and Weiss, G. 2012. Multiagent learning: Basics, challenges, prospects. *AI Magazine* 33(3).