Graph Traversal Methods for Reasoning in Large Knowledge-Based Systems

Abhishek Sharma[1] **Kenneth D. Forbus**[2]

[1]Cycorp, Inc. 7718 Wood Hollow Drive, Suite 250, Austin, TX 78731

[2]Northwestern University, 2133 Sheridan Road, Evanston, IL 60208

abhishek@cyc.com, forbus@northwestern.edu

Abstract

Commonsense reasoning at scale is a core problem for cognitive systems. In this paper, we discuss two ways in which heuristic graph traversal methods can be used to generate plausible inference chains. First, we discuss how Cyc's predicate-type hierarchy can be used to get reasonable answers to queries. Second, we explain how connection graph-based techniques can be used to identify script-like structures. Finally, we demonstrate through experiments that these methods lead to significant improvement in accuracy for both Q/A and script construction.

Introduction and Motivation

Commonsense reasoning is a fundamental problem for cognitive systems. While information extraction-based systems are often successful in finding answers to users' queries, inference-based systems have the significant advantages of general reasoning and answer explanation. Unfortunately, constructing a knowledge base (KB) for such systems remains a tedious task. One way to avoid this task is to exploit learning strategies to populate a KB automatically, and use feedback to teach reasoning. For example, text-reading systems [Matuzek *et al*. 2005; Forbus *et al*. 2007] have proven successful in providing ground facts for use in KB, though they have shown little success in gleaning correct, fully quantified logical axioms. Being able to learn plausible patterns of inference over ground facts would significantly improve the scalability of KB construction, and they would help solve a second problem in reasoning with large KBs: typically, the set of logically quantified axioms has a miserably low coverage. While prior work has explored plausible inference schemes (e.g., [Collins 1978]) our work is more squarely focused on developing systems that can learn to reason. In the end, human attention is a scarce resource, and our goal must be to minimize the amount of feedback users need to provide.

The problem of finding plausible chains between concepts also arises when cognitive systems have to stitch a given set of concepts into a coherent situation, such as when a script-like explanation is needed for a given set of input concepts. Since concepts can have arbitrary relations between them, it is difficult to identify relevant queries for such problems. What kinds of reasoning methods are needed for solving such ill-defined problems?

In this paper, we propose that path-finding methods on a graph representation of the contents of a commonsense KB provide a unifying theme for solving these problems. The amount of knowledge needed for filtering incorrect paths varies with the task. While Q/A tasks are likely to need heavily constrained, focused search, other problems (e.g., finding a script for a given set of concepts) can be satisfactorily solved by weaker knowledge. Specifically, this paper makes two contributions: (1) We show how to integrate graph search, higher-order knowledge representation, and reinforcement learning to discern reliable patterns of plausible reasoning from ground facts. Given a fully grounded query, we show how to incrementally search the facts that mention the entities therein, guided by a set of *plausible inference patterns* (PIPs). We also show that the quality of inference chains of PIPs can be learned through reinforcement learning. (2) We argue that the problem of explaining a set of concepts by linking them in a situation should be seen as a connection graph finding problem. We discuss the types of representations needed to address this problem.

This paper is organized as follows: we begin by discussing related work. We then cover the basics of PIPs and how they are used. Next, we show how reinforcement learning is used to learn the quality of PIPs. We then

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

describe our connection graph-based algorithm for identifying scripts. Next, we present experimental results. We conclude by discussing future work.

Related Work

A number of researchers from the fields of information retrieval, natural language processing, databases and logical inference have contributed to the advancement of QA technologies [Brill *et al.* 2002] [Prager *et al.* 2004]. Overviews of QA techniques can be found in [Belduccinni *et al.* 2008, Molla and Vicedo 2007], and a comparison of challenging problems and approaches has been discussed in a recent IBM report [Ferrucci *et al.* 2009]. Learning surface patterns from natural language text has also been discussed in [Molla 2006]. Our work differs from this in that we are trying to improve the performance of a plausible inference-based Q/A system by learning to reason. We note that other frameworks for learning to reason have been explored in [Khardon 1999], but their efficacy in improving Q/A performance is not known. Reinforcement learning has been used for learning control rules for guiding inference in ResearchCyc KB [Taylor *et al.* 2007], but to the best of our knowledge, there has been no prior work to develop a method for providing plausible explanations for queries (without using logically quantified axioms) using a learning framework. Similarly, although there has been work on script identification [Miikkulainen 1990] and event detection [Moore & Essa 2002], we are not aware of any system which uses a path-finding approach for producing script-like explanations.

Representation and Reasoning

Our major source of KB contents is Cyc, and so we have chosen to adopt key Cyc conventions in this paper. We summarize those conventions here [Matuszek *et al.* 2006].

Cyc represents concepts as *collections*. Each collection is a kind or type of thing whose instances share a certain property, attribute, or feature. For example, Cat is the collection of all and only cats. Collections are arranged hierarchically by the genls relation. (genls <*sub*> <*super*>) means that anything that is an instance of <*sub*> is also an instance of <*super*>. Predicates are also arranged in hierarchies. In Cyc terminology, (genlPreds <*s*> <*g*>) means that <*g*> is a generalization of <*s*>. We make extensive use of Cyc's predicate type hierarchy. PredicateType is a collection of collections and each instance of PredicateType is a collection of predicates. The predicates in a given predicate category represented in the KB are typically those sharing some common feature(s) considered significant enough that the collection of all such predicates is useful to include. Instances of PredicateType include TemporalPartPredicate, SpatialPredicate, and PropositionalAttitudeSlot.

The task of answering questions without using logically quantified axioms is difficult because it requires sifting through any number of arbitrary relations between predicates, any one of which could explain the query. To avoid this, we have chosen the simpler approach of building a small sub-graph of relations around the entities in the query and then assessing the quality of inference chains between them. This intuition is similar to connection graphs [Faloutsos *et al.* 2004] and relational pathfinding, where the domain is viewed as a (possibly infinite) graph of constants linked by the relations that hold between the constants [Richards & Mooney 1992]. More formally, the KB can be seen as a graph G = (V, E) where V is the set of nodes (or constants) and E is the set of edges. An edge, *e*, exists between two nodes v_1 and v_2 if $e(v_1, v_2)$ or $e(v_2, v_1)$ are true in the KB. A path from vertex *a* to *b* is an ordered sequence $a = v_0.v_1.v_2 \ldots v_m = b$ of distinct vertices in which each adjacent pair (v_{j-1}, v_j) is linked by an edge. Since prior knowledge is important for biasing learning, we leverage existing axioms in the KB to create PIPs that are used to keep only the more likely inference chains. These PIPs are created by replacing predicates in axioms by their predicate types. PIPs are accepted if they are generated by more than N axioms. (In this work, N = 5). We provide a concrete example for illustration.

Let us assume that the system has been asked to provide a plausible inference for the query (acquaintedWith BillClinton HillaryClinton). A small section of the KB relevant to answering this query is shown in Figure 1. For simplicity, let us assume that we have just one PIP:

FamilyRelationSlot(?x,?y) AND FamilyRelationSlot(?y,?z) → PersonalAssociationPredicate(?x,?z) **[PIP1]**

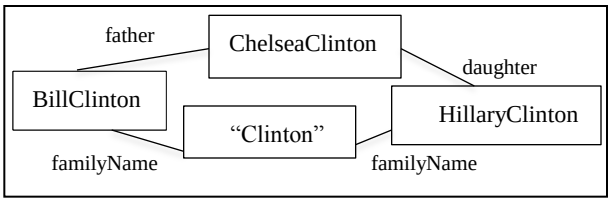

Figure 1: Plausible Inference Example.

This pattern represents the knowledge that two predicates of type FamilyRelationSlot can plausibly combine to infer assertions involving personal associations. This representation has been chosen because we believe that predicate types such as SubEventPredicate, PhysicalPartPredicate, and CausalityPredicate provide a meaningful level of abstraction for identifying PIPs. For instance, all predicates of type SubEventPredicate can be

used for proving eventPartiallyOccursAt queries[1]. Similarly, all predicates of type PhysicalPartPredicate are relevant for proving objectFoundInLocation queries[2]. Therefore, learning knowledge in terms of predicate types is easier and more natural.

Algorithm: **find-plausible-explanations: FPE(KB, query)**
Input: *query:* A ground query
Local Variable: *solutions* initialized to {}.
Output: A set of facts which would justify *query*.

1. **for** all patterns *r* in KB, where *r* = $\{p_1 \wedge p_2 \wedge ... \wedge p_n \rightarrow q\}$
2. *type* ← predicate type in *q*.
3. *pred* ← predicate in *query*
4. **if** (isa *pred type*) **then**
5. $\theta' \leftarrow$ *unify* the variables in *q* and *query*.
6. **for** j ← 1 to n
7. $query_j \leftarrow$ substitute bindings from θ' in p_j.
8. $Solution_j \leftarrow$ find solutions for $query_j$.
9. **if** bindings from $Solutions_j$ are consistent **then** update θ' **else goto** step 12.
10. **end for**
11. If $Solution_j$ exists for all $1 \leq j \leq n$ then update *solutions*
12. **end if**
13. **end for**
14. **return** *solutions*.

Figure 2: Algorithm for finding plausible explanations

The relative tractability of this formulation can also be seen when noting the difference between the sizes of search spaces. Learning to distinguish between correct and incorrect derivations of length *k* involves searching in a space of size N^k, where N is the size of vocabulary. In Cyc, the number of predicates is 24 times larger than the number of predicate types. Therefore, learning PIPs in terms of predicate types is significantly easier. The algorithm *find-plausible-explanations* is described in Figure 2. In the example introduced above, *r* would be bound to PIP1 in step 1. In step 2, *type* is bound to PersonalAssociationPredicate. Since the predicate acquaintedWith is an instance of this collection, the test in step 4 succeeds, and we try to prove the antecedents of the rule in steps 6–10. Essentially, this means that we are looking for a path between the nodes labeled BillClinton and HillaryClinton traversing two edges labeled with predicates of type FamilyRelationSlot. In Figure 1, a small section of the graph is shown. In step 7, we create a query FamilyRelationSlot(?x, BillClinton). In step 8, solutions for this query would be found by querying for (p ?x BillClinton) where p is an instance of FamilyRelationSlot. An assertion as (father ChelseaClinton BillClinton) would be a solution for this query. This would lead to a query as FamilyRelationSlot (ChelseaClinton , HillaryClinton), which would be answered with the help of facts like (mother ChelseaClinton HillaryClinton). In Figure 1, the second path involving two edges labeled 'familyName' would not be selected because no PIPs use predicates of type ProperNamePredicate-Strict to entail PersonalAssociationPredicate predicates. Similarly, the following PIP would help in proving (objectFoundInLocation ArmyBase-Grounds-FtShafter-Oahu HawaiianIslands) (see Figure 3):

SpatialPredicate(?x, ?y) AND Group-Topic(?z,?y) → SpatialPredicate(?x, ?z) … **[PIP2]**

The pattern PIP2 shown above would lead to an incorrect answer if we use bordersOn as an instance of SpatialPredicate in the consequent. In the next section, we discuss how reinforcement learning helps us in solving this problem.

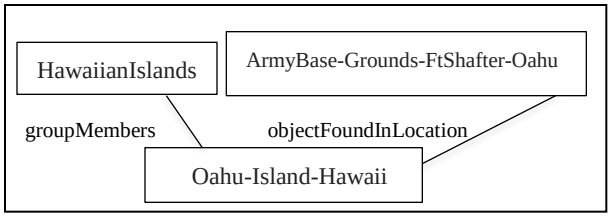

Figure 3: Another plausible inference example.

This inference scheme also simplifies inference by condensing inference chains. For example, wife is a PersonalAssociationPredicate, and so the inference from wife to acquaintedWith is a one-step process. On the other hand, using the normal predicate hierarchy involves multi-step inferences. The inference chain from wife to acquaintedWith, for example, is a four-step reasoning chain[3]. Since the number of predicate types is less than the number of predicates, the predicate type hierarchy maps the predicates to a smaller space. This speeds up the search because the average path length between two nodes in this smaller space is less than what we encounter in a typical predicate hierarchy. This plays an important role in improving inference. The FPE algorithm can be easily extended to handle queries with variables. This would entail checking that the node at the search frontier satisfies the argument constraint of the predicate.

Learning to Reason

Many learning systems learn the correct level of generalization by trial and error. Our approach gets initial

[1] Some examples of SubEventPredicate predicates are firstSubEvents, cotemporalSubEvents, finalSubEvents etc.

[2] Some examples of PhysicalPartPredicate are physicalParts, internalParts, northernRegion etc.

[3] The four steps are acquaintedWith → mutualAcquaintances → mate → spouse → wife.

PIPs by replacing predicates in axioms with their predicate types. These generalizations certainly increase the deductive closure but can lead to incorrect answers.

The task of designing a system that could learn to identify incorrect search steps from minimal user feedback is complicated by the fact that a typical user may not be able to identify the incorrect search choice(s) made during a multistep reasoning process. Thus, the learner ought to be able to work with delayed feedback about the correctness of the final answer and learn to find plausible inferences for queries. We believe that reinforcement learning is a reasonable method for solving this problem. Formally, the model consists of (a) a discrete set of states, S; (b) a discrete set of agent actions, A; (c) a reward function R: $S \times A \rightarrow \{-1, 1\}$; and (d) a state transition function T: $S \times A \rightarrow \prod(S)$, where a member of $\prod(S)$ is a probability distribution over the set S [Kaelbling *et al.* 1996]. In this context, a *state* is the list of predicate types already used during the partially complete search process. At each step of the reasoning process, the inference engine has choice points at which it chooses or rejects different alternatives. To do this, it must assess how useful a particular predicate type is for completing the proof given the predicate types already chosen in the current search path. The *actions* are the selection of a particular predicate type for completing the partial assignment of variables. The *value function* (or V(s)) is the inference engine's current mapping from the set of possible states to its estimates of the long-term reward to be expected after visiting a state and continuing the search with the same policy. Q(*s*, *a*) represents the value of taking the action *a* in state *s*. We use the value iteration algorithm [Kaelbling *et al.* 1996] for learning the plausibility of search paths, and a delayed reward model with user-provided rewards of +1 and -1 for correct and incorrect answers, respectively.

Connection Graph Methods for Identifying Script-like Structures

In many AI applications, we need to find a small set of assertions that best capture the relationships between a set of concepts. The primary aim is to stitch the concepts together in a script by adding missing relations between them. Here we discuss the different knowledge representation and reasoning challenges that must be addressed to solve this problem. As discussed above, we represent the contents of the KB as a graph, where concepts are nodes and edges represent relations between them. Given such a structure, the problem can be formulated as:

Input: An edge-weighted undirected graph G, a set of concepts $S = \{S_1, \ldots, S_N\}$, and an integer budget *n*.

Output: A connected subgraph containing S and at most *n* other nodes, that maximizes a goodness criterion.

Input: [BlowingOutCandles, Applauding-Clapping, EatingEvent]
Relevant Facts:
(properSubEventTypes LightingTheCandlesOnABirthdayCake LightingACandle)
(preconditionFor-EventTypeEventType BlowingOutCandles LightingACandle)
(properSubEventTypes BirthdayParty LightingTheCandlesOnABirthdayCake)
(candidateProperSubSituationTypes BirthdayParty ApplaudingTheBlowingOutOfBirthdayCakeCandles)
(properSubEventTypes ApplaudingTheBlowingOutOfBirthdayCakeCandles Applauding-Clapping)
(properSubEventTypes BirthdayParty ServingTheCakeToGuests)
(superEventTypes ServingTheCakeToGuests CelebratoryEatingOfTheCake)
(properSubEventTypes CelebratoryEatingOfTheCake EatingEvent)

Figure 4: A Sample Output

For example, given an input like {BlowingOutCandles, Applauding-Clapping, EatingEvent}, we would like to infer that the situation refers to a birthday party and explain why these features are related to it (See Figure 4). Our method for finding a connection graph is based on work in the knowledge discovery community [Faloutsos *et al.* 2004, Ramakrishnan *et al.* 2005]. We describe their approach in brief here. The algorithm has two central components: (1) a candidate generation algorithm, and (2) a display generation component. The candidate generation component maintains a list of pending nodes and expands the frontiers of the graph starting from a given set of nodes. The display generation component [Faloutsos *et al.* 2004] aims to identify a small subset of nodes and edges that represent the most relevant relation between the input nodes, using the model of an electrical circuit. A modified version of this algorithm is shown in Figure 5. The input to the algorithm is a set of concepts. It also uses a set of useful predicates, P, and a set of inconsistent path patterns, Q. The set P contains predicates that are more useful for representing semantic information[4]. Consider the example shown in Figure 4. In steps 1-6 of the algorithm shown in Figure 5, we create a graph around the input concepts. For example, the query in step 3 would look like (properSubEventTypes ?x BlowingOutCandles). The

[4] We used this set to exclude bookkeeping and NL predicates (e.g., nameString). properSubEventTypes, eventTypeOccursAtLocationType, and typePrimaryFunction are some elements of P.

results from the query are added to the graph in step 4. In step 7, we use the display generation algorithm for extracting a small connection graph. This involves using dynamic programming and a model of an electrical circuit to identify most relevant paths between the input nodes [Faloutsos *et al.* 2004]. This might be sufficient for lightly constrained domains, but this approach becomes less useful in a domain where more semantic processing is needed. Therefore, we augment the algorithm with a post-processing step (step 8) which prunes implausible paths in the graph. A set of implausible path patterns, Q, is an input to the algorithm. For example, a pattern of the type (eventTypeOccursAtLocationType ?x ?y) AND (eventTypeOccursAtLocationType ?z ?y) would represent the fact that sharing a common location type is less useful for finding connections between script constituents. This would ensure that we do not hypothesize a link between RoadConstructing and IceClimbing just because they occur outdoors. If all concepts in the input are not in a connected component of the pruned graph, then we repeat the step with a relaxed size constraint (step 12).

Algorithm: **find-plausible-script (FPS)**
Input: A set of input concepts: S, A set of predicates: P, A set of filtered paths: Q, An integer budget: N

1. **for** each concept c belonging to S
2. **for** each predicate p belonging to P
3. Retrieve all facts involving p and c
4. Add the nodes to the graph
5. **end for**
6. **end for**
7. *OutputGraph* ← Use display generation algorithm to extract a connection graph of size N
8. *PrunedGraph* ← Remove all edges from *OutputGraph* which are inconsistent with Q.
9. **if** all elements of S are in a component **then**
10. **return** *PrunedGraph*
11. **else**
12. set N ← N+1 and **goto** step 1.

Figure 5: Algorithm for identifying script-like structures

Experimental Method and Results

To show that these ideas generate more answers compared to traditional deductive reasoning methods, we conducted a set of experiments. Five sets of questions were selected based on the availability of ground facts in KB and their relevance in learning by reading [Forbus *et al* 2007]. These questions' templates were (1) Where did <*Event*> occur? (2) Who is affected by <*Event*>? (3) Where is <*SpatialThing*>? (4) Who performed the <*Event*>? and (5) Where is <*GeographicalRegion*>? Each question template expands to a disjunction of formal queries. Queries were generated by randomly selecting facts for these questions from the KB. For a baseline comparison, we included all axioms for these predicates and their subgoals to a depth of 3. We used a simple backchainer working on an LTMS-based inference engine [Forbus & de Kleer, 1993]. The depth of backchaining was also limited to three and each query was timed out after three minutes. All experiments were performed on a 3.2 GHz Pentium Xeon processor with 3GB of RAM. 25% of the queries were used as the training set for learning the V(s) values. Answers whose V(s) values were more than a threshold were accepted.

Table 1 compares the performance of the FPE algorithm and reinforcement learning against the baseline for the test set (i.e. the remaining 75% of queries). Each experiment corresponds to queries from a particular template (i.e., Expt. 1 concerns the location of events). Column **T** is the total number of queries, and **AW** is the number that could be answered given the KB contents, as determined by hand inspection. The columns **P** and **R** indicate precision and recall, respectively. The user assessed 334 unique answers (from the training set) and the feedback was used for learning the V(s) values. The accuracy of answers provided by the FPE algorithm was 73%. We then removed answers whose V(s) values were below the threshold. The total number of new answers at this stage was 1010 and the accuracy improved from 73% to 94%. The FPE algorithm mainly reduces false negatives, whereas reinforcement learning reduces false positives. Together, they improve on the baseline by a factor of 2.2 (i.e. by 120%) with an average accuracy of 94%. The results shown in Table 1 are statistically significant ($p < 0.01$).

Exp. No.	Query sets	T	AW	P	R
1	Baseline	833	412	1.00	0.51
	FPE	833	412	0.95	0.87
2	Baseline	200	61	1.00	0.42
	FPE	200	61	0.92	0.77
3	Baseline	1834	433	1.00	0.32
	FPE	1834	433	0.92	0.88
4	Baseline	953	226	1.00	0.34
	FPE	953	226	0.93	0.93
5.	Baseline	1309	724	1.00	0.43
	FPE	1309	724	0.97	0.94

*Table 1: Summary of inference results. Experiment numbers are the same as query numbers***.**

To evaluate the script recognition system, we used the algorithm for comprehending events in videos. A computer

vision and speech recognition system was used to generate low-level features from 103 videos. These features are mapped to the collections from the knowledge base, such as the collections marked as input in Figure 4. These features were fed to the algorithm shown in Figure 5. The scripts in videos were roughly divided into following five types: (1) wedding event, (2) skateboarding (3) making a sandwich, (4) flash mob and (5) parkour. The sixth script type includes all scripts which were different from five types mentioned above. The first column in Table 2 shows these script numbers. In the third column, we report the proportion of facts in the output which were directly relevant for understanding the script. Since we are evaluating the efficacy of connection-graph methods for this problem, we can establish our baseline by replacing steps 7 and 8 in Figure 5 with an algorithm that finds the shortest paths between the nodes in the input. The results shown in Table 2 are statistically significant ($p < 0.01$).

Script Type	Method	% correct	Improvement w.r.t. baseline
1	Baseline	69.0	-
	FPS	96.0	39%
2	Baseline	67.4	-
	FPS	84.6	25%
3	Baseline	61.9	-
	FPS	89.0	44%
4	Baseline	50.0	-
	FPS	100.0	100%
5	Baseline	33.5	-
	FPS	93.0	177%
6	Baseline	25.6	-
	FPS	49.0	91%

Table 2: Evaluation of find-plausible-script (FPS) algorithm

Conclusion

Plausible commonsense reasoning is a fundamental problem for cognitive systems, and we believe that our approach provides a promising solution. We have shown how different graph traversal methods can be used to alleviate the difficulties created by missing knowledge. The use of predicate types for representing PIPs leads to a succinct, easily learnable and tractable representation. With the FPE algorithm mainly reducing false negatives, and reinforcement learning reducing false positives, we get a 120% improvement over the baseline with an average accuracy of 94%. The use of connection graph methods of identifying script-like explanations produces good results.

While these experiments used the contents of ResearchCyc, we believe they are applicable to any large-scale KB whose predicate types were classified sensibly. Our technique is especially suitable for knowledge capture because it exploits ground facts, which are much easier to gather than logically quantified facts. We believe that this technique can be used to help bootstrap intelligent systems and reduce dependence on handcrafted axioms. Our results suggest following further lines of work. For one, being able to refine PIPs to use more specific predicate types would improve accuracy and coverage. In addition, PIPs could be used as an intermediate stage for postulating new logically quantified statements, perhaps by using a technique like relational reinforcement learning [Dzeroski *et al*. 2001] to carry out the refinements.

Our work on identifying scripts can be extended in at least four ways. Firstly, scripts often have temporal dependencies between their events, and we would like to include temporal constraint processing in our model. Secondly, though we have used a list of implausible patterns to prune less useful paths, we believe this approach can be improved by employing a grammar of plausible paths [Navigli 2008]. Thirdly, we should try to remove the stricture that all concepts be part of a connected component. For example, concepts like Sun and Night might be present in many video events. Although it is possible to connect such concepts to different events in a given video, we believe that we might need to prune those input concepts that have low estimates of information gain. Finally, we have found that correct paths might not be available in the search space due to two reasons: (a) The KB might not have relevant knowledge[5] or (b) The system is not considering relevant predicates due to resource constraints. Due to the absence of correct paths, the algorithm might connect concepts via overly general nodes. We would like to detect and prune such incorrect paths.

Acknowledgements

This work benefited from discussions with Doug Lenat. This work was supported by the Office of Naval Research.

References

Belduccinni, M. Baral, C. and Lierler, Y. 2008. *Knowledge Representation and Question Answering*. In Vladimir Lifschitz and Frank van Harmelen and Bruce Porter, ed In Handbook of Knowledge Representation.

Brill, E., Dumais, S. and Banko, M. 2002. An analysis of the AskMSR question-answering system. *Proceedings of ACL*, pages 257-264.

Clark, P., Thompson, J. and Porter, B. 2000 Knowledge Patterns. *Proceedings of KR*, page 591-600.

Collins, A. 1978. Human Plausible Reasoning. BBN Report No. 3810.

Dzeroski, S., de Raedt, L. and Driessens, K. 2001 Relational Reinforcement Learning. *Machine Learning*, 43, pp. 7-52

[5] Lack of relevant knowledge in KB is the primary reason behind relatively low numbers for Script type 6 in Table 2.

Faloutsos, C., McCurley, K. S. and Tomkins, A. 2004 Fast Discovery of Connection Subgraphs. *Proceedings of KDD*, pages 118-127.

Ferrucci, F. and Nyberg, E. *et al* 2009 Towards the Open Advancement of Question Answering Systems. IBM Research Report. RC24789 (W0904-093), IBM Research, New York.

Forbus, K. D. and de Kleer, J. 1993 *Building Problem Solvers*. MIT Press

Forbus, K. D., Riesbeck, C., Birnbaum, L., Livingston, K., Sharma A., and Ureel, L. (2007). Integrating Natural Language, Knowledge Representation and Reasoning, and Analogical Processing to Learn by Reading. *Proceedings of AAAI*, pages 1542-1547

Khardon, R. 1999 Learning Function-Free Horn Expressions. *Machine Learning*, 37, pp. 241-275.

Kaelbling, L. P., Littman, M. and Moore, A. 1996. Reinforcement Learning: A Survey. *Journal of AI Research*, 4, pp. 237-285.

Matuszek, C., Witbrock, M., Kahlert, R., Cabral, J., Schneider, D., Shaw, P., and Lenat, D. 2005. Searching for common sense: Populating Cyc from the web. *Proceedings of AAAI*, pages 1430-1435.

Matuszek, C., Cabral, J., Witbrock, M. and De Olivieira, J. 2006. An Introduction to the Syntax and Content of Cyc. *AAAI Spring Symposium*, pages 44-49.

Miikkulainen, R. 1990. Script Recognition with Hierarchical Feature Maps. *Connection Science*, pages 83-101

Molla, D. 2006 Learning of Graph-based Question Answering Rules. *Proceedings of HLT/NAACL Workshop on Graph Algorithms for Natural Language Processing*. pages 37-44.

Molla, D. and Vicedo, J. L. 2007. Question Answering in Restricted Domains: An Overview. *Computational Linguistics*, 33 (1), pp. 41-61

Moore, D. and I. Essa. 2002. Recognizing Multitasked Activities from Video Using Stochastic Context-Free Grammar, *Proceedings of AAAI*, pages 770-776.

Navigli, R. 2008. A Structural Approach to the Automatic Adjudication of Word-Sense Disagreements. *Natural Language Engineering.* Vol. 14, 4, pp. 547-573

Prager, J., Chu-Carroll, J. and Czuba, K. 2004. Question Answering Using Constraint Satisfaction: QA-by-Dossier-with-Constraints. *Proceedings of ACL*, pages 60-65.

Ramakrishnan, S., Milnor, W. H., Perry, M. and Sheth, A. 2005. Discovering Informative Connection Subgraphs in Muti-relational Graphs. *ACM SIGKDD Explorations Newsletter*, 7(2), pp. 56-63.

Richards, B. and Mooney, R. 1992. Learning Relations by Pathfinding. *Proceedings of AAAI*, pages 50-55.

Taylor, M., Matuszek, C., Smith, P. and Witbrock, M. 2007. Guiding Inference with Policy Search Reinforcement Learning. *Proceedings of FLAIRS*, pages 146-151.