Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

# Identifying Useful Inference Paths in Large Commonsense Knowledge Bases by Retrograde Analysis

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

**Abstract**

Commonsense reasoning at scale is a critical problem for modern cognitive systems. Large theories have millions of axioms, but only a handful are relevant for answering a given goal query. Irrelevant axioms increase the search space, overwhelming unoptimized inference engines in large theories. Therefore, methods that help in identifying useful inference paths are an essential part of large cognitive systems. In this paper, we use retrograde analysis to build a database of proof paths that lead to at least one successful proof. This database helps the inference engine identify more productive parts of the search space. A heuristic based on this approach is used to order nodes during a search. We study the efficacy of this approach on hundreds of queries from the Cyc KB. Empirical results show that this approach leads to significant reduction in inference time.

## Introduction and Motivation

Cognitive systems need large bodies of general commonsense knowledge about the external world to perform complex, real-world tasks robustly [Lenat & Feigenbaum 1991, Forbus *et al*. 2007]. Therefore, domain-general knowledge-based systems (KBSs) contain millions of commonsense axioms and other general facts. KBSs often draw deductively valid conclusions from known facts to make plans and to reason about the external world. Typically, the goal query is provable from a very small subset of millions of axioms and facts available in the knowledge base (KB). However, due to the intractability of reasoning in first-order logic (FOL), unoptimized inference engines find it difficult to distinguish between a handful of relevant axioms and the millions of other axioms/facts available in the KB. Without search control knowledge, inference engines cannot answer even relatively simple queries within minutes. Therefore, it is common thinking that resolution-based theorem provers are overwhelmed when they are expected to work on large KBs [Hoder & Voronkov 2011]. Efficient reasoning in such systems is critical for developing large-scale realistic cognitive systems.

Inference algorithms of expressive KBSs (e.g., backward inference in Cyc, tableaux algorithms in description logic (DL)) typically represent the search space as a graph, the structure of which is determined by the axioms applicable to nodes in the graph. Generally, hundreds of rules simultaneously apply to a node and the order of rule and node expansion can have a significant effect on efficiency [Tsarkov & Horrocks 2005, Sharma *et al.* 2016]. Therefore, ordering heuristics play an important role in the optimization of reasoning in large KBSs.

Large KBSs have a very large set of axioms Ax, and a goal G. Since only a tiny subset, S, of Ax is sufficient for proving G, researchers have worked to identify a small set R, such that $S \subset R \subset Ax$. The relevant set of axioms, R, can be used in a variety of ways. The inference engines of KBSs can prune axioms not in R, or can use them in ordering algorithms by prioritizing resolution steps that use axioms in R. In this paper, we argue that retrograde analysis can be used to identify useful parts of the search space. We propose an algorithm that uses this analysis to identify useful proof paths by using a bottom-up level-by-level path expansion approach. The resulting database of proof paths helps us in identifying a small number of axioms that can be useful in a given search state. This database is used to order nodes during a search. Experimental results show that this approach helps in significantly reducing inference time.

This paper is organized as follows. We start by discussing relevant previous work. Our approach to generating a database of useful axiom sequences is presented next. We conclude by discussing our results and plans for future work.

## Related Work

Researchers have examined the use of machine learning to identify the best heuristics[1] for solving problems [Bridge *et al*. 2014] and to select a small set of axioms/lemmas that is most relevant for answering a set of queries [Sharma and Forbus 2013, Kaliszyk *et al*. 2015, Kaliszyk and Urban 2015, Alama *et al.* 2014]. [Hoder and Voronkov 2011] present a symbol-based axiom selection scheme, which is based on the co-occurrence of symbols in axioms. [Meng & Paulson 2009] apply a relevance based filtering approach in which a clause is added to a set of relevant clauses if it is sufficiently close to an existing relevant clause. [Taylor *et al*. 2007] use reinforcement learning to guide inference, whereas [Tsarkov and Horrocks 2005] study different types of rule-ordering heuristics (e.g., preference between ∃ and ⊔ rules) and expansion-ordering heuristics (e.g., descending order of frequency of usage of each of the concepts in the disjunction). In contrast, our work was initially inspired by research in the computer games community where exhaustive offline search has been used to construct pattern and endgame databases that have proven to be quite useful for traversal in huge search spaces [Culberson & Schaeffer 1998, Thompson 1986]. Our approach of extracting useful inference paths is similar to existing methods in the graph mining community [Kuramochi & Karypis 2001]. Work in other fields (e.g., database community [Chaudhuri 1998], SAT reasoning [Hutter *et al.* 2014], answer set programming [Brewka *et al*. 2011]) is less relevant because the studies do not address the complexity of deep and cyclic search graphs that arise from expressive first-order reasoning. To the best of our knowledge, no work in the AI community has used methods similar to ours to control inference in large commonsense reasoning systems.

## Background

We assume familiarity with the Cyc representation language [Lenat and Guha 1990, Matuszek *et al.* 2006, Taylor *et al*. 2007]. In Cyc, concept hierarchies are represented by the 'genls' relation. For instance, (genls Person Mammal) holds. During backward inference, the rule P(x) → Q(x) is used to *transform* a query like Q(a) into P(a). The link between Q(a) and P(a) is a type of *transformation link.* Every node in the search graph is timestamped with an *id*. A node *y* is called a *successor* of *x* if there is a path consisting of transformation links from *x* to *y* and *id*(*x*) < *id*(*y*). A node *x* is a *parent* of *y* if a transformation link exists between *x* and *y* and *id(x)* < *id(y)*. *Parents*(*x*) and *Successors*(*x*) denote the sets of all parents and successors of node *x* respectively. Let S be the set of all nodes in a search graph. Then, a *transformation link set* p = {a(1), a(2), …,a(n)} is a set of transformation links that transform an initial state $s_0$ to an intermediate state $s_n$. *TRule(a)* and *Substitutions(a)* denote the rule and bindings associated with the transformation link *a*.

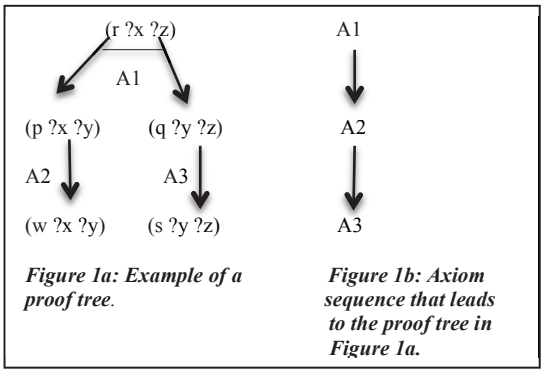


***Figure 1a: Example of a proof tree.***

***Figure 1b: Axiom sequence that leads to the proof tree in Figure 1a.***

Cognitive systems often need to make deep inference chains. Let us introduce some notation with the help of the following example:

(p ?x ?y) ∧(q ?y ?z) → (r ?x ?z) … (Axiom A1)
(w ?x ?y) → (p ?x ?y) …(Axiom A2)
(s ?x ?y) → (q ?x ?y) …(Axiom A3)
(m ?x ?y) → (w ?x ?y) …(Axiom A4)

Given a query of the type (r ?x ?z), the inference engine would create the proof tree shown in Figure 1a. Since we are primarily interested in sequences of axiom choices during resolution, we aim to understand the utility of axiom sequence chosen by the inference engine (see Figure 1b).

In Table 1, we introduce the notation that is used in the next section. $Candidates^k$ refers to the set of all proof paths consisting of *k* axioms. Each element of $Candidates^k$ is a tuple <head, RuleSet>, where head is the rule used at the root node of the proof tree, and *RuleSet* is the set of all rules used in the proof. For instance, the proof path shown in Figure 1b is an element of $Candidates^3$, and it is represented as <A1, {A1, A2, A3}>. Some of the these proof paths are productive, and solutions for open variables of the root node can be found. The set of all axiom sequences that correspond to productive paths forms the set $G^k$, and graph.count refers to the number of solutions for the top-level query. Therefore, if the proof path shown in Figure 1a leads to two solutions for the root node, then the axiom sequence shown in Figure 1b would be an element of $G^3$, and graph.count for the corresponding graph would be 2. Finally, we use RootPred(graph) to refer to the predicate in the root node of the graph.

[1] Examples of heuristics (or strategies) include "give priority to axioms in clause selection" and "sort symbols by inverse frequency".

*Table 1: Notation*

| Notation | Description |
|---|---|
| $G^k$ | The set of proof trees that lead to successful proof(s) and use *k* axioms for resolution. |
| $Candidates^k$ | The set of candidates proof paths that use *k* axioms. |
| graph.count | The number of solutions for the root node when the inference engine is allowed to only use axioms in *graph.* |
| RootPred(graph) | The predicate in the root node of the graph. |
| Rules(graph) | The set of rules used in the graph. |
| Head(graph) | The rule used at the root node of the proof tree. |

Reasoning in Cyc KB 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) 21,743 role inclusion axioms (e.g., P(x, y) → Q(x, y)), (ii) 2,601 inverse role axioms (e.g., P(x, y) → Q (y, x)), (iii) 365,593 concepts and 986,965 concept inclusion axioms (i.e., 'genls' facts), (iv) 817 transitive roles, (v) 99,238 complex role inclusion axioms (e.g., P(x, y) ∧ Q (y, z) → R (x, z)), and (vi) 31,897 binary roles and 7,980 roles with arities greater than 2. The KB has 21.7 million assertions and 652,037 individuals. To control search in such a large KBS, inference algorithms use various strategies. They distinguish between a set of clauses known as the *set of support*[2] that defines the important facts about the problem and a set of *usable axioms* that is outside the set of support (e.g., see the OTTER theorem prover [Russell and Norvig 2003]). At every step, such theorem provers 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 → $\mathbb{R}$ that assesses the quality of a node, and $w_i$ is the weight of $h_i$. The net score of a node *s* is $\Sigma_i w_i f_i(s)$, and the node with the highest score is selected for further expansion. Cyc uses a number of heuristic modules for controlling the search. One of these modules prefers rules that have a higher success rate. Other heuristic modules use decision trees and linear regression-based models [Sharma *et al*. 2016]. In the next section, we discuss a new heuristic module for ordering nodes during a search.

[2] For instance, the negated query is often used as the set of support.

## Retrograde Analysis

The basic idea behind this approach involves finding the value of using axioms at the end of proof construction, and backing up the values towards the root node (cf, Bjornsson *et al.* 2005). Let us illustrate this with the help of an example. Consider the axioms shown below:

(isa ?PUNISH Punishing)∧(performedBy ?PUNISH UNSecurityCouncil)
→ (performedBy ?PUNISH UnitedNationsOrganization) ...(Axiom A5)
(isa ?INS1 HidingOneself) ∧ (isa ?INS2 Agent-PartiallyTangible)
(objectHidden ?INS1 ?INS2) → (performedBy ?INS1 ?INS2) ..(Axiom A6)

Given a query of the type (performedBy ?event UnitedNationsOrganization), an inference engine would backchain on Axiom A5 and lead to a sub-query (performedBy ?event UNSecurityCouncil). It would then use Axiom A6 to answer it, and that would lead to the sub-goal (objectHidden ?event UNSecurityCoucil). Such search paths are unlikely to succeed. Implausible search paths are pervasive in large KBSs and recent research work has focused on steering the search away from these doomed paths [Sharma *et al.* 2016]. In this paper, we have taken the approach that identifying useful proof paths can be seen as finding connected sub-graphs that appear in at least one successful proof. We argue that the candidate longer proof paths can be generated by adding one axiom at a time to smaller successful proof paths. The high-level structure of our algorithm is shown in Figure 2.

The algorithm generates small proof trees first and extends them by adding a single axiom to them in a bottom-up fashion. In step 1 of the algorithm, we identify all axioms that could be the leaves of successful proof trees. The last step of a proof reduces to querying the antecedent of an axiom, and inference algorithms use ground atomic formulas (GAFs) to complete the proof. Therefore, if an axiom appears in the last step of a successful proof, then GAFs must suffice to find at least one solution for its antecedent.

**Example 1**: Consider axiom A3 used in the proof tree shown in Figure 1a. The query corresponding to its antecedent is (s ?y ?z). If the KB has no GAFs to answer that query, then the inference engine must use other axioms to answer it. Consequently, (s ?y ?z) cannot be a leaf node, and A3 cannot be used in the last step of a proof.

**Example 2**: If the KB has no instances of HidingOneself then the GAFs alone would not suffice to solve the query represented by the antecedent of axiom A6 (shown below) and A6 cannot appear in the last step of a proof.

(isa ?INS1 HidingOneself) ∧ (isa ?INS2 Agent-PartiallyTangible) ∧ (objectHidden ?INS1 ?INS2)

Algorithm 1: *Chains*(KB, α, depth)

1. $G^1 \leftarrow$ Detect all chains of length 1 in KB.
2. $k \leftarrow 2$.
3. **while** ($G^{k-1} \neq \Phi$ and k < depth)
4. $Candidates^k \leftarrow$ GenCandidates ($G^1,\ldots,G^{k-1}$, KB)
5. **for each** candidate $graph^k \in Candidates^k$ ***do***
6. $graph^k.count \leftarrow$ EstimateSolutionCount($graph^k$, KB)
7. $G^k \leftarrow \{ graph^k \in Candidates^k | graph^k.count > \alpha\}$
8. $k \leftarrow k+1$
9. **return** $G^1, G^2, \ldots, G^{depth}$

***Figure 2: A high-level description of our algorithm***

The main computational loop (steps 3-8) extends the chains detected in step 2. In step 4, we generate candidate subgraphs of size *k* by integrating smaller sub-graphs. We estimate the number of solutions for each of the candidate subgraphs and select those that have more solutions than a user-specified threshold (steps 5-7). In this paper, the user-specified threshold, α, was set to zero.

**Candidate Generation:** The function 'GenCandidates' (see Figure 3) takes as input selected subgraphs of size 1 to *k*-1, and produces candidates subgraphs of size *k*.

**Algorithm** 2: GenCandidates
**Input**: $G^1, G^2, \ldots, G^{k-1}$
**Output**: $Candidates^k$

1. $Candidates^k \leftarrow \Phi$
2. **for each** axiom *ax* in KB **do**
3. antecedent-preds ← Predicates in the antecedent literals of ax
4. **for each** subset *sb* of antecedent-preds **do**
5. **for each** p in *sb* **do**
6. rg(p) ← {g | g∈ $G_i$, 1≤i≤k-1, p is relevant to Head(g)}
7. **end for**
8. cross-products ← $rg(p_1)\times\ldots\times rg(p_n)$, n = |sb|.
9. **for each** cp in cross-products **do**
10. $g^k \leftarrow$ <ax, {ax} ∪ CPRules(cp) >
11. add $g^k$ to $Candidates^k$ **if** |CPRules(cp)| = k-1
12. **end for**
13. **end for**
14. **end for**
15. **return** $Candidates^k$

***Figure 3: Algorithm for candidate generation***

The '*GenCandidates'* algorithm iterates over all axioms in the KB (step 2), and reasons about how they can be used successfully at the root node. Since only some of the antecedent literals might need backchaining (i.e., remaining literals might be solved via GAF lookups), we consider all subsets of antecedent predicates in step 4. We find proof graphs for queries involving these predicates in steps 5 through 13. In step 6, we generate rg(p), the set of all relevant graphs for the predicate 'p' that use less than *k* axioms[3]. We compute the cartesian product of the relevant sets in step 8. Given an n-tuple cp= ($x_1, x_2,\ldots,x_n$), each $x_i$ is a proof path (see example in Figure 1b) that can be useful for one of the predicates in the antecedent of axiom *ax*. The set of all rules used in the n–tuple, CPRules(cp), is given by $\bigcup_{i=1}^{n} \boldsymbol{Rules}(\boldsymbol{x_i})$. The algorithm includes those proof graphs that use exactly *k* rules. We illustrate this process with an example.

Consider axiom A1 shown on page 2. The set of predicates in its antecedents is {p, q}. In step 4, the algorithm finds proof graphs for all subsets of {p, q}. When the subset in step 4 is {p, q}, we calculate both rg(p) and rg(q) (via step 6). Proof paths containing axioms A2 and A3 are elements of rg(p) and rg(q), respectively. The cross-product (calculated in step 8) contains the tuple (<A2, {A2}>, <A3, {A3}>). Since the aforementioned tuple uses two rules, the condition in step 11 is satisfied when *k* is 3, and a longer proof path represented by <A1, {A1, A2, A3}> (shown in Figure 1b) is included in *Candidates*[3].

Let us consider another iteration of steps 4-13 in which the subset is bound to {p}, and *k* is 3. rg(p) contains <A2, {A2}> and <A2, {A2, A4}>. The latter tuple uses two axioms and is extended with axiom A1 to create <A1, {A1, A2, A4}> (shown in Figures 4a and 4b).

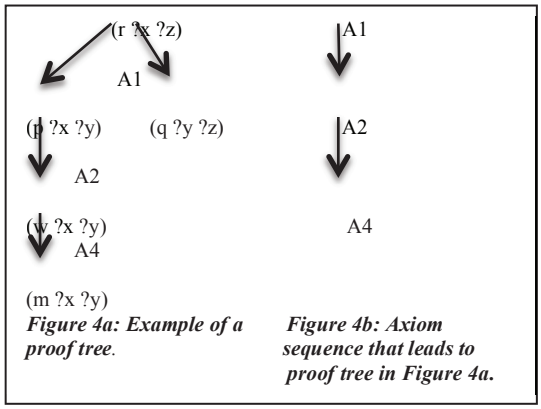


***Figure 4a: Example of a proof tree.***

***Figure 4b: Axiom sequence that leads to proof tree in Figure 4a.***

Figure 4a shows that while two axioms are needed to prove the first literal, the second literal is answered via GAF lookup. The function 'EstimateSolutionCount' (see Figure 2) estimates the productivity of a given proof chain by restricting the inference engine to use the axioms in the proof chain, and then asking the query represented by the root node[4].

[3] In step 6, we say that a predicate q is relevant to p if answering a query involving 'p' could involve answering a query involve 'q'. For example, the predicate 'properSubEventTypes' is relevant to 'subEventTypes' because the former is a specialization of the latter.

[4] The root node query (e.g., (performedBy ?x ?y)) might have some answers that are derived by mere GAF lookup or by using only a subset of the rules in the proof chain. The function 'EstimateSolutionCount' in-

The complexity of the 'GenCandidates' algorithm is reasonable. If *m* is the number of literals in the antecedent of an axiom, then steps 5-12 are repeated $2^m$ times. The number of n-tuples in step 8 is less than $p^m$, where p is the number of proof paths for a predicate. Therefore, the generation of candidate subgraphs takes $O(N.2^m p^m)$ time, where N is the number of axioms in the KB. In practice, *m* is small[5] (well-factored axioms have small number of literals in the antecedent); therefore, the time-complexity is essentially determined by N and *p*. Table 2 shows the number of useful proof chains of different sizes. We see that while the inference engine has to examine millions of proof chains[6], only a small fraction of these proof chains are useful for *any* query.

*Table 2: Number of useful proof paths of different sizes*

| $k$ | $\|G^k\|$ |
|---|---|
| 1 | 33,026 |
| 2 | 16,837 |
| 3 | 14,058 |
| 4 | 8,179 |

The 'Chains' algorithm (Figure 2) takes $O(d.N.2^m p^m.t_0)$ time, where d is the maximum number of rules in the proof chains and $t_0$ is the time required to estimate the number of solutions for a given proof tree (step 6). In the next section, we evaluate the efficacy of these proof paths.

## Experimental Evaluation

The selection of benchmark problem instances is important in the design and empirical evaluation of heuristics. We used three principles in this evaluation: (i) Heuristics should be tested only on the hardest reasoning problems. Since the size of the search space increases with the size and expressiveness of KB, we tested our heuristics on the largest commonsense KB that uses full FOL[7]. (ii) While the study of artificially generated problem instances has led to a good understanding of how syntactic properties determine problem hardness, the right methodology for generating such problems has not received sufficient attention in the commonsense reasoning community. Therefore, we focus on queries pertaining to real world events and applications. These queries have been created by knowledge engineers and programmers for different projects (e.g., Project HALO [Friedland *et al*. 2004], HPKB project [Cohen *et al*. 1998]) and for testing the question-answering (Q/A) capability of the inference engine. (iii) To expand the frontier of inference capabilities, heuristics should be able to solve problems that are difficult for the inference engine. Therefore, first, we excluded simple queries that do not need any backward inference (e.g., (genls Dog LivingOrganism), (isa BarackObama Person)). Second, to identify queries that are difficult for the inference engine, we divided the queries into four groups based on their time requirements[8] . The column labeled "TR" in Table 3 shows the time requirements of the queries in the test set. These experiments included both fully and partially bound queries. Since ordering heuristics guide the inference engine towards more productive parts of the search space, they help in reducing the inference time when the user asks a pre-determined number of answers.

***Table 3: Experimental Results***

| TS | TR | Method | #Q | %A | Time (hours) | S | C (%) |
|---|---|---|---|---|---|---|---|
| 1 | [0,0.5] | Baseline | 600 | 97 | 0.79 | - | - |
| | | Opt | 600 | 98 | 0.38 | 2.07 | 1 |
| 2 | [0.5,10] | Baseline | 309 | 60 | 17.84 | - | - |
| | | Opt | 309 | 91 | 2.52 | 7.07 | 51 |
| 3 | [10.40] | Baseline | 225 | 23 | 63.13 | - | |
| | | Opt | 225 | 80 | 11.27 | 5.60 | 247 |
| 4a | [40,60] | Baseline | 140 | 8 | 109.22 | - | |
| | | Opt | 140 | 97 | 5.08 | 21.50 | 1112 |
| 4b | [40,60] | Baseline | 137 | 20 | 66.91 | - | |
| | | Opt | 137 | 97 | 3.51 | 19.06 | 385 |

Meaning of Column labels: TS:Test Sets, TR: Time requirements of test sets (in minutes), #Q: Number of queries, %A: % queries answered, S: Speedup with respect to baseline, C: % Improvement in proportion of queries answered with respect to baseline.

On the other hand, these heuristics lead to more answers when the user seeks all answers for a given query. The English translation of one of the queries is shown below:

What causes a decrease in circulating calcium during cardiovascular circulation?

The question shown above leads to the following query[9]:

cludes only those answers that use all rules in the proof chain in their derivation.

[5] Rules in Cyc KB have (on average) less than 3 literals in their antecedent. (The average value of *m* is 2.55).

[6] For example, more than 2000 axioms are relevant for answering 'isa' queries. Therefore, the inference engine would have to examine (in the worst case) more than 4 million paths if the relevant proof path involves backchaining on 'isa' literals twice.

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

[8] For instance, the baseline inference engine needed 0 to 0.5 minutes for each query in the first test set. Queries in the fourth set were divided into two parts (sets 4a and 4b) due to their large time requirements.

[9] (causes-SubSitTypeSubSitType S CAUSE-TYPE EFFECT-TYPE) means that in situations of type S, events of type CAUSE-TYPE cause events of type EFFECT-TYPE.

(causes-SubSitTypeSubSitType CardiovascularCirculation ?cause (DecreaseOnSlotFn (QuantitySlotForArg2Fn CalciumIon amountPresentAt))))

Recall that the inference engine uses a set of heuristic modules for ordering nodes during search and the net score of a node is given by $f(s) = \Sigma_i w_i f_i(n)$. We define a new heuristic module with the following quality function:

$$f_c(n) = \begin{cases} \mathbf{1}\ \boldsymbol{if}\ \exists \boldsymbol{g} \in \boldsymbol{ProofGraphs},\ \boldsymbol{TRules(n)} \subseteq \boldsymbol{Rules(g)} \\ \mathbf{0}\ \boldsymbol{otherwise} \end{cases}$$

$$TRules(n) = \bigcup \boldsymbol{TRule}(\boldsymbol{p_i(n)})$$

$$\text{ProofGraphs} = \bigcup_{\mathbf{1}}^{\boldsymbol{n}} \boldsymbol{G^k} \quad k = 1, 2, 3, 4.$$

In the expression shown above, $p_i(n)$ is the $i^{th}$ element of the transformation link set of node $n$ and TRules(n) is the set of all rules that have been used to derive node $n$. The function $f_c(n)$ returns 1 when an element of $G^k$ contains all rules that have been used to derive node $n$. Given the function $fc$ discussed above, we can include a heuristic module $(w_c, f_c)$ where $w_c$ is the "weight" of the module. The baseline version in Table 3 is obtained by setting $w_c$ to 0. We can assess the utility of graphs in $G^k$ by setting $w_c$ to 1 (see rows labeled "Opt" in Table 3). The experimental data was collected on a 4-core 3.40 GHz Intel processor with 32 GB of RAM. We see that proof chains selected by retrograde analysis provide significant speedup and the highest performance gain has been obtained for problem sets that are most difficult for the inference engine.

## Conclusion and Discussion

Efficient commonsense reasoning at scale is critical for building large-scale cognitive systems [Matuszek et al 2005, Sharma & Forbus 2010][10]. The intractability of reasoning with first-order logic provides opportunities for designing new search control heuristics. Research in the computer games community has shown that large-scale offline search (cf, pattern databases [Culberson & Schaeffer 1998], endgame databases [Thomson 1986]) can be used to assess and cache the utility of actions. Inspired by the success of these databases, in this paper, we build a database that helps the inference engine compute the utility of resolution steps. Building this database involves performing a bottom-up level-by-level analysis of search space. We show that this method has sufficient discriminatory power because it selects only a small fraction of millions of proof paths. Experimental results show that this heuristic leads to significant speedup on the hardest problems. These results suggest two lines of future work. First, we want to conduct even larger experiments to ensure the validity of these methods. Second, coupling this approach with the research in decision-theoretic models and work on proof planning [Smith 1989, Greiener 1991] may a more complete theoretical model for designing new heuristics.

[10] For a discussion on tangentially related applications that benefit from commonsense reasoning, see [Abhishek & Basu 2005] and [Abhishek & Rajaraman 2003].


### Acknowledgements

We thank Mary Shepherd, Doug Lenat, Dave Schneider and Saket Joshi for their useful comments and suggestions.


## References


Abhishek and A. Basu. 2005. Iconic Interfaces for Assistive Communication. In *Encyclopedia of Human Computer Interaction*, ed. Ghaoui, C. 295-302. IGI Global.

Abhishek and Rajaraman, V. 2005. A Computer Aided Shorthand Expander. *IETE Technical Review*, Vol. 22, no. 4, 295-302.

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

Bjornsson, Y. Schaeffer, J. and Sturtevant, N. 2005. Partial Information Endgame Databases, *Proc. of 11th Intl. Conf. on Advances in Computer Games,* pages 11-22.

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, L. 2014. Machine Learning for first-order Theorem Proving. *Journal of Automated Reasoning*, 53(2):141–172.

Cohen, P. *et al.*. 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), pp. 318-334.

Chaudhuri, S. 1998. An Overview of Query Optimization in Relational Systems, *PODS,* 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. In *Proceedings of Twenty Second National Conference On Artificial Intelligence (AAAI)*, pp. 1542-1547, Vancouver, BC.

Friedland, N., Allen, P., Matthews, G., Witbrock, M., Curtis, J. and Shepard, B. *et al.*. 2004 Project Halo: Towards a Digitial Aristotle. *AI Magazine*, 25(4), 29-47.

Greiner, R. 1991. Finding Optimal Derivation Strategies in Redundant Knowledge Bases, *Artificial Intelligence,* 50(1), pp. 95-115.

Hoder, K. and Voronkov, A. 2011. Sine qua non for Large Theory Reasoning. In *Proceedings of the CADE-23*, pages 299-314

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

Hutter, F., Xu, L, Hoos, H. and Leyton-Brown, K. 2014. Algorithm Runtime Prediction: Methods and Evaluation. *Artificial Intelligence*, 206, pages 79-111

Kaliszyk, C., J. Urban and J. Vyskocil. 2015. Efficient Semantic Features for Automated Reasoning Over Large Theories. *Proceedings of IJCAI*, pp. 3084-3090, Buenos Aires, Argentina.

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

Kuramochi, M. and Karypis,G. 2001. Frequent Subgraph Discovery, *Proc. of ICDM,* pp. 313-320.

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

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 AAAI*, Pittsburgh, PA.

Matuszek, C., Cabral, J., Witbrock, M., DeOliveira, J. 2006. An Introduction to the Syntax and Content of Cyc. *AAAI Spring Sympoisum*, Palo Alto, CA

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

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

Sharma, A. and Forbus, K. D. 2010. Modeling the Evolution of Knowledge and Reasoning in Learning Systems, *Proceedings of AAAI Fall Symposium: Commonsense Knowledge*, Arlington, VA

Sharma, A. and Forbus, K. D., 2013. Automatic Extraction of Efficient Axiom Sets from Large Knowledge Bases, *Proceedings. of AAAI*, Bellevue, WA.

Sharma,A.,Witbrock, M. and Goolsbey, K. 2016. Controlling Search in Very Large Knowledge Bases: A Machine Learning Approach. *Proc. of Advances in Cognitive Systems,* Evanston, IL.

Smith, D. E. 1989. Controlling Backward Inference, *Artificial Intelligence*, 39 (2), pp. 145-208.

Taylor, M., Matuszek, C., Smith, P and Witbrock, M. 2007. Guiding Inference with Policy Search Reinforcement Learning, *Proc. of FLAIRS*, Key West, FL

Thompson, K. 1986. Retrograde Analysis of Certain Endgames. Journal of Intl. Computer Chess Association, Vol 9, pp. 131-139

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