The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Simulation-Based Approach to Efficient Commonsense Reasoning in Very Large Knowledge Bases

Abhishek Sharma, Keith M. Goolsbey
Cycorp Inc., 7718 Wood Hollow Drive, Suite 250, Austin, TX 78731
abhishek@cyc.com, goolsbey@cyc.com

Abstract

Cognitive systems must reason with large bodies of general knowledge to perform complex tasks in the real world. However, due to the intractability of reasoning in large, expressive knowledge bases (KBs), many AI systems have limited reasoning capabilities. Successful cognitive systems have used a variety of machine learning and axiom selection methods to improve inference. In this paper, we describe a search heuristic that uses a Monte-Carlo simulation technique to choose inference steps. We test the efficacy of this approach on a very large and expressive KB, Cyc. Experimental results on hundreds of queries show that this method is highly effective in reducing inference time and improving question-answering (Q/A) performance.

Introduction

Deductive reasoning is an important component of many cognitive systems. Modern cognitive systems need large bodies of general knowledge to perform complex tasks (Lenat & Feigenbaum 1991, Forbus et al 2007). However, efficient reasoning systems can be built only for small-to-medium sized knowledge bases (KBs). Very large knowledge bases contain millions of rules and facts about the world in highly expressive languages. Due to the intractability of reasoning in such systems, even simple queries are timed-out after several minutes. Therefore, researchers believe that resolution-based theorem provers are overwhelmed when they are expected to work on large expressive KBs (Hoder & Voronkov 2011).

The goal query in knowledge-based systems (KBS) is typically provable from a small number of ground atomic formulas (GAFs) and rules. However, unoptimized inference engines can find it difficult to distinguish between a small set of relevant rules and the millions of irrelevant ones. Hundreds of thousands of axioms that are irrelevant for the query can inundate the reasoner with millions of paths. Therefore, to make the search more efficient in such a KBS, an inference engine is expected to assess the utility of further expanding each incomplete path. A naïve ordering algorithm can cause unproductive backtracking. To solve this problem, researchers have used two types of search control knowledge: (i) Axiom/premise selection heuristics: These heuristics attempt to find the small number of axioms that are the most relevant for answering a set of queries, and (ii) Certain researchers have worked on ordering heuristics for improving the order of rule and node expansions.

In the current work, we describe a simulation-based approach for learning an ordering heuristic for controlling search in large knowledge-based systems (KBS). The key idea is to simulate several thousand paths from a node. New nodes are added to a search tree, and each node contains a value that predicts the expected number of answers from expanding the tree from that node. The search tree expansion guides simulations along promising paths, by selecting the nodes that have the highest potential values. The algorithm produces a highly selective and asymmetric search tree that quickly identifies good axiom sequences for queries. Moreover, the evaluation function is not handcrafted: It depends solely on the outcomes of the simulated paths. This approach has the characteristics of a statistical anytime algorithm: The quality of evaluation function improves with additional simulations. The evaluation function is used to order nodes in a search. Experimental results show that: (i) this approach helps in significantly reducing inference time and, (ii) by guiding the search towards more promising parts of the tree, this approach improves the question-answering performance in time-constrained cognitive systems.

This paper is organized as follows: We start by discussing relevant previous work. Our approach to using simulations to learn a search heuristic is explained next. We conclude after analyzing the experimental results.

Related Work

Learning of search control knowledge plays an important role in the optimization of KBS for at least two reasons: First, the inference algorithms of KBS (e.g., backward chaining, tableaux algorithms in description logic (DL)), typically represent their search space as a graph. Hundreds of rules can apply to each node in a large KBS, and researchers have shown that the specific order of node and rule expansion can have a significant effect on efficiency (Tsarkov & Horrocks 2005). Further still, first-order logic (FOL) theorem provers have been used as tools for such reasoning with very expressive languages (e.g., OWL DL, the Semantic Web Rule Language (SWRL)), where the language does not correspond to any decidable fragment of FOL, or where reasoning with the language is beyond the scope of existing DL algorithms (Tsarkov et al. 2004, Horrocks & Voronkov 2006). Researchers have also examined the use of machine learning techniques to identify the best heuristics for problems (Bridge et al. 2014). There has been work as well on the premise selection algorithms (Hoder & Voronkov 2011, Sharma & Forbus 2013, Meng & Paulson 2009, Kaliszyk et al. 2015, Kaliszyk & Urban 2015, Alama et al. 2014). In contrast, we focus on designing ordering heuristics that will enable the system to work with all axioms. In (Tsarkov & Horrocks 2005), the authors proposed certain rule-ordering heuristics, and expansion ordering heuristics (e.g., a descending order of frequency of usage of symbols). In contrast, we believe that rule-ordering heuristics should be based on the search state. In other recent work (Sharma, Witbrock & Goolsbey 2016, Sharma & Goolsbey 2017), researchers have used different machine learning techniques to improve the efficiency of theorem provers, ore approach is different from all aforementioned research because none have shown how Monte Carlo tree search/UCT-based approach can be used to improve reasoning in a very large and expressive KBS. The work in other fields (Chaudhuri 1998, Hutter et al. 2014, Brewka et al. 2011) is also less relevant because it does not address the complexity of reasoning needed for large and expressive KBS.

Background

We assume familiarity with the Cyc representation language (Lenat & Guha 1990). In Cyc, concepts are represented as collections. For example, "Cat" is the set of cats and only cats. Concept hierarchies are represented by the "genls" relation. For example, (genls Telephone Artifact-Communication) holds. The "isa" relation is used to link things to any kind of collections of which they are instances. For instance, (isa MicrosftInc PubliclyHeldCorporation) holds. For any entity e, Cyc keeps track of the most specific collections of which it is an instance. This set is referred to as MostSpecificCollections (e).

Reasoning with Cyc representation language is difficult due to the size of the KB and the expressiveness of the language. The Cyc representation language uses full first-order logic with higher-order extensions. Some examples of highly expressive features of its language include: (a) Cyc has more than 2000 rules with quantification over predicates, (b) it has 267 relations with variable arities, and (c) Although first-order logic with three variables is undecidable (Tsarkov et al. 2004), Cyc ahs several thousand rules with more than three variables. The number of rules in Cyc with three, four and five variables are 48160, 23813 and 14014 respectively. Moreover, 29716 rules have more than five variables. In its default inference mode, the Cyc inference engine uses the following types of rules/facts during inference: (i) 28,429 role inclusion axioms; (ii) 3,623 inverse role axioms, (iii) 494,405 concepts and 1.1 million concept inclusion axioms; (iv) 814 transitive roles; (v) 120,547 complex role inclusion axioms; (vi) 77,170 other axioms; (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 efficiency search in such a large KBS, inference engines often use control strategies. They define: (i) set of support, i.e. the set of important facts about the problem.; and (ii) the set of usable axioms, i.e. the set of all axioms outside the set of support. At every inference step, the inference engine has to select an element from the set of usable axioms and resolve it with an element of the set of support. To perform efficient 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 & Norvig 2003). Cyc uses a set of heuristic modules to identify the best clause from the set of support. If S is the set of all states, and A is the set of actions (i.e. inference steps), then a heuristic module is a tuple h_i: (w_i, f_i) , where f_i is a function f_i: $S \times A \rightarrow R$, which assesses the quality of an inference step, and w_i is the weight of h_i. The net score of an inference step is $\sum w_i f_i(s, a)$ and the inference step with the highest score is selected. Cyc uses several heuristics including the success rates of rules, decision trees (Sharma, Witbrock and Goolsbey 2016), regression-based models (Sharma, Witbrock, and Goolsbey 2016) and a large database of useful rule sequences (Sharma & Goolsbey 2017). We can define a policy that uses these heuristics:

$$\Pi_{BASELINE}(s) = \text{argmax}_a \sum w_i f_i(s, a) \quad \ldots(1)$$

In (1), we use all heuristics mentioned above to calculate the score of an inference step. In its default inference mode, Cyc uses a policy function $\pi_{BASELINE}(s)$, to guide

the search. This is used as the “baseline” in the experiments discussed below.
We consider the problem of finding proofs for first-order formulas in large KBs. The search begins with a root state s_o[1]. At each turn, the inference engine selects an action that is an element of A(s), where *s* is the current state, and A(s) is the set of inference steps applicable to the state *s*. This sequential sampling of states and actions continues, without backtracking, until the search finishes upon reaching a terminal state with outcome z. The aim of the reasoner is to maximize z. A policy π is a function that maps states to actions. $\pi(s)$ is defined as $\max_a Q(s, a)$ where Q(s, a) is used to denote the value of selecting action *a* in state *s*. It is simply the expected reward of action a.

$$Q(s, a) = N(s, a)^{-1} * \sum I_i(s, a) * z_i$$

where N(s, a) is the number of times action a has been selected in state s, N(s) is the total number of times the simulaton has passed through state *s*, z_i is the outcome of the i^th simulation, and $I_i(s, a)$ is 1 if action a was selected in state *s* in the i^th simulation, and 0 otherwise (Browne et al 2012).

Simulation-Based Learning

The basic premise of this approach is that we can learn to search for answers in large expressive KBS by taking random samples in the search space and these samples could be used to approximate the true value of choosing an inference step. Monte Carlo tree search (MCTS) algorithms use Monte Carlo simulations to evaluate the quality of nodes in a search tree. The search tree contains a node for each state s that has been generated during simulations. Each state s in the tree stores three types of values: N(s), N(s, a) and Q(s, a) for every action that applies to s. the generation of search trees is guided by the outcomes of previous explorations, and the estimates become progressively more accurate (Browne et al. 2012, Gelly & Silver 2007).

Figure 1 shows the high-level approach of the simulation algorithm. The algorithm takes a state and depth cutoff as input. The state contains information about the query that has to be answered. The algorithm can be divided into two distinct phases: (i) a tree policy is used until depth d: During this stage, the algorithm selects actions according to knowledge contained with the search tree (ii) when the depth of the node is greater than d, then a default policy is used to complete the simulation. Finally, the outcome of the simulation is “backed up” through the selected nodes to update their statistics.

[1] In search problems, an agent needs to find the correct move for each position encountered during search. Therefore, each node generated in a search graph is a state, and is represented by a set of features (discussed below).

Tree policy: In step 2(a) of the algorithm in Figure 1, we use the UCT algorithm to build a tree over the state space with the current node as the root node. The execution of the TreePolicy uses the following mechanism: if there exists an action in A(s), the set of possible actions in state s, which has never been selected, the algorithm defaults to selecting it before any sampled action. Otherwise, the UCT algorithm selects an action that maximizes the upper confidence interval given by

```
Input: A state s
       A depth cutoff, d

    1. Create a root node with states.
    2. While within computational budget,
       do
           a. s' ← TreePolicy (s, d)
           b. z ← DefaultPolicy (s')
           c. UpdateValues (s', z)
```

Figure 1: High-Level Description of the Simulation Algorithm

$$Q^*(s, a) \leftarrow Q(s, a) + c * (\log (N(s) / N(s, a))^{1/2} \qquad \ldots(2)$$

The first term in (2), Q(s, a) favors actions that have led to better outcomes in the past. The second term provides a balance between exploiting actions that currently appear sub-optimal but may turn out to be better in the long run (Gelly & Silver 2011, Bella & Fern 2009, Finnsson & Bjornsson 2008). When an action is chosen, N(s, a) increases, and all other actions become more likely to be selected. The value of the exploration parameter c helps is in biasing the search toward or against exploration. In this paper, we report results of several experiments that show how performance changes with c. the tree policy is used to generate a tree until depth d.

Default policy: our algorithm uses a default policy for expanding nodes that have depth greater than the pre-specified depth limit. Cyc’s existing mechanism for selecting inference steps (discussed above) is used as the default policy. The default policy is executed for a fixed duration of time (set to 30 seconds).

Updating Values: In step 2 (c) of the algorithm shown above, the algorithm reaches a terminal state and observes the number of answers obtained from the simulation, then the following updates are made for any state action pair on the simulation path:

$$N(s, a) \leftarrow N(s, a) + 1$$
$$N(s) \leftarrow N(s) + 1$$
$$Q(s, a) \leftarrow Q(s, a) + N(s, a)^{-1} * [\, R - Q(s, a)] \quad \ldots (3)$$

Search Space Formulation

We now define a search space that can be used the UCT algorithm. We used 2,698 features to represent an abstract state in search space. These features were identified by selecting the commonly occurring types in the most specific collections of entities in the KB [defined as MostSpecificCollections() on page 2]. These features can be grouped into three types: (i) 2,664 features were used represent whether a certain argument of the focal literal[2] is a sub-type of a given collection[3]; (ii) 33 features were used to represent whether a certain argument of the focal literal is an instance of a given collection[4]; and (iii) the depth of the sub-goal in the tree. Each node in the search space is described by a vector of 2,698 features. Edges in our search space correspond to resolution steps that transform one node into another. Let us consider an example. During training, an agent might create a root node with query Q1 from the training set[5].

(relationAllExists parts Nivalenol Carbon) … (Q1)

During the MCTS simulation, the UCT algorithm might lead the algorithm to choose the inference step A1 shown below:
(memberOfList ?element ?list) .
(completeAtomicComposition-List ?compound ?list ?coefficients) $\rightarrow$
(relationAllExists parts ?compound ?element) …(A1)

The selection of inference step A1 will lead to a child state with the query Q2:

(memberOfList Carbon ?list) AND
(completeAtomicComposition-List Nivalenol ?list ?coefficients) …(Q2)

The Monte Carlo tree search continues for a given number of simulations, and it learns the relative contribution of each feature to the likelihood of deriving an answer. The action value function Q(s, a) is approximated by a partial tabular representation $\mu \subseteq S \times A$, where S and A are the sets of all states and actions respectively. μ contains the search tree of all visited states and it is a subset of all (state, action) pairs. Unfortunately, this tabular representation does not allow for easy generalization between states. Therefore, we consider a simple k-nearest neighbor algorithm to generalize from similar states. We define the following:

$Q_{NN}(s, a) = k^{-1} \sum Q(s(i), a)$,

where s(i) is an element of NN(k, s) …(4)

In (4), NN(k, s) denotes the k-nearest neighbor of state s, where we use the Manhattan distance to compute the distance between two states.

Given (4), we can define a heuristic module with the following cost and policy function:

$f_{MCTS}(s, a) = Q_{NN}(s, a)$ … (5)
$\pi_{MCTS}(s) = \text{argmax}_{a}\, Q_{NN}(s, a)$ … (6)

This heuristic module uses the k-nearest neighbor algorithm on the output of the Monte Carlo tree search algorithm, an to evaluate the quality of inference steps. In the next section, we use this heuristic module to order inference steps and discuss its performance.

Experimental Results

The selection of benchmark problems for training models and evaluating algorithms is a critical aspect of research. Our decision to select problem instances was based on following principles: (a) Artificially generated problems have played an important role in the development of SAT algorithms However, the generation of artificial problems has not received sufficient attention in the commonsense reasoning community. Therefore, we focused on the problems from the real world. These queries were created by the knowledge engineers and the programmers to test the performance of different applications and the inference engine[6]. Query Q1 discussed above is an example of a query that is part of our test sets. (b) We believe that heuristics and algorithms should be tested using the most difficult problems. The Cyc KB has thousands of queries of various levels of difficulty. Some queries are quite simple, and can be answered in a few milliseconds (e.g., (isa BarackObama Person)). On the other hand, some require generation of a large search space, and cannot be answered in several minutes. In this work, we have included problems from the latter group, and the results from the baseline experiment show that many queries cannot be answered within 10 minutes. Moreover, the Cyc KB is the largest and most

[2] In a conjunctive or disjunctive query, the inference engine might decide to resolve one of the literals in the query. The literal that is resolved is called the focal literal.
[3] An example of the feature for query Q1 would be that second and third arguments of the focal literal are sub-types of "Trichothecene" and "NonMetal" respectively.
[4] An example of this feature for the query Q1 would be that the first argument of the focal literal is an instance of TransitiveBinaryPredicate.
[5] (relationAllExists parts A B) means that for any instance A1 of A, there exists an instance B1 of B, such that (parts A1 B1) holds.

[6] The query parameters (e.g., the number of desired answers) were set by knowledge engineers and programmers. In most cases, we were expected to find one answer for a fully bound query (e.g., query Q1 discussed above).

expressive knowledge base that is amenable to deductive reasoning[7]. Therefore, our algorithm has been tested on one of the most difficult reasoning problems.

We divided the queries into four parts. One of the four parts was used as a test set, while the queries from the remaining three were used for training purposes[8]. This process was repeated with each of the four parts to produce four experiments. In each experiment, we tried to answer the following questions:

- How does search performance change with the number of simulations?
- How does search performance change with the value of c?

The results of these four experiments are shown in Figures 2-5. We compared the performance of $\pi_{BASELINE}(s)$ and $\pi_{MCTS}(s)$ in these experiments. In the results, speedup in experiment e is defined as:

$$\text{Speedup}(s) = \text{Time}(\pi_{BASELINE}, e) / \text{Time}(\pi_{MCTS}, e)$$

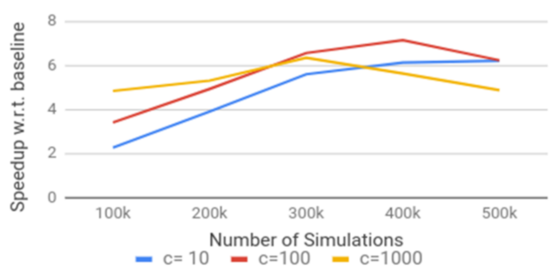

Figure 2a: Effect of exploration factor ('c') and number of simulations on search performance ('Speedup') in Experiment 1.

where Time (π, e) refers to the time required by the inference engine to answer queries in the experiment e when it uses the search policy π. The graphs also show how the proportion of queries that could be answered changes with the value of 'c' and the number of simulations. The experimental data reported here was collected on a 4-core 3.40 GHz Intel processor with 32 GB of RAM. Due to the large time requirements of some of these queries, we restricted the cutoff time for each query to 10 minutes. In Table 1, we show the parameter values for the best results obtained for each of the four experiments. For example, in the first experiment we could answer 46% of 266 queries in the baseline experiment. The best results from the MCTS simulation was obtained when c was set to 100, and we allowed 400,000 simulations to learn the Q-values. This led to a speedup of 7.1 and we could answer 92% of all queries.

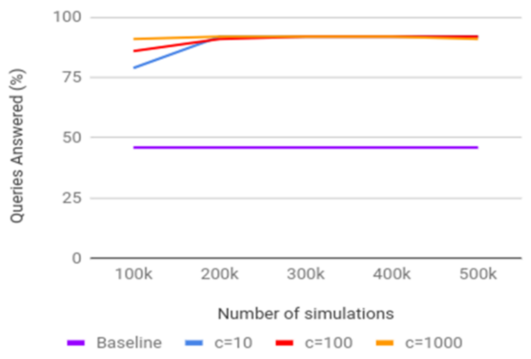

Figure 2b: Effect of exploration factor ('c') and number of simulations on Q/A performance in Experiment 1.

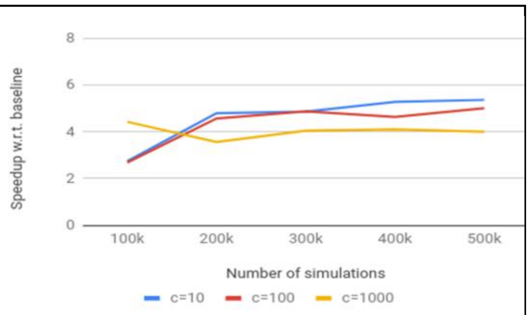

Figure 3a: Effect of exploration factor ('c') and number of simulations on search performance ('Speedup') in Experiment 2.

The results demonstrate the following:

- The search policy learned from MCTS simulation leads to significant speedup compared to the baseline.
- Moreover, we see that MCTS-based learning has also led to significant improvement in the Q/A performance.
- For small values of simulations (~100k), higher values of c (i.e., c = 1000) led to the best performance. However, the performance asymptotes soon. And the best performance is obtained for small to medium values of c (i.e., c = 10 or 100).

[7] 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).
[8] For example, in experiment 2, query set 2 was used for testing purposes, whereas queries from sets 1, 3 and 4 formed the training set.

We also divided the queries by "degree of difficulty" as measured by the time required to answer them by the baseline version of the inference engine. The speedup and improvement in the Q/A performance for the best MCTS settings are shown in Table 2. For example, there were 597 queries that needed between 0 and 100 seconds in the baseline version. The best speedup obtained from MCTS simulations for these 597 queries was 1.23 and there was no change in the number of answerable queries.

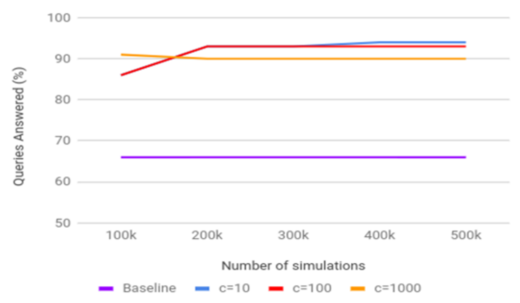

Figure 3b: Effect of exploration factor ('c') and number of simulations on Q/A performance in Experiment 2.

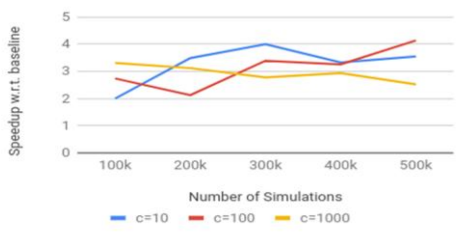

Figure 4a: Effect of exploration factor ('c') and number of simulations on search performance ('Speedup') in Experiment 3.

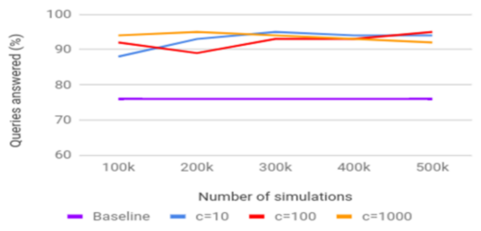

Figure 4b: Effect of exploration factor ('c') and number of simulations on Q/A performance in Experiment 3.

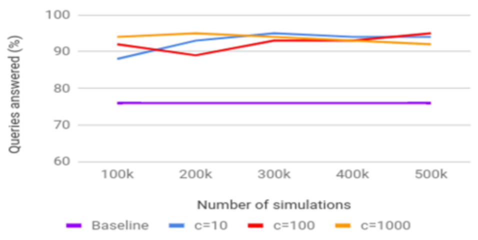

Figure 5a: Effect of exploration factor ('c') and number of simulations on search performance ('Speedup') in Experiment 4.

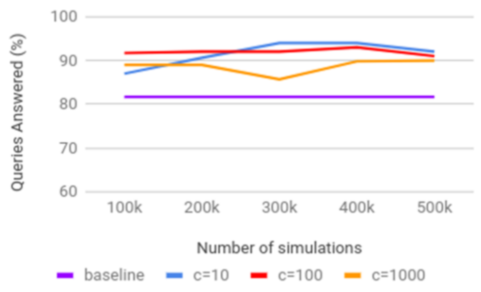

Figure 5b: Effect of exploration factor ('c') and number of simulations on Q/A performance in Experiment 4.

Conclusions

Deductive reasoning is an important issue for building large cognitive systems. To make deductive reasoning more efficient, in this work, we have proposed a sample-based search paradigm for learning search control knowledge. In this approach, we learn from simulated episodes that can be sampled from the model. Results for hundreds of queries from a very large and expressive KB show that this approach can lead to a significant reduction in inference time. It can also lead to notable improvement in Q/A performance. These results suggest several areas for future work: (i) First, we want to test these methods on a larger set of queries to ensure their generality, (ii) since feature selection plays an important role in the performance of learning algorithms, we will experiment with other schemes of feature selection, and (iii) we will design techniques to make Monte Carlo searches more efficient by including more domain knowledge in the simulation algorithm.

E	Π	#Q	%A	C	TS	S	I(%)
1	B	266	46	-	-	-	-
	MCTS	266	92	100	400k	7.1	100
2	B	254	66	-	-		-
	MCTS	254	94	100	500k	5.4	42
3	B	261	76	-	-	-	-
	MCTS	261	94	100	500k	4.1	24
4	B	241	82	-	-		-
	MCTS	241	94	10	300k	3.3	14

Table 1: The experiment numbers are shown in the first column (labeled 'E'), and the second column shows the search policy that was used in the experiment: baseline (B) or MCTS. The third column (labeled #Q) shows the number of queries in each of the experiments. The proportion of queries answered is shown in the fourth column. The next column (labeled 'c') shows the value of c that led to these results. The sixth column (labeled 'TS') shows the number of simulations used in the Monte Carlo search. The column labeled 'S' shows the speedup obtained in the experiment. Finally, the last column (labeled I(%) shows the improvement in the number of queries answered w.r.t. to the baseline).

Time(sec.)	#Q	S	I(%)
0-100	597	1.23	0
101-200	50	3.33	-2
201-300	38	13.79	0
301-400	9	11.79	0
401-500	1	18.2	0
>500	327	6.44	14600

Table 2: Column 1 shows the time requirement for the query in baseline version. The second column shows the number of queries in the group. The third column shows the speedup w.r.t. to the baseline. And the fourth column (labeled I(%)) shows the percent improvement in the number of queries that were answered.

Acknowledgements

We thank Doug Lenat and Saket Joshi for many helpful discussions and suggestions.

References

Engelmore, R., and Morgan, A. eds. 1986. *Blackboard Systems.* Reading, Mass.: Addison-Wesley.

Alama, J.; Heskes, T; Kuhlwein, D.; Tsivivadze, E.; and Urban, J. 2014. Premise Selection for Mathematics by Corpus Analysis and Kernel Methods. *Journal of Automated Reasoning,* 52(2): 191-213.

Balla, R. and Fern, A. 2009. UCT for Tactical Assault Planning in Real-Time Strategy Games. *Proceedings of the IJCAI,* 40-45.

Brewka, G.; Eiter, T.; and Truszcynski, M. 2011. Answer Set Programming at a Glance. *Communications of the ACM,* 54(12), 91-103.

Bridge, J. P.; Holden, S.; and Paulson, I. 2014. Machine Learning for First-Order Theorem Proving. *Journal of Automated Reasoning,* 53(2), 141-272.

Browne, C.; Powle, E., Whitehouse, D., Lucas, S., Cowling, P. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and Games.* 4(1), 1-55.

Cohen, P. 1998. The DARPA High Performance Knowledge Bases Project. *AI Magazine,* 19(4), 25-48.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern Databases, *Computational Intelligence,* 14(3), 318-334.

Chaudhuri, S. 1998. An Overview of Query Optimization in Relational Systems, *Proceedings of PODS,* 34-43.

Finnsson, H.; and Bjornsson, Y. 2008. Simulation-Based Approach to General Game Playing. *Proceedings of AAAI,* 259-264.

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 AAAI,* 1542-1547.

Gelly, S.; and Silver, D. 2011. Monte Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence,* 175, 1856-1875.

Gelly, S.; and Silver, D. 2007. Combining Online and Offline Knowledge in UCT, *Proceedings of the ICML,* 273-280.

Hoder, K.; and Voronkov, A. 2011. Sine qua non for Large Theory Reasoning. *Proc. of CADE,* 299-314.

Horrocks.I.; and Voronkov, A. 2006. Reasoning Support for Expressive Ontology Languages Using A Theorem Prover. *Foundations of Information and Knowledge Systems.* 201-218.

Hutter, F.;, and Leyton-Brown, K. 2014. Algorithmic Runtime Prediction: Methods and Evaluation. *Artificial Intelligence,* 206, 79-111.

Kaliszyk, C.; Urban, J.; and Vyskocil. J. 2015. Efficient Semantic Features for Automated Reasoning Over Large Theories. *Proceedings of the IJCAI,* 3084-3900.

Kaliszyk, C.; and Urban, J. 2015. Learning Assisted Theorem Proving with Millions of Axioms. *Journal of Symbolic Computation,* 69, 109-128.

Lenat, D. B.;, Feigenbaum, E. 1991. On the Thresholds of Knowledge. *Artificial Intelligence,* 47 (1-3), 185-250.

Lenat, D. B.; and Guha, R. 1990. *Building Knowledge-based Systems: Representation and Inference in the Cyc Project*. Addison Wesley.

Matuszek, C.; Witbrock, M.; Shah, P.; and Lenat, D. 2005. Searching for Common Sense: Populating Cyc from the Web. *Proceedings of the AAAI,* 1430-1435.

Matuszek, C.; Cabral, J.; Witbrock, M.; DeOliveira, J.. 2006. An Introduction to the Syntax and Conent of Cyc. *AAAI Spring Symposium,* 44-49.

Meng, J.; and Paulson, C. 2009. Lightweight Relevance Filtering for Machine Generated Resolution Problems. *Journal of Applied Logic,* 7(1), 41-57.

Robles, D.; Rohlfshagen, P.; and Lucas, S. 2011. Learning Non-Random Moves for Playing Othello: Improving Monte Carlo Tree Search. *Proceedings of IEEE Conference on Computational Intelligence and Games.* 10-16.

Russell, S.; and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach.* Pearson Education.

Sharma, A.; and Forbus, K. D. 2013. Automatic Extraction of Efficient Axiom Sets from Large Knowledge Bases. *Proceedings of AAAI,* 1248-1254.

Sharma, A.; Witbrock, M.; and Goolsbey, K. M. 2016. Controlling Search in Very Large Knowledge Bases: A Machine Learning Approach. *Advances in Cognitive Systems,* 200-216.

Sharma, A.; and Goolsbey, K. M. 2017. Identifying Useful Inference Paths in Large Commonsense Knowledge Bases By Retrograde Analysis. *Proceedings of AAAI,* 4437-4443.

Tsarkov, D.; and Horrocks, I. 2005. Ordering Heuristics for Description Logic Reasoning. *Proceedings of IJCAI,* 609-614.

Tsarkov, D., Riazanov, A., Bechofer, S.; and Horrocks, I. 2004. Using Vampire to Reason with OWL. *Proceedings of the Semantic Web-ISWC,* 471-485.