Economic Attention Networks: Associative Memory and Resource Allocation for General Intelligence

Matthew Ikle', Joel Pitt, Ben Goertzel, George Sellman

Adams State College (ASC), Singularity Institute for AI (SIAI), Novamente LLC and SIAI, ASC
1405 Bernerd Place, Rockville MD 20851, USA
ben@goertzel.org, stephan@bugaj.com

Abstract

A novel method for simultaneously storing memories and allocating resources in AI systems is presented. The method, Economic Attention Networks (ECANs), bears some resemblance to the spread of activation in attractor neural networks, but differs via explicitly differentiating two kinds of "activation" (Short Term Importance, related to processor allocation; and Long Term Importance, related to memory allocation), and in using equations that are based on ideas from economics rather than approximative neural modeling. Here we explain the basic ideas of ECANs, and then investigate the functionality of ECANs as associative memories, via mathematical analysis and the reportage of experimental results obtained from the implementation of ECANs in the OpenCog integrative AGI system.

Introduction

One of the critical challenges confronting any system aimed at advanced general intelligence is the allocation of computational resources. The central nature of this issue is highlighted by Hutter's (2004) mathematical results showing that if one formalizes intelligence as the achievement of complex computable goals, then there are very simple software programs that can achieve arbitrarily high degrees of intelligence, so long as they are allotted huge amounts of computational resources. In this sense, coping with space and time limitations is the crux of the AGI problem.

Not surprisingly, given its central nature, the management of computational resources ties in with a variety of other concrete issues that AGI systems confront, in ways depending on the specific system in question. In the approach we will describe here, resource allocation is carried out by the same structures and dynamics as associative memory, whereas the relationship between resource allocation and other system processes like reasoning and procedure learning involves feedback between distinct software components.

We will describe here a specific approach to resource allocation and associative memory, which we call Economic Attention Networks or ECANs. ECANs have been designed and implemented within an integrative AGI framework called OpenCog (which overlaps with the related Novamente Cognition Engine system; see Goertzel, 2006). However, ECANs also have meaning outside the OpenCog context; they may be considered nonlinear dynamical systems in roughly the same family as attractor neural networks such as Hopfield nets (Amit, 1992). The main focus of this paper is the study of ECANs as associative memories, which involves mathematical and experimental analyses that are independent of the embedding of ECANs in OpenCog or other AGI systems. But we will also discuss the implications of these results for specific interactions between ECANs and other OpenCog components

Economic Attention Networks

First we summarize the essential ideas of ECANs; in later sections two specific variants of ECAN equational formalizations are presented.

An ECAN is a graph, consisting of un-typed nodes and links, and also links that may be typed either HebbianLink or InverseHebbianLink. It is also useful sometimes to consider ECANs that extend the traditional graph formalism and involve links that point to links as well as to nodes. The term Atom will be used to refer to either nodes or links. Each Atom in an ECAN is weighted with two numbers, called STI (short-term importance) and LTI (long-term importance). Each Hebbian or InverseHebbian link is weighted with a probability value.

The equations of an ECAN explain how the STI, LTI and Hebbian probability values get updated over time. The metaphor underlying these equations is the interpretation of STI and LTI values as (separate) artificial currencies. The motivation for this metaphor has been elaborated somewhat in (Goertzel, 2007) and will not be recapitulated here. The fact that STI (for instance) is a currency means that the total amount of STI in the system is conserved (except in unusual instances where the ECAN controller

decides to introduce inflation or deflation and explicitly manipulate the amount of currency in circulation), a fact that makes the dynamics of an ECAN dramatically different than that of, say, an attractor neural network (in which there is no law of conservation of activation).

Conceptually, the STI value of an Atom is interpreted to indicate the immediate urgency of the Atom to the ECAN at a certain point in time; whereas the LTI value of an Atom indicates the amount of value the ECAN perceives in the retention of the Atom in memory (RAM). An ECAN will often be coupled with a "Forgetting" process that removes low-LTI Atoms from memory according to certain heuristics.

STI and LTI values will generally vary continuously, but the ECAN equations we introduce below contain the notion of an AttentionalFocus (AF), consisting of those Atoms in the ECAN with the highest STI values. The AF is given its meaning by the existence of equations that treat Atoms with STI above a certain threshold differently.

Conceptually, the probability value of a HebbianLink from A to B is the odds that if A is in the AF, so is B; and correspondingly, the InverseHebbianLink from A to B is weighted with the odds that if A is in the AF, then B is not. A critical aspect of the ECAN equations is that Atoms periodically spread their STI and LTI to other Atoms that connect to them via Hebbian and InverseHebbianLinks; this is the ECAN analogue of activation spreading in neural networks.

Based on the strong correspondences, one could plausibly label ECANs as "Economic Neural Networks"; however we have chosen not to go that path, as ECANs are not intended as plausible neural models, but rather as nonlinear dynamical systems engineered to fulfill certain functions within non-brain-emulative AGI systems.

Integration into OpenCog and the NCE

The OpenCog AGI framework, within which the current ECAN implementation exists, is a complex framework with a complex underlying theory, and here we will only hint at some of its key aspects. OpenCog is an open-source software framework designed to support the construction of multiple AI systems; and the current main thrust of work within OpenCog is the implementation of a specific AGI design called OpenCogPrime (OCP), which is presented in the online wikibook (Goertzel, 2008). Much of the OpenCog software code, and many of the ideas in the OCP design, have derived from the open-sourcing of aspects of the proprietary Novamente Cognition Engine, which has been described extensively in previous publications.

The first key entity in the OpenCog software architecture is the AtomTable, which is a repository for weighted, labeled hypergraph nodes and hyperedges. In the OpenCog implementation of ECANs, the nodes and links involved in the ECAN are stored here. OpenCog also contains an object called the CogServer, which wraps up an AtomTable as well as (among other objects) a Scheduler that schedules a set of MindAgent objects that each (when allocated processor time by the Scheduler) carry out cognitive operations involving the AtomTable. The essence of the OCP design consists of a specific set of MindAgents designed to work together in a collaborative way in order to create a system that carries out actions oriented toward achieving goals (where goals are represented as specific nodes in the AtomTable, and actions are represented as Procedure objects indexed by Atoms in the AtomTable, and the utility of a procedure for achieving a goal is represented by a certain set of probabilistic logical links in the AtomTable, etc.). OpenCog is still at an experimental stage but has been used for such projects as statistical language analysis, probabilistic inference, and the control of virtual agents in online virtual worlds (see opencog.org).

So, in an OpenCog context, ECAN consists of a set of Atom types, and then a set of MindAgents carrying out ECAN operations such as HebbianLinkUpdating and ImportanceUpdating. OCP also requires many other MindAgents carrying out other cognitive processes such as probabilistic logical inference according to the PLN system (Goertzel et al, 2008) and evolutionary procedure learning according to the MOSES system (Looks, 2006). The interoperation of the ECAN MindAgents with these other MindAgents is a subtle issue that will be briefly discussed in the final section of the paper, but the crux is simple to understand.

The CogServer is understood to maintain a kind of central bank of STI and LTI funds. When a non-EAN MindAgent finds an Atom valuable, it sends that Atom a certain amount of Stimulus, which results in that Atom's STI and LTI values being increased (via equations to be presented below, that transfer STI and LTI funds from the CogServer to the Atoms in question). Then, the ECAN ImportanceUpdating MindAgent carries out multiple operations, including some that transfer STI and LTI funds from some Atoms back to the CogServer.

Definition and Analysis of Variant 1

We now define a specific set of equations in accordance with the ECAN conceptual framework described above. We define $\mathbf{H}_{STI} = [s_1, \cdots, s_n]$ to be the vector of STI values, and $\mathbf{C} = \begin{bmatrix} c_{11}, \cdots, c_{1n} \\ \vdots \quad \ddots \quad \vdots \\ c_{n1}, \cdots, c_{nn} \end{bmatrix}$ to be the connection matrix of Hebbian probability values, where it is assumed that the existence of a HebbianLink or InverseHebbianLink between A and B are mutually exclusive possibilities. We also define $\mathbf{C}_{LTI} = \begin{bmatrix} g_{11}, \cdots, g_{1n} \\ \vdots \quad \ddots \quad \vdots \\ g_{n1}, \cdots, g_{nn} \end{bmatrix}$ to be the matrix of LTI values for each of the corresponding links.

We assume an updating scheme in which, periodically, a number of Atoms are allocated Stimulus amounts, which

causes the corresponding STI values to change according to the equations

$$\forall i: \quad s_i = s_i - \text{rent} + \text{wages},$$

where rent and wages are given by

$$\text{rent} = \begin{cases} \langle \text{Rent} \rangle \cdot \max\left(0, \dfrac{\log\left(\dfrac{20 s_i}{\text{recentMaxSTI}}\right)}{2}\right), & \text{if } s_i > 0 \\ 0, & \text{if } s_i \le 0 \end{cases}$$

and

$$\text{wages} = \begin{cases} \dfrac{\langle \text{Wage} \rangle \langle \text{Stimulus} \rangle}{\sum_{i=1}^{n} p_i}, & \text{if } p_i = 1 \\ \dfrac{\langle \text{Wage} \rangle \langle \text{Stimulus} \rangle}{n - \sum_{i=1}^{n} p_i}, & \text{if } p_i = 0 \end{cases},$$

where $\mathbf{P} = [p_1, \cdots, p_n]$, with $p_i \in \{0,1\}$ is the cue pattern for the pattern that is to be retieved.

All quantities enclosed in angled brackets are system parameters, and LTI updating is accomplished using a completely analogous set of equations.

The changing STI values then cause updating of the connection matrix, according to the "conjunction" equations. First define

$$\text{norm}_i = \begin{cases} \dfrac{s_i}{\text{recentMaxSTI}}, & \text{if } s_i \ge 0 \\ \dfrac{s_i}{\text{recentMinSTI}}, & \text{if } s_i < 0 \end{cases}.$$

Next define

$$\text{conj} = \text{Conjunction}(s_i, s_j) = \text{norm}_i \times \text{norm}_j$$

and

$$c'_{ij} = \langle \text{ConjDecay} \rangle \, \text{conj} + (1 - \text{conj}) c_{ij}.$$

Finally update the matrix elements by setting

$$c_{ij} = \begin{cases} c_{ji} = c'_{ij}, & \text{if } c'_{ij} \ge 0 \\ c'_{ij}, & \text{if } c'_{ij} < 0 \end{cases}.$$

We are currently also experimenting with updating the connection matrix in accordance with the equations suggested by Storkey (1997, 1998, 1999.)

A key property of these equations is that both wages paid to, and rent paid by, each node are positively correlated to their STI values. That is, the more important nodes are paid more for their services, but they also pay more in rent.

A fixed percentage of the links with the lowest LTI values is then forgotten (which corresponds equationally to setting the LTI to 0).

Separately from the above, the process of Hebbian probability updating is carried out via a diffusion process in which some nodes "trade" STI utilizing a diffusion matrix **D**, a version of the connection matrix **C** normalized so that **D** is a left stochastic matrix. **D** acts on a similarly scaled vector **v**, normalized so that **v** is equivalent to a probability vector of STI values.

The decision about which nodes diffuse in each diffusion cycle is carried out via a decision function. We currently are working with two types of decision functions: a standard threshold function, by which nodes diffuse if and only if the nodes are in the AF; and a stochastic decision function in which nodes diffuse with probability $\dfrac{\tanh(\text{shape}(s_i - \text{FocusBoundary})) + 1}{2}$, where shape and FocusBoundary are parameters.

The details of the diffusion process are as follows. First, construct the diffusion matrix from the entries in the connection matrix as follows:

$$\text{If } c_{ij} \ge 0, \text{ then } d_{ij} = c_{ij},$$
$$\text{else, set } d_{ji} = -c_{ij}.$$

Next, we normalize the columns of **D** to make **D** a left stochastic matrix. In so doing, we ensure that each node spreads no more that a $\langle \text{MaxSpread} \rangle$ proportion of its STI, by setting

$$\text{if } \sum_{i=1}^{n} d_{ij} > \langle \text{MaxSpread} \rangle:$$

$$d_{ij} = \begin{cases} d_{ij} \times \dfrac{\langle \text{MaxSpread} \rangle}{\sum_{i=1}^{n} d_{ij}}, & \text{for } i \ne j \\ d_{jj} = 1 - \langle \text{MaxSpread} \rangle \end{cases}$$

else:

$$d_{jj} = 1 - \sum_{\substack{i=1 \\ i \ne j}}^{n} d_{ij}$$

Now we obtain a scaled STI vector **v** by setting

$$\text{minSTI} = \min_{i \in \{1,2,\cdots,n\}} s_i \text{ and } \text{maxSTI} = \max_{i \in \{1,2,\cdots,n\}} s_i$$

$$v_i = \frac{s_i - \text{minSTI}}{\text{maxSTI} - \text{minSTI}}$$

The diffusion matrix is then used to update the node STIs

$$\mathbf{v}' = \mathbf{D}\mathbf{v}$$

and the STI values are rescaled to the interval $[\text{minSTI}, \text{maxSTI}]$.

In both the rent and wage stage and in the diffusion stage, the total STI and LTI funds of the system each separately form a conserved quantity: in the case of diffusion, the vector **v** is simply the total STI times a probability vector. To maintain overall system funds within homeostatic bounds, a mid-cycle tax and rent-adjustment can be triggered if necessary; the equations currently used for this are

- $\langle \text{Rent} \rangle = \frac{\text{recent stimulus awarded before update} \times \langle \text{Wage} \rangle}{\text{recent size of AF}}$;
- $\text{tax} = \frac{x}{n}$, where x is the distance from the current AtomSpace bounds to the center of the homeostatic range for AtomSpace funds;
- $\forall i: \; s_i = s_i - \text{tax}$

Investigation of Convergence Properties

Now we investigate some of the properties that the above ECAN equations display when we use an ECAN defined by them as an associative memory network in the manner of a Hopfield network.

We consider a situation where the ECAN is supplied with memories via a "training" phase in which one imprints it with a series of binary patterns of the form $\mathbf{P} = [p_1, \cdots, p_n]$, with $p_i \in \{0,1\}$. Noisy versions of these patterns are then used as cue patterns during the retrieval process.

We obviously desire that the ECAN retrieve the stored pattern corresponding to a given cue pattern. In order to achieve this goal, the ECAN must converge to the correct fixed point.

Theorem: For a given value of *e* in the STI rent calculation, there is a subset of hyperbolic decision functions for which the ECAN dynamics converge to an attracting fixed point.

Proof: Rent is zero whenever $s_i \le \frac{\text{recentMaxSTI}}{20}$, so we consider this case first. The updating process for the rent and wage stage can then be written as $f(s) = s + \text{constant}$. The next stage is governed by the hyperbolic decision function

$$g(s) = \frac{\tanh(\text{shape}(\text{s} - \text{FocusBoundary})) + 1}{2}.$$

The entire updating sequence is obtained by the composition $(g \circ f)(s)$, whose derivative is then

$$(g \circ f)' = \frac{\text{sech}^2(\text{f(s)}) \cdot \text{shape}}{2} \cdot (1),$$

which has magnitude less than 1 whenever -2 < shape < 2. We next consider the case $s_i > \frac{\text{recentMaxSTI}}{20}$. The function f now takes the form

$$f(s) = s - \frac{\log(20s/\text{recentMaxSTI})}{2} + \text{constant},$$

and we have

$$(g \circ f)' = \frac{\text{sech}^2(\text{f(s)}) \cdot \text{shape}}{2} \cdot \left(1 - \frac{1}{s}\right).$$

which has magnitude less than 1 whenever $|\text{shape}| < \left|\frac{2 \cdot \text{recentMaxSTI}}{\text{recentMaxSTI} - 20}\right|$. Choosing the shape parameter to satisfy $0 < \text{shape} < \min\left(2, \left|\frac{2 \cdot \text{recentMaxSTI}}{\text{recentMaxSTI} - 20}\right|\right)$ then guarantees that $\left|(g \circ f)'\right| < 1$. Finally, g∘f maps the closed interval [recentMinSti, recentMaxSTI] into itself, so applying the Contraction Mapping Theorem completes the proof.

Definition and Analysis of Variant 2

The ECAN variant described above has performed completely acceptably in our experiments so far; however we have also experimented with an alternate variant, with different convergence properties. In Variant 2, the dynamics of the ECAN are specifically designed so that a certain conceptually intuitive function serves as a Liapunov function of the dynamics.

At a given time t, for a given Atom indexed i, we define two quantities: *$OUT_i(t)$ = the total amount that Atom i pays in rent and tax and diffusion during the time-t iteration of ECAN ; $IN_i(t)$ = the total amount that Atom i receives in diffusion, stimulus and welfare during the time-t iteration of ECAN*. Note that welfare is a new concept to be introduced below. We then define $\text{DIFF}_i(t) = |\text{IN}_i(t) - \text{OUT}_i(t)|$; and define AVDIFF(t) as the average of $\text{DIFF}_i(t)$ over all i in the ECAN.

The design goal of Variant 2 of the ECAN equations is to ensure that, if the parameters are tweaked appropriately, AVDIFF can serve as a (deterministic or stochastic, depending on the details) Liapunov function for ECAN dynamics. This implies that with appropriate parameters the ECAN dynamics will converge toward a state where AVDIFF=0, meaning that no Atom is making any profit or incurring any loss. It must be noted that this kind of convergence is not always desirable, and sometimes one might want the parameters set otherwise. But if one wants the STI components of an ECAN to converge to some

specific values, as for instance in a classic associative memory application, Variant 2 can guarantee this easily.

In Variant 2, each ECAN cycle begins with rent collection and welfare distribution, which occurs via collecting rent via the Variant 1 equation, and then performing the following two steps. Step A: calculate X, defined as the positive part of the total amount by which AVDIFF has been increased via the overall rent collection process. Step B: redistribute X to needy Atoms as follows: *For each Atom z, calculate the positive part of (OUT - IN), defined as deficit(z). Distribute (X + e) wealth among all Atoms z, giving each Atom a percentage of X that is proportional to deficit(z), but never so much as to cause OUT < IN for any Atom (the welfare being given counts toward IN).* Here e>0 ensures AVDIFF decrease; e=0 may be appropriate if convergence is not required in a certain situation.

Step B is the welfare step, which guarantees that rent collection will decrease AVDIFF. Step A calculates the amount by which the rich have been made poorer, and uses this to make the poor richer. In the case that the sum of deficit(z) over all nodes z is less than X, a mid-cycle rent adjustment may be triggered, calculated so that step B will decrease AVDIFF. (I.e. we cut rent on the rich, if the poor don't need their money to stay out of deficit.)

Similarly, in each Variant 2 ECAN cycle, there is a wage-paying process, which involves the wage-paying equation from Variant 1 followed by two steps. Step A: calculate Y, defined as the positive part of the total amount by which AVDIFF has been increased via the overall wage payment process. Step B: exert taxation based on the surplus Y as follows: *For each Atom z, calculate the positive part of (IN - OUT), defined as surplus(z). Collect (Y + e1) wealth from all Atom z, collecting from each node a percentage of Y that is proportional to surplus(z), but never so much as to cause IN < OUT for any node (the new STI being collected counts toward OUT).*

In case the total of surplus(z) over all nodes z is less than Y, one may trigger a mid-cycle wage adjustment, calculated so that step B will decrease AVDIFF. I.e. we cut wages since there is not enough surplus to support it.

Finally, in the Variant 2 ECAN cycle, diffusion is done a little differently, via iterating the following process: If AVDIFF has increased during the diffusion round so far, then choose a random node whose diffusion would decrease AVDIFF, and let it diffuse; if AVDIFF has decreased during the diffusion round so far, then choose a random node whose diffusion would increase AVDIFF, and let it diffuse. In carrying out these steps, we avoid letting the same node diffuse twice in the same round. This algorithm does not let all Atoms diffuse in each cycle, but it stochastically lets a lot of diffusion happen in a way that maintains AVDIFF constant. The iteration may be modified to bias toward an average decrease in AVDIFF.

The random element in the diffusion step, together with the logic of the rent/welfare and wage/tax steps, combines to yield the result that for Variant 2 of ECAN dynamics, AVDIFF is a stochastic Lyaponov function. The details of the proof of this will be given elsewhere due to space considerations but the outline of the argument should be clear from the construction of Variant 2. And note that by setting the e and e1 parameter to 0, the convergence requirement can be eliminated, allowing the network to evolve more spontaneously as may be appropriate in some contexts; these parameters allow one to explicitly adjust the convergence rate.

One may also derive results pertaining to the meaningfulness of the attractors, in various special cases. For instance, if we have a memory consisting of a set M of m nodes, and we imprint the memory on the ECAN by stimulating m nodes during an interval of time, then we want to be able to show that the condition where precisely those m nodes are in the AF is a fixed-point attractor. However, this is not difficult, because one must only show that if these m nodes and none others are in the AF, this condition will persist. Rigorous proof of this and related theorems will appear in a follow-up paper.

Associative Memory

We have carried out experiments gauging the performance of Variant 1 of ECAN as an associative memory, using the implementation of ECAN within OpenCog, and using both the conventional and Storkey Hebbian updating formulas. Extensive discussion of these results (along with Variation 2 results) will be deferred to a later publication due to space limitations, but we will make a few relevant comments here.

As with a Hopfield net memory, the memory capacity (defined as the number of memories that can be retrieved from the network with high accuracy) depends on the sparsity of the network, with denser networks leading to greater capacity. In the ECAN case the capacity also depends on a variety of parameters of the ECAN equations, and the precise unraveling of these dependencies is a subject of current research. However, one interesting dependency has already been uncovered in our preliminary experimentation, which has to do with the size of the AF versus the size of the memories being stored.

Define the size of a memory (a pattern being imprinted) as the number of nodes that are stimulated during imprinting of that memory. In a classical Hopfield net experiment, the mean size of a memory is usually around, say, .2-.5 of the number of neurons. In typical OpenCog associative memory situations, we believe the mean size of a memory will be one or two orders of magnitude smaller than that, so that each memory occupies only a relatively small portion of the overall network.

What we have found is that the memory capacity of an ECAN is generally comparable to that of a Hopfield net with the same number of nodes and links, if and only if the ECAN parameters are tuned so that the memories being imprinted can fit into the AF. That is, the AF threshold or (in the hyperbolic case) shape parameter must be tuned so that the size of the memories is not so large that the active nodes in a memory cannot stably fit into the AF. This

tuning may be done adaptively by testing the impact of different threshold/shape values on various memories of the appropriate size; or potentially a theoretical relationship between these quantities could be derived, but this has not been done yet. This is a reasonably satisfying result given the cognitive foundation of ECAN: in loose terms what it means is that ECAN works best for remembering things that fit into its focus of attention during the imprinting process.

Interaction between ECANs and other OpenCog Components

Our analysis above has focused on the associative-memory properties of the networks, however, from the perspective of their utility within OpenCog or other integrative AI systems, this is just one among many critical aspects of ECANs. In this final section we will discuss the broader intended utilization of ECANs in OpenCog in more depth.

First of all, associative-memory functionality is directly important in OpenCogPrime because it is used to drive concept creation. The OCP heuristic called "map formation" creates new Nodes corresponding to prominent attractors in the ECAN, a step that (according to our preliminary results) not only increases the memory capacity of the network beyond what can be achieved with a pure ECAN but also enables attractors to be explicitly manipulated by PLN inference.

Equally important to associative memory is the capability of ECANs to facilitate effective allocation of the attention of other cognitive processes to appropriate knowledge items (Atoms). For example, one key role of ECANs in OCP is to guide the forward and backward chaining processes of PLN (Probabilistic Logic Network) inference. At each step, the PLN inference chainer is faced with a great number of inference steps from which to choose; and a choice is made using a statistical "bandit problem" mechanism that selects each possible inference step with a probability proportional to its expected "desirability." In this context, there is considerable appeal in the heuristic of weighting inference steps using probabilities proportional to the STI values of the Atoms they contain. One thus arrives at a combined PLN/EAN dynamic as follows:

1. An inference step is carried out, involving a choice among multiple possible inference steps, which is made using STI-based weightings (and made among Atoms that LTI weightings have deemed valuable enough to remain in RAM)
2. The Atoms involved in the inference step are rewarded with STI and LTI proportionally to the utility of the inference step (how much it increases the confidence of Atoms in the system's memory)
3. The ECAN operates, and multiple Atom's importance values are updated
4. Return to Step 1 if the inference isn't finished

An analogous interplay may occur between ECANs and the MOSES procedure learning algorithm that also plays a key role in OCP.

It seems intuitively clear that the same attractor-convergence properties highlighted in the present analysis of associative-memory behavior, will also be highly valuable for the application of ECANs to attention allocation. If a collection of Atoms is often collectively useful for some cognitive process (such as PLN), then the associative-memory-type behavior of ECANs means that once a handful of the Atoms in the collection are found useful in a certain inference process, the other Atoms in the collection will get their STI significantly boosted, and will be likely to get chosen in subsequent portions of that same inference process. This is exactly the sort of dynamics one would like to see occur. Systematic experimentation with these interactions between ECAN and other OpenCog processes is one of our research priorities going forwards.

References

Amit, Daniel (1992). *Modeling Brain Function*. Cambridge University Press.

Goertzel, Ben (2006). *The Hidden Pattern*. Brown Walker.

Goertzel, Ben (2007). *Virtual Easter Egg Hunting*. In *Advances in Artificial General Intelligence*, IOS Press.

Goertzel, Ben (2008). *OpenCogPrime: Design for a Thinking Machine*, online at http://www.opencog.org/wiki/OpenCogPrime:WikiBook

Goertzel, Ben, Matthew Ikle', Izabela Goertzel and Ari Heljakka. *Probabilistic Logic Networks*. Springer.

Hutter, Marcus (2004). *Universal AI*. Springer.

Looks, Moshe (2006). *Competent Program Evolution*. PhD thesis in CS department, Washington University at St. Louis.

Storkey A.J. (1997) *Increasing the capacity of the Hopfield network without sacrificing functionality*, ICANN97 p451-456.

Storkey, Amos (1998). *Palimpsest Memories: A New High Capacity Forgetful Learning Rule for Hopfield Networks*.

Storkey A.J. and R. Valabregue (1999) *The basins of attraction of a new Hopfield learning rule*, Neural Networks 12 869-876.

An Inferential Approach to Mining Surprising Patterns in Hypergraphs

Chapter · July 2019

DOI: 10.1007/978-3-030-27005-6_6

An Inferential Approach to Mining Surprising Patterns in Hypergraphs

Nil Geisweiller and Ben Goertzel

SingularityNET Foundation, The Netherlands
{nil,ben}@singularitynet.io

Abstract. A novel pattern mining algorithm and a novel formal definition of surprisingness are introduced, both framed in the context of formal reasoning. Hypergraphs are used to represent the data in which patterns are mined, the patterns themselves, and the control rules for the pattern miner. The implementation of these tools in the OpenCog framework, as part of a broader multi-algorithm approach to AGI, is described.

Keywords: Pattern Miner · Surprisingness · Reasoning · Hypergraphs.

1 Introduction

Pattern recognition is broadly recognized as a key aspect of general intelligence, as well as of many varieties of specialized intelligence. General intelligence can be envisioned, among other ways, as the process of an agent recognizing patterns in itself and its environment, including patterns regarding which of its actions tend to achieve which goals in which contexts [5].

The scope of pattern recognition algorithms in AI and allied disciplines is very broad, including many specialized algorithms aimed at recognizing patterns in particular sorts of data such as visual data, auditory data or genomic data. Among more general-purpose approaches to pattern recognition, so-called "pattern mining" plays a prominent role. Mining here refers to the process of systematically searching a body of data to find a large number of patterns satisfying certain criteria. Most pattern mining algorithms are greedy in operation, meaning they start by finding simple patterns and then try to combine these to guide their search for more complex patterns, and iterate this approach a few times. Pattern mining algorithms tend to work at the *syntactic* level, such as subtree mining [2], where patterns are subtrees within a database of trees, and each subtree represents a concept containing all the trees consistent with that subtree. This is both a limit and a strength. Limit because they cannot express arbitrary abstractions, and strength because they can be relatively efficient. Moreover even purely syntactic pattern miners can go a long way if much of the semantic knowledge is represented in syntax. For instance if the data contains `human(John)` and `human` $\Rightarrow$ `mortal` a purely syntactic pattern miner will not be able to take into account the implicit datum `mortal(John)` unless a step

of inference is formerly taken to make it visible. Another shortcoming of pattern mining is the volume of patterns it tends to produce. For that reason it can be useful to rank the patterns according to interestingness [12]. One can also use pattern mining in combination with other pattern recognition techniques, e.g. evolutionary programming or logical inference.

Here we present a novel approach to pattern mining that combines semantic with syntactic understanding of patterns, and that uses a sophisticated measure of pattern surprisingness to filter the combinatorial explosion of patterns. The surprisingness measure and the semantic aspect of patterns are handled via embedding the pattern mining process in an inference engine, operating on a highly general hypergraph-based knowledge representation.

1.1 Contribution

A pattern miner algorithm alongside a measure of surprisingness designed to find patterns in hypergraph database are introduced. Both are implemented on the OpenCog framework [6], on top of the *Unified Rule Engine*, URE for short, the reasoning engine of OpenCog. Framing pattern mining as reasoning provides the following advantages:

1. Enable hybridizations between syntactic and semantic pattern mining.
2. Allow to handle the full notion of surprisingness, as will be further shown.
3. Offer more transparency. Produced knowledge can be reasoned upon. Reasoning steps selected during mining can be represented as data for subsequent mining and reasoning, enabling meta-learning by leveraging URE's inference control mechanism.

The last point, although already important as it stands, goes further than it may at first seem. One of the motivations to have a pattern miner in OpenCog is to mine inference traces, to discover control rules and apply these control rules to speed up reasoning, akin to a Heuristic Algorithmic Memory [9] for reasoning. By framing not only pattern mining but more generally learning as reasoning we hope to kickstart a virtuous self-improvement cycle. Towards that end more components of OpenCog, such as MOSES [8], an evolutionary program learner, are in the process of being ported to the URE.

Framing learning as reasoning is not without drawbacks as more transparency comes at a computational cost. However by carefully partitioning transparent/costly versus opaque/efficient computations we hope to reach an adequate balance between efficiency and open-endedness. For instance in the case of evolutionary programming, decisions pertaining to what regions of the program space to explore is best processed as reasoning, given the importance and the cost of such operation. While more systematic operations such as evaluating the fitness of a candidate can be left as opaque. One may draw a speculative analogy with the distinction between conscious and unconscious processes.

1.2 Outline

In Section 2 a pattern mining algorithm over hypergraphs is presented; it is framed as reasoning in Section 3. In Section 4 a definition of surprisingness is provided, and a more specialized implementation is derived from it. Then, in Section 5 an example of how it can be framed as reasoning is presented, both for the specialized and abstract definitions of surprisingness.

2 Pattern Mining in Hypergraph Database

2.1 AtomSpace: Hypergraph Database

Let us first rapidly recall what is the AtomSpace [6], the hypergraph knowledge store with which we shall work here. The AtomSpace is the OpenCog AGI framework's primary data storage solution. It is a labeled hypergraph particularly suited for representing symbolic knowledge, but is also capable of representing sub-symbolic knowledge (probabilities, tensors, etc), and most importantly combinations of the two. In the OpenCog terminology, edges of that hypergraph are called *links*, vertices are called *nodes*, and *atoms* are either links or nodes.

For example one may express that cars are vehicles with

```
(Inheritance (Concept "car") (Concept "vehicle"))
```

`Inheritance` is a link connecting two concept nodes, `car` and `vehicle`. If one wishes to express the other way around, how much vehicles are cars, then one can attach the inheritance with a *truth value*

```
(Inheritance (stv 0.4 0.8) (Concept "vehicle") (Concept "car"))
```

where 0.4 represents a probability and 0.8 represents a confidence.

Storing knowledge as hypergraph rather than collections of formulae allows to rapidly query atoms and how they relate to other atoms.

2.2 Pattern Matching

OpenCog comes with a *pattern matcher*, a component that can query the AtomSpace, similar in spirit to SQL, but different in several aspects. For instance queries are themselves programs represented as atoms in the AtomSpace. This insures reflexivity where queries can be queried or produced by queries.

Here's an example of such a query

```
(Get (Present (Inheritance (Variable "$X") (Variable "$Y"))
              (Inheritance (Variable "$Y") (Variable "$Z"))))
```

which fetches instances of transitivity of inheritance in the AtomSpace. For instance if the AtomSpace contains

```
(Inheritance (Concept "cat") (Concept "mammal"))
(Inheritance (Concept "mammal") (Concept "animal"))
(Inheritance (Concept "square") (Concept "shape"))
```

it retrieves

```
(Set (List (Concept "cat") (Concept "mammal") (Concept "animal")))
```

where `cat`, `mammal` and `animal` are associated to variable `$X`, `$Y` and `$Z` according to the prefix order of the query, but `square` and `shape` are not retrieved because they do not exhibit transitivity. The construct `Set` represents a set of atoms, and `List` in this context represents tuples of values. The construct `Get` means retrieve. The construct `Present` means that the arguments are patterns to be conjunctively matched against the data present in the AtomSpace. We also call the arguments of `Present`, *clauses*, and say that the pattern is a *conjunction of clauses*.

In addition, the pattern matcher can rewrite. For instance a transitivity rule could be implemented with

```
(Bind (Present (Inheritance (Variable "$X") (Variable "$Y"))
               (Inheritance (Variable "$Y") (Variable "$Z")))
      (Inheritance (Variable "$X") (Variable "$Z")))
```

The pattern matcher provides the building blocks for the reasoning engine. In fact the URE is, for the most part, pattern matching + unification. The collection of atoms that can be executed in OpenCog, to query the atomspace, reason or such, forms a language called *Atomese*.

2.3 Pattern Mining as Inverse of Pattern Matching

The pattern miner solves the inverse problem of pattern matching. It attempts to find queries that would retrieve a certain *minimum* number of matches. This number is called the *support* in the pattern mining terminology [1, 2].

It is worth mentioning that the pattern matcher has more constructs than `Get`, `Present` and `Bind`; for declaring types, expressing preconditions, and performing general computations. However the pattern miner only supports a subset of constructs due to the inherent complexity of such expressiveness.

2.4 High Level Algorithm of the Pattern Miner

Before showing how to express pattern mining as reasoning, let us explain the algorithm itself.

Our pattern mining algorithm operates like most pattern mining algorithms [2] by greedily searching the space of frequent patterns while pruning the parts that do not reach the minimum support. It typically starts from the most abstract one, the *top* pattern, constructing specializations of it and only retain those that have enough support, then repeat. The apriori property [1] guaranties that

no pattern with enough support will be missed based on the fact that patterns without enough support cannot have specializations with enough support. More formally, given a database $\mathcal{D}$, a minimal support S and an initialize collection $\mathcal{C}$ of patterns with enough support, the mining algorithm is as follows

1. Select a pattern P from $\mathcal{C}$.
2. Produce a *shallow specialization* Q of P with support equal to or above S.
3. Add Q to $\mathcal{C}$, remove P if all its shallow specializations have been produced.
4. Repeat till a termination criterion has been met.

The pattern collection $\mathcal{C}$ is usually initialized with the top pattern

```
(Get (Present (Variable "$X")))
```

that matches the whole database, and from which all subsequent patterns are specialized. A shallow specialization is a specialization such that the expansion is only a level deep. For instance, if $\mathcal{D}$ is the 3 inheritances links of Subsection 2.2 (cat is a mammal, a mammal is an animal and square is a shape), a shallow specialization of the top pattern could be

```
(Get (Present (Inheritance (Variable "$X") (Variable "$Y"))))
```

which would match all inheritance links, thus have a support of 3. A subsequent shallow specialization of it could be

```
(Get (Present (Inheritance (Concept "cat") (Variable "$Y"))))
```

which would only match

```
(Inheritance (Concept "cat") (Concept "mammal"))
```

and have a support of 1. So if the minimum support S is 2, this one would be discarded. In practice the algorithm is complemented by heuristics to avoid exhaustive search, but that is the core of it.

3 Framing Pattern Mining as Reasoning

The hardest part of the algorithm above is step 1, selecting which pattern to expand; this has the biggest impact on how the space is explored. When pattern mining is framed as reasoning such decision corresponds to a *premise or conclusion selection*. Let us formalize the type of propositions we need to prove in order to search the space of patterns. For sake of conciseness we will use a hybridization between mathematics and Atomese, it being understood that all can be formalized in Atomese. Given a database $\mathcal{D}$ and a minimum support S we want to instantiate and prove the following theorem

$$S \leq \texttt{support}(P, \mathcal{D})$$

which expresses that pattern P has enough support with respect to the data base $\mathcal{D}$. To simplify we introduce the predicate $\texttt{minsup}(P, S, \mathcal{D})$ as a shorthand for $S \leq \texttt{support}(P, \mathcal{D})$. The primary inference rule we need is (given in Gentzen style),

$$\frac{\texttt{minsup}(Q, S, \mathcal{D}) \qquad \texttt{spec}(Q, P)}{\texttt{minsup}(P, S, \mathcal{D})} \text{ (AP)}$$

expressing that if Q has enough support, and Q is a specialization of P, then P has enough support, essentially formalizing the apriori property (AP). We can either apply such rule in a forward way, top-down, or in a backward way, bottom-up. If we search from more abstract to more specialized we want to use it in a backward way. Meaning the reasoning engine needs to choose P (*conclusion selection* from $\texttt{minsup}(P, S, \mathcal{D})$) and then construct a specialization Q. In practice that rule is actually written backward so that choosing P amounts to a *premise selection*, but is presented here this way for expository purpose. The definition of `spec` is left out, but it is merely a variation of the subtree relationship accounting for variables.

Other *heuristic* rules can be used to infer knowledge about `minsup`. They are heuristics because unlike the apriori property, they do not guaranty completeness, but can speed-up the search by eliminating large portions of the search space. For instance the following rule

$$\frac{\texttt{minsup}(P, S, \mathcal{D}) \qquad \texttt{minsup}(Q, S, \mathcal{D}) \qquad R(P \otimes Q)}{\texttt{minsup}(P \otimes Q, S, \mathcal{D})} \text{ (CE)}$$

expresses that if P and Q have enough support, and a certain combination $P \otimes Q$ has a certain property R, then such combination has enough support. Such rule can be used to build the conjunction of patterns. For instance given P and Q both equal to

```
(Get (Present (Inheritance (Variable "$X") (Variable "$Y"))))
```

One can combine them (joint by variable `$Y`) to form

```
(Get (Present (Inheritance (Variable "$X") (Variable "$Y"))
              (Inheritance (Variable "$Y") (Variable "$Z"))))
```

The property R here is that both clauses must share at least one joint variable and the combination must have its support above or equal to the minimum threshold.

4 Surprisingness

Even with the help of the apriori property and additional heuristics to prune the search, the volume of mined patterns can still be overwhelming. For that it is helpful to assign to the patterns a measure of *interestingness*. This is a broad notion and we will restrict our attention to the sub-notion of *surprisingness*, that can be defined as what is *contrary to expectations*.

Just like for pattern mining, surprisingness can be framed as reasoning. They are many ways to formalize it. We tentatively suggest that in its most general sense, surprisingness may be the considered as the difference of outcome between different inferences over the same conjecture.

Of course in most conventional logical systems, if consistent, different inferences will produce the same result. However in para-consistent systems, such as PLN for *Probabilistic Logic Network* [4], OpenCog's logic for common sense reasoning, conflicting outcomes are possible. In particular PLN allows propositions to be believed with various degrees of truth, ranging from total ignorance to absolute certainty. Thus PLN is well suited for such definition of surprisingness.

More specifically we define surprisingness as the *distance of truth values between different inferences over the same conjecture*. In PLN a *truth value* is a second order distribution, probabilities over probabilities, Chapter 4 of [4]. Second order distributions are good at capturing uncertainties. Total ignorance is represented by a flat distribution (Bayesian prior), or a slightly concave one (Jeffreys prior [7]), and absolute certainty by a Dirac delta function.

Such definition of surprisingness has the merit of encompassing a wide variety of cases; like the surprisingness of finding a proof contradicting human intuition. For instance the outcome of Euclid's proof of the infinity of prime numbers might contradict the intuition of a beginner upon observation that prime numbers rapidly rarefy as they grow. It also encompasses the surprisingness of observing an unexpected event, or the surprisingness of discovering a pattern in seemingly random data. All these cases can be framed as ways of constructing different types of inferences and finding contradictions between them. For instance in the case of discovering a pattern in a database, one inference could calculate the empirical probability based on the data, while an other inference could calculate a probability estimate based on variable independences.

The distance measure to use to compare conjecture outcomes remains to be defined. Since our truth values are distributions the *Jensen-Shannon Distance*, JSD for short [3], suggested as surprisingness measure in [11], could be used. The advantage of such distance is that it accounts well for uncertainty. If for instance a pattern is discovered in a small data set displaying high levels of dependencies between variables (thus surprising relative to an independence assumption), the surprisingness measure should consider the possibility that it might be a fluke since the data set is small. Fortunately, the smaller the data set, the flatter the second order distributions representing the empirical and the estimated truth values of the pattern, consequently reducing the JSD.

Likewise one can imagine the following experiments. In the first experiment a coin is tossed 3 times, a probability p_1 of head is calculated, then the coin is tossed 3 more times, a second probability p_2 of head is calculated. p_1 and p_2 might be very different, but it should not be surprising given the low number of observations. On the contrary, in the second experiment the coin is tossed a billion times, p_1 is calculated, then another billion times, p_2 is calculated. Here even tiny differences between p_1 and p_2 should be surprising. In both cases the Jensen-Shannon Distance seems to adequatly accounts for the uncertainty.

A slight refinement of our definition of surprisingness, probably closer to human intuition, can be obtained by fixing one type of inference provided by the current model of the world from which rapid (and usually uncertain) conclusions can be derived, and the other type of inference implied by the world itself, either

via observations, in the case of an experiential reality, or via crisp and long chains of deductions in the case of a mathematical reality.

4.1 Independence-based Surprisingness

Here we explore a limited form of surprisingness based on the independence of the variables involved in the clauses of a pattern, called I-Surprisingness for Independence-based Surprisingness. For instance

```
(Get (Present (Inheritance (Variable "$X") (Variable "$Y"))
              (Inheritance (Variable "$Y") (Variable "$Z"))))
```

has two clauses

```
(Inheritance (Variable "$X") (Variable "$Y"))
```

and

```
(Inheritance (Variable "$Y") (Variable "$Z"))
```

If each clause is considered independently, that is the distribution of values taken by the variable tuples ($X, $Y) appearing in the first clause is independent from the distribution of values taken by the variable tuples ($Y, $Z) in the second clause, one can simply use the product of the two probabilities to obtain an probability estimate of their conjunctions. However the presence of joint variables, here $Y, makes this calculation incorrect. The connections need to be taken into account. To do that we use the fact that a pattern of connected clauses is equivalent to a pattern of disconnected clauses combined with a condition of equality between the joint variables. For instance

```
(Get (Present (Inheritance (Variable "$X") (Variable "$Y"))
              (Inheritance (Variable "$Y") (Variable "$Z"))))
```

is equivalent to

```
(Get (And (Present (Inheritance (Variable "$X") (Variable "$Y1"))
                   (Inheritance (Variable "$Y2") (Variable "$Z")))
          (Equal (Variable "$Y1") (Variable "$Y2"))))
```

where the joint variables, here $Y, have been replaced by variable occurrences in each clause, $Y1 and $Y2. Then we can express the probability estimate as the product of the probabilities of the clauses, times the probability of having the values of the joint variables equal.

5 I-Surprisingness Framed as Reasoning and Beyond

The proposition to infer in order to calculate surprisingness is defined as

$$\texttt{surp}(P, \mathcal{D}, s)$$

where `surp` is a predicate relating the pattern P and the database $\mathcal{D}$ to its surprisingness s, defined as

$$s := \texttt{dst}(\texttt{emp}(P, \mathcal{D}), \texttt{est}(P, \mathcal{D}))$$

where `dst` is the Jensen-Shannon distance, `emp` is the empirical second order distribution of P, and `est` its estimate. The calculation of $\texttt{emp}(P, \mathcal{D})$ is easily handled by a *direct evaluation* rule that uses the support of P and the size of $\mathcal{D}$ to obtain the parameters of the beta-binomial-distribution describing its second order probability. However, the mean by which the estimate is calculated is let unspecified. This is up to the reasoning engine to find an inference path to calculate it. Below is an example of inference tree to calculate `surp` based on I-Surprisingness

$$\dfrac{P \quad \mathcal{D} \qquad \dfrac{\dfrac{P \quad \mathcal{D}}{\texttt{emp}(P, \mathcal{D})}\,(\text{DE}) \qquad \dfrac{P \quad \mathcal{D}}{\texttt{est}(P, \mathcal{D})}\,(\text{IS})}{\texttt{dst}(\texttt{emp}(P, \mathcal{D}), \texttt{est}(P, \mathcal{D}))}\,(\text{JSD})}{\texttt{surp}(P, \mathcal{D}, \texttt{dst}(\texttt{emp}(P, \mathcal{D}), \texttt{est}(P, \mathcal{D})))}\,(\text{S})$$

where

- (S) is a rule to construct the `surp` predicate,
- (JSD) is a rule to calculate the Jensen-Shannon Distance,
- (DE) is the direct evaluation rule to calculate the empirical second order probability of P according to $\mathcal{D}$,
- (IS) is a rule to calculate the estimate of P based on I-Surprisingness described in Section 4.1.

That inference tree uses a single rule (IS) to calculate the estimate. Most rules are complex, such as (JSD), and actually have the heavy part of the calculation coded in C++ for maximum efficiency. So all that the URE must do is put together such inference tree, which can be done reasonably well given how much complexity is encapsulated in the rules.

As of today we have only implemented (IS) for the estimate. In general, however, we want to have more rules, and ultimately enough so that the estimate can be inferred in an open-ended way. In such scenario, the inference tree would look very similar to the one above, with the difference that the (IS) rule would be replaced by a combination of other rules. Such approach naturally leads to a dynamic surprisingness measure. Indeed, inferring that some pattern is I-Surprising requires to infer its empirical probability, and this knowledge can be further utilized to infer estimates of related patterns. For instance, if say an I-Surprising pattern is discovered about pets and food. A pattern about cats and food might also be measured as I-Surprising, however the fact that cat inherits pet may lead to constructing an inference that estimates the combination of cat and food based on the combination of pet and food, possibly leading to a much better estimate, and thus decreasing the surprisingness of that pattern.

6 Discussion

The ideas presented above have been implemented as open source C++ code in the OpenCog framework, and have been evaluated on some initial test datasets, including a set of logical relationships drawn from the SUMO ontology [10]. The results of this empirical experimentation are omitted here for space reasons and will be posted online as supplementary information. These early experiments provide tentative validation of the sensibleness of the approach presented: using inference on a hypergraph based representation to carry out pattern mining that weaves together semantics and syntax and is directed toward a sophisticated version of surprisingness rather than simpler objective functions like frequency.

Future work will explore applications to a variety of practical datasets, including empirical data and logs from an inference engine; and richer integration of these methods with more powerful but more expensive techniques such as predicate logic inference and evolutionary learning.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. Proceedings of the 20th International Conference on Very Large Data Bases (1994)
2. Chi, Y., Muntz, R., Nijssen, S., N. Kok, J.: Frequent subtree mining - an overview. Fundam. Inform. **66**, 161–198 (01 2005)
3. Endres, D., Schindelin, J.: A new metric for probability distributions. Information Theory, IEEE Transactions on **49**, 1858 – 1860 (08 2003)
4. Goertzel, B., Ikle, M., Goertzel, I.F., Heljakka, A.: Probabilistic Logic Networks. Springer US (2009)
5. Goertzel, B., Pennachin, C., Geisweiller, N.: Engineering General Intelligence, Part 1: A Path to Advanced Agi Via Embodied Learning and Cognitive Synergy. Atlantis Press (2014)
6. Goertzel, B., Pennachin, C., Geisweiller, N.: Engineering General Intelligence, Part 2: The CogPrime Architecture for Integrative, Embodied AGI. Atlantis Press (2014)
7. Jeffreys, H.: An Invariant Form for the Prior Probability in Estimation Problems. Proceedings of the Royal Society of London Series A **186**, 453–461 (1946)
8. Looks, M., Sc, B., Missouri, S.L., Louis, S.: Abstract competent program evolution by moshe looks (2006)
9. Özkural, E.: Towards heuristic algorithmic memory. In: Schmidhuber, J., Thórisson, K.R., Looks, M. (eds.) Artificial General Intelligence. pp. 382–387. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
10. Pease, A.: Ontology: A practical guide. Articulate Software Press, Angwin, CA (01 2011)
11. Pienta, R., Lin, Z., Kahng, M., Vreeken, J., Talukdar, P.P., Abello, J., Parameswaran, G., Chau, D.H.P.: Adaptivenav: Discovering locally interesting and surprising nodes in large graphs. IEEE VIS Conference (Poster) (2015)
12. Vreeken, J., Tatti, N.: Interesting Patterns, pp. 105–134. Springer International Publishing, Cham (2014)