Pattern Mining for General Intelligence: The FISHGRAM Algorithm for Frequent and Interesting Subhypergraph Mining

Jade O'Neill[1] *and Ben Goertzel*[2,3,4] *and Shujing Ke*[1,2,4] *and Ruiting Lian*[2,4] *and Keyvan Sadeghi*[2] *and Simon Shiu*[1] *and Dingjie Wang*[2,4] *and Gino Yu*[2]

[1] Hong Kong Poly U, Dept. of Computer Science
[2] Hong Kong Poly U, School of Design
[3] Novamente LLC
[4] Dept. of Cognitive Science, Xiamen University

Abstract. Fishgram, a novel algorithm for recognizing frequent or otherwise interesting sub-hypergraphs in large, heterogeneous hypergraphs, is presented. The algorithm's implementation the OpenCog integrative AGI framework is described, and concrete examples are given showing the patterns it recognizes in OpenCog's hypergraph knowledge store when the OpenCog system is used to control a virtual agent in a game world. It is argued that Fishgram is well suited to fill a critical niche in OpenCog and potentially other integrative AGI architectures: scalable recognition of relatively simple patterns in heterogeneous, potentially rapidly-changing data.

1 Introduction

Pattern recognition is a core aspect of general intelligence. In general it is an extremely difficult problem (uncomputable, under many formulations), but in various special cases it may be tractable and even efficiently soluble in the large-scale and in real time. Different AGI architectures handle pattern recognition in a great diversity of ways; some via a unified approach to recognizing all patterns relevant to an AGI system's goals, others via a collection of different pattern recognition processes with different foci, strengths and weaknesses.

The problem addressed here is the creation of a pattern recognition algorithm suitable for the scalable recognition of simple patterns in large, heterogeneous, potentially real-time sets of data. One question arising immediately when one considers such an algorithm is the nature of the data representation. Here we assume a hypergraph representation (a very general representation that is suitable for basically any kind of discrete data and some varieties of continuous data as well), and present an algorithm called Fishgram (Frequent
Interesting Subhypergraph Mining), which mines frequent or otherwise interesting subhypergraphs from (large or small) hypergraphs. Algorithmically, Fishgram is in the same broad family as frequent itemset and subgraph mining algorithms; but it involves many specific choices made to ensure its practical utility in an AGI context.

Fishgram was designed primarily for use within the OpenCog integrative AGI framework [?], and has been implemented in this context. It represents patterns as a conjunction (AndLink) of OpenCog Links, which usually contain OpenCog VariableNodes. Concrete examples of Fishgram's utilization within OpenCog will be presented here, in the context of OpenCog's application to control virtual agents in a game world. In this example context, the patterns recognized by Fishgram are combinations of predicates representing basic perceptions (e.g. what kind of object something is, objects being near each other, types of blocks, and actions being performed by the user or the AI). However, the Fishgram algorithm is not intrinsically restricted to the agent control domain nor to OpenCog, and could be used much more broadly, e.g. in a narrow AI data mining context, or in the context of any other integrative AGI architecture that is able to present its knowledge in hypergraph format.

2 The OpenCog Integrative AGI Framework

OpenCog is an open-source AGI software framework, which has been used for various practical applications, and also for the in-progress implementation of the OpenCogPrime design aimed ultimately toward AGI at the human level and beyond. OpenCog has been used for commercial applications in the area of natural language processing and data mining; e.g. see [?]. It has also been used to control virtual agents in virtual worlds, at first using an OpenCog variant called the OpenPetBrain [?], and more recently in a more general way using a Minecraft-like virtual environment [?].

Conceptually founded on the "patternist" systems theory of intelligence outlined in [?], OpenCogPrime combines multiple AI paradigms such as uncertain logic, computational linguistics, evolutionary program learning and connectionist attention allocation in a unified architecture. Cognitive processes embodying these different paradigms interoperate together on a common neural-symbolic hypergraph knowledge store called the Atomspace. The interaction of these processes is designed to encourage the self-organizing emergence of high-level network structures in the Atomspace, including superposed hierarchical and heterarchical knowledge networks, and a self-model network enabling meta-learning.

OCP relies on multiple memory types (all intersecting via the AtomSpace, even when also involving specialized representations), including the declarative, procedural, sensory, and episodic memory types that are widely discussed in cognitive neuroscience [?], plus attentional memory for allocating system resources generically, and intentional memory for allocating system resources in a goal-directed way. Declarative memory is addressed via probabilistic inference; procedural memory via probabilistic evolutionary program learning; episodic memory via simulation; intentional memory via a largely declarative goal system; attentional memory via an economics-based dynamical system similar to an attractor neural network.

The essence of the OCP design lies in the way the structures and processes associated with each type of memory are designed to work together in a closely coupled way, the operative hypothesis being that this will yield cooperative emergent intelligence.

2.1 The Atomspace Representation

OpenCog's "Atomspace" knowledge representation is a generalized hypergraph formalism which comprises a specific vocabulary of Node and Link types, used to represent declarative knowledge and also, indirectly, other types of knowledge as well. There is a specific vocabulary of a couple dozen node and link types with semantics carefully chosen to reflect the needs of OpenCog's cognitive processes. Simple examples of OpenCog links, in the notation commonly used with OpenCog, are:

```
InheritanceLink Ben_Goertzel animal <.99>

EvaluationLink <.7>
        chase
        ListLink
                cat
                mouse
```

Examples using nodes with English-word labels provide convenient examples, but in fact most nodes in a practical OpenCog system will generally be automatically learned and not correspond directly to any human-language concept.

What's important about the AtomSpace knowledge representation is mainly that it provides a flexible means for compactly representing multiple relevant forms of knowledge, in a way that allows them to interoperate – where by "interoperate" we mean that e.g. a fragment of a chunk of declarative knowledge can link to a fragment of a chunk of attentional or procedural knowledge; or a chunk of knowledge in one category can overlap with a chunk of knowledge in another category (as when the same link has both a (declarative) truth value and an (attentional) importance value).

3 The Fishgram Algorithm

Fishgram was developed because OpenCog, which represents knowledge internally using a hypergraph called the Atomspace, needed a fast, scalable, greedy subhypergraph mining algorithm. At first, an attempt was made to find an existing subgraph mining algorithm that would suit the purpose (since mapping hypergraphs into graphs can be done straightforwardly). It was found that no existing algorithms fit the bill, so a novel algorithm was developed.

This reflects a pattern that we have found to occur fairly often in the development of OpenCog. When OpenCog requires a component that relies on the concepts already studied extensively in some area of computer science, it usually turns out that no existing algorithm or software package sufficiently meets OpenCog's requirements. Existing algorithms and software packages have generally been designed and implemented to operate stand-alone, or within software pipelines oriented to particular narrow tasks; and nearly always, it seems that making algorithms to inter-operate with other algorithms and structures in an AGI context places different requirements.

In this case, we found that frequent *itemset* mining algorithms are not conveniently applicable to subhypergraph mining, as representing hypergraphs in the required tabular format is awkward and introduces large inefficiencies. On the other hand, we found that most other graph mining algorithms were designed with molecular datasets in mind (see [**?**] [**?**] for overviews of the frequent subgraph mining literature). The OpenCog AtomSpace is a different sort of graph from these in various ways. For example, in the Atomspace

- there are many possible relations between each pair of nodes (much like in a semantic network)
- many relations involve more than two objects, and there are also properties predicates about a single object. So the relations are effectively directed links of varying arity.
- there are many events represented, and many states can change over time (e.g. an egg changes state while it's cooking)

Unlike other subgraph mining algorithms, Fishgram is designed for general knowledge in an embodied agent.

The largest inspirations for the Fishgram algorithm were the GSpan frequent subgraph mining algorithm [**?**], and the handling of variable bindings in standard inductive learning systems like FOIL [**?**]. Among the main differences between Fishgram and GSpan are Fishgram's use of breadth-first search, and its more flexible management of variable bindings in a roughly FOIL-like way.

Fishgram uses a breadth-first search, rather than depth-first search as is the case with most subgraph mining algorithms. This is appropriate for use in an intelligent agent which is looking to learn a broad variety of regularities in its environment – a very different use case from searching for specific patterns in a molecular database. Also, Fishgram does an embedding-based search, searching for patterns that can be embedded multiple times in a large graph. Molecular datasets have many separate graphs for separate molecules; embodied perceptions are closer to a single, fairly well-connected graph. Depth-first search would be very slow on such a graph, as there are many very long paths and the search would mostly find those. Whereas in an embodied-agent-control use case, the useful patterns tend to be compact and repeated many times.

The design of Fishgram makes it easy to experiment with multiple different scoring functions, from simple ones like frequency to much more sophisticated functions such as interaction information [**?**]. It also makes it easy to guide and customize the pattern search in various ways. In typical Fishgram uses, one may specify a certain category of entities about which one is particularly interested to recognize patterns (e.g. virtual-world objects, in a virtual agent control context), and one may also specify whether one is especially interested in spatial, temporal patterns or neither.

3.1 Pseudocode

Pseudocode for Fishgram is as follows. For simplicity, this assumes a certain set of "distinguished" entities has been identified, and that temporal but not spatial patterns are of interest.

```
initial layer = every pair (relation, binding)

while previous layer is not empty:
        foreach (conjunction, binding) in previous layer:
                let incoming = all (relation, binding) pairs
                               containing an "distinguished entity"
                               in the conjunction
                let possible_next_events = all (event, binding) pairs
                               where the event happens during or shortly
                               after the last event in conjunction
                foreach (relation, relation_binding) in incoming
                               and possible_next_events:
                         (new_relation, new_conjunction_binding) =
                                    map_to_existing_variables(conjunction,
                                    binding, relation, relation_binding)
                        if new_relation is already in conjunction, skip it
                        new_conjunction = conjunction + new_relation
                        if new_conjunction has been found already, skip it
                        otherwise, add (new_conjunction,
                                  new_conjunction_binding)
                                  to the current layer

map_to_existing_variables(conjunction, conjunction_binding,
                               relation, relation_binding)
        r', s' = a copy of the relation and binding using new variables
        foreach variable v, object o in relation_binding:
                foreach variable v2, object o2 in conjunction_binding:
                        if o == o2:
                               change r' and s' to use v2 instead of v
        return r',s
```

To generalize the above to recognize spatial as well as temporal patterns, it suffices to introduce *possible_nearby_events* analogous to *possible_next_events* in the above.

3.2 Preprocessing

The Fishgram implementation includes several preprocessing steps that make it easier for the main Fishgram search to find patterns. There is a filter system, so that things which seem irrelevant can be excluded from the search. And, one can explicitly specify a list of "distinguished entities" that have to be treated by Fishgram as variables. For example, in a typical virtual world application, any predicate that refers to objects (including agents) will be given a variable so it can refer to any object. Other predicates or InheritanceLinks can be added to a pattern, to restrict it to specific kinds of objects, as will be shown in the examples given below. So there is a step which goes through all of the links in the AtomSpace, and records a list of predicates with variables, such as X is red or X eats Y. This makes the search part simpler, because it never has to decide whether something should be a variable or a specific object.

Also, in the current implementation, there is some customization to ease the recognition of temporal patterns in an agent-control context. The increased predicate is added to potential patterns via a preprocessing step. The OpenCog agent's goals have a fuzzy TruthValue representing how well the goal is achieved at any point in time, so that e.g. EnergyDemandGoal represents how much energy the virtual robot has at some point in time. The "increased" predicate, in this case, records times that a goal's TruthValue increased.

3.3 The Search Process

The Fishgram search, as depicted above, is breadth-first. It starts with all predicates (or InheritanceLinks) found by the preprocessing step. Then it finds pairs of predicates involving the same variable. Then they are extended to conjunctions of three predicates, and so on. Many relations apply at a specific time, for example the agent being near an object, or an action being performed. These are included in a sequence, and are added in the order they occurred.

Fishgram remembers the examples for each pattern. If there is only one variable in the pattern, an example is a single object; otherwise each example is a vector of objects for each variable in the pattern. Each time a relation is added to a pattern, if it has no new variables, some of the examples may be removed, because they don't satisfy the new predicate. It needs to have at least one variable in common with the previous relations. Otherwise the patterns would combine many unrelated things.

In frequent itemset mining (for example APRIORI [**?**]), there is effectively one variable, and adding a new predicate will often decrease the number of items that match. It can never increase it. The number of possible conjunctions increases with the length, up to some point, after which it decreases. But when mining for patterns with multiple objects there is a much larger combinatorial explosion of patterns. Various criteria can be used to prune the search.

The most basic criterion is the frequency. Only patterns with at least N examples will be included, where N is an arbitrary constant. You can also set a maximum number of patterns allowed for each length (number of relations), and only include the best ones. The next level of the breadth-first search will only search for extensions of those patterns. Similar dynamics may be used with criteria more sophisticated than frequency.

4 Example Patterns

What we see here is that, in this particular Atomspace, the algorithm found a significant number of patterns of moderate length and reasonably high frequency.

To illustrate Fishgram's operation, we present some concrete examples obtained via running Fishgram on a small AtomSpace, derived via allowing an OpenCog agent to control a simulated robot agent in a small virtual world containing a house and some batteries. The Atomspace was obtained via running OpenCog to control the agent in this environment for roughly 5 minutes. The environment contained 32 objects, and 98 timestamps corresponding to moments at which events occurred. A preprocessing step noticed TimeNodes the agent's EnergyDemandGoal's TruthValue increased or decreased. The events involved, in which Fishgram recognized patterns, included appearance and disappearance of objects, and grabbing, eating and holding on the part of the agent.

Following is a list of the EvaluationLinks and InheritanceLinks in this small, test Atomspace:

```
(EvaluationLink is_edible:PredicateNode (ListLink $0)): 9,
(EvaluationLink is_toy:PredicateNode (ListLink $0)): 1,
(EvaluationLink is_small:PredicateNode (ListLink $0)): 3,
(EvaluationLink isHoldingSomething:PredicateNode (ListLink $0)): 1,
(EvaluationLink at_home:PredicateNode (ListLink $0)): 1,
(InheritanceLink $0 Battery:ConceptNode): 9,
(InheritanceLink $0 ClawSwitch:ConceptNode): 1,
(InheritanceLink $0 Pet:ConceptNode): 2,
(InheritanceLink $0 Avatar:ConceptNode): 2,
(InheritanceLink $0 BatterySwitch:ConceptNode): 1,
(InheritanceLink $0 Soccerball:ConceptNode): 1,
(InheritanceLink $0 pet_home:ConceptNode): 1,
(InheritanceLink $0 Home:ConceptNode): 1,
(InheritanceLink $0 robotic:ConceptNode): 1,
(InheritanceLink $0 Player:ConceptNode): 1,
(InheritanceLink $0 egg:ConceptNode): 3,
(InheritanceLink $0 dish:ConceptNode): 3,
(InheritanceLink $0 TheLiftButton:ConceptNode): 1,
(InheritanceLink $0 TheLift:ConceptNode): 1,
(InheritanceLink $0 Crate:ConceptNode): 1,
(InheritanceLink $0 table:ConceptNode): 1,
(InheritanceLink $0 chair:ConceptNode): 1,
(InheritanceLink $0 pan:ConceptNode): 1,
(InheritanceLink $0 stoveButton:ConceptNode): 1,
(InheritanceLink $0 cookTop:ConceptNode): 1,
(InheritanceLink $0 Lightning Cloud:ConceptNode): 1
```

Figure **??** depicts the number of patterns of different sizes recognized by Fishgram on this particular Atomspace. What we see there is that, in this particular Atomspace, the algorithm found a significant number of patterns of moderate length and low but non-trivial frequency. Few high-frequency patterns of any length were found.

To give a more concrete sense of what Fishgram is doing, following is some example output from Fishgram from this Atomspace:

```
(AndLink
        (EvaluationLink is_edible:PredicateNode (ListLink $1000041))
```

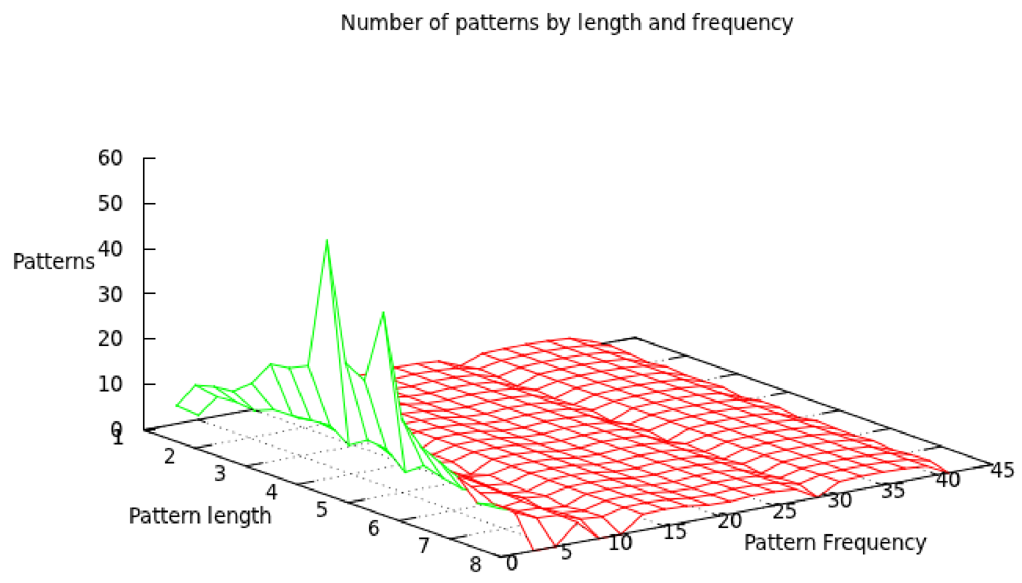

Fig. 1. Statistics of Patterns Recognized by Fishgram in an Example OpenCog Atomspace.

```
        (InheritanceLink $1000041 Battery:ConceptNode)
)
```

This means a battery which can be eaten by the virtual robot. The variable \$1000041 refers to the object (battery).

Fishgram can also find patterns containing a sequence of events. In this case, there is a list of EvaluationLinks or InheritanceLinks which describe the objects involved, followed by the sequence of events.

```
(AndLink
        (InheritanceLink $1007703 Battery:ConceptNode)
        (SequentialAndLink
                (EvaluationLink isHolding:PredicateNode
                                        (ListLink $1008725 $1007703)))
        )
)
```

This means the agent was holding a battery (denoted \$1007703); note there is also a variable for the agent itself. This pattern would also apply to the user (or another AI) holding a battery, because the pattern does not refer to the AI character specifically.

Fishgram can find patterns where it performs an action and achieves a goal. There is code to create implications based on these conjunctions . There is code that outputs causal patterns using a postprocessing system, which uses a conjunction to create a nested structure of ImplicationLinks, PredictiveImplicationLinks and SequentialAndLinks. After finding many conjunctions, it can produce ImplicationLinks based on some of them. Here is an example where the AI-controlled virtual robot discovers how to get energy.

```
(ImplicationLink
        (AndLink
                (EvaluationLink is_edible:PredicateNode (ListLink $1011619))
                (InheritanceLink $1011619 Battery:ConceptNode)
        )
        (PredictiveImplicationLink
                (EvaluationLink actionDone:PredicateNode
                                (ListLink (ExecutionLink
                                        eat:GroundedSchemaNode
                                           (ListLink $1011619))))
                (EvaluationLink increased:PredicateNode
                                (ListLink (EvaluationLink
                                        EnergyDemandGoal:PredicateNode)))
        )
)
```

5 Conclusions and Future Work

Our work with Fishgram so far has validated the general viability of the Fishgram algorithm within an OpenCog integrative AGI approach to embodied agent control. However, the current, initial implementation of Fishgram has significant limitations, which we plan to remedy incrementally in the context of utilizing Fishgram to help OpenCog control intelligent agents.

One limitation worth noting is that the current Fishgram algorithm cannot handle patterns involving numbers, although it could be extended to do so. The two options would be to either have a separate discretization step, creating predicates for different ranges of a value; or alternatively, to have predicates for mathematical operators. It would be possible to search for a split point like in decision trees – so that a number would be chosen, and only things above that value (or only things below that value) would count for a pattern. It would also be possible to have multiple numbers in a pattern, and compare them in various ways.

Another issue worth considering is scalability. Like essentially all data mining algorithms, Fishgram can achieve scalability only at the cost of aggressive pruning of candidate combinations. This aspect has not prevented classic data mining algorithms from being applied at very large scale. However, from a general intelligence perspective, it seems clear that Fishgram will need to be complemented by other algorithms that, via incorporation of more intelligent search or pruning heuristics, are able to find more complex patterns even from very large knowledge bases.

While our work so far has focused on recognizing frequent patterns, it will be important for future applications to supplement the frequency criterion with a measure of statistical interestingness, which ensures that the relations in a pattern are genuinely correlated with each other. Using frequency as the criterion results in many spurious frequent patterns, because anything which is frequent will occur together with other things, whether they are relevant or not. For example breathing while typing is a frequent pattern, because people breathe at all times. But moving your hands while typing is a much more interesting pattern. As people only move their hands some of the time, a measure of correlation would prefer the second pattern. Based on our study of the matter, we have tentatively concluded that the best measure may be interaction information, which is a generalization of mutual information that applies to patterns with more than two predicates [**?**], or variations on interaction information intended to identify multi-variable synergies even more finely [**?**]. In a learning-oriented AGI paradigm like OpenCog, an early-stage AGI does not have much knowledge of real-world structures and dynamics built in, so it must rely on statistical measures like these to find useful patterns.