ACS Poster Collection (2018) 1−12

Published 8/2018; Paper 16

Learning Search Policies in Large Commonsense Knowledge Bases by Randomized Exploration

Abhishek Sharma ABHISHEK@CYC.COM
Keith M. Goolsbey GOOLSBEY@CYC.COM
Cycorp, Inc. 7718 Wood Hollow Dr, Suite 250, Austin, TX 78731

Abstract

Deep deductive reasoning is a critical problem for cognitive systems. Very large knowledge bases have millions of axioms, of which relatively few are relevant for answering any given goal query. A large number of irrelevant axioms overwhelm theorem provers, and even simple queries cannot be answered in reasonable time. Therefore, heuristics which help in identifying useful inference paths are an essential component of cognitive systems. In this paper, we learn search control knowledge by taking random samples in the search space and evaluating the quality of resulting paths. These random probes help us to identify relevant inference paths for queries. Experimental results on difficult queries from the Cyc KB show that this approach can be applied effectively to reduce inference time.

1. Introduction and Motivation

Many knowledge-based systems (KBS) rely on deductive reasoning capabilities for various reasoning tasks. However, deduction in large expressive knowledge bases (KBs) is intractable. Very large knowledge-based systems contain several million ground atomic formulas (GAFs) and axioms. Only a few of these GAFs and axioms are relevant for answering any goal query. Irrelevant axioms increase search space, and traditional resolution-based theorem provers are overwhelmed by irrelevant paths [Hoder & Voronkov 2011]. Therefore, many AI systems use only small to medium-sized KBs for efficient query processing. Such systems cannot perform well in realistic situations because cognitive systems need large bodies of general knowledge to perform real-world tasks robustly [Lenat & Feigenbaum 1991, Forbus *et al.* 2007]. Therefore, the development of fast deductive reasoning algorithms is critical to create realistic cognitive systems.

Deductive reasoning algorithms in expressive knowledge-based systems (KBS) typically represent the search space as a graph, whose structure is determined by the rules that apply to nodes. Generally hundreds of rules simultaneously apply to nodes, and the order of node and rule expansion has a significant effect on efficiency [Tsarkov & Horrocks 2005, Sharma *et al.* 2016].

Therefore, research in ordering heuristics plays an important role in improving the efficiency of deductive reasoning algorithms.

In this paper, we examine whether randomized exploration of search space can be used to predict the most promising inference step. An agent starts at a root node, and chooses a path. The path is expanded, and the number of answers obtained by the path is an indicator of its quality. Very little can be learned from a single path. However, we show that useful search control knowledge can be gleaned from a multitude of random path explorations. At the end of this process, we learn a function that would map the features of search paths to the number of answers resulting from them. This function is then used to assess and order search paths during inference. Experimental results show that this approach leads to significant reduction in inference time.

This paper is organized as follows: We start by discussing relevant work. In Section 3, we discuss the basics of Cyc representation language and its inference engine. Our approach of learning by randomized exploration is discussed next. We conclude by discussing experimental results and future work.

2. **Related Work**

In the last decade, there has been interest in reasoning with expressive languages like OWL description logic (DL) and the Semantic Web Rule Language (SWRL). Researchers have used first-order theorem provers with these languages because reasoning with them is beyond the scope of existing DL algorithms or because the language does not correspond to any decidable fragment of first-order logic (FOL) [Tsarkov *et al* 2004, Horrocks & Voronkov 2006]. The work in the field of optimizing inference in large expressive knowledge bases can be divided into three types: (i) axiom selection or premise selection algorithms, where researchers have focused on selecting a small set of axioms/lemmas that is most useful for answering a set of queries [Sharma & Forbus 2013, Kaliszyk *et al.* 2015, Kaliszyk and Urban 2015, Alama *et al.* 2014]; others have used co-occurrence-based analysis to design a symbol-based axiom selection scheme [Hoder & Voronkov 2011]. (ii) Learning of ordering heuristic: recently, researchers have used machine learning techniques (e.g., decision trees and regression-based models) to control searches [Sharma *et al.* 2016], and (iii) offline search; in [Sharma & Goolsbey 2017], the authors have proposed an offline procedure that performs an exhaustive breadth-first search to generate sequences of useful action sequences. Our approach is mainly inspired by the work in reinforcement learning and the Monte Carlo tree search community, where researchers have shown that tree search and random sampling can be combined to learn search policies that perform well in computationally difficult problems [Browne *et al.* 2012]. We are not aware of any work in the AI community that has addressed these issues[1]. Work in other communities [Chaudhuri 1998, Hutter *et al.* 2014, Brewka *et al.* 2011] is less relevant because these do not address the complexity of reasoning with expressive languages in large knowledge bases.

[1] The work presented in [Taylor et al 2007] does not address improving deep deductive reasoning.

3. Background

We assume familiarity with the basics of Cyc representation language [Lenat & Guha 1990, Matuszek *et al.* 2006]. In Cyc, concepts are called *Collections*, and concept hierarchies are represented by the "genls" relation. For example, (genls Chair-PieceOfFurniture InanimateObject) holds. Similarly, the role and inverse-role hierarchies are represented by the "genlPreds" and "genlInvese" relations, respectively (e.g., (genlPreds touches near) and (genlInverse lessThan greaterThan) hold). While performing backward inference, the Cyc inference engine uses rules to transform goal nodes into logically equivalent child nodes. For example, the rule P(x) →Q(x) could be used to transform Q(a) into P(a). The link between Q(a) and P(a) is a type of *transformation link*. Given a multi-literal query of the type (pred1 arg1 arg2 …) ^ (pred2 …) ^ ..., an inference step involves resolving one of the literals. The literal on which the inference engine chooses to backchain is called the *focal literal*. Cyc estimates and keeps track of the generality of all terms (referred to as GeneralityEstimate (term)) in the KB based on their position in the ontology. The leaf nodes in the ontology have a generality estimate of zero, whereas the most general term (the collection "Thing") has the highest generality estimate.

Reasoning with Cyc representation language is difficult due to the sheer size of the KB and the expressiveness of the CycL representation language. In its default inference mode, the Cyc inference engine uses the following types of axioms/facts during backward inference: (i) 28,429 role inclusion axioms (e.g., P(x, y) → Q(x, y)); (ii) 3,623 inverse role axioms (e.g., P(x, y) → Q (y, x)), (iii) 494,045 concepts and 1.1 million concept inclusion axioms (i.e., 'genls' facts); (iv) 817 transitive roles; (v) 120,547 complex role inclusion axioms (e.g., P(x, y) ∧ Q (y, z) → R (x, z)); (vi) 77,170 axioms of the type P(x, …) →Q(w, …) (these axioms are not included in role inclusion, complex role inclusion or concept inclusion axioms mentioned above); and (vii) 35,528 binary roles and 10,508 roles with arities greater than two. The KB has 27.3 million assertions and 1.14 million individuals. To control search performance in such a large KBS, inference algorithms often use heuristics. They distinguish between a set of clauses known as the *set of support*, which define the important facts about the problem, and a set of usable axioms that are outside the set of support (e.g., see the OTTER theorem prover [Russell and Norvig 2003]). At every step, a theorem prover would resolve an element of the set of support against one of the usable axioms. To perform best-first search, a heuristic control strategy measures the "weight" of each clause in the set of support, picks the "best" clause, and adds to the set of support the immediate consequences of resolving it with the elements of the usable list [Russell and Norvig 2003]. Cyc uses a set of *heuristic modules* to identify the best clause from the set of support. A heuristic module is a tuple $h_i = (w_i, f_i)$, where f_i is a function $f_i: S \rightarrow \mathbb{R}$ that assesses the quality of a node, and w_i is the weight of h_i. The net score of a node s is given by (1), and the node with the highest score is selected for further expansion:

$$\sum \quad w_i f_i(s) \qquad \ldots(1)$$

Currently used heuristics by the Cyc inference engine include: (a) success rate of rule, (b) an ordering heuristic that uses decision trees [Sharma *et al.* 2016], (c) an ordering heuristic that uses linear regression models [Sharma *et al.* 2016], and (d) a heuristic that uses a database of useful axiom sequences [Sharma & Goolsbey 2017]. In the next section, we propose a new heuristic to control a search.

4. Randomized Exploration of Search Space

The basic idea behind this approach is best explained with a few examples. Consider the following query:

(performedBy IranOccupiesTumbIslands-1971 ?who) ..(Q1)

The rule shown below could be used to backchain on Q1.

(isa ?sub-event PhysicalEvent) $\wedge$(performedBy ?event ?actor) $\wedge$
(subSituations ?event ?sub-event) $\wedge$ (isa ?event BrushingOnesTeeth) $\rightarrow$
(performedBy ?sub-event ?actor) …(Axiom A0)

Reasoning with A0 to solve Q1 would lead the inference engine to try to prove how a military occupation could be a sub-event of a teeth-brushing event[2]. We note that the only explicit constraint on the variable ?sub-event is quite general (i.e., it is constrained to be a physical event). Therefore simple type checking on explicitly provided constraints cannot detect this implausible path. Recently, researchers have used decision trees for addressing this problem [Sharma *et al* 2016]. However, they have not discussed how they generate negative examples for their training set. In the absence of negative examples, the decision tree learning algorithm can over-generalize. Moreover, that work has not shed light on how performance scales with the number of training examples.

Our approach is based on the well-known idea that we can learn to make optimal decisions in a given domain by sampling actions and building a search tree using those results [Browne *et al.* 2012]. A high-level description of our algorithm is found in Figure 1, and Table 1 describes the notation used in that algorithm.

The algorithm discussed below uses a specific set of features to represent the states that are generated during deductive proofs. The list of features used in this work are shown in Table 1.

[2] The predicate subSituations is used to relate situations to further situations which are their parts. (subSituations A B) means that the situation B is a part of the situation A, such that a complete description of A would have to involve a description of B. For example, (subSituations WorldWarII BombingOfHiroshima) holds.

500 collections were used to generate some of these features. These collections were selected by analyzing the GAFs in the knowledge base and selecting 500 collections from the middle ontology[3] that is used most often in these GAFs.

The *RandomizedModelLearning* algorithm takes a training set (as a set of queries called *Queries*), and a set of 3,003 features as the input. Step 1 of the algorithm continues until a given computational budget is exhausted. A query *q* is selected from the training set at random in Step 2. Let us assume that q is (partTypes EukaryoticCellCycle AnaphaseI), and depth_cutoff is set to 2. In Step 5, we find all axioms that apply to the query. Since Cyc reasons over the role subsumption hierarchy, this step involves including 448 axioms that apply to the specializations of the 'partTypes' predicate[4]. In Step 6, we then select one of these axioms at random. Let us assume that Axiom A1 was chosen[5].

(relationAllExists properPhysicalParts ?x ?y) →(properPhysicalPartTypes ?x ?y) … (Axiom A1)

Note that Axiom A1 is applicable to the query because 'properPhysicalPartTypes' is a specialization of 'partTypes. Using A1 to backchain on query (Step 7) will lead to a child node with query q*: (relationAllExists properPhysicalPartTypes EukaryoticCellCycle AnaphaseI). In Step 8, we then query for q*, and that query does not lead to any answer.

Table 1: Notation

Features	The set of features that are used by the learning algorithm
Queries	The set of queries in the training set
NumberOfAnswers(query)	The number of answers produced by asking the query for 30 seconds.
StoreInTrainingSet	The function that stores the features of the query and the reward in the training

[3] In this work, a term is said to be part of the middle ontology if its generality estimate is between M and N. Through trial and error, we set M and N to be 2,000 and 200,000.

[4] The predicate 'partTypes' is used to relate types of individuals to the types of parts they have. (partTypes A B) means that for every instance A1 of A, there exists an instance B1 of B such that B1 is a part of A1. For example, (partTypes Eye Retina) holds.

[5] (relationAllExists PRED A B) means that for every instance A1 of A, there exists some instance B1 of B such that (PRED A1 B1) holds. For example, (relationAllExists temporalBoundsContain CalendarWeek Wednesday) means that every calendar week contains a Wednesday.

	set.
Query	(p_1 t_{11} t_{12} … t_{1N}) ^ (p_2 t_{21} t_{22} …)^ ...
Focal literal	One of the literals in a multi-literal query is chosen for resolution (or backchaining). The chosen literal is called the focal literal.
Collection$_i$ (i = 1, 2, .., 500)	The set of 500 collections chosen as features for queries.
w(a, j)	The weight associated with feature f_j for axiom *a*.
Feature f_1	Number of literals in query
Feature f_2	Number of free variables in query
Feature f_3	Depth of node
Features f_4-f_{533}	Each feature f_i is 1 if (isa arg1 <col>) holds and 0 otherwise. Here <col> is one of the features (i.e., Collection$_i$) , and arg1 is the first argument to the focal literal.
Features f_{534}-f_{1033}	Each feature f_i is 1 if (isa arg2 <col>) holds and 0 otherwise. Here <col> is one of the features (i.e, Collection$_i$), and arg2 is the second argument to the focal literal.
Features f_{1034}-f_{1533}	Each feature f_i is 1 if (isa arg3 <col>) holds and 0 otherwise. Here <col> is one of the features (i.e, Collection$_i$), and arg3 is the third argument to the focal literal.
Features f_{1534}-f_{2033}	Each feature f_i is 1 if (genls arg1 <col>) holds and 0 otherwise. Here <col> is one of the features (i.e, Collection$_i$), and arg1 is the first argument to the focal literal.
Features f_{2034}-f_{2533}	Each feature f_i is 1 if (genls arg2 <col>) holds and 0 otherwise. Here <col> is one of the features (i.e, Collection$_i$), and arg2 is the second argument to the focal literal.
Features f_{2534}-f_{3003}	Each feature f_i is 1 if (genls arg3 <col>) holds and 0 otherwise. Here <col> is one of the features (i.e, Collection$_i$), and arg3 is the third argument to the focal literal.

```
Algorithm: RandomizedModelLearning
Input: A set of queries, Queries
       A depth cutoff, depth_cutoff
       A set of features, Features
Output: A model for ordering rules

  1. while within computational budget do
  2.         Pick a query q from Queries at random
```

```
3.          depth ← depth_cutoff,
4.          while depth > 0
5.              axioms ← All axioms that apply to predicates in q.
6.              axiom* ← Select an axiom from axioms
7.              Use axiom* to backchain on q and create a child node. Let q* be the child
                node's query.
8.              reward ← NumberOfAnswers (q*)
9.              StoreInTrainingSet (q, axiom*, reward, Features, depth_cutoff-depth).
10.             q ← q*
11.             depth ← depth -1
12. for each axiom a in the KB, do
13.     for each e in TrainingExamples (a) do
14.         prediction ← $\sum_{j=1}^{n} w(a,j)$*x_j [e]
15.         error ← Reward (e) - prediction
16.         w(a, j) ← w(a, j) + α * error * x_j [e]
```

Figure 1: The RandomizedModelLearning Algorithm

The algorithm chooses to select axiom A1 to solve query q, and this choice leads to no answer at depth 0 of the tree. This information is stored in the training set in Step 9. Each entry in the training set takes the following form:

<Axiom, number of literals, number of free variables, depth, f_4, f_5, …, f_{3003}, reward>.

Let us also assume the following definition for features f_4 and f_{1535}.

f_4 is 1 if (isa arg1 CyclicalProcessType) holds ...(C1)
f_{1535} is 1 if (genls arg1 BiologicalProcess) holds. ...(C2)

Here q is a single-literal query, and (properPhysicalPartTypes EukaryoticCellCycle AnaphaseI) is the focal literal. Therefore, C1 and C2 can be rewritten as[6]:

f_4 is 1 if (isa EukaryoticCellCycle CyclicalProcessType) holds ...(C3)
f_{1535} is 1 if (genls EukaryoticCellCycle BiologicalProcess) holds. ...(C4)

The conditions in C3 and C4 hold; therefore both f_4 and f_{1535} are 1.

[6] In C1 and C2, arg1 refers to the first argument to the focal literal. Therefore, it can be replaced by EukaryoticCellCycle.

In Steps 12-16, the algorithm iterates over all the axioms and learns a linear model for them. The utility of using a rule is expressed as a linear function of the features in Step 14, and we learn the weight of those features based on the standard Widrow-Hoff rule [Russell & Norvig 2003].

The complexity of RandomizedModelLearning algorithm is reasonable. If N is the number of axioms in KB, then Steps 5-7 can be completed in O(kN) time, where k is less than 1.[7] Step 8 is executed for a fixed duration of time T (T is set to 30 seconds in this work). Step 9 involves iterating over all features; therefore it takes O(|Features|) time. Steps 4-11 can be completed in O(depth_cutoff * (kN+|Features|). The loop in Step 12 executes once for every axiom in the KB. Step 14 takes O(|Features|) time because we have to combine evidence from all features. Therefore, Steps 12-16 take O(N*|TrainingSet| *|Features|) time, where TrainingSet is the set of all training examples.

4. Experimental Evaluation

The selection of benchmark problems for testing algorithms and heuristics is one of the most important design decisions in research. We used the following principles in this evaluation: (i) AI researchers have often used artificially generated problem instances for testing SAT heuristics. However, the generation of such problem instances has not received enough attention in the commonsense reasoning community. Therefore, we test our heuristics on queries pertaining to real-world events and applications. Programmers and knowledge engineers generated these queries to test the performance of inference engine and different applications (e.g., Project HALO [Friedland et al. 2004], HPKB project [Cohen et al. 1998]), BELLA [Lenat & Durlach 2014]). (ii) Heuristics should be tested on the most difficult problems. To accomplish this: (a) We have tested our heuristics on the largest commonsense KB that uses full FOL[8]; (b) The Cyc KB has thousands of queries of varying levels of difficulty: some of them can be answered in a few milliseconds by focused graph search. On the other hand, some queries require us to generate a deep and large search tree. In this work, we are aiming at improving deep backward inference. Therefore, we excluded simple queries that can be answered by simple lookup or shallow one-step backward inference (e.g., (isa BarackObama Person)). Instead, we focused on queries that are typically slow because they produce a huge search space. We identified a set of 2,000 such queries. Reasoning with these queries was quite difficult for the Cyc system. The results for baseline experiment show that the average time requirements for queries was quite high. In fact, a significant fraction of queries could not be answered in less than 10 minutes.

Recall that the inference engine uses a set of heuristics for ordering nodes during search. The net score of a node s is given by $f(s) = w_0 + w_1 * g_n(s)$, where $g_n()$ is the function learnt

[7] Since Cyc has indexing support for identifying relevant axioms for a predicate, we do not need to iterate over all axioms in the KB.

[8] KBs like ConceptNet might have more GAFs than the Cyc KB, but they do not have axioms for deductive reasoning. Researchers have shown that Cyc-based problems are 1-3 orders of magnitude larger than other problems [see Table 1 in Hoder & Voronkov 2011].

from the RandomizedModelLearning algorithm with *n* training examples, and w_0 is the score returned by heuristics not discussed in this paper. When *n* is 0 (i.e., when there is no learning), the function $g_n()$ returns 0, and the node ordering corresponds to the baseline version for our experiments. By changing *n*, we can assess how performance varies with the size of training set.

We perform a k-fold cross-validation, by dividing the queries into five sets of 400 queries each. One of these sets was kept for validation, whereas the remaining four were used for training purposes. This process was repeated five times with each query subset used for validation. The results of these experiments are shown in Table 2. In this paper, we examine how the number of randomized explorations improve search performance. As discussed above, when the size of the training set is zero, the question answering (Q/A) performance corresponds to the baseline for the experiment. In Table 2, the number of queries is shown in Column 3 (labeled "#Q"). The next column (labeled "%A") shows the proportion of queries that were answered in the experiment. The total time requirements of the queries are shown next. The column labeled "Speedup" shows the improvement in time requirements over the baseline. The last column (labeled "C") shows the improvement in proportion of questions answered over the baseline. The experimental data were collected on a 4-core 3.40 GHz Intel processor with 32 GB of RAM.

Table 2: Experimental Results

Query set	Size of training set	#Q	%A	Time (hours)	Speedup	C(%)
1	0	400	49.25	35.0	-	-
	18,079	400	83.75	11.4	3.07	70
	46,248	400	95.75	3.4	10.29	94
2	0	400	43.50	41.3	-	-
	17,241	400	92.75	5.4	7.64	113
	46,326	400	96.00	3.0	13.40	120
3	0	400	53.75	32.38	-	-
	8,496	400	94.75	3.88	8.34	76

	18,874	400	95.50	3.36	9.63	77
	78,358	400	95.75	3.23	10.02	78
4	0	400	34.50	42.63	-	-
	38,011	400	58.00	24.58	1.73	68
	246,439	400	68.25	15.48	2.75	97
	1,368,720	400	75.00	11.18	3.81	117
5	0	400	36.50	40.91	-	-
	42,899	400	61.00	24.69	1.65	67
	282,255	400	76.00	12.51	3.27	108
	1,287,025	400	78.50	10.20	4.01	115

We see that our approach has led to significant speedups in all cases. Since these models identify useful parts of the search space, they have improved Q/A performance too.

5. Conclusions

This paper has examined an important problem: How can cognitive systems become proficient in deriving deductive proofs in an unknown environment? We have presented a model in which we can sample decisions by conducting random simulations. We can learn search control knowledge from these simulations because the true of value for choosing an axiom in the given state is correlated with the observed result of simulation. Given the large state space, we approximate a state by a set of features, and use function approximation to learn the value of choosing an action given its feature representation. Experiments show that our approach has the characteristics of a statistical anytime algorithm: more computing power generally leads to better performance. These results suggest three lines of future work: (i) to ensure the generality of these methods we would like to test them on a larger set of queries. (ii) we would like to choose actions non-uniformly and use them to guide a search toward more promising states; (iii) we would like to experiment with other schemes of feature selection and analyze how they affect efficiency.

6. References

Alama, J., Heskes, T., Kulhwein, D., Tsivtsivadze, E., & Urban, J. (2014). Premise selection for mathematics by corpus analysis and kernel methods. *Journal of Automated Reasoning*, *52*, 191–213.

Browne, C., Powley, E., et al. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*, 1–49.

Brewka, G., Eiter, T., & Truszczynski, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*, 91–103.

Bridge, J. P., Holden, S., & Paulson, L. (2014). Machine learning for first-order theorem proving. *Journal of Automated Reasoning*, *53*, 141–172.

Cohen, P., et al. (1998). The DARPA high-performance knowledge bases project. *AI Magazine*, *19*, 25–48.

Chaudhuri, S. (1998). An overview of query optimization in relational systems. *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (pp. 34–43).

Forbus, K. D., Riesbeck, C., Birnbaum, L., Livingston, K., Sharma A., & Ureel, L. (2007). Integrating natural language, knowledge representation and reasoning, and analogical processing to learn by reading. *Proceedings of the Twenty-Second National Conference on Artificial Intelligence* (pp. 1542–1547). Vancouver, British Columbia.

Friedland, N., Allen, P., Matthews, G., Witbrock, M., Curtis, J., & Shepard, B. et al. (2004). Project Halo: Towards a digital Aristotle. *AI Magazine*, *25*, 29–47.

Hoder, K., & Voronkov, A. (2011). Sine qua non for large theory reasoning. *Proceedings of the International Conference on Automated Deduction* (pp. 299–314). Springer.

Hutter, F., Xu, L, Hoos, H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence*, *206*, 79–111.

Horrocks, I., & Voronkov, A. (2006). Reasoning support for expressive ontology languages using a theorem prover. *Proceedings of the Eighth International Symposium on Foundations of Information and Knowledge Systems* (pp. 201–218). Springer.

Kaliszyk, C., Urban, J., & Vyskocil, J. (2015). Efficient semantic features for automated reasoning over large theories. *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 3084–3090). Buenos Aires, Argentina.

Kaliszyk, C., & Urban, J. (2015). Learning assisted theorem proving with millions of lemmas. *Journal of Symbolic Computation*, *69*, 109–128.

Lenat, D. B., & Feigenbaum, E. (1991). On the thresholds of knowledge. *Artificial Intelligence*, *47*, 185–250.

Lenat, D. B., & Guha, R. (1990). *Building knowledge-based systems: Representation and inference in the Cyc Project*. Addison Wesley.

Lenat, D. B., & Durlach, P. (2014). Reinforcing math knowledge by immersing students in a simulated learning-by-teaching experience. *International Journal of Artificial Intelligence in Education*.

Matuszek, C., Witbrock, M., Kahlert, R., Cabral, J., Schneider, D., Shah, P., & Lenat, D. (2005). Searching for common sense: Populating Cyc from the Web. *Proceedings of the Twentieth National Conference on Artificial Intelligence*. Pittsburgh, PA.

Matuszek, C., Cabral, J., Witbrock, M., & DeOliveira, J. (2006). An introduction to the syntax and content of Cyc. *Proceedings of the AAAI Spring Symposium*. Palo Alto, CA.

Meng, J., & Paulson, L. C. (2009). Lightweight relevance filtering for machine-generated resolution problems. *Journal of Applied Logic*, *7*, 41–57.

Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach*. Pearson Education.

Sharma, A., & Forbus, K. D., (2013). Automatic extraction of efficient axiom sets from large knowledge bases. *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. Bellevue, WA.

Sharma, A., Witbrock, M., & Goolsbey, K. (2016). Controlling search in very large knowledge bases: A machine learning approach. *Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems*. Evanston, IL: Cognitive Systems Foundation.

Sharma, A., & Goolsbey, K. M. (2017). Identifying useful inference paths in large commonsense knowledge bases by retrograde analysis. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. San Francisco, CA.

Smith, D. E. (1989). Controlling backward inference. *Artificial Intelligence*, *39*, 145–208.

Taylor, M., Matuszek, C., Smith, P., & Witbrock, M. (2007). Guiding inference with policy search reinforcement learning. *Proceedings of the Florida Artificial Intelligence Research Society Conference*. Key West, FL.

Tsarkov, D., & Horrocks, I. (2005). Ordering heuristics for description logic reasoning. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence* (pp. 609–614).

Tsarkov, D., Riazanov, A., Bechhofer, S., & Horrocks, I. (2004). Using Vampire to reason with OWL. *Proceedings of the Semantic Web Conference* (pp. 471–485).