Computational Intelligence, Volume , Number 000, 2015

Cognitive Agents and Machine Learning by Example: Representation with Conceptual Graphs

ALEXANDROS GKIOKAS, ALEXANDRA I. CRISTEA

Computer Science Department, University of Warwick,CV3 7AL, Coventry, United Kingdom

As *Machine Learning* and *Artificial Intelligence* progress, more complex tasks can be addressed, quite often by cascading or combining existing models and technologies, known as the bottom-up design. Some of those tasks are addressed by agents, which attempt to simulate or emulate higher cognitive abilities that cover a broad range of functions; hence those agents are named *cognitive agents*. We formulate, implement and evaluate such a cognitive agent, which combines *learning by example* with machine learning. The mechanisms, algorithms and theories to be merged when training a cognitive agent to read and learn how to represent knowledge, have not, to the best of our knowledge, been defined by the current state-of-the-art research. The task of *learning to represent knowledge* is known as *semantic parsing*, and we demonstrate that it is an ability that may be attained by cognitive agents using machine learning, and the knowledge acquired can be represented by using *conceptual graphs*. By doing so, we create a cognitive agent that simulates properties of 'learning by example', whilst performing semantic parsing with good accuracy. Due to the unique and unconventional design of this agent, we first present the model, and then gauge its performance, showcasing its strengths and weaknesses.

Key words: Learning by Example, Machine Learning, Cognitive Agents, Semantic Parsing, Conceptual Graphs.

1. INTRODUCTION

The task of reading a sentence and representing it using an ontology model is called *semantic parsing* (Popescu et al., 2003) a process which uses semantics in order to translate text to a knowledge representation structure. This process is normally algorithmic, based upon heuristics, statistics or other rules (Tang and Mooney, 2001; Shi and Mihalcea, 2005; Wong and Mooney, 2006), or in more recent research, relies on Deep learning (Andor et al., 2016; Weiss et al., 2015; Zhang and McDonald, 2012) . Such a process might simulate the mechanism through which we acquire knowledge and information (Vlachos and Clark, 2014; Vlachos, 2012). A *cognitive agent* simulates the functions which enable humans to perform semantic parsing, rather than implementing semantic parsing as a machine-oriented mechanism. Therefore it is sound to presume that a cognitive agent capable of performing such a function with relative ease, would be one step closer to becoming able to autonomously learn indefinitely.

Hence, a major target for an intelligent cognitive agent is the ability to learn how to represent knowledge from *extracted information*, and in this use-case, from raw text (Craven et al., 1999). The criteria, as set by (Lawniczak and Di Stefano, 2010) are perception, reasoning with and judgement of information or knowledge, as well as the most important achievement, the ability to **learn new knowledge** from information.

This ability may be allowing for *simulative biomimicry*, if built upon biologically inspired AI models. Regardless of the mechanisms used, the goal is the same: project

the input text and symbols onto a knowledge representation (KR) structure, which correctly describes all currently known *semiotic information* about that input. Doing so can involve the usage of semantics, statistics, annotated meta-data related to grammatical properties, ontologies, and many other characteristics and attributes.

Hereinafter, we describe and analyse an agent which combines different AI models, in order to achieve the goal of *learning how to correctly represent information onto conceptual graphs*. The goal is to attain semantic parsing, by teaching the agent how to perform such a task. The algorithms involved in performing semantic parsing are not based on any prior knowledge. Instead, the agent first learns by examples and observation, then self-analyses the learnt material, and is finally tested on unknown material.

The work described hereinafter addresses some of Bach's *Synthetic Intelligence* requirements (Bach, 2009), e.g., perception, experience, cognition and emotion (via *reinforcement*) and Haikonen's (Haikonen, 2012, 2009) cognitive intelligence topics such as meaning and representation and their relation to information, association and memory, and a rudimentary form of reasoning (discussed in Section 2.1).

The unique novelty in our approach is that we avoid previously used rigid approaches, rule-based models, heuristics and templates, (Tang and Mooney, 2001; Shi and Mihalcea, 2005; Pradhan et al., 2004; Zhong et al., 2011; Poon and Domingos, 2009) which albeit performing well, are severely limited to the domains, languages, or even to the input size. On the contrary, we describe, evaluate and provide evidence that a cognitive agent based upon machine learning models, can provide robust and adaptive semantic parsing, by *learning by example*, regardless of the domain. Moreover, that a cognitive agent can learn indefinitely, by cascading reinforcement

learning with artificial neural sub-controllers, data-mining, semantics and statistics. We have designed it in a manner which enables it to learn from examples the user provides, and self-organises and controls its internal experience and knowledge, in order to enable accurate and continuous learning, whilst being able to be retrained and updated. This agent starts in a blank/*tabula rasa* state. When trained by a user, it is able to recreate the semantic parsing process, thus performing the task in the same manner as the trainer.

Moreover, previous research in semantic parsing and other related fields rely on *feature vectors*, and not on the actual symbols (tokens,words) (Clarke, 2015; Grefenstette et al., 2014; Pradhan et al., 2004). This is due to the computational power required, memory space needed to process individual tokens and the increase in execution time, when working with raw information. Thus the norm for the past decade or so, has been to work on text using feature vectors and vector spaces, rather than directly on the actual text, with only a handful exceptions where a direct representations was used (Liang and Potts, 2015; Wong and Mooney, 2006).

Current more recent research in semantic parsing, using Deep Learning and other types of AI models, still rely on feature vectors (Grefenstette et al., 2014; Clarke, 2015). These researches vary different models, architectures, designs and algorithms on the aforementioned vectors. On the other hand, non-vector related research uses statistical approaches (Wong and Mooney, 2006) and relies on word-alignment and other well-formatted rules. Reinforcement learning and *imitation-based* approaches (Vlachos and Clark, 2014; Andreas et al., 2013; Vlachos, 2012) use machine learning as an *action-selection* mechanism dealing with features, rather than as a spatial-temporal mechanism learning the knowledge representation structure.

Recent *state-of-the-art* made significant breakthrough in accuracy, attributed to *deep learning*. The most recent breakthrough is from Google (Andor et al., 2016; Weiss et al., 2015; Zhang and McDonald, 2012) and the use of *TensorFlow* (Abadi et al., 2016) library as used by *SyntaxNet* (Petrov, 2016). Google relied mostly on POS tagging (the *Parsey McParseface* POS tagger) rather than Semantics, yet achieved the best F_1 scores to date (F_1 results range from 94.44% on news data, 95.40% on question-answers and 90.17% on web data).

Other *state-of-the-art* is the research and platform by the *spaCy* (Honnibal and Johnson, 2015) start-up in Germany. Current *software, tools and platforms* considered state-of-the-art are shown in Table 1 (*State of the Art NLU Software Tools*); for a full list and analysis Choi et al (Choi et al., 2015) have gauged performance, accuracy, speed, etc., yet some platforms have been renamed, and some appear to be un-maintained or deprecated since then.

[Table 1 about here.]

Our main contribution is thus that we provide a radical new approach to semantic parsing, where the agent learns a whole temporal-spatial sequence on how to construct knowledge representations of the input.

In the above context, the agent treats the construction of the CG as a temporal process, e.g., a *Shift-Reduce* operation (Sagae, 2009; Sagae and Lavie, 2006; Shieber, 1983) whilst the *spatial aspect* of the processing focuses on operations on the graph being constructed. Such an approach, to the best of our knowledge, has never been attempted; this process combines graph operations as an *action* to a

Markovian state represented by the graph itself. As such, a symbolic KR (the CG) is being learnt by a learning mechanism.

Prior research in using conceptual graphs for semantic parsing, compared to Meaning Representation Languages (MRL) or other KR schemes, is sparse and inconclusive (Zhong et al., 2011; Montes-y Gómez et al., 2002). Moreover, the latter research relies solely on heuristics, or semi-supervised learning. Instead, we support the notion that heuristics should be replaced by machine learning, to eliminate the necessity of prior knowledge of the domain and hard-coded rules or templates. However, the notion of using CG to extract knowledge, as well as reason with it (Kamsu-Foguem and Chapurlat, 2006) isn't new. Manipulation of knowledge graphs in real-life applications has been researched in the past (Ruiz et al., 2014; Kamsu-Foguem et al., 2013), and as such it stands to reason that usage of CG in cognitive agents could enable better human-agent interaction, especially for *knowledge transference*.

The Machine-Learning based cognitive agent is further compared with the most representative and successful heuristic and semi-supervised state of the art solutions, to clarify and illustrate its strengths and weaknesses.

2. LEARNING MODEL

2.1. Theoretical Model

Figure 1 (Agent Schema) demonstrates a high level of the components of the agent[1], and how they interact with each-other (Gkiokas, 2016).

[Figure 1 about here.]

[1] Please note that not all the components shown in Figure 1 are being used in this version of the agent.

The agent uses KR to store knowledge acquired from raw information. The KR model that does the actual representation of knowledge is the conceptual graph (CG) model of Sowa (1999). Conceptual graphs are finite connected bipartite graphs with entities partitioned as either concepts or relations (Chein and Mugnier, 2008). Conceptual graphs were chosen because they are simplistic and minimal models without an excess of meta-data. Other advantages are the simplification of the representation and relations through labelled edges, their expressiveness, which is similar to natural language, and their accuracy and highly structural information (Rasli et al., 2014; Zhong et al., 2011). Furthermore, other researchers (Croitoru et al., 2007) state that conceptual graphs are intuitive and semantically sound means of knowledge representation. Most importantly, conceptual graphs have been demonstrated to offer a computationally tractable and sound way of representing text and natural language (Montes-y Gómez et al., 2002). Within the cognitive agent's memory, we describe a conceptual graph $G_t = (V_t, E_t)$ at a moment in time t, as shown in (1), with nodes $V = (C_t, R_t)$ being either concepts or relations, and using edges (E) between node classes, to form a bipartite graph. This model is implemented as an *adjacency list* for graph G_t which also describes a state s_t.

$$G_t = (C_t, R_t, E_t). \tag{1}$$

We have used an *adjacency list*, as it has $O(|V| + |E|)$ storage complexity, $O(1)$ vertex and edge addition, and $O(|V|)$ query complexity (Cormen et al., 2001). Using an *adjacency matrix* would offer faster queries, but it has larger memory requirements and vertex addition complexity. Thus, due to the fact that the agent

stores thousands of conceptual graphs in its memory, we chose the *adjacency list*, as it has the fastest addition and most conservative memory requirements. Indeed, these operations occur very frequently when learning graphs, since the agent manipulates an empty graph into a populated one, by adding nodes and edges. We do not remove vertexes or edges, and thus an *incidence list* would not be beneficial, as neither would be an *incidence matrix* (Coxeter, 1973).

In order to learn the construction of a conceptual graph, the cognitive agent uses *reinforcement learning* (Sutton and Barto, 1998) a neural-temporal learning mechanism, which we have implemented using the *Q-Learning* algorithm (Sutton and Barto, 1998, equation 6.6) - one of the most frequently used algorithms due to its performance. We chose Reinforcement Learning because it is a biologically plausible mathematical representation of behaviouristic Psychology learning, by simulating how agents learn by associating a cumulative reward with their actions (Watkins, 1989; Sutton, 1984; Galef Jr, 1988). Furthermore, using Reinforcement Learning allows us to represent and handle symbolic representations (e.g., the graphs) directly on a ML algorithm in a *Markovian* sense (Howard, 1970; Bellman, 1957).

At the highest and most abstract level, the agent observes examples provided by the user, decomposes them, learns by their decomposition, and then becomes able to (a) recreate them, or (b) use them to approximate how to perform *highly similar tasks*. Figure 2 (*Agent Observation and Recreation*) demonstrates that high level approach in simplistic terms.

[Figure 2 about here.]

The Q-learning algorithm is shown in formula (2) where $Q(s_t, a_t)$ denotes the

policy value of taking action a_t in state s_t. The reward R is obtained only at the terminal state and is back-propagated to the previous states. The constant α is the learning rate, and constant γ is the discount factor of the next policy's value; both constants α and γ have a range between 0 and 1.

[Figure 3 about here.]

The rewarding cycle, as shown in Figure 3 (*Agent Reward*), is not continuous and takes place once when the agent has finished performing the task. In this case the *environment* is in fact the Conceptual Graph being manipulated, and the reward is associated to a specific input and output.

A low learning rate tends to ignore updates, whereas a high learning rate considers the most recent updates. Similarly, a low discount rate makes the algorithm opportunistic with respect to the most recent rewards, whereas a conservative-long term approach requires a high discount value (Even-Dar and Mansour, 2004). After trial and error and through empirical testing we decided to use $\alpha = 0.7$ and $\gamma = 0.3$, the reason being that due to the way the cognitive agent learns, by being presented examples only once, we prefer fast learning of the most recent reward. Because the same episode will not be re-iterated multiple times (as it is the case with *probabilistic Q-learning*, or when re-experiencing same or similar episodes), faster learning implies less time spent on training, a notion similarly described in *One-Shot Imitation learning* (Duan et al., 2017) for Robotics.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_{(t)} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]. \quad (2)$$

This implementation enables a unique approach to KR construction: machine learn-

ing can be used directly on symbolic information and the KR byproduct, as the state s_t denotes a KR structure at a moment in time: the conceptual graph G_t of that instance. Using the *Markov Decision Process* (MDP) employed by Q-Learning to describe the process (named an *episode*) of creating a conceptual graph, is also the basis upon which the agent learns how to perform the projection operation of information onto KR. Thus, the agent processes the actual text, structures and labels, rather than the features of the input. This approach affects *what the agent is learning*; not features or meta-data from vectorised information or vector space models, but the *actual representation*, as it is being created from the input information.

The agent aims to accumulate background knowledge (conceptual graphs) and experience (policies - episodes) when trained, in order to maximise its accuracy and overall performance. It is employing *classification methods* (see Sections 2.3.2, 2.3.3 and 2.3.4), which are combined as sub-controllers of the main reinforcement learning algorithm. The forefathers of reinforcement learning (Sutton and Barto, 1998) mentioned that an ensemble of learning and other function approximation methods are necessary, in order to be able to reuse policies. The *states and actions*, the building blocks of reinforcement learning, are learnt via *Q* policies. By learning the correct policies about the projection of text onto conceptual graphs, the approximation of states and actions enables reusability of previous experiences (episodic and non-episodic), when presented with new input (see Section 6).

In reinforcement learning an episode $\vec{e}^t$ is a sequence of *states* s_t joined together by their *actions* a_t, where *t* is a *time-step* that describes a moment in time, thus creating a MDP. Each episode is made up of a variable number of states and has a root $s_{t=1}$ and terminal state $s_{t=n}$ (see definition 3).

$$\vec{e}^{t} : \{(s_{t=1}, a_{t=1}), (s_{t=2}, a_{t=2}), \cdots, (s_{t=n}, a_{t=n})\}. \tag{3}$$

Each state describes the transition and projection of *symbolic information* (text) towards a conceptual graph. The state is described here in a *Markovian manner*, by a set of *tokenised* words T_t, and the current conceptual graph G_t, for the time-step *t* (see definition 4).

$$s_t : \left\{ \begin{array}{l} T_t : \{w_1, w_2, \cdots, w_n\} \\ \\ G_t : \{C_t, R_t, E_t\} \end{array} \right\}. \tag{4}$$

As the agent *projects information* (e.g., tokens, words, symbols) as *nodes* onto the graph G_t, these tokens are removed from the next state's s_{t+1} token set T_{t+1}. Thus, the token set is a stack from which words are removed as the episode progresses, and the graph is populated with nodes, it is in fact a *shift-reduce* (Shieber, 1983) operation. When the token set T_t becomes empty, the agent has to connect the nodes of graph G_t, by using edges. Once the agent is certain that no more edges must be created, it terminates the episode, by producing a terminal state. The *conceptual graph* of that terminal state is the final product of the projection: **the representation of the information given as input**.

An action a_t can be the decision to convert a token to a *concept* or *relation*, as conceptual graphs are bipartite graphs. The decision to take action a_t is what creates new states, the transitional link from s_t to s_{t+1} in the episode $\vec{e}^{t}$. In addition to creating nodes, an action can also be the decision to create *edges* between concepts and relations, and thus creating a connected conceptual graph.

[Figure 4 about here.]

The Figure 4 shows how *Shift-Reduce* is used in the MDP to populate the graph with nodes, and then how it is used to create edges between the nodes. This process is *deterministic* (and therefore we use the deterministic version of Q-Learning) because the agent can determine the result of its action at any given moment. Furthermore, the MDP has no hidden states or partial hidden states, albeit the semiotics of a state are not assumed to be all known.

The actions are created by the agent, and can be generated by *random, semantic, probabilistic* algorithms, or classified via neural networks, as those are the models and algorithms most often used in recent related research (Choi et al., 2015; Andor et al., 2016; Weiss et al., 2015; Clarke, 2015; Zhang and McDonald, 2012; Grefenstette et al., 2014; Vlachos and Clark, 2014; Andreas et al., 2013; Vlachos, 2012; Zhong et al., 2011; Poon and Domingos, 2009; Shi and Mihalcea, 2005; Pradhan et al., 2004; Tang and Mooney, 2001). Each of the described approaches, models and algorithms is described below, and evaluated in Section 6.

In the literature (Sutton and Barto, 1998) actions are created and then evaluated via a fitness function. As actions directly affect the next state and therefore the produced outcome, a good action generation mechanism is crucial in a cognitive agent and it is highly beneficial to avoid searching; instead it is preferred to approximate new actions based on previous ones. We examine all individual action creation mechanisms in Sections 2.3.1, 2.3.2, 2.3.3 and 2.3.4.

Cascading various algorithms was done empirically, as well as by relying on previous literature; we tried various combinations, and every time recorded their

overall accuracy. The methods tested (and described hereinafter) were based on previous research using Heuristics, Probabilities, Semantics and Machine Learning.

During training, the graph *G* and the associated input sentence are given as a *paradigm* or *example*, which the agent observes and analyses, in order to infer the episode and learn from it. Once the agent has recreated an episode for that graph, it *presumes* that the example is correct, and thus reinforces it with a positive reward, hence reinforcing the episode that created that graph.

As part of the training process (see section 3.1), a *decomposition* algorithm disassembles an existing graph *G* into an episode $\vec{e}^t$, and its corresponding states and actions. The graph is part of the training data, and has already been created by a human user as the *example* from which to learn. The actions are inferred by observing the graph changes between states: word to node conversion and graph edge creation. The decomposition algorithm is a naive heuristic, but is required in order to infer the associated states and actions; its output is a pair of s_t, a_t, part of the episode used to train the agent. During testing, the agent may be presented with known, unknown, or partially known input. When the agent is given *known input*, it stays *on-Policy* and simply outputs what Q-Learning dictates (see formula 2). For partially known input, the classification mechanisms are responsible for creating new actions by reusing previous episodes. The classification mechanisms depend on reusing previous $Q(s_t, a_t)$ policies, either directly or indirectly. In the event where unknown input is given, the agent has to explore and thus discover new policies, by attempting to approximate known states and actions.

2.2. Input pre-processing

When the agent is given a sentence as input, that input is tokenised, using standard white-space tokenizing, and certain English particles are removed (e.g.: ”a”, ”an”, ”the”), as well as common symbols such as commas, question-marks, and full-stops. This is done in order to offer a ’cleaner’ input to the agent, as it is recommended by (Jurafsky and Martin, 2000). Furthermore, a *part-of-speech* processor obtains the tags for each tokenised word in the sentence (Tsuruoka et al., 2011). In addition to the aforementioned pre-processing, a *Vector Space Model* (Turney et al., 2010) is built as a sparse matrix, and indexes all input sentences, so that *attributional semantics* can be used, to find input sentences similar to the ones stored in the agent’s memory. However, no rules or templates are applied, the input is not pre-annotated with semantics, ontologies, entities, or in any other way.

2.3. Action Decision

The importance of correct actions needs to be emphasised: incorrect actions lead to incorrect states, which eventually create incorrect conceptual graphs. *The accuracy of the terminal state and its corresponding conceptual graph is what dictates the overall performance of the agent*. The agent learns how to create the actual representation, by learning to perform the correct sequence of *actions* for each corresponding *state*. Thus it does not *classify or categorise* a conceptual graph; it creates correct or incorrect *terminal* graphs. It is those graphs we used to reward the agent, and also to infer how *accurate* it is, in comparison to the expected graph output. We discuss accuracy metrics in Section 6.1, using Sørensen coefficient (12) and Jaccard index (13).

The cognitive agent has a variety of action creation mechanisms at its disposal, and some are used in a cascading or preferential manner (meaning, one algorithm takes precedence over another), whereas other action creation mechanisms (i.e., *neural networks*) are used as standalone. We have implemented and evaluated various action-selection mechanisms from previous research, ranging from Relational Semantics (Fellbaum, 1998) to a statistical approach, a naive Bayesian and others. The action-decision mechanisms are used only when testing the agent; during training, the actions are inferred from the observed *examples*.

An explanation of each algorithm follows. We use a variety, not only to test which works better, but to compare to other research, which relies on similar algorithms.

2.3.1. *Random Action.* A random action is based upon a uniform random distribution, and utilises the Mersenne twister (Matsumoto and Nishimura, 1998) pseudo-random generator (PRNG). The Mersenne twister is the most widely used PRNG and it offers a fast and secure implementation for random integers. In the action-decision deployment scenario, it randomly decides if a word is a *concept* or a *relation*, and if two nodes should be connected by an *edge*.

2.3.2. *Semantic Action.* A semantic action uses WordNet (Fellbaum, 1998) to obtain *hypernyms, hyponyms and synonyms* as graphs, which it then traverses, in order to detect if two queried words are *semantically* connected, and what their semantic similarity is. We chose WordNet, as it is the only dictionary-based framework widely used for discovering semantic relations; it has also been evaluated and tested for more than a decade, especially for English, which is the language of our input

sentences. Other languages would require both a WordNet database to be used, and a POS tagger capable of processing that language.

Quantifying the semantic similarity, denoted by $\delta_{[n,n']}$, is done using the formula (5), as below.

$$\delta_{[n,n']} = w_{s_i} \sum_{k=0}^{k=layers(s_i)} t_{[n,n']} \Big(1 + \beta \big(d_{[n,n']} \big) \Big). \tag{5}$$

The nodes *n* and *n'* represent *words*, *tokens* or *labels*, which become the queries to WordNet. When a graph (representing a hyponym, hypernym or synonym tree) is processed, the distance travelled is stored in $t_{[n,n']}$, whereas the traversal direction $d_{[n,n']}$ (forwards or backwards) is alleviated by the constant β (empirically set to 0.1). The min-max normalised sense weight w_{s_i} (set by WordNet) biases towards the most frequent senses, which always appear first, according to the way they are sorted in WordNet dictionaries, from the most frequent to the least frequent. By using this *semantic similarity*, the agent may decide to recreate a known action a_t, which was operating on node *n*, because it is very similar to node *n'*, for which a policy $Q(s_t, a_t)$ already exists.

2.3.3. *Probabilistic Action.* The cognitive agent, after being trained with examples, analyses its episodic memory and all policies acquired. By *data mining* its own episodic memory statistics, it acquires frequencies of events and observations, such as the rate of token to node conversion, or edges existing between nodes. By obtaining statistics from *observing the frequency of events*, the agent is able to calculate empirical binomial probabilities. Probabilities are calculated by querying the probability of an edge existing for a node tuple, denoted as $P(\exists(e_{[n,n']}))$ the frequency of events, when an edge from *n* to *n'* has been observed to exist (see

equation 6). The opposite also holds true: $\not\exists(e_{[n,n']})$ denotes the fact that such an edge *has not been observed to exist* but could have been created due to the presence of n and n'. Thus, formula (6) describes the empirical probability of an edge existing, with respect to the total observations for that edge and the *nodes that it can connect*.

$$P(\exists(e_{[n,n']})) = \frac{\sum\left(\exists(e_{[n,n']})\right)}{\sum\left(\exists(e_{[n,n']})\right) + \sum\left(\not\exists(e_{[n,n']})\right)}. \tag{6}$$

When computing edge probabilities, we may use either the token value (the label) or the POS Tag of the token (its syntactic attribute). A combinatorial probability may also be used, in the event that both edge probabilities are known. All probabilities recorded have a *range* between 0 and 1, and are recorded for *edges* and *nodes*. Token to node probability (equation 7) is recorded for both tokens and POS tags, i.e., measuring how frequently a specific tag (or token/word) is classified as a concept or relation.

$$P(Concept) = \frac{\#ofEvents(t = Concept)}{\#ofObservations(t)}. \tag{7}$$

In the above formula (7), the number of events where a token *t* was being classified as a concept, are divided by the number of total observations made about *t*. Exactly the same rule applies for calculating relation probabilities. In this formula, *t* can be either a token, or a token's part-of-speech tag. Token distance is a probability value based on the observation of distance events (e.g., how far were two tokens when connected as nodes).

2.3.4. *Neural Actions.* Two *multi-layer feed forward artificial neural networks (ANN)* are created and trained after the *data-mining* phase. One network is trained using POS Tags and token distance, the other is trained using POS tags, tokens and token distance. The reason for doing this is that $P(token)$ is not always available, whereas $P(POS)$ is always available (as computed by formula (7)). After being trained, and after performing *data-mining* on the action/policy observations, the probability values can be calculated for every edge observed. We do not train the ANN on *token-to-node* action selection, because probabilistic actions perform a highly accurate node recognition (as further discussed in section 6).

The probability values for edges (see equation (6)) and the scaled and normalised token distance between the tokens within the input sentence, are the actual ANN input data. The architecture of the networks has been optimised by using the *cascading algorithm* (Nissen, 2003), and was further optimised, by using *early stopping* (Yao et al., 2007) through cross validation of the *mean-square error*. Optimisation was necessary as the problem of over-fitting became an issue, mostly due to the usage of noise in large training sets. Empirical hyper-parameter optimisation, albeit a topic on its own accord, directly affects the agent's output and as such we examined how different parameters and approaches could be used to yield the best possible neural-based action selection. Furthermore, we used training data sub-sampling and random shuffling, in order to ensure correct representational efficiency. The selection of the neural network parameters (hidden neurons) is a big challenge, with over 20 years of research in the area (Sheela and Deepa, 2013; Hagan et al., 1996), and no definite answer. Here, we used an anecdotal formula (Stackexchange, 2015) which was derived from (Sheela and Deepa, 2013), in order to infer a value for hidden

neurons (see eq. 8), where N_h is the number of hidden neurons needed, N_i is the number of input neurons, and N_o is the number of output neurons, whilst N_s is the number of training samples. The constant *alpha* is normally said to be a value between 5 and 10, but through trial and error we established that a value of 12 was more suitable for our intents and purposes, as it avoided overfitting.

$$N_h = \frac{N_s}{alpha * (N_i + N_o)}. \tag{8}$$

Neural action controllers can be said to act as filters or classifiers, which decide if an edge should be created or not. They act as sub-controllers, processing the extracted probabilities from the data-mining done by the agent, when iterating its episodic memory.

2.4. State Classification

Similar states are classified using only the original input sentence. This is performed via the use of a Vector Space Model (Turney et al., 2010), which has the ability to find very similar input in the agent's memory. This model allows the agent to assume that highly similar input *may have similar output graphs*. Classification was done by training an ANN to take into account the similarity between states, using the *min-max* normalised VSM similarity from equation (9) as the input value which affects the action-decision mechanism. In the equation, m is the vector space, and n denotes the vectorised input sentence, where p_n, t_m are the matrix coefficient elements respective to a sentence p and word t.

$$A_{m,n} = \begin{bmatrix} p_1t_1 & p_1t_2 & p_1t_3 & \dots & p_1t_m \\ p_2t_1 & p_2t_2 & p_2t_3 & \dots & p_2t_m \\ p_3t_1 & p_3t_2 & p_3t_3 & \dots & p_3t_m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_nt_1 & p_nt_2 & p_nt_3 & \dots & p_nt_m \end{bmatrix}. \quad (9)$$

3. AGENT ALGORITHMS

3.1. Training

During training, the agent learns how to project input onto *conceptual graphs*. An inference algorithm breaks down conceptual graphs to their nodes and edges, and then infers *node actions*, by observing which words were converted to what node types; it then finds the edges which connected those nodes, thus inferring *edge actions*. The algorithm (1) is inspired by *programming by example* (Lieberman, 2001); in essence, it is neuro-dynamic programming (Bertsekas and Tsitsiklis, 1995), which creates an MDP, in the form of an episode $\vec{e}^t$.

The training is done in batches, and thus results in many episodes and their corresponding conceptual graphs, populating the agent's memory. Each batch size depends upon the actual data-set used to train the agent (see Section 4). Every time the agent is trained, it is done without it having any *a-priori* knowledge about the domain or the data-set, with the only exception being the POS tag annotation by laPOS (Tsuruoka et al., 2011) (see Sections 2.3.3 and 2.3.4).

Every episode acquired during training is positively rewarded, as the agent assumes it is correct; an inherent property of learning by example, which has many ad-

Algorithm 1 Batch Training Loop

$tuples = load(trainset)$
for $tuple[graph, text] \in tuples$ **do**
  $episode = array[s_t, a_t]$
  for $word \in tuple.text$ **do**
    $a_t = findNodeInGraph(word, paradigm.graph)$
    $episode.add(s_t, a_t)$
    $g_t = s_t$
    $addNode(g_t, a_t)$
    $s_{t+1} = g_t$
    $Remove(s_{t+1}, word)$
  for $edge \in tuple.graph$ **do**
    $a_t = findEdgeInGraph(g_t)$
    if $s_t \neg terminal$ **then**
      $episode.add(s_t, a_t)$
      $g_t = s_t$
      $addEdge(g_t, a_t)$
      $s_{t+1} = g_t$
  $UpdatePolicies(episode, 1.0)$

vantages (Nehaniv and Dautenhahn, 2007) but also some pitfalls. Admittedly some limitations do exist: since this agent learns from the user, should the user provide erroneous examples, the agent will simply learn them. Ambiguity and contradictory paradigms also create dissonance within the agent's memory, and in some cases can severely hinder the agent's performance. A phenomenon, which, as known, is inherent from the imitative ability of animals and humans. Dissonance is filtered via statistics, but this approach also affects special cases such as patterns where general probabilities do not apply.

Furthermore, because the focus is Natural Language (NL), a common issue often encountered is that of ambiguity (Gorrell, 2006), which is related to both the syntactic and semantic properties of NL. That ambiguity can (and often does) create *noise* or *erroneous* and *contradictory* training samples, which propagate in the agent's

memory. Because the agent does not attribute importance to the information source, or the frequency that a specific sample may appear, e.g., all samples are given equal policy weights in the memory, hence an exception to a rule, i.e., an outlier might influence the overall agent memory. A scoring or bias system, could potentially address this issue; we've used classical probabilities in order to filter out the outliers, but admittedly more sophisticated approaches such as feature selection and detection may increase accuracy. Perhaps, most interesting of all, *judgement* or *scoring* of the training source (e.g., person or resource) could provide a more robust agent, but the complexity of such a system would increase, and would have to be unsupervised since the agent would have to be capable of adjusting policies in its memory.

3.2. Data-Mining

The agent *data-mines* (Algorithm 2) its own episodic memory after being trained, in order to deduce and calculate probability values, as described in section 2.3.3, and the agent updates look-up tables for those probability values. The *modus operandi* is straightforward: the agent will create all possible permutations of tokens into nodes, and then observe which ones actually exist in a given episode, as actions in the known $Q(s_t, a_t)$ policies. Similarly, it creates edge permutations for existing nodes within an episode's graph, and then observes which ones did exist. This creates a *mapping* of the *possible* action search space for a given episode input, which the agent then uses to update the probability values for tokens, POS Tags, and token distances. The actual size and values recorded depend on the data-set used to train the agent. Hence, larger data-sets create larger and potentially more accurate probability look-up tables. By taking this approach, we manage to create statistical data (as shown in

6.5), which is then used either by the *Probability Action* mechanism (section 2.3.3), or the artificial neural network action mechanism (section 2.3.4).

Algorithm 2 Data-Mining edges

```
for episode ∈ memory do
    s_terminal = episode.end()
    g_t = s_terminal
    for relation ∈ g_t do
        for concept ∈ g_t do
            action = Edge(relation, concept)
            if action ∈ episode then
                Record(PR(relation, concept, True)
            else
                Record(PR(relation, concept, False)
    for concept ∈ g_t do
        for relation ∈ g_t do
            action = Edge(relation, concept)
            if action ∈ episode then
                Record(PR(relation, concept, True)
            else
                Record(PR(relation, concept, False)
```

3.3. Testing

Testing is done after the agent has been trained, and it has data-mined its episodic memory space. Thus, testing assumes that the agent is trained on some subset that is representational of the information given when being tested, and that some kind of action-decision mechanism exists (e.g., neural, random, semantic, etc). If the produced conceptual graph of the terminal state is *equivalent* (identical or isomorphic) *to the one associated with the tuple's input*, then the agent self-rewards its decisions positively, else the episode is regarded as erroneous and is therefore rewarded negatively (see Section 6 for results).

Algorithm 3 Testing loop

$policies = load(Memory)$
$examples = load(Paradigms)$
for $test \in examples$ **do**
  $graph = test.graph$
  $input = test.text$
  $s_t = new(input)$
  if $s_t \notin policies$ **then**
    $episode = new(empty)$
    while $a_t = Decide(s_t)$ **do**
      $s_t = infer(input, a_t)$
      $episode.add(s_t, a_t)$
      $s_{t+1} = calculateNext(s_t, a_t)$
      $s_t = s_{t+1}$
    $terminal = s_t$
    if $terminal.graph \equiv graph$ **then**
      $UpdatePolicies(episode, 1.0)$
    elseIf $terminal.graph \not\equiv graph$
      $UpdatePolicies(episode, -1.0)$
  else
    while $\exists\big(Q(s_t, a_t)\big) \wedge s_t \neg Terminal$ **do**
      $Online(Q(s_t, a_t))$

4. CONCEPTUAL GRAPH DATA-SETS

The domain data used in this paper is taken from *health related news articles, scientific discoveries* and *lifestyle articles*, and has been published on GitHub[2]. The reason for our selection is that, whilst allowing for reasonable complexity (empirically observed during construction and annotation of the data-set), in terms of unknown words and more complex graph structures, such articles tend at the same time to be quite straightforward and factual, containing state-of-fact or discovery, thereby being less ambiguous in their description. Using factual or statement oriented sentences isn't a requirement, but evidence suggests that this type of text input

[2] https://github.com/alexge233/conceptual_graph_set

is *more structured* and *less ambiguous*, for example Google's Syntaxnet (Andor et al., 2016; Weiss et al., 2015) which provided the best to date F_1 scores of 94.44% on news data and 95.40% on question-answer data, but only 90.17% on Web data. This type of performance is consistent across various studies (Andor et al., 2016; Weiss et al., 2015; Zhang et al., 2014; Martins et al., 2013). Our intention therefore was to test with varying degrees of sentence complexity (small, medium and large sentences) without dealing with Web-related data.

The articles were collected from a variety of on-line resources (RSS feeds) from *BBC, Sky News, USA Today, Science Daily and Knox News*. Both the title and actual content of the RSS feeds were used to create the data-set. The titles were used to create smaller graphs, whereas the article contents were used to expand on the same topic and create larger and more complex graphs. The content is also further partitioned into many graphs, by using full-stops as the delimiter. In total, the data-set has 1199 entries, and each entry is a tuple: a text sentence and its corresponding conceptual graph. The dataset size was limited by the fact that it had to be generated manually. However, the total data-set size was deemed fit for these experiments, as other datasets in use were of similar or smaller size (see Table 2 - *Commonly used Data-sets*).

[Table 2 about here.]

The data-set has a variety of entries: some are short sentences (and thereby small graphs) and some are long and complex. All subsets were created randomly in the same fashion, by shuffling tuple entries and then randomly choosing the ones with a certain average sentence length. Each subset was further partitioned into a training

set and a testing set. The larger subsets also contain entries from the smaller subsets, but the agent is never tested with different sets in the same experiment, to avoid encountering these overlapping entries. The average length of a sentence increases from set to set, from 5 words per input up to 30 words per input. Figure 5 (*Graph Example*) shows an example of a simple CG.

[Figure 5 about here.]

The most commonly used data-sets for testing similar tasks in the literature (e.g., semantic parsing) are displayed in Table 2. As can be seen, only two sets are larger than our set. However, these sets cannot be directly compared, for reasons as follows. ATIS3 (Hemphill et al., 1990), which is significantly larger, having evolved over the course of decades, offers *transcribed utterances* dissimilar to scientific news and feeds. It aims to provide a *question-answer* scenario, that is not applicable to our cognitive agent scenario. Similarly, the Penn-Treebank 3 (Marcus et al., 1993) contains *Wall Street Journal* stories, with syntactic annotation. They are not appropriate for our agent, as they contain abbreviations, US-English, stock names and symbols, etc. The *RoboCup* data-set (Chen and Mooney, 2008; David L. Chen, 2010) organises the data in Meaning Representation Language (MRL) logical form tuples, containing phrases used in the *RoboCup* soccer championship, quite dissimilar from our domain. The BioNLP11ST (by the BioNL Shared Task organisation) is a data-set focusing on co-reference, entity relations and gene renaming. The last and largest difference is that all aforementioned data-sets use either MRL, logical formulae, or other similar KR structures. With the exception of (Campbell and Musen, 1992) who proposed to create a data-set for clinical data, to the best of our knowledge, no other

conceptual graph data-set currently exists. Still, even with these differences, the comparison to the data-sets in Table 2 serves to demonstrate that our data-set size is comparable with other datasets in current use, however, the average word length per sentence is smaller than what is often used. The reason for that is mostly related to practicality; the CG dataset was created for the purpose of testing this agent, and as such the effect of large and complex sentences is already known and established from previous research (Choi et al., 2015). What we are mostly interesting is not gauging how well the agent compares to other research, but if it performs its intended tasks, which are however comparable to other research. As seen later in Section 6, the agent does stay on par with other related NL research.

[Figure 6 about here.]

However it is important to note that our data-set partitioning is biased towards smaller average sentences, with the overall average size being 8 words per sentence, as shown in Figure 6 (*Dataset Characteristics*). The variation observed in accuracy (later discussed in Section 6), is mostly attributed to that wide range of changing sub-set datasets being tested.

We hypothesise that whilst the complexity of the tuple entries is not entirely dependent upon the *input word count*, it is a good indicator, as the larger the input in words/tokens, the larger the search space becomes for the agent, and hence the solution to the input is more difficult. Other simple complexity measures include the *node to edge ratio*, the *average path length* of a graph, all indicators of the structural complexity of a graph; we did not examine more complex graph measures

such as *betweenness, radius, closeness, clusterization* (Wright, 1977; Barooah and Hespanha, 2007) as those were outside of the scope of our work.

The average VSM (Vector Space Model) similarity is yet another metric, which indicates how *attributionally similar* is the topmost similar episode in the memory of the agent, to the one being produced (Turney et al., 2010). We average the topmost VSM similarity for all inputs, in order to establish if the agent has already processed some input which is similar. The implication is that subsets with higher VSM similarity would be easier for the agent to evaluate, as its acquired policies during training would *most likely be applicable* to the given input when testing.

5. EXPERIMENTAL METHODOLOGY

Experimentation is done by first training the cognitive agent, using one of the training subsets, and then testing it with the corresponding testing subset, following the principles of *Simulated Experiments* (Winsberg, 2003). This approach ensures that tuple entries in the training set will never be present in the testing set. Each experiment was done using a random subset of the dataset, and the memory of the agent is deleted across different experiments, so that we can average the accuracy and account for randomness and noise. The experimental methodology remains constant for all experiments, and a variety of meta-data is logged, in order to infer accuracy, performance, and monitor algorithm usage. Moreover, instead of training the agent on the same training-set, we use random samples of the training set, which are mutually exclusive. Ensuring consistent performance is done not only by randomising the training-sets, but also by testing the agent on average 10 times, using each training

and its respective testing set (Cavazzuti, 2012). This approach known as *Randomized Complete Block Design* (Higgins, 2003; Winer et al., 1971) organises in small blocks of experiments similar input-length sub-sets randomly picked, and proceeds to repeat each one ten times, and then averages their performance.

5.1. Training

The training methodology is as follows: (i) delete the agent's previous memory, (ii) instruct the agent to load the training set, (iii) train the agent, (iv) perform data-mining, (v) save all memory (policies, probabilities, etc.) on the disk. The training phase is entirely separated from the testing phase. The agent is trained incrementally, by observing and analysing one paradigm at a time.

5.2. Testing

The testing methodology is as follows: (i) load its memory (from training), (ii) do testing phase, (iii) log every produced graph, (iv) log VSM similarity, (v) log graph node-edge ratio, word input length, and other metrics, (v) log amount of random, probabilistic, semantic and neural actions. At the end of each testing experiment, a script is run which extracts certain metrics obtained during execution.

Those metrics are averaged to evaluate: (i) average node similarity, (ii) average edge similarity, (iii) average graph similarity, (iv) average input length. The total number of experiments used 12 blocks of random subsets executing each one 10 times.

6. RESULTS

6.1. Overall Agent Performance

Due to the nature of conceptual graphs using sets of relations, concepts and edges, it is justifiable to treat graph similarity as a problem of set matching. As our implementation of conceptual graphs uses *adjacency lists* (see equation (1)) we have identified and used two different methods to calculate graph similarity: the *Similarity Coefficient* (Rijsbergen, 1979) also known as *Sørensen index* or *Dice's coefficient* (see equation (10)) and the *Jaccard Index* (Real and Vargas, 1996) or *Jaccard coefficient* (see equation (11)), which compute the similarity between two sets A and B.

$$S(A \sim B) = \frac{2 \mid A \cap B \mid}{\mid A \mid + \mid B \mid}. \tag{10}$$

$$J(A \sim B) = \frac{\mid A \cap B \mid}{\mid A \mid + \mid B \mid - \mid A \cap B \mid}. \tag{11}$$

Both *similarity coefficients* serve the purpose of measuring how similar the agent's output, e.g., a produced graph G is, to the *ideal* or *target* output graph G'. The difference between *Sørensen* and *Jaccard* coefficients is that *Sørensen* ignores the amount of different items in the set, whereas *Jaccard* penalises the set difference, thus resulting in a smaller similarity value if the two sets contain many different items.

When establishing the agent accuracy, a norm widely used in binary classification is the F_1 score (Brodersen et al., 2010). However, this score is not directly applicable to our work, due to the fact that the notions of *precision* and *recall* (Brodersen et al., 2010) do not exist in our agent; it isn't performing direct classification but a

generation of CG as output. The Jaccard coefficient is a more strict measure which ideally would be used, however the Dice-Sørensen coefficient has *the same form* as the F_1 score (Intan et al., 2015, p. 158) and therefore functions as the primary accuracy quantifier. It is also important to note that similarity is respective to the first graph in Dice-Sørensen: the formula quantifies only how similar G is to G' and its parameters are non-anadrome.

We use the term *average graph similarity*, referring to the similarity coefficient of the *target* output graph and the *actual* graph for both (10) and (11), a value scaled and normalised between zero and one. Using a similarity coefficient with *concepts, relations and edges* from the graphs G and G' provides the basis upon which we have computed the agent's performance.

For Sørensen coefficient we weighted nodes $V_G = (C, R)$ of both classes equally to edges, in order to avoid biasing the final value favourably towards nodes. Equal importance is thus attributed to all entities of the graph, and hence to all actions of the agent, due to the fact that agent actions a_t construct the final graph.

$$S(G \sim G') = \frac{\frac{1}{2} \cdot \big(S(C_G \sim C_{G'}) + S(R_G \sim R_{G'})\big) + S(E_G \sim E_{G'})}{2}. \tag{12}$$

The *Jaccard* coefficient is used differently: each set C, R and E uses its size as a ratio, thus the final value not only penalises different items in sets, but if a set is larger, then it weighs more in the final score.

$$J(G \sim G') = \frac{\mid C_G \mid \cdot J(C_G, C_{G'}) + \mid R_G \mid \cdot J(R_G, R_{G'}) + \mid E_G \mid \cdot J(E_G, E_{G'})}{\mid C_G \mid + \mid R_G \mid + \mid E_G \mid}. \tag{13}$$

When two graphs are identical or isomorphic, their *graph similarity* is 1, in both

Boolean terms and as a percentile. The opposite also holds true, two graphs which have no common nodes or edges will have a *graph similarity* of 0.

Figure 7 shows how Sørensen and Jaccard accuracy for all experiments change, with respect to different data-sets. As the average sentence length (input) decreases, the agent becomes much more accurate. As expected, the subset with the smallest input was able to reproduce most accurately the output, whilst the subsets with a higher count of words and thus complexity, had significantly lower accuracy. Sørensen/F_1 core and Jaccard have a very narrow distribution curve for the average accuracy of all experiment subsets. However, we report the average accuracy for the largest data-set (the one containing the most complex and large input) in Table 3.

[Figure 7 about here.]

The first observation is that the overall accuracy and graph similarity are not linearly related; in fact quite the contrary appears to take place, where small edge similarity fluctuations have a disproportionate effect on the graph accuracy. Due to the fact that node accuracy remains constantly high and above 95% for all experiments, we can only attribute this drop to the decrease in edge accuracy; it is infact the only metric which seems to decrease, as input complexity increases, hence influencing the *graph similarity*. The implication of this effect is important, as it signifies that edge accuracy is what hinders the overall agent accuracy, albeit a small but noticable decline in node accuracy could also affect edge accuracy.

In order to further examine the effect of node and edge accuracy, we logged the percentage of node and edge actions (out of the total actions) for every experiment. We discovered that edge actions were on average 55.61%, whereas node actions

were 44.39%. This ratio could further augment the negative effect incorrect edges have on correct output: for example 88.65% of edge similarity could in fact play a more significant role in the output, when more edge actions than node actions are performed. Thus the *Jaccard* index we implemented (shown in equation (13)) weighing edges, appears to have a sharper decline, very similar to that of the edge similarity decline, whilst node similarity remains highly accurate.

6.2. Complexity and Accuracy

As Figure 7 (*Agent accuracy*) shows, the agent was able to *consistently provide* an F_1 Score of 92.78% for the entire data-set (e.g., all sentence input sizes).

Taking into account the fact that the data-sets in the right side of the plot in Figure (6) *have increasing complexity* (see Section 4), this indicates that the cognitive agent manages to stay on par with previous related research (Zhong et al., 2011), and demonstrates the ability to construct complex conceptual graphs.

In Figure 8 we have used the logged data from our experiments: Sørensen coefficient, Jaccard index, graph node-edge ratio, average graph path length and edge search space. We performed Principal Component Analysis (Wold et al., 1987) (PCA) on: (a) the similarity coefficients (Sørensen and Jaccard) as *Accuracy (PCA)*, and (b) the word input length and edge search space as *Input Complexity (PCA)*, thereby representing the overall similarity with respect to edge search space and word input changes. In order to plot multiple dimensional data, we have projected the most significant Eigenvectors on a single (lower) dimension: one dimension for Accuracy (PCA), and one dimension for graph Input complexity (PCA). Thus the *"complexity"* metric is the compressed row of the PCA from the *input word length and edge*

search space, and the *"accuracy"* metric is the compressed row of the PCA from the *Jaccard and Sørensen* data columns. This was done for all experimental results, as seen in Figure 8, whilst identifying how complexity affects accuracy and agent performance. In Figure 8, the graph path-size (how "deep" a graph is), the graph *node-edge* ratio $\frac{|V|}{|E|}$, and the synthetic (PCA) "complexity" metric are shown, as a Q-normalised 3D surface. The right plot in Figure 8 shows the 4th dimension as a colour heatmap, the "accuracy", and how graph attributes and complexity relate to graph accuracy.

[Figure 8 about here.]

The first observation (see Figure 8) is that *complexity* is directly related to agent accuracy. Another, less obvious factor that seems to influence agent accuracy is a small *node-edge* ratio $\frac{|V|}{|E|}$, which implies that sparsely connected graphs are harder to construct, compared to dense or fully connected graphs. An untested hypothesis which would explain this phenomenon, is that sparse graphs tend to branch in a variety of different ways, thus making sub-pattern recognition harder. Observing Figure 8 clearly demonstrates that graphs which are "column-like", i.e., have few branches, are a lot easier to construct. The last important observation is that albeit graph path-size increases as complexity increases (which is to be expected), it does not have a detrimental effect on graph construction.

6.3. Data-mining Results

Using *Data-Mining* to perform probability value extraction via graph permutations essentially maps the possible actions for a specific input. This approach is **not**

a complete mapping of the action search space, but it is not a random sub-sampling either: it is a mapping of all possible actions related to a specific input. We did not use boot-strapping, random sub-sampling or other techniques to acquire data for training the ANN, because smaller training sets were empirically found to *generalise* too much, and offer little advantages over simple probabilistic algorithms. Thus, we arrived to the conclusion that, although a permutation mapping of a large action space was time-consuming, it warranted *more representational* probability samples, due to the fact that data-mining provided considerably larger frequency/probability samples. Furthermore, the conclusion was further supported by the fact that ANNs trained with larger training data outperformed every other algorithm, as discussed in Section 6.4.

[Figure 9 about here.]

We also examined the distribution of the data-mined data after training. Figure 9 depicts the data-mined probability histograms. The first histogram $Pr(edge[Token])$ shows the probability (see equation (6)) of an edge connecting two tokens, based on their label, e.g., their *token value*. It has an unusual distribution, with a mean $\overline{x} = 0.27$ and standard distribution $\sigma = 0.37$, evident by the high frequency samples at the extremes. This is not a sampling error but indicates that most observed events were either correct, or incorrect, due to edges created using token probabilities. Because most samples have near-zero probability, the agent *learnt* which edges to avoid based on tokens values alone. The middle histogram $Pr(Edge[POSTag])$ is the probability (see equation (6)) of an edge based on Part-of-Speech tags. It has a mean of $\overline{x} = 0.27$ and a standard deviation $\sigma = 0.11$. The observation of this

probability's distribution implies that POS tags offer a rather general approach to calculating edges, and more often than not, do not offer a high degree of certainty for an action a_t. The bottom histogram is not a probability, but the normalised and scaled (0 to 1) values of token distance within a sentence, which have been observed to become connected by an edge. We data-mined it because it aids the agent by correctly inferring if an edge, which would otherwise be highly probable to exist, should be in fact filtered out, due to an extreme distance of the two tokens inside the sentence. It has a mean of $\overline{x} = 0.04$ and a standard deviation of $\sigma = 0.55$.

[Figure 10 about here.]

In Figure 10 we used the data shown as histograms previously in Figure 9, and plotted it in three dimensions. The top left *point-cloud* is the representation in space of all 30,960 samples taken from *low-value* and *high value* actions. The visual demonstration showcases the correlation of the samples to the *entire action search space*, not taking into account the affinely extended real number system and its implications, but assuming discrete integral values. The uppermost left plot in Figure 10 shows that in the entire search space, only a small amount has been sampled, when using graph permutation actions from the episodic memory of the agent. The topmost right plot in Figure 10 is a three-dimensional grid surface, which connects the points, and thus generalises the data, by using the *Q-norm* function for all data-points. The bottom left plot contains the *three dimensional Convex Hull* (Chazelle, 1993) of the points, superimposed on the cloud. The bottom right plot contains the same Convex Hull, superimposed on the normalised data grid surface. All plots in Figure 10 have as *X axis* the edge probability based on tokens (first histogram in

Figure 9), the edge probability based on POS tags (second histogram in Figure 9) for the *Y axis*, and the normalised token distance (third histogram in Figure 9) as the *Z axis*. One evident conclusion from Figure 10 is that the samples are not large, when taking into to the actual search space, yet they provide enough representational value to allow the agent to become sufficient. Considering that the search space may become larger, as more tokens (words) are introduced, the constraining factors are the fixed set of POS Tags, and the fact that token distances are scaled and normalised.

The second conclusion related to this form of data-mining via action permutations, is that the data appears to be non-linearly separable, due to the fact that the convex hull visually appears to intersect closely clustered data-points (Toussaint, 1983); we did not however use any applicable methodology (Elizondo, 2006) to verify this observation. This observation justifies the usage of a multi-layered ANN, as it is beneficial when compared to other algorithms, since it allows to efficiently map and classify data collected from data-mining post-training as *high value actions*.

6.4. Algorithm Comparison

We compared the various action algorithms employed within the cognitive agent. The baseline performance measured as a "*random walk*" was the random action controller (Section 2.3.1). All other action selection controllers were tested either isolated (e.g., the only action selection mechanism active) or fused together, in a cascading mode, starting from probabilistic, to semantic and random, or semantic to probabilistic and random. The neural action controller was tested in combination with other algorithms, but found to perform significantly better on its own, whereas

the best cascade of the remaining controllers was probabilistic, to semantic and random.

Cascading Semantic with probabilistic, and probabilistic with semantic showed that preferential execution of probabilistic before falling back to Semantic was significantly better as average graph similarity. Probabilistic execution was optimised, by combining probabilities and by using them in a preferential manner, where token probability was chosen over POS tag probability. An explanation for this phenomenon is the debate of generality over granularity; probabilities based on POS tags are presumed to be generalising action decisions, whereas token-based probabilities offer a much more granular approach, but are not always available, since the agent may be given unknown tokens or unseen tokens. Semantic action selection, when run semi-isolated (using only a random selector as a fall-back), showed a marginally better than random accuracy. As aforementioned, VSM similarity only shows that *some episodes* are similar up to a certain degree, and because the Semantics algorithm relies on finding VSM-similar episodes, it most often was unusable. Furthermore, smaller data-sets have a low average VSM similarity, thus feature vectoring was not of much use to the agent. Other data-sets had higher average VSM similarity, but also a higher graph complexity.

A comparison of the probabilistic and neural controllers provides some insight as to why neural-based approaches may in fact be more suitable than statistical-based approaches in the field of cognitive agents. The neural controller is trained using probability values and scaled distance values, but it develops the ability to filter, classify and differentiate good from bad actions, whereas the probabilistic controller generalises a lot of actions. This is evidenced by the fact that, although the ANN

controller uses both probability values and scaled/normalised arithmetic values, it outperforms the empirical probability controller. We have optimised and tried numerous experiments with probability-based controllers, including a naive Bayesian filter; however, we found that a large multilayer feed-forward neural network always provided better results.

6.5. Artificial Neural Networks as Action Controllers

Due to the nature of neural networks, and more specifically their random weight initialisation, variable performance was observed. Shuffling training data, and using *Batch Training*, combined with the optimisation techniques mentioned in Section (2.3.4), we proceeded to optimise ANNs, due to strong indications that they would outperform other algorithms. Initially, the ANNs were not consistent throughout experiments, and were trained on-the-fly, right after data-mining. This induced a very generalised classification of actions, which proved inefficient and inaccurate. Eventually we accumulated data during multiple data-mining passes and gained a larger training sample for the ANN.

What Figure 11 demonstrates, is the progression from a small and generalising ANN, which used a small training sample, towards a fine tuned (and large) ANN, which used a highly representational training sample. The X axes represent the *POS Tag probability values*, the Z axes represent the *token/node distance* within a sentence (normalised and scaled to -1 and 1) and the Y axes show the *token edge probabilities* (formula (6)). That is the same data acquired during data-mining (Section 3.2, 6.3, Figure 10), with the action value superimposed as a colour. The Action value is the fourth dimension, described by the heat-map of the graph, where

green are *correct* actions, and red *incorrect* actions. The early ANN was small (less than 10 neurons) and used a small training sample, but as the agent kept data-mining, larger training samples were acquired. The steepness of the angle in the ANN shows the token distance sampling, whereas the spikes and crevices show particular areas of extreme token distance samples. The final ANN was quite large (300 neurons) and with 3 layers, of which one is a hidden layer.

[Figure 11 about here.]

We proceeded to train multiple neural networks, and used the best optimised one for all experiments, regardless of the data-set. The *best* ANNs for our purpose were empirically selected, not only via the hyper-parametrisation techniques aforementioned, but though cross-validation by experimentation using various data-sets as input. We avoided over-training by using early stopping (Yao et al., 2007), whilst retaining good generalisation. Due to the nature of optimising hyper-parameters and ANN architecture, it is plausible that the reported accuracy may further be improved.

The optimal neural network was selected out of a group of many networks, because it provided consistently the best results across *all* data-sets with which it was tested. Continuous updating of the probabilities look-up table enabled us to create a very large training sample set for the ANNs. Furthermore, by iterating the episodic memory of the agent, and trying all sorts of permutations as operations on graphs, and then filtering the *known high Q-value actions*, we were able to create an accurate map of high value actions and associate them with input and episodes. Figure 11 shows that only a small amount of high probability values for edge tokens

(Y axis) and only a small fraction of high probability POS Tag edges (X axis) are associated with *high value* $Q(s_t, a_t)$ *policies*.

6.6. Comparison to State-of-the-Art

In order to determine how well the cognitive agent performed, in comparison to some of the most recent and related research, we have used the reported accuracy by their respective authors, as shown in Table (3).

[Table 3 about here.]

Most of the results provided in Table (3) are obtained by using *annotated* or formatted data. This needs to be emphasised, as we did not annotate data (with the exception of the POS-tagging), nor did we use utterances, question-answer scenarios, bi-grams or tri-grams. Our data was partitioned, with *input text length* ranging from 4 to 30 words, as resulted from the harvested data sets. Choi et al mention that most parsers have a UAS accuracy of 93.49 to 95.5 for sentences under 10 terms, which declines to 81.66 and 86.61 for sentences larger than 50 terms. Examining Table (3), (Zhong et al., 2011) used a manual template model, and reported 85% to 88% accuracy on concept entities, and 74% to 95% accuracy on relations (possesive, function, broader, style, content). Our average entity accuracy was higher (96.90%). Moreover, Zhong et al. do not mention conceptual graph accuracy, nor do they provide information about their datasets, or the edge accuracy. In comparison, our cognitive agent combines unsupervised learning (reinforcement learning via examples), with semi-supervised learning (for the ANN), and self-supervised (data-mining) in order to achieve a very high graph similarity.

Vlachos et al, in their various research studies (Vlachos, 2012; Andreas et al., 2013; Vlachos and Clark, 2014) used a variety of models, techniques and data-sets. Most notable is their action selection using *reinforcement learning*, which was average (reported 55%). However, they used the *question-answer* scenario with utterances, which is different from conceptual graph creation, which would be considered an easier type of data to learn. Furthermore, the data-sets they used are fundamentally different, as they are annotated and use MRLs instead of conceptual graphs. A parallelism that can be drawn upon comparison to their work is that node prediction, constant argument prediction, string argument prediction, node argument prediction, focus/negation prediction, are processes/algorithms similar to conceptual graph entity and relation recognition. Node prediction in MRL could be hypothetically substituted by graph node recognition/prediction in conceptual graphs, whereas the argument prediction is essentially the correct connection of predicates with their arguments, via edges (as per the conceptual graph literature).

Other researchers (Poon and Domingos, 2009) used similar models (neural networks) but very different approaches: they implement λ-form clusters and predefined formal meaning representations. They report an accuracy of 88%, and emphasise the usage of *Deep Learning*, a very insightful conclusion that could further aid the progression of cognitive agents. A comparison here may be the assumption that rigid rules and pre-definitions might in fact hinder accuracy, since they impose restrictions on patterns that might be exceptions to those rules.

Another comparison is with (Wong and Mooney, 2006) and the WASP statistical system. They used tree structures for argumentation (relation extraction) and achieved very similar results. Current state-of-the-art research from Berkley (Durrett

and Klein, Durrett and Klein) reports 90.97% F_1 score, which mentions using text sizes of up to 40 words (albeit without any further details, such as average sentence word length). They use neural-based models and train them on *anchors* using various features. Taking into consideration that their average input size is unknown, we can conclude that a cognitive agent such as the one described here, manages to marginally surpass their accuracy, if the average sentence length is similar. Other researchers (Socher et al., 2011) using recurrent neural networks (RNN) also report similar results, a strong indication that neural-based models can recognize, associate and learn, not only specific relations which statistical models also can learn, but patterns (e.g., *anchors*) and recurring graph structures.

Finally, we need to stress the fact that the above comparison serves only as a reference point. The aforementioned research is the closest reference to a cognitive agent performing semantic parsing, and although the mechanisms, implementation and datasets are different, the comparison aims to create a valid reference to the research done in the recent past. What is of relevance in this comparison is that similar KR schemes and technologies have similar accuracy and performance. Thus it is justifiable to presume that ML-based cognitive agent systems are comparable to heuristic semantic parsing for *Natural language processing*.

Furthermore, the agent described is not bound by rules, logic forms or syntax, but, on the contrary, is designed to learn and identify patterns arising from training sets in a self-constrained manner.

7. DISCUSSIONS AND CONCLUSIONS

In this paper, we have proposed a cognitive agent which is capable of learning to represent knowledge as conceptual graphs, in other words, perform semantic parsing, based on machine learning, with limited prior knowledge. Not only is this possible, but the agent also learns by examples from humans, with a good accuracy in the same ballpark as current state-of-the-art. Moreover, we have demonstrated the feasibility of this approach on realistic input sentences of different sizes (including large sized ones, unlike in some of the prior research), from real-life text repositories. Such an approach, as witnessed by the recent commercialisation of AI by tech giants (IBM, Amazon, NVIDIA, Google, Apple, Microsoft, etc) can be applicable in personal assistants, knowledge assistants, expert systems, predictive analytic agents, chatbots and dialogue systems, and many more. Furthermore, because this approach performs real-time (during input propagation), it has the potential to be used in real applications with some further optimisation, such as training with big data in order to increase its accuracy when dealing with large sentences.

Concluding, we can say that the unique properties of conceptual graphs, their adaptability and mathematical properties, are an advantage to semantic parsing. Templates, rules and expressions often hard-coded in rule-based natural language processing (NLP), can instead be identified and learnt, and heuristic rules and phenomena, such as functions with arguments, exist as relations and the edges within a graph. However, all that added mutability and adaptability of conceptual graphs comes at certain cost; their creation is just as complex as MRL and other KR structures, if not more complex, due to the absence of constrains and rules. Furthermore, because

CG are bipartite graphs, failure to correctly classify a word or token as a *relation* or *concept*, propagates that error to the *edge creation*. We believe this to be a limitation of CG, which reduces the overall accuracy; the assumption being that the same agent using MRL could potentially have a higher accuracy, especially when dealing with larger sentences, which is when we've witnessed both node and edge accuracy decreasing, the first affecting the latter.

The ability to manipulate knowledge at such a low-level as that of a mathematical construct (a graph) creates new obstacles and questions: how does knowledge complexity affect KR and semantic parsing, what underlying properties and attributes of the Natural Language are usable and which aren't, and many more questions.

We believe that the current performance of the cognitive agent may be further limited by the fact that it tries to apply general probabilities (even through ANN filtering and classification) and that is a justified but somewhat limiting approach. We arrived to this conclusion, by observing how larger and more complex input makes the agent less accurate.

The intrinsic value of empirical probabilities is abstracting, and the probability values of specific tokens become too granular, thus focusing too much on specific words and their edges, rather than identifying graph patterns, and sub-patterns. Part of our future planned work is to consider enabling the agent to detect graph patterns and features using deep learning and sparse encoding, and ignore generalising statistical properties.

The current agent, as described and implemented, showcases a clearly novel approach: constructing representations as a temporal-spatial sequence is indeed a subject of machine learning, even at the most symbolic level of the information. Such

a notion has not been explored before, where the norm was to use feature vectors. Furthermore, the approach of combining Machine Learning (here, reinforcement learning and ANN) with KR construction is not limited to conceptual graphs; any type of KR model as well as logic representation could potentially be learnt using this approach. It has been shown that MRL and RDF can be translated to CG, and as such it is plausible that even without a graph being the representation, a spatial-temporal sequence could *Markovianly* describe another KR form. This has the added benefit that the agent or system implementing such an approach is *learning* the actual spatial-temporal process.

The gravity of that notion transcends semantic parsing, as the cognitive agent moves beyond traditional heuristics or neural models, into an area where demonstration is observed, learnt, analysed and then reused, in order to keep learning indefinitely in an unsupervised manner. Furthermore, due to the nature of neural-dynamic programming, and its basis upon human brain physiology (Bertsekas and Tsitsiklis, 1995), we avoid heuristic constrains and other finite-state related issues (such as uncertainty).

This agent is a step towards autonomous cognitive agents, able not only to *read* and *understand* information and knowledge, but perform other types of high level cognitive functions, using the same spatial-temporal ML approach. The main contribution of this work is that a neuro-dynamic agent, learnt to do semantic parsing via observation alone. This is, in essence, a cognitive agent, learning how to read text, only by being ”shown” by a human user - similar to how children learn how to read.

REFERENCES

ABADI, MARTÍN, ASHISH AGARWAL, PAUL BARHAM, EUGENE BREVDO, ZHIFENG CHEN, CRAIG CITRO, GREG S CORRADO, ANDY DAVIS, JEFFREY DEAN, MATTHIEU DEVIN, and OTHERS. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *In* arXiv preprint arXiv:1603.04467.

ANDOR, DANIEL, CHRIS ALBERTI, DAVID WEISS, ALIAKSEI SEVERYN, ALESSANDRO PRESTA, KUZMAN GANCHEV, SLAV PETROV, and MICHAEL COLLINS. 2016. Globally normalized transition-based neural networks. *In* arXiv preprint arXiv:1603.06042.

ANDREAS, JACOB, ANDREAS VLACHOS, and STEPHEN CLARK. 2013. Semantic parsing as machine translation. *In* ACL (2), pp. 47–52.

BACH, JOSCHA. 2009. Principles of synthetic intelligence PSI: an architecture of motivated cognition, Volume 4. Oxford University Press.

BAROOAH, PRABIR, and JOÃO P HESPANHA. 2007. Estimation on graphs from relative measurements. Control Systems, IEEE, **27**(4):57–74.

BELLMAN, RICHARD. 1957. A markovian decision process. Technical report, DTIC Document.

BERTSEKAS, DIMITRI P, and JOHN N TSITSIKLIS. 1995. Neuro-dynamic programming: an overview. *In* Decision and Control, 1995., Proceedings of the 34th IEEE Conference on, Volume 1, IEEE, pp. 560–564.

BRODERSEN, KAY H, CHENG SOON ONG, KLAAS E STEPHAN, and JOACHIM M BUHMANN. 2010. The balanced accuracy and its posterior distribution. *In* Pattern Recognition (ICPR), 2010 20th International Conference on, IEEE, pp. 3121–3124.

CAMPBELL, KEITH E, and MARK A MUSEN. 1992. Representation of clinical data using snomed iii and conceptual graphs. *In* Proceedings of the Annual Symposium on Computer Application in Medical Care, American Medical Informatics Association, p. 354.

CAVAZZUTI, MARCO. 2012. Optimization Methods: From Theory to Design Scientific and Technological Aspects in Mechanics. Springer Science & Business Media.

CHAZELLE, BERNARD. 1993. An optimal convex hull algorithm in any fixed dimension. Discrete & Computational Geometry, **10**(1):377–409.

CHEIN, MICHEL, and MARIE-LAURE MUGNIER. 2008. Graph-based knowledge representation: computational foundations of conceptual graphs. Springer Science & Business Media.

CHEN, DAVID L., and RAYMOND J. MOONEY. 2008. Learning to sportscast: A test of grounded language acquisition. *In* Proceedings of 25th International Conference on Machine Learning (ICML-2008), Helsinki, Finland.

CHOI, JINHO D, JOEL TETREAULT, and AMANDA STENT. 2015. It depends: Dependency parser comparison using a web-based evaluation tool. *In* Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL, pp. 26–31.

CLARKE, DAOUD. 2015. Simple, fast semantic parsing with a tensor kernel. *In* arXiv preprint arXiv:1507.00639.

CORMEN, THOMAS H.., CHARLES ERIC LEISERSON, RONALD L RIVEST, and CLIFFORD STEIN. 2001. Introduction to algorithms, Volume 6. MIT press Cambridge.

COXETER, HAROLD SCOTT MACDONALD. 1973. Regular polytopes. Courier Corporation.

CRAVEN, MARK, JOHAN KUMLIEN, and OTHERS. 1999. Constructing biological knowledge bases by extracting information from text sources. *In* ISMB, Volume 1999, pp. 77–86.

CROITORU, MADALINA, BO HU, SRINANDAN DASMAHAPATRA, PAUL LEWIS, DAVID DUPPLAW, ALEX GIBB, MARGARIDA JULIA-SAPE, JAVIER VICENTE, CARLOS SAEZ, JUAN MIGUEL GARCIA-GOMEZ, and OTHERS. 2007. Conceptual graphs based information retrieval in health agents. *In* Computer-Based Medical Systems, 2007. CBMS'07. Twentieth IEEE International Symposium on, IEEE, pp. 618–623.

DAVID L. CHEN, JOOHYUN KIM, RAYMOND J. MOONEY. 2010. Training a multilingual sportscaster: Using perceptual context to learn language. Journal of Artificial Intelligence Research, **37**:397–435.

DUAN, YAN, MARCIN ANDRYCHOWICZ, BRADLY STADIE, JONATHAN HO, JONAS SCHNEIDER, ILYA SUTSKEVER, PIETER ABBEEL, and WOJCIECH ZAREMBA. 2017. One-shot imitation learning. *In* arXiv preprint arXiv:1703.07326.

DURRETT, GREG, and DAN KLEIN. Neural crf parsing.

ELIZONDO, DAVID. 2006. The linear separability problem: Some testing methods. Neural Networks, IEEE Transactions on, **17**(2):330–344.

EVEN-DAR, EYAL, and YISHAY MANSOUR. 2004. Learning rates for q-learning. The Journal of Machine Learning Research, **5**:1–25.

FELLBAUM, CHRISTIANE. 1998. WordNet. Wiley Online Library.

GALEF JR, BENNETT G. 1988. Imitation in animals: history, definition, and interpretation of data from the psychological laboratory. Social learning: Psychological and biological perspectives, **28**.

GKIOKAS, ALEXANDROS. 2016. Imitation learning in artificial intelligence. Ph. D. thesis, University of Warwick.

GORRELL, PAUL. 2006. Syntax and parsing, Volume 76. Cambridge University Press.

GREFENSTETTE, EDWARD, PHIL BLUNSOM, NANDO DE FREITAS, and KARL MORITZ HERMANN. 2014. A

deep architecture for semantic parsing. *In* Proceedings of the ACL 2014 Workshop on Semantic Parsing.

HAGAN, MARTIN T, HOWARD B DEMUTH, MARK H BEALE, and OTHERS. 1996. Neural network design. Pws Pub. Boston.

HAIKONEN, PENTTI OA. 2009. Qualia and conscious machines. International Journal of Machine Consciousness, **1**(02):225–234.

HAIKONEN, PENTTI O. 2012. Consciousness and robot sentience, Volume 2. World Scientific.

HEMPHILL, CHARLES T, JOHN J GODFREY, and GEORGE R DODDINGTON. 1990. The atis spoken language systems pilot corpus. *In* Proceedings of the DARPA speech and natural language workshop, pp. 96–101.

HIGGINS, JAMES J. 2003. Introduction to modern nonparametric statistics.

HONNIBAL, MATTHEW, and MARK JOHNSON. 2015. An improved non-monotonic transition system for dependency parsing. *In* Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, p. 13731378.

HOWARD, RONALD A. 1970. Dynamic programming and Markov processes. MIT Press.

INTAN, ROLLY, CHI-HUNG CHI, HENRY N PALIT, and LEO W SANTOSO. 2015. Intelligence in the Era of Big Data: 4th International Conference on Soft Computing, Intelligent Systems, and Information Technology, ICSIIT 2015, Bali, Indonesia, March 11-14, 2015. Proceedings, Volume 516. Springer.

JURAFSKY, DAN, and JAMES H MARTIN. 2000. Speech & language processing. Pearson Education India.

KAMSU-FOGUEM, BERNARD, and VINCENT CHAPURLAT. 2006. Requirements modelling and formal analysis using graph operations. International Journal of Production Research, **44**(17):3451–3470.

KAMSU-FOGUEM, BERNARD, FABIEN RIGAL, and FÉLIX MAUGET. 2013. Mining association rules for the quality improvement of the production process. Expert systems with applications, **40**(4):1034–1045.

LAWNICZAK, ANNA T, and BRUNO N DI STEFANO. 2010. Computational intelligence based architecture for cognitive agents. Procedia Computer Science, **1**(1):2227–2235.

LIANG, PERCY, and CHRISTOPHER POTTS. 2015. Bringing machine learning and compositional semantics together. Annu. Rev. Linguist., **1**(1):355–376.

LIEBERMAN, HENRY. 2001. Your wish is my command: Programming by example. Morgan Kaufmann.

MARCUS, MITCHELL P, MARY ANN MARCINKIEWICZ, and BEATRICE SANTORINI. 1993. Building a large annotated corpus of english: The penn treebank. Computational linguistics, **19**(2):313–330.

MARTINS, ANDRÉ FT, MIGUEL B ALMEIDA, and NOAH A SMITH. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. *In* Proceedings of the Conference, p. 617.

MATSUMOTO, MAKOTO, and TAKUJI NISHIMURA. 1998. Mersenne twister: a 623-dimensionally equidis-

tributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS), **8**(1):3–30.

MONTES-Y GÓMEZ, MANUEL, ALEXANDER GELBUKH, and AURELIO LÓPEZ-LÓPEZ. 2002. Text mining at detail level using conceptual graphs. *In* Conceptual Structures: Integration and Interfaces. Springer, pp. 122–136.

NEHANIV, CHRYSTOPHER L, and KERSTIN DAUTENHAHN. 2007. Imitation and social learning in robots, humans and animals: behavioural, social and communicative dimensions. Cambridge University Press.

NISSEN, STEFFEN. 2003. Implementation of a fast artificial neural network library (fann). Report, Department of Computer Science University of Copenhagen (DIKU), **31**.

PETROV, SLAV. 2016. Announcing syntaxnet: The worlds most accurate parser goes open source. *In* Google Research Blog.

POON, HOIFUNG, and PEDRO DOMINGOS. 2009. Unsupervised semantic parsing. *In* Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1, EMNLP '09, Association for Computational Linguistics, Stroudsburg, PA, USA. ISBN 978-1-932432-59-6. pp. 1–10. http://dl.acm.org/citation.cfm?id=1699510.1699512.

POPESCU, ANA-MARIA, OREN ETZIONI, and HENRY KAUTZ. 2003. Towards a theory of natural language interfaces to databases. pp. 149–157.

PRADHAN, SAMEER S, WAYNE WARD, KADRI HACIOGLU, JAMES H MARTIN, and DANIEL JURAFSKY. 2004. Shallow semantic parsing using support vector machines. *In* HLT-NAACL, pp. 233–240.

RASLI, RUZIANA BINTI MOHAMAD, FAUDZIAH AHMAD, and SITI SAKIRA KAMARUDDIN. 2014. A comparative study of conceptual graph and concept map. Journal of Engineering and Applied Sciences, **9**(9):1442–1446.

REAL, RAIMUNDO, and JUAN M VARGAS. 1996. The probabilistic basis of jaccard's index of similarity. *In* Systematic biology, pp. 380–385.

RIJSBERGEN, C. J. VAN. 1979. Information Retrieval (2nd ed.). Butterworth-Heinemann, Newton, MA, USA. ISBN 0408709294.

RUIZ, PAULA POTES, BERNARD KAMSU FOGUEM, and BERNARD GRABOT. 2014. Generating knowledge in maintenance from experience feedback. Knowledge-Based Systems, **68**:4–20.

SAGAE, KENJI. 2009. Analysis of discourse structure with syntactic dependencies and data-driven shift-reduce parsing. *In* Proceedings of the 11th International Conference on Parsing Technologies, Association for Computational Linguistics, pp. 81–84.

SAGAE, KENJI, and ALON LAVIE. 2006. A best-first probabilistic shift-reduce parser. *In* Proceedings of the COLING/ACL on Main conference poster sessions, Association for Computational Linguistics, pp. 691–698.

SHEELA, K GNANA, and SN DEEPA. 2013. Review on methods to fix number of hidden neurons in neural networks. Mathematical Problems in Engineering, **2013**.

SHI, LEI, and RADA MIHALCEA. 2005. Putting pieces together: Combining framenet, verbnet and wordnet for robust semantic parsing. *In* Computational linguistics and intelligent text processing. Springer, pp. 100–111.

SHIEBER, STUART M. 1983. Sentence disambiguation by a shift-reduce parsing technique. *In* Proceedings of the 21st annual meeting on Association for Computational Linguistics, Association for Computational Linguistics, pp. 113–118.

SOCHER, RICHARD, CLIFF C LIN, CHRIS MANNING, and ANDREW Y NG. 2011. Parsing natural scenes and natural language with recursive neural networks. *In* Proceedings of the 28th international conference on machine learning (ICML-11), pp. 129–136.

SOWA, JOHN F. 1999. Knowledge representation: logical, philosophical, and computational foundations. Course Technology.

STACKEXCHANGE. 2015. How to choose the number of hidden layers and nodes in a feed-forward neural network? `http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw/1097#1097`. [Online; accessed 2015-09-30].

SUTTON, RICHARD STUART. 1984. Temporal credit assignment in reinforcement learning. Ph. D. thesis, University of Massachusetts Amherst.

SUTTON, RICHARD S., and ANDREW G. BARTO. 1998. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA. ISBN 0-262-19398-1.

TANG, LAPPOON R, and RAYMOND J MOONEY. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. *In* Machine Learning: ECML 2001. Springer, pp. 466–477.

TOUSSAINT, GODFRIED T. 1983. Solving geometric problems with the rotating calipers. *In* Proc. IEEE Melecon, Volume 83, p. A10.

TSURUOKA, YOSHIMASA, YUSUKE MIYAO, and JUN'ICHI KAZAMA. 2011. Learning with lookahead: can history-based models rival globally optimized models? *In* Proceedings of the Fifteenth Conference on Computational Natural Language Learning, Association for Computational Linguistics, pp. 238–246.

TURNEY, PETER D, PATRICK PANTEL, and OTHERS. 2010. From frequency to meaning: Vector space models of semantics. Journal of artificial intelligence research, **37**(1):141–188.

VLACHOS, ANDREAS. 2012. An investigation of imitation learning algorithms for structured prediction. *In* EWRL, Citeseer, pp. 143–154.

VLACHOS, ANDREAS, and STEPHEN CLARK. 2014. A new corpus and imitation learning framework for context-dependent semantic parsing. Transactions of the Association for Computational Linguistics, **2**:547–559.

WATKINS, CHRISTOPHER JOHN CORNISH HELLABY. 1989. Learning from delayed rewards. Ph. D. thesis, University of Cambridge England.

WEISS, DAVID, CHRIS ALBERTI, MICHAEL COLLINS, and SLAV PETROV. 2015. Structured training for neural network transition-based parsing. *In* arXiv preprint arXiv:1506.06158.

WINER, BEN JAMES, DONALD R BROWN, and KENNETH M MICHELS. 1971. Statistical principles in experimental design, Volume 2. McGraw-Hill New York.

WINSBERG, ERIC. 2003. Simulated experiments: Methodology for a virtual world. Philosophy of science, **70**(1):105–125.

WOLD, SVANTE, KIM ESBENSEN, and PAUL GELADI. 1987. Principal component analysis. Chemometrics and intelligent laboratory systems, **2**(1):37–52.

WONG, YUK WAH, and RAYMOND J. MOONEY. 2006. *In* Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, HLT-NAACL '06, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 439–446.

WRIGHT, E MAITLAND. 1977. The number of connected sparsely edged graphs. Journal of Graph Theory, **1**(4):317–330.

YAO, YUAN, LORENZO ROSASCO, and ANDREA CAPONNETTO. 2007. On early stopping in gradient descent learning. Constructive Approximation, **26**(2):289–315. ISSN 0176-4276. . http://dx.doi.org/10.1007/s00365-006-0663-2.

ZHANG, HAO, and RYAN MCDONALD. 2012. Generalized higher-order dependency parsing with cube pruning. *In* Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Association for Computational Linguistics, pp. 320–331.

ZHANG, YUAN, TAO LEI, REGINA BARZILAY, TOMMI JAAKKOLA, and AMIR GLOBERSON. 2014. Steps to excellence: Simple inference with refined scoring of dependency trees. *In* Proceedings of the 52nd Annual

Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, pp. 197–207.

ZHONG, MAOSHENG, JIANYONG DUAN, and JIAN ZOU. 2011. Indexing conceptual graph for abstracts of books. *In* Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on, Volume 3, IEEE, pp. 1816–1820.

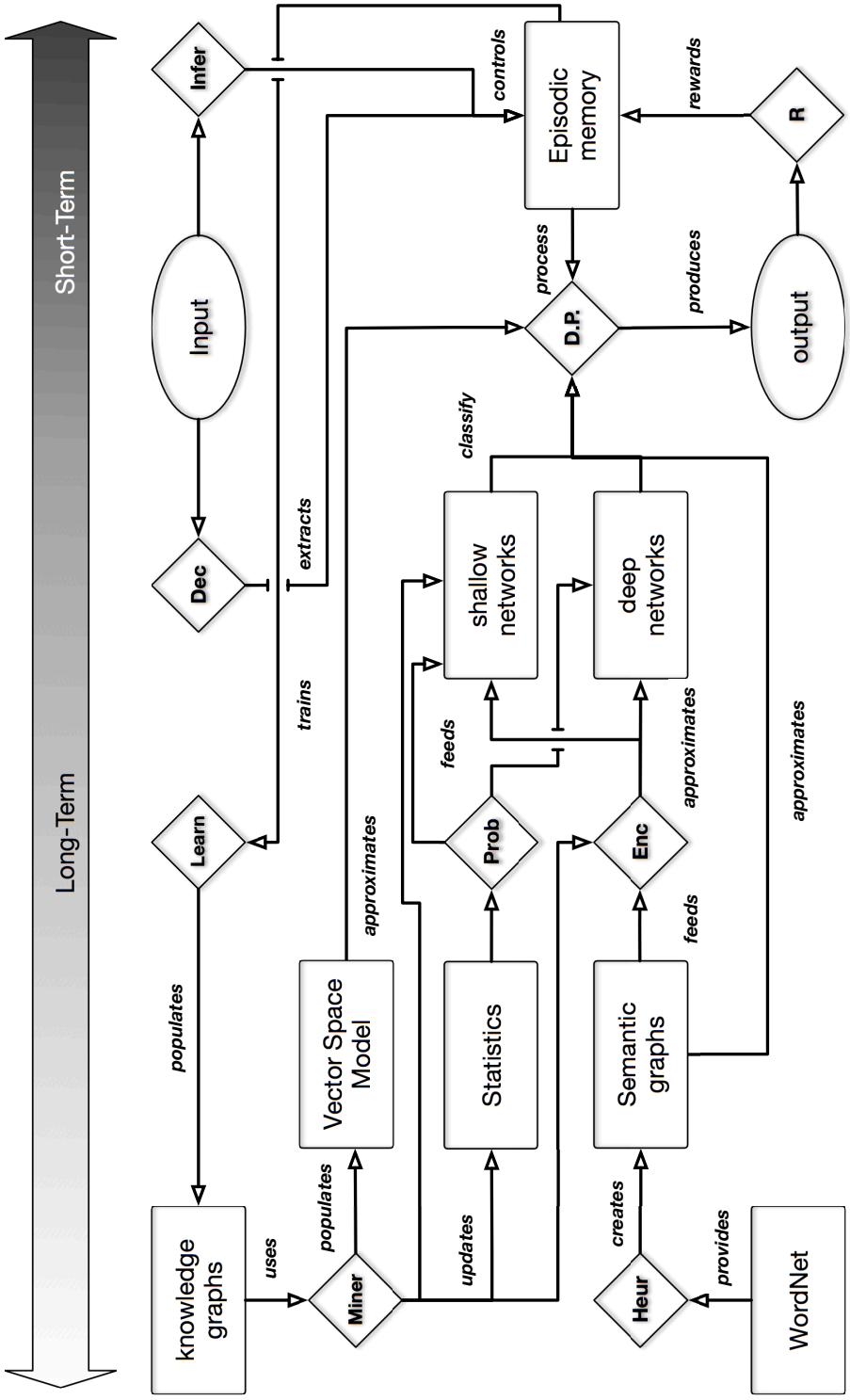

FIGURE 1: Agent Schema: overall components of the agent and how they connect with each-other.

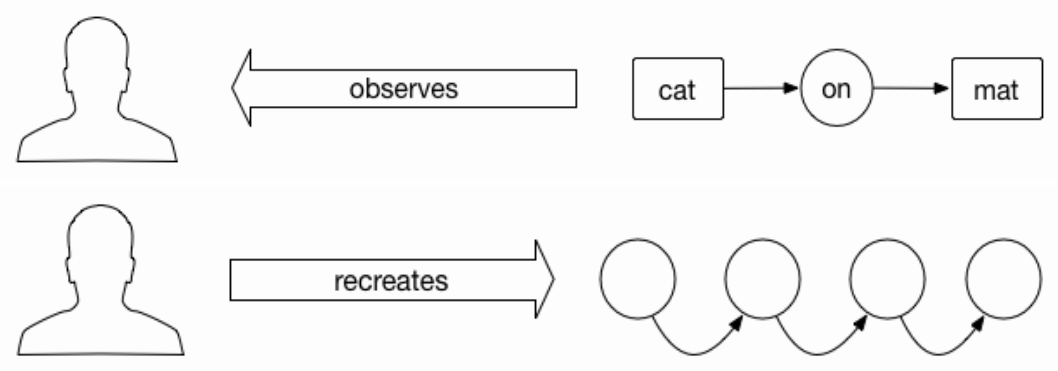

FIGURE 2: Agent Observation: A graph is observed and then recreated.

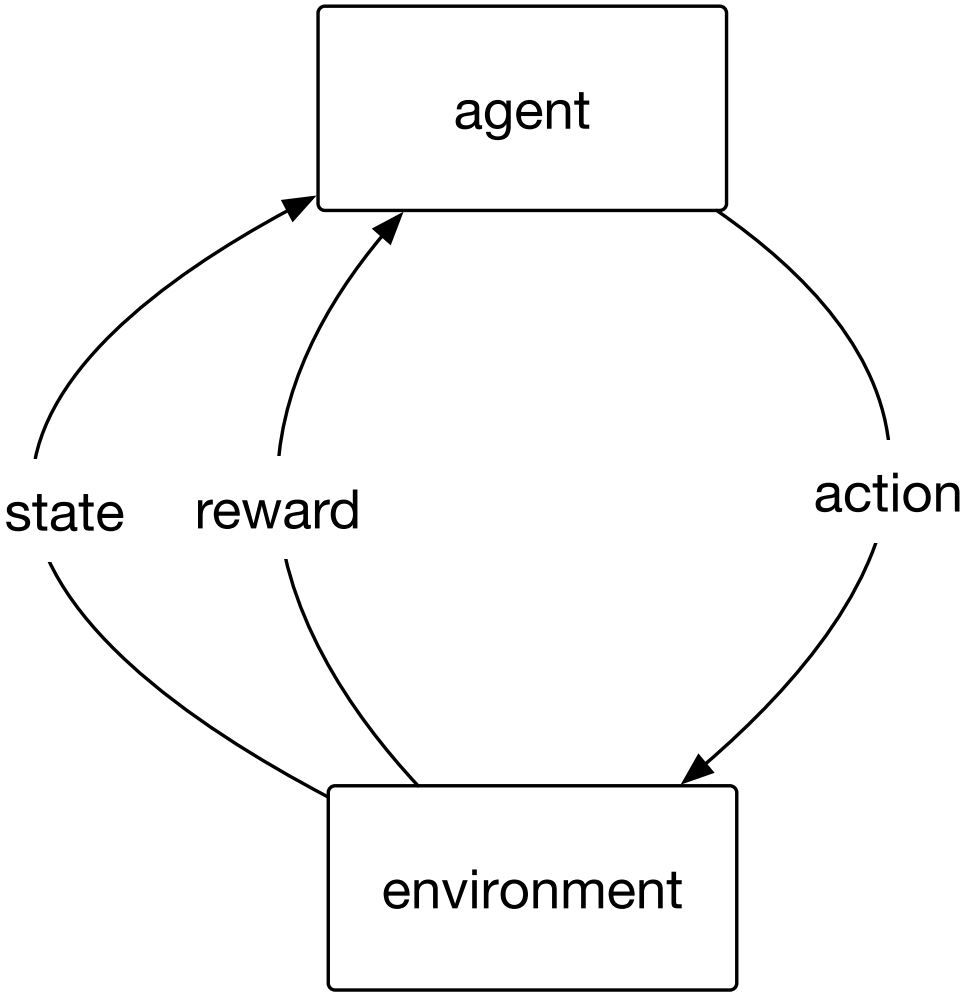

FIGURE 3: Agent Reward: Temporal MDP is rewarded.

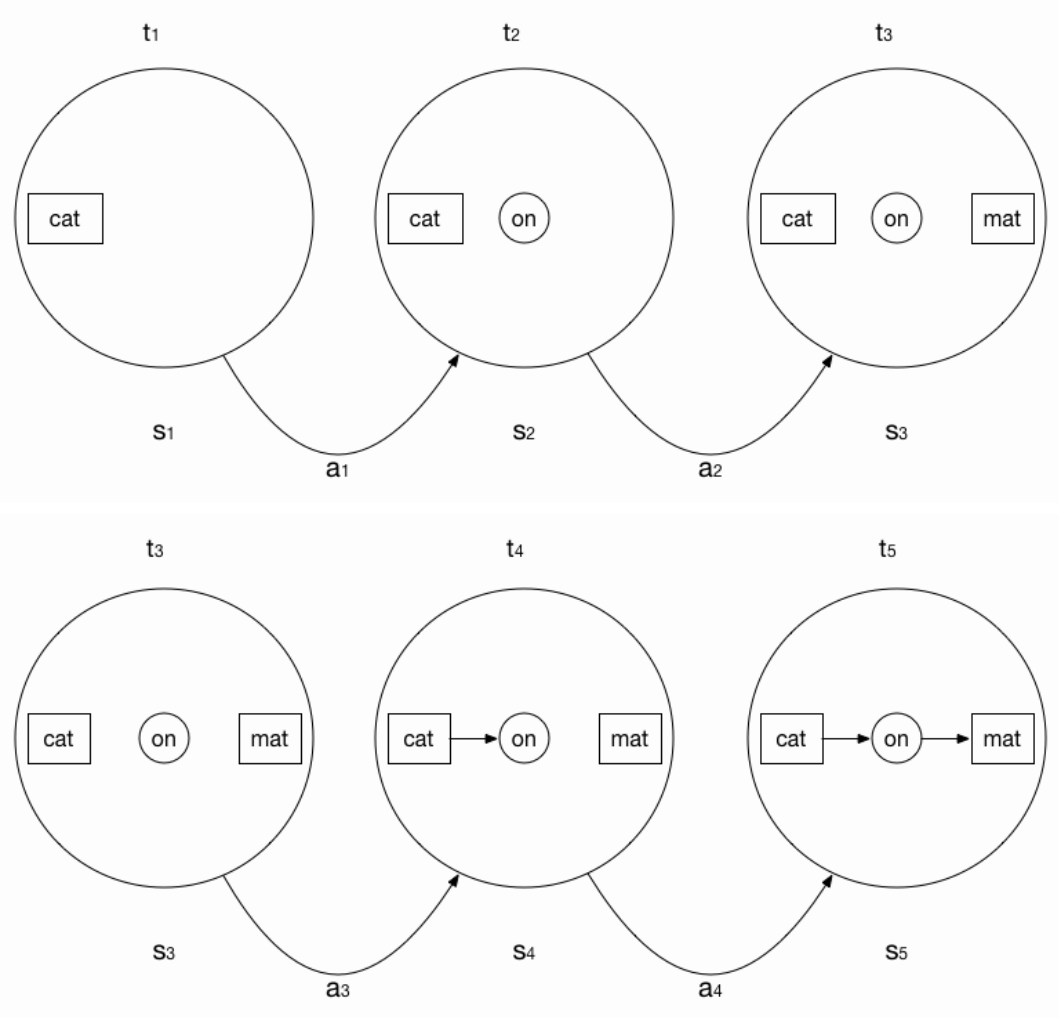

FIGURE 4: CG population: Temporal Shift-Reduce populates the graph with nodes and connects them.

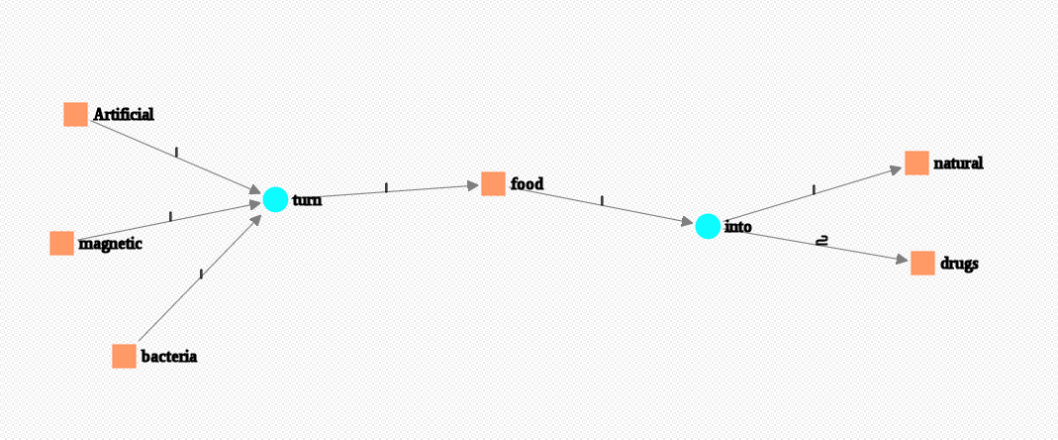

FIGURE 5: Graph Example: A simple linear graph.

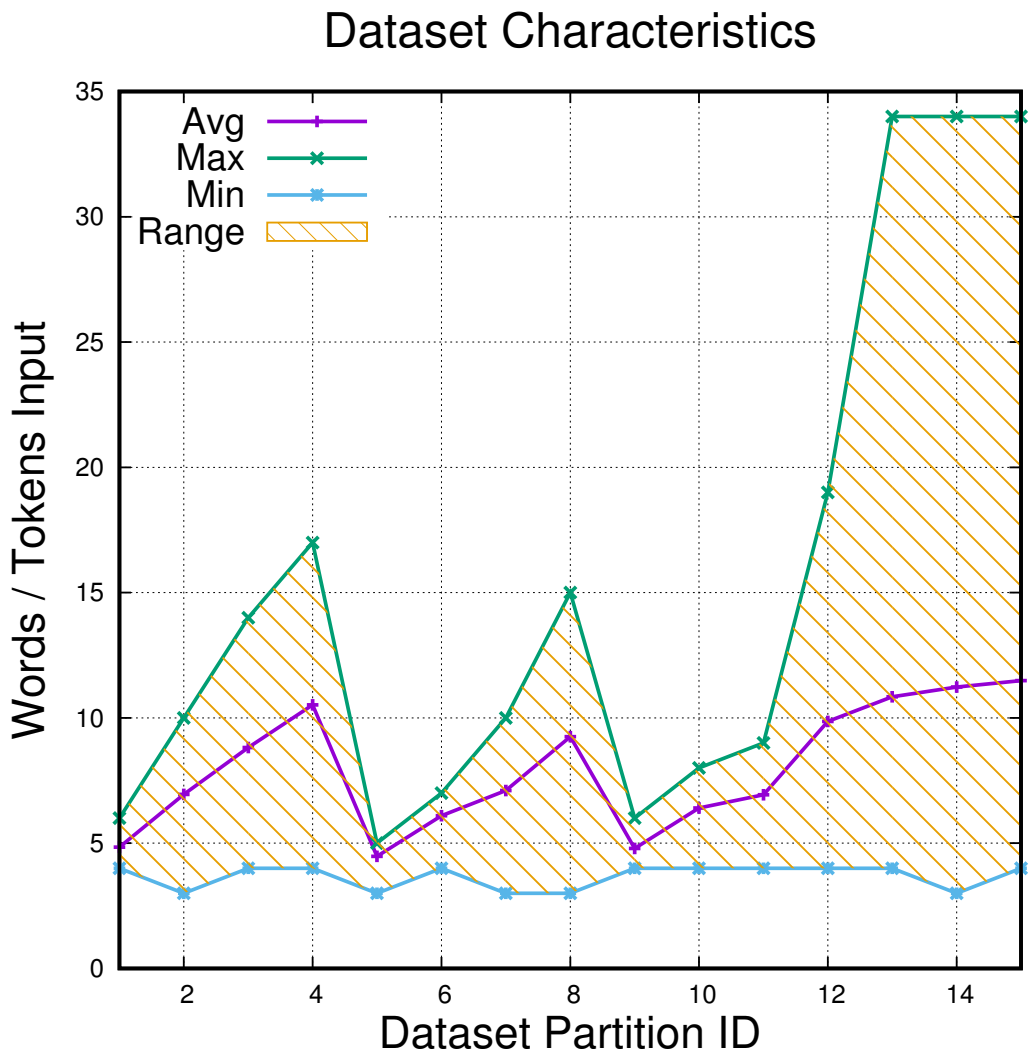

FIGURE 6: Dataset Characteristics: Subset Partitioning, Average Sentence Size, Average Minimum Size, Average Maximum Size.

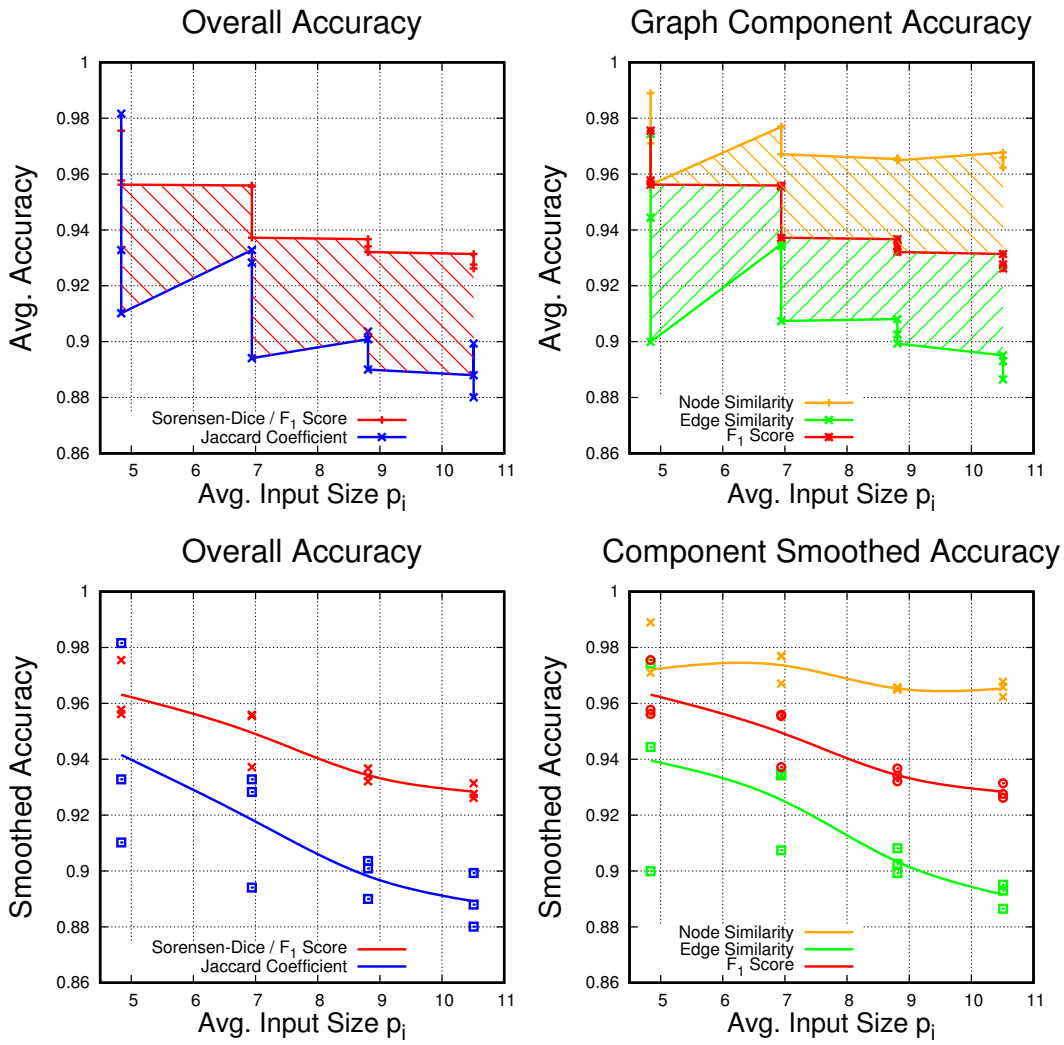

FIGURE 7: Agent Accuracy: correlation of input complexity (average sentence length) to different datasets.

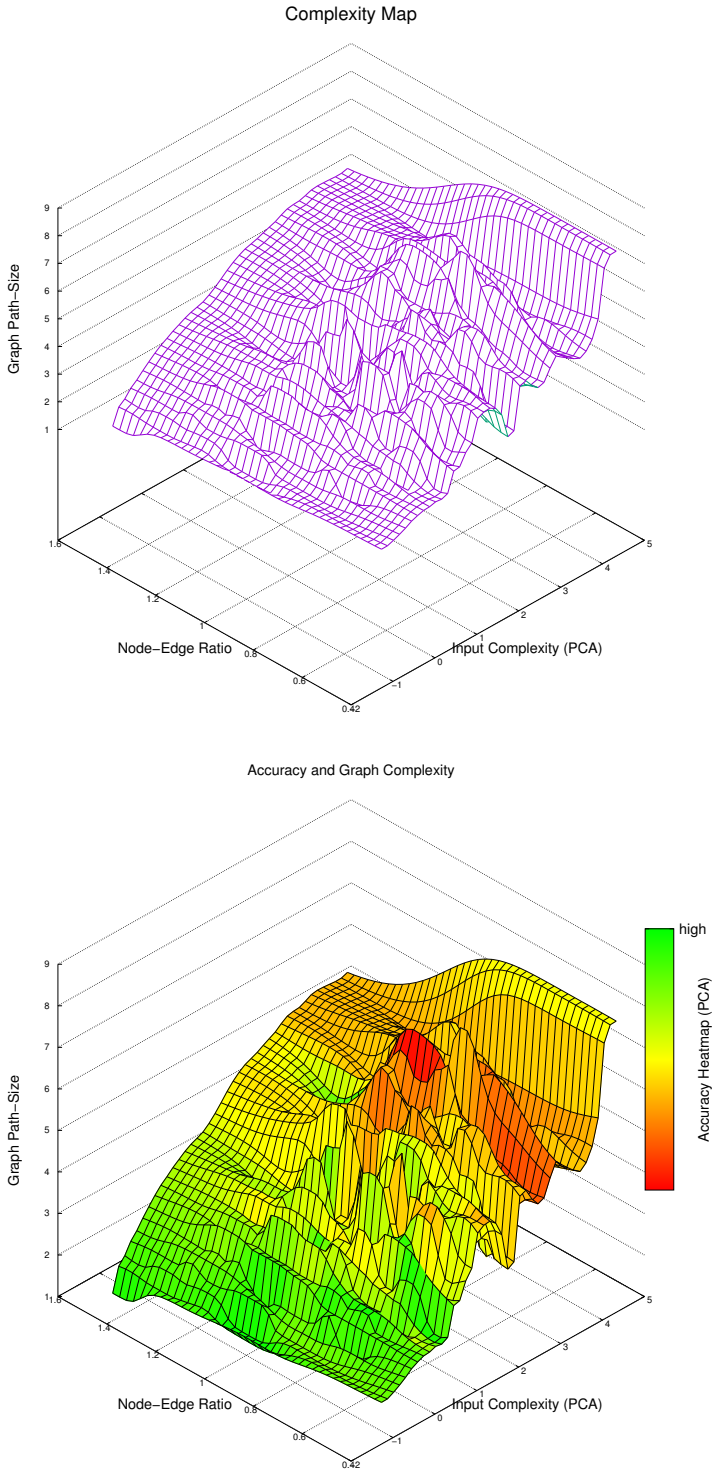

FIGURE 8: Graph complexity and Agent accuracy: Correlation between Node and Edge sizes, Average Sentence Size, and Sorensen/Jaccard PCA Accuracy.

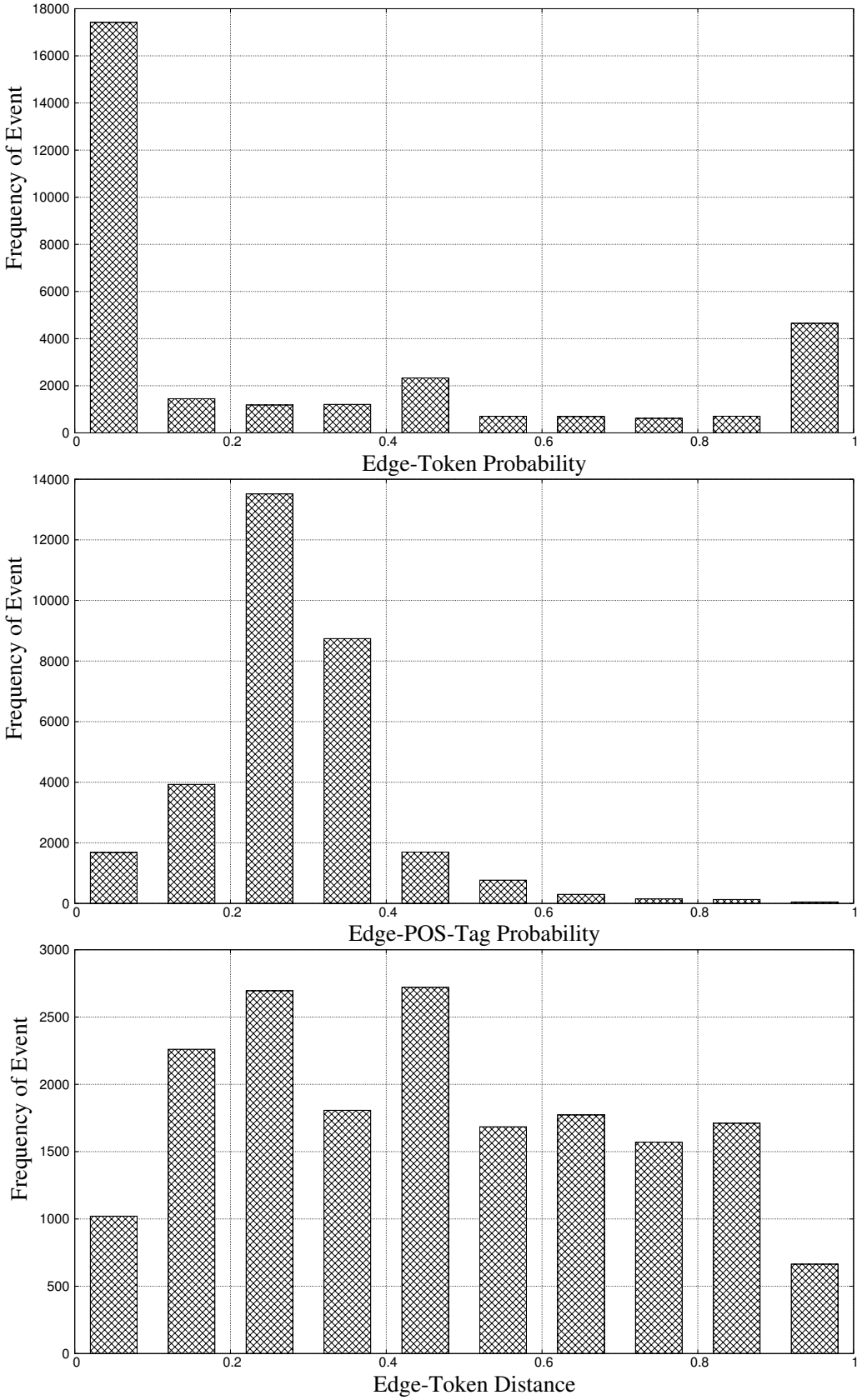

FIGURE 9: Probability Distribution Histograms: Distribution of event-frequency metrics data-mined by the agent.

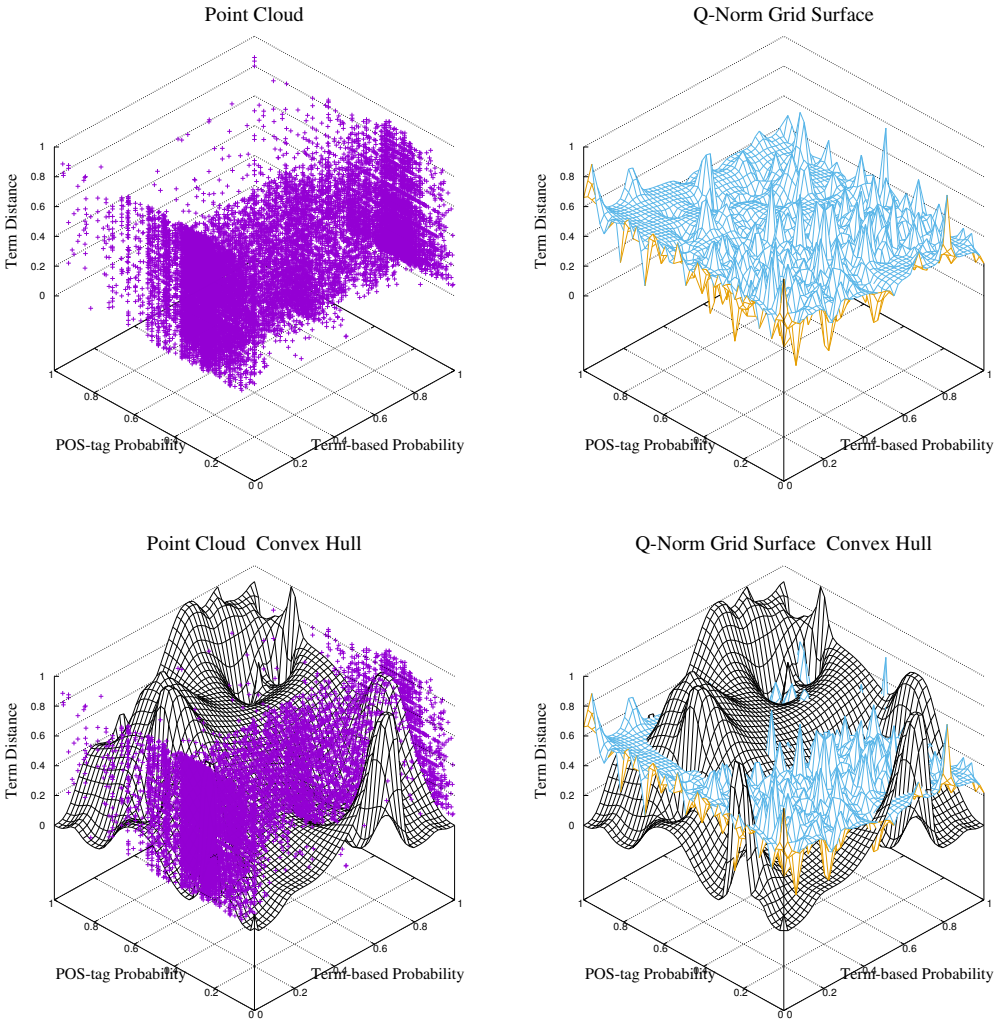

FIGURE 10: Probability Distribution Cloud and 3D Surface Q-normalised: Demonstration of non-linear separability of the state space.

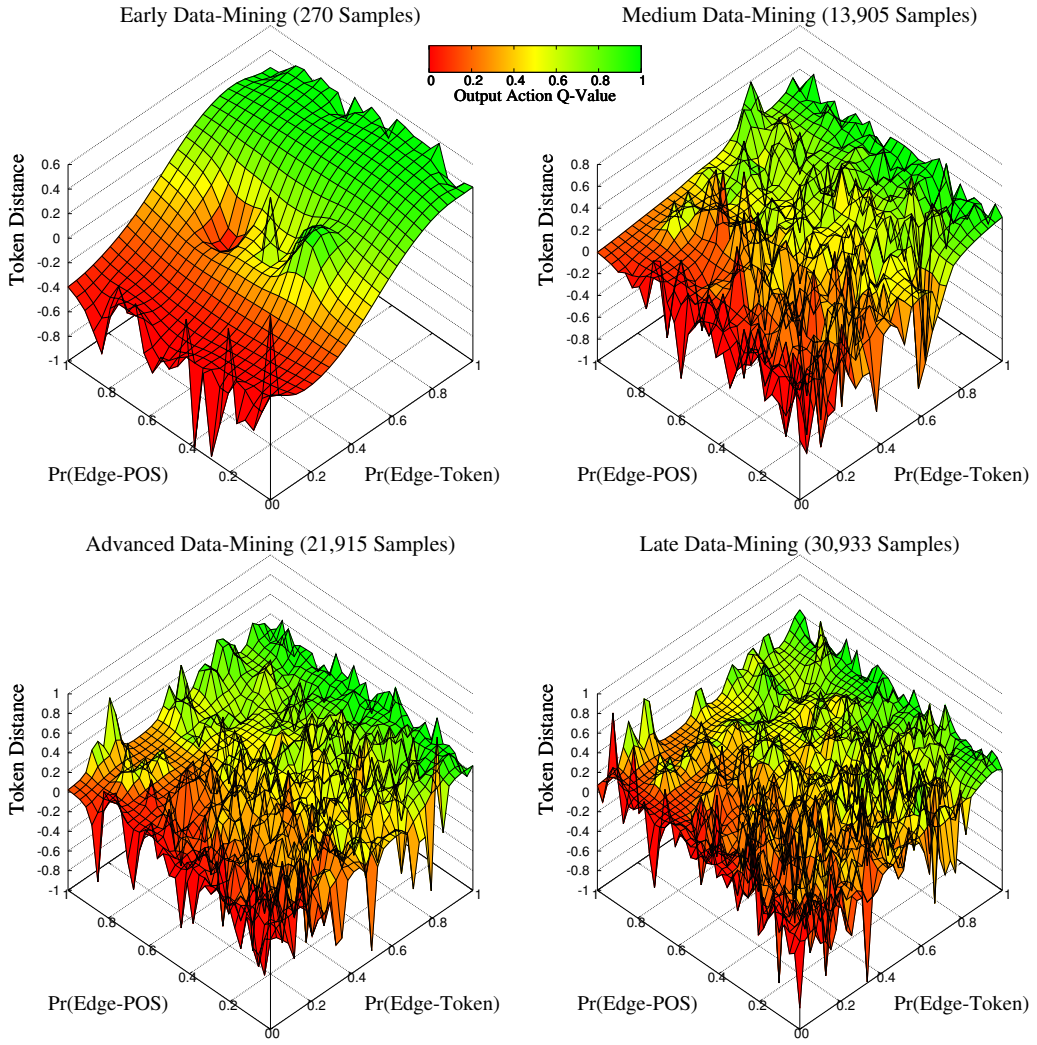

FIGURE 11: ANN Classification Progression: Improvement of ANN using larger experience samples.

Table 1: State of the Art NLU Software Tools.

Name	**URL**
Deep-Syntactic Parser	https://github.com/talnsoftware/deepsyntacticparsing
MaltEval	http://www.maltparser.org/malteval.html
NLP4J	https://github.com/emorynlp/nlp4j
RedShift	https://github.com/syllog1sm/Redshift
RBGParser	https://github.com/taolei87/RBGParser
SNN	http://nlp.stanford.edu/software/nndep.shtml
SpaCy	https://spacy.io
SyntaxNet	https://github.com/tensorflow/models
TedEval	http://www.tsarfaty.com/unipar
TurboParser	http://www.cs.cmu.edu/~ark/TurboParser

Table 2: Commonly used Data-sets

Name	**Training entries**	**Testing entries**	Average Sentence Length
ATIS3	7,300	1,000	20.5
Penn-Treebank	2499	unknown	25.6
Our Data-set	1199	variable	10.92
BioNLP11ST	800	260	unknown
GeoQuery	600	280	6.87
RoboCup Data-set	300	unknown	22.52

Table 3: Performance comparison

Name / Platform	**Reported Accuracy**	**Data Type**
Andor et al (SyntaxNet)	94.44%	News Data
Parsey McParseface	94.15%	News Data
Weiss et al. (SyntaxNet)	93.91%	News Data
Zhang and McDonald	93.32%	News Data
Martins et al	93.10%	News Data
Our Work	**92.78%**	**News Data (CG)**
Durett, Klein (Neural CRF)	90.97 %	Various Data-sets
Socher, Ling, Ng, Manning (RNN)	90.29%	WSJ Treebank
Poon, Domingos (USP)	88%	Biomedical abstracts
Vlachos, Clark, Jacob	53% - 81.8%	GeoQuery
Vlachos, Clark (DAGGER)	78.9%	Tourist Information (MRL)
Wong, Mooney (WASP)	76.77%	GeoQuery
Zhong, Duan, Zou	74% - 95%	Science book abstracts (CG)
Wong, Mooney (WASP)	72%	RoboCup
Vlachos (R.Learning)	55%	BioNLP11ST