Automatically Generating Problems and Solutions for Natural Deduction

Umair Z. Ahmed
IIT Kanpur
umair@iitk.ac.in

Sumit Gulwani
MSR Redmond
sumitg@microsoft.com

Amey Karkare
IIT Kanpur
karkare@cse.iitk.ac.in

Abstract

Natural deduction, which is a method for establishing validity of propositional type arguments, helps develop important reasoning skills and is thus a key ingredient in a course on introductory logic. We present two core components, namely solution generation and practice problem generation, for enabling computer-aided education for this important subject domain. The key enabling technology is use of an offline-computed data-structure called *Universal Proof Graph* (UPG) that encodes *all* possible applications of inference rules over all *small* propositions abstracted using their *bitvector-based truth-table representation*. This allows an efficient forward search for solution generation. More interestingly, this allows generating fresh practice problems that have given solution characteristics by performing a backward search in UPG. We obtained around 300 natural deduction problems from various textbooks. Our solution generation procedure can solve many more problems than the traditional forward-chaining based procedure, while our problem generation procedure can efficiently generate several variants with desired characteristics.

1 Introduction

Natural deduction [Gentzen, 1964] is a method for establishing the validity of propositional type arguments, where the conclusion of an argument is actually derived from the premises through a series of discrete steps. A proof in natural deduction consists of a sequence of propositions, each of which is either a premise or is derived from preceding propositions by application of some inference/replacement rule and the last of which is the conclusion of the argument.

Significance of Teaching Natural Deduction From a pragmatic perspective, it helps develop the reasoning skills needed to construct sound arguments of one's own and to evaluate the arguments of others. It instills a sensitivity for the formal component in language, a thorough command of which is indispensable to clear, effective, and meaningful communication. Such a logical training provides a fundamental defense against the prejudiced and uncivilized attitudes that threaten the foundations of our democratic society [Hurley, 2011].

From a pedagogical perspective, natural deduction is a useful tool in relieving *math anxiety* that terrifies countless students. It is a gentle introduction to mastering the use of logical symbols, which carries into other more difficult fields such as algebra, geometry, physics, and economics.

Natural deduction is typically taught as part of an introductory course on logic, which is a central component of college education and is generally offered to students from all disciplines regardless of their major. It is thus not a surprise that a course on logic is being offered on various online education portals including Coursera [Coursera,], Open Learning Initiative [oli,], and even Khan Academy [kha,].

Significance of Solution Generation Solution generation is important for several reasons. First, it can be used to generate sample solutions for automatically generated problems. Second, given a student's incomplete solution, it can be used to complete the solution. This can be much more illustrative for a student as opposed to providing a completely different sample solution to a student. Third, given a student's incomplete solution, it can also be used to generate hints on the next step or an intermediate goal.

Significance of Problem Generation Generating fresh problems that have specific solution characteristics is a tedious task for the teacher. Automating this has several benefits.

Generating problems that are similar to a given problem has two applications. First, it can help *avoid copyright issues*. It may not be legal to publish problems from textbooks on course websites. Hence, instructors resort to providing indirect pointers to textbook problems as part of assignments. A problem generation tool can provide instructors with a fresh source of problems to be used in their assignments or lecture notes. Second, it can help *prevent plagiarism* in classrooms or massive open online courses since each student can be provided with a different problem of the same difficulty level.

Generating problems that have a given difficulty level and that exercise a given set of concepts can help *create personalized workflows* for students. If a student solves a problem correctly, then the student may be presented with a problem that

(a)

Rule Name	Premises	Conclusion
Modus Ponens (MP)	p → q, p	q
Modus Tollens (MT)	p → q, ¬ q	¬ p
Hypothetical Syllogism (HS)	p → q, q → r	p → r
Disjunctive Syllogism (DS)	p ∨ q, ¬ p	q
Constructive Dilemma (CD)	(p → q) ∧ (r → s), p ∨ r	q ∨ s
Destructive Dilemma (DD)	(p → q) ∧ (r → s), ¬ q ∨ ¬ s	¬ p ∨ ¬ r
Simplification (Simp)	p ∧ q	q
Conjunction (Conj)	p, q	p ∧ q
Addition (Add)	p	p ∨ q

(b)

Rule Name	Proposition	Equivalent Proposition
De Morgan's Th.	¬ (p ∧ q)	¬ p ∨ ¬ q
	¬ (p ∨ q)	¬ p ∧ ¬ q
Distribution	p ∨ (q ∧ r)	(p ∨ q) ∧ (p ∨ r)
Double Negation	p	¬ ¬ p
Transposition	p → q	¬ q → ¬ p
Implication	p → q	¬ p ∨ q
Equivalence	p ≡ q	(p → q)∧ (q → p)
	p ≡ q	(p ∧ q)∨ (¬ p ∧ ¬ q)
Exportation	(p ∧ q) → r	p → (q → r)

Figure 1: (a) Sample Inference Rules. (b) Sample Replacement Rules.

(a)

Step	Proposition	Reason
P1	$x_1 \vee (x_2 \wedge x_3)$	Premise
P2	$x_1 \rightarrow x_4$	Premise
P3	$x_4 \rightarrow x_5$	Premise
1	$(x_1 \vee x_2) \wedge (x_1 \vee x_3)$	P1, Distribution
2	$x_1 \vee x_2$	**1, Simp.**
3	$x_1 \rightarrow x_5$	**P2, P3, HS.**
4	$x_2 \vee x_1$	2, Commutativity
5	$\neg\neg x_2 \vee x_1$	4, Double Neg.
6	$\neg x_2 \rightarrow x_1$	5, Implication
7	$\neg x_2 \rightarrow x_5$	**6, 3, HS.**
8	$\neg\neg x_2 \vee x_5$	7, Implication
9	$x_2 \vee x_5$	8, Double Neg.

(b)

Step	Truth-table	Reason
P1	η_1 = 1048575	Premise
P2	η_2 = 4294914867	Premise
P3	η_3 = 3722304989	Premise
1	η_4 = 16777215	**P1, Simp.**
2	η_5 = 4294923605	**P2, P3, HS.**
3	η_6 = 1442797055	**1, 2, HS.**

(c)

Node	Candidate Proposition	Target Witness	Source Witness
η_4	q_2	$[q_1]$	$[q_{P1}]$
η_5	q_3	$[q_{P2}, q_{P3}]$	$[q_{P2}, q_{P3}]$
η_6	q_7	$[q_6, q_3]$	$[q_2, q_3]$

(d)

Step	Proposition	Reason
P1	$x_1 \equiv x_2$	Premise
P2	$x_3 \rightarrow \neg x_2$	Premise
P3	$(x_4 \rightarrow x_5) \rightarrow x_3$	Premise
1	$(x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_1)$	P1, Equivalence
2	$x_1 \rightarrow x_2$	**1, Simp.**
3	$(x_4 \rightarrow x_5) \rightarrow \neg x_2$	**P3, P2, HS.**
4	$\neg\neg x_2 \rightarrow \neg(x_4 \rightarrow x_5)$	3, Transposition
5	$x_2 \rightarrow \neg(x_4 \rightarrow x_5)$	4, Double Neg.
6	$x_1 \rightarrow \neg(x_4 \rightarrow x_5)$	**2, 5, HS.**
7	$x_1 \rightarrow \neg(\neg x_4 \vee x_5)$	6, Implication
8	$x_1 \rightarrow (\neg\neg x_4 \wedge \neg x_5)$	7, De Morgan's
9	$x_1 \rightarrow (x_4 \wedge \neg x_5)$	8, Double Neg.

Figure 2: (a) Natural Deduction Proof (with application of inference rules highlighted in bold).
(b) Abstract Proof with truth-tables shown using a 32-bit integer representation.
(c) Steps in converting abstract proof to natural deduction proof. q_j denotes the proposition at step j from (a).
(d) Natural Deduction Proof of a similar problem.

is more difficult than the last problem, or exercises a richer set of concepts. If a student fails to solve a problem, then the student may be presented with a simpler problem.

Novel Technical Insights Our observations include:

- Small-sized hypothesis: Propositions that occur in educational contexts use a small number of variables and have a small size (Fig. 4(a)). The number of such small-sized propositions is bounded (Fig. 3(a)).
- Truth-Table Representation: A proposition can be abstracted using its truth-table, which can be represented using a bitvector representation [Knuth, 2011]. (Application of inference rules over bitvector representation reduces to performing bitwise operations.) This provides two key advantages: (i) It partitions small-sized propositions into a small number of buckets (Fig. 3(b)). (ii) It reduces the size/depth of a natural deduction proof tree, making it easier to search for solutions or generate problems with given solution characteristics.
- Offline Computation: The symbolic reasoning required to pattern match propositions for applying inference rules can be performed (over their truth-table bitvector representation) and stored in an offline phase. This has two advantages: (i) It alleviates the cost of symbolic reasoning by a large constant factor by removing the need to perform any symbolic matching at runtime. (ii) It enables efficient backward search for appropriate premises starting from a conclusion during problem generation, which we model as a *reverse of solution generation*.

Contributions We make following contributions.

- We propose leveraging the following novel ingredients for building an efficient computer-aided education system for natural deduction: small-sized proposition hypothesis, truth-table based representation, and offline computation.
- We present a novel two-phased methodology for solution generation that first searches for an abstract solution and then refines it to a concrete solution.
- We motivate and define some useful goals for problem generation, namely similar problem generation and parameterized problem generation. We present a novel methodology for generating such problems using a process that is reverse of solution generation.
- We present detailed experimental results on 279 benchmark problems collected from various textbooks. Our solution generation algorithm can solve 84% of these problems, while the baseline traditional algorithm could only solve 57% of these problems. Our problem generation algorithm is able to generate few thousands of *similar problems* and *parameterized problems* on average per instance in a few minutes.

(a)

n	Total	s				
		1	2	3	4	5
1		2	10	1.3×10^2	2.4×10^3	4.6×10^4
2		4	52	1.5×10^3	5.9×10^4	2.5×10^6
3	∞	6	126	5.8×10^3	3.3×10^5	2.1×10^7
4		8	232	1.4×10^4	1.1×10^6	9.8×10^7
5		10	370	2.9×10^4	2.8×10^6	3.1×10^8

(b)

n	Total (2^{2^n})	s				
		1	2	3	4	5
1	4	2	4	4	4	4
2	16	4	16	16	16	16
3	256	6	38	152	232	256
4	65536	8	70	526	3,000	13,624
5	4,294,967,296	10	112	1,252	12,822	1,22,648

Figure 3: (a) Number of propositions (not including those with double or more negations) over n variables and size at most s. (b) Number of truth-tables over n variables and size at most s.

2 Natural Deduction

Let $x_1, \ldots, x_n$ be n Boolean variables. A *proposition* over these Boolean variables is a Boolean formula consisting of Boolean connectives over these variables.

Definition 1 (Natural Deduction Problem) *A* natural deduction problem *is a pair $(\{p_i\}_{i=1}^m, c)$ of a set of propositions $\{p_i\}_{i=1}^m$ called* premises *and a proposition c called* conclusion*. A natural deduction problem is* well-defined *if the conclusion is implied by the premises, but not by any strict subset of those premises.*

Definition 2 (Natural Deduction Proof) *Let $\mathcal{I}$ and $\mathcal{R}$ be sets of inference rules and replacement rules respectively. A* natural deduction proof *for a problem $(\{p_i\}_{i=1}^m, c)$ using $\mathcal{I}$ and $\mathcal{R}$ is a step-by-step derivation of conclusion c from premises $\{p_i\}_{i=1}^m$ by making use of some inference rule from $\mathcal{I}$ or some replacement rule from $\mathcal{R}$ at each step.*

Inference and Replacement Rules An *inference rule I* is made up of some premises $\texttt{Premises}(I)$ and a conclusion $\texttt{Conclusion}(I)$. Fig. 1(a) lists down some sample inference rules. Unlike the inference rules, which are basic argument forms, a *replacement rule* is expressed in terms of pairs of logically equivalent statement forms, either of which can replace the other in a proof sequence. Fig. 1(b) lists down some sample replacement rules. Our system does not leverage any knowledge of a specific inference/replacement rule. The only interface that it requires of a rule is the capability to generate the target proposition from source propositions.

Example 1 Consider the natural deduction problem $(\{x_1 \vee (x_2 \wedge x_3), x_1 \rightarrow x_4, x_4 \rightarrow x_5\}, x_2 \vee x_5)$. Fig. 2(a) shows a natural deduction proof for it with inference rule applications in bold.

3 Universal Proof Graph

We start out by describing our key data-structure that is used for both solution and problem generation. It encodes all possible applications of inference rules over all propositions of small size, abstracted using their truth-table representation. Note that a truth-table representation does not distinguish between equivalent propositions such as q and $\neg\neg q$.

Definition 3 (Truth-Table Bitvector Representation) *Let q be a proposition over n Boolean variable. Its truth-table, which assigns a Boolean value to each of the 2^n possible assignments to the n Boolean variables, can be represented using a 2^n bitvector [Knuth, 2011]. We denote this bitvector by $\tilde{q}$.*

Definition 4 (Abstract Proof) *Let $\mathcal{I}$ be a set of inference rules. An* abstract proof tree *for a natural deduction problem $(\{p_i\}_{i=1}^m, c)$ is any step-by-step deduction for deriving $\tilde{c}$ from $\{\tilde{p_i}\}_{i=1}^m$ using the abstract version of some inference rule from $\mathcal{I}$ at each step. An abstract version of an inference rule $\tilde{I}$ has premises $\{\tilde{q} \mid q \in \texttt{Premises}(I)\}$ and conclusion $\tilde{q'}$, where $q' = \texttt{Conclusion}(I)$. An abstract proof A is* minimal *if for every other abstract proof A', the set of truth-tables derived in A' is not a strict subset of A.*

An abstract proof for a natural deduction problem is smaller than a natural deduction proof. This is because an abstract proof operates on truth-tables, i.e., equivalence classes of propositions, and does not need to encode the replacement steps to express equivalent rewrites within a class. Note that a natural deduction proof for a given problem over inference rules $\mathcal{I}$ and replacement rules $\mathcal{R}$ can always be translated into an abstract proof for that problem over $\mathcal{I}$. However, translating an abstract proof into natural deduction proof depends on whether or not $\mathcal{R}$ contains sufficient replacement rules.

Example 2 Consider the problem mentioned in Example 1 and its natural deduction proof in Fig. 2(a). An abstract proof for the problem is given by the bold statements in its natural deduction proof along with interpreting each proposition q as its truth-table $\tilde{q}$. This abstract proof is shown in Fig. 2(b).

Definition 5 (Size of a Proposition) *We define the size $\texttt{Size}(q)$ of a proposition q to be the number of variable occurrences in it.*

$$\begin{aligned} \texttt{Size}(x) &= 1 \\ \texttt{Size}(\neg q) &= \texttt{Size}(q) \\ \texttt{Size}(q_1 \ \mathit{op} \ q_2) &= \texttt{Size}(q_1) + \texttt{Size}(q_2) \end{aligned}$$

Let $P_{n,s}$ denote the set of all propositions over n variables that have size at most s. Fig. 3(a) shows the number of propositions over n variables as a function of s. *Note that even though the number of syntactically distinct propositions over n variables is potentially infinite, the number of such propositions that have bounded size is bounded.*

Definition 6 (Size of a Truth-Table) *We define the size of a truth-table t to be the smallest size of a proposition q such that $\tilde{q} = t$. We select any such smallest-sized proposition as the* canonical proposition *for truth-table .*

Let $T_{n,s}$ denote the set of all truth-tables over n variables that have size at most s. Fig. 3(b) mentions statistics related to

(a)

n	s				
	2	3	4	5	6
1	2		1		
2	32	6	6		2
3	27	61	15	3	2
4	15	34	27	1	
5	3	30	20	2	
6		7	20	1	1
7		3	7		1
8		2	3	2	
9		1		1	1

(b)

n	# Edges	Time (hrs.)
1	20	< 1
2	960	< 1
3	14424	< 1
4	68422	< 3
5	207322	< 8

Figure 4: (a) Benchmarks: Total of 339 natural deduction problems obtained from 5 textbooks [Howard-Snyder *et al.*, 2008; Bergmann, 2008; vander Nat, 2010; Hurley, 2011; Cederblom and Paulsen, 2011], distributed across various partitions defined by number of Boolean variables n and max size s of each proposition. Note that 82% of the problems have $s \leq 4$ and $n \leq 5$.
(b) UPG statistics for $s = 4$ and $n \leq 5$.

the number of truth-tables in $T_{n,s}$ for various values of n and s. *Note that the cardinality of $T_{n,s}$ is much smaller than the cardinality of $P_{n,s}$.*

Definition 7 (Universal Proof Graph (UPG)) *An* $(n, s, \mathcal{I})$ Universal proof graph (UPG) *is a hyper-graph whose nodes are labeled with truth-tables from $T_{n,s}$ and edges are labeled with an inference rule from $\mathcal{I}$.*

Algorithm We now describe our algorithm for computing the $(n, s, \mathcal{I})$-UPG. The algorithm makes use of functions `Eval`, `EvalS`, and `Size`, which are defined later.

1. The node set of $(n, s, \mathcal{I})$-UPG is $T_{n,s}$, i.e., the set of all truth-tables of size s. We compute $T_{n,s}$ by enumerating all propositions q of size s over n variables and adding $\tilde{q}$ to the node set. During this computation, we also maintain a reverse mapping called `Canonical` that maps each truth-table to a canonical proposition.
2. The edge set computation involves adding the following hyper-edge $\mathcal{E}$ for each inference rule $I \in \mathcal{I}$ and each state ρ that maps each free variable in I to a truth-table in $T_{n,s}$: The i^{th} source node of the hyper-edge is $\texttt{Eval}(p_i, \rho)$ and the target node is $\texttt{Eval}(q, \rho)$, where p_i is the i^{th} premise in $\texttt{Premises}(I)$ and $q = \texttt{Conclusion}(I)$. We add this edge only when (i) $\texttt{Size}(\texttt{EvalS}(q, \rho)) \leq s$ and $\texttt{Size}(\texttt{EvalS}(p_i, \rho)) \leq s$ (for all i), and (ii) $\texttt{EvalS}(q, \rho)$ and $\texttt{EvalS}(p_i, \rho)$ (for all i) can be rewritten into the corresponding canonical propositions using the given set of replacement rules. This optimization avoids adding too many edges that result from too involved reasoning on large-sized propositions–which is something that our replacement rule finding engine will fail to do. Each hyper-edge $\mathcal{E}$ is annotated with the set of all tuples $([\texttt{EvalS}(p_1, \rho), \ldots, \texttt{EvalS}(p_n, \rho)], \texttt{EvalS}(q, \rho))$ obtained from any inference rule I and any state ρ that yields $\mathcal{E}$. This set is referred to as $\texttt{PTuples}(\mathcal{E})$.

The function $\texttt{EvalS}(q, \rho)$ substitutes each free variable x in q by $\texttt{Canonical}(\rho(x))$. The function `Eval` is as follows:

$$\begin{aligned} \texttt{Eval}(q_1 \wedge q_2, \rho) &= \texttt{Eval}(q_1, \rho) \,\&\, \texttt{Eval}(q_2, \rho) \\ \texttt{Eval}(q_1 \vee q_2, \rho) &= \texttt{Eval}(q_1, \rho) \parallel \texttt{Eval}(q_2, \rho) \\ \texttt{Eval}(\neg q, \rho) &= \sim \texttt{Eval}(q, \rho) \\ \texttt{Eval}(x, \rho) &= \rho(x) \end{aligned}$$

Here $\&, \parallel, \sim$ denote bitwise-and, bitwise-or, and bitwise-not operators respectively.

Key Ideas of the Algorithm Note that the naive approach to computing the node set (by enumerating all truth-tables and filtering out truth-tables with size at most s) has two key challenges: It is not easy to identify whether a given truth-table has small size. Furthermore, the total number of truth-tables over n variables is huge (2^{2^n}). Instead we compute $T_{n,s}$ by enumerating all small propositions one by one.

Similarly, note that the naive approach to computing the edge set (by enumerating over all possible tuples of source nodes and target node, and identifying if a given inference rule is applicable) has two key challenges: It is not easy to identify whether a given inference rule is applicable at the truth-table representation, and the number of such tuples is huge. Instead we enumerate all edges corresponding to a given inference rule by enumerating over all possible applications of that rule.

Results Fig. 4(b) describes the number of edges of UPGs and the time taken to compute them for various values of $n \leq 5$ and $s = 4$. We chose s to be 4 since most problems in practice have $s \leq 4$ (See Fig. 4(a)). We experimented with $n \leq 5$ since it allows use of standard 32-bit integer for bitvector representation of a truth-table.

4 Solution Generation

In this section, we discuss how to generate a solution (i.e., a natural deduction proof) to a natural deduction problem.

A naive approach to generating a minimal natural deduction proof would be to perform a breadth-first search starting from premises p_i and applying some inference or replacement rule at each step to generate a new proposition until the conclusion is reached. We refer to this as the *Traditional algorithm*. We next present our two-phase *UPG-based algorithm* that first computes an abstract proof using the UPG and then extends it to a natural deduction proof.

Algorithm Given the problem $(\{p_i\}_{i=1}^m, c)$, with n variables and max proposition size s, a set $\mathcal{I}$ of inference rules, and a set $\mathcal{R}$ of replacement rules, we do the following:

1. Compute an abstract proof for the problem $(\{p_i\}_{i=1}^m, c)$. This is done by performing a forward search for conclusion $\tilde{c}$ starting from $\{\tilde{p_i}\}_{i=1}^m$ in $(n, s, \mathcal{I})$-UPG. We do a breadth-first search to compute a minimal abstract proof.
2. Concretize the abstract proof to a natural deduction proof. This is done by generating a set of *candidate propositions* for each node η in the abstract proof tree in

Total Steps	Avg Abstract Steps (j)	Num. of problems	% Success Traditional	Avg time UPG (sec)	Avg time Trad. (sec)
1	0.6	18	100	0.0	0.0
2	0.9	21	100	0.0	0.2
3	0.9	25	100	0.0	0.9
4	1.2	23	70	0.0	1.2
5	1.8	16	69	0.1	0.2
6	2.0	19	47	0.1	0.6
7	1.7	14	71	0.9	0.6
8	1.9	20	70	1.2	0.3
9	2.0	14	29	0.2	1.9
10	2.7	6	33	0.2	1.5
11	2.5	6	17	1.9	0.0
12	2.6	10	60	4.4	2.0
13	2.7	6	33	3.0	0.1
14	3.0	2	50	0.4	4.4
15	3.1	8	13	2.9	0.1
16	2.6	8	38	1.0	0.4
17	2.8	6	17	2.2	0.1
19	3.3	3	33	3.0	0.1
20	2.0	2	0	0.3	-
21	4.0	1	100	6.4	8.7
22	3.0	2	0	1.4	-
23	3.0	1	0	4.3	-
24	2.0	1	100	1.3	0.1
25	4.0	1	0	8.7	-
27	4.0	1	0	9.3	-

Figure 5: Solution Generation Results (with timeout of 10 sec): Performance of Traditional Algorithm on problems that UPG-based algorithm could solve grouped by total steps in overall proof. Last column shows average over only those problems that Traditional Algorithm could solve.

a topological order as follows. For each premise node, this set contains only the corresponding premise. Let $\mathcal{E}$ be the hyper-edge from the UPG whose target node yielded η, and let $([q_1, \ldots, q_n], q) \in \texttt{PTuples}(\mathcal{E})$. If for each i^{th} child node of η, any of the candidate propositions q_i' associated with it can be rewritten to q_i, then q is added as a candidate proposition for η. The rewriting of a proposition to another one is attempted by performing a bounded breadth-first search over transitive applications of replacement rules from $\mathcal{R}$. $[q_1', \ldots, q_n']$ is called the *source-witness tuple* for q and $[q_1, \ldots, q_n]$ is called the *target-witness tuple* for q.

3. The natural deduction proof is now obtained by iteratively selecting a *source proposition* and a *target proposition* for each node in reverse topological order. These propositions are obtained respectively from the source-witness tuple and target-witness tuple of the parent node. Each node is then replaced by a series of replacement rules that convert the source proposition into the target proposition.

Example 3 Fig. 2(c) briefly outlines how the above algorithm generates the natural deduction proof in Fig. 2(a) from the abstract proof in Fig. 2(b). Column "Candidate Proposition" shows one of the candidate propositions produced by step 2 of the algorithm (and identified by step 3 of the algorithm) along with the corresponding source/target witnesses.

Results We tried our UPG-based algorithm over the 279 benchmark problems (Fig. 4(a)) that have $n \leq 5$ and $s \leq 4$. These problems were picked across 21 different exercise sets, each requiring use of a specific set $\mathcal{I}$ of inference rules. We obtained $(n, s, \mathcal{I})$-UPG as $\bigcup_{I \in \mathcal{I}} (n, s, \{I\})$-UPG instead of having to compute the UPG for each given subset $\mathcal{I}$ from scratch. Our tool was able to generate an abstract proof for 88% of these problems. Among these 88% problems, our tool was able to concretize the abstract proof to a natural deduction proof for 96% instances, thereby achieving an overall success rate of 84%. In contrast, the overall success rate of the traditional algorithm is 57%.

Fig. 5 shows the distribution of the 84% problems that the UPG-algorithm is able to solve with respect to the number of total steps required to solve them. Col. "% Success" shows that the traditional algorithm is unable to several of these problems that have larger number of steps. Col. 2 shows that the (average) number of steps of the abstract proof is significantly smaller than the total number of steps in Col. 1—this partly explains the higher success rate of the UPG-based algorithm. There are a small number of instances where the traditional algorithm works better than the UPG-based algorithm. This happens when the traditional algorithm is able to find an overall shorter proof that has more number of abstract steps than the proof found by the UPG-based algorithm (which has smallest number of abstract steps, but involves significantly large number of replacement steps).

The UPG-based algorithm does fail to solve certain problems. This is because it restricts its search to those proofs all of whose propositions have size $s \leq 4$. However, for certain problems, proofs involve intermediate propositions whose size is greater than 4. (Note that this can happen even if the premises and conclusion have size at most 4.)

5 Problem Generation

We now show how to generate fresh problems that have a given solution characteristic. The key algorithmic insight here is to model problem generation as reverse of solution generation, and the needed backward search is enabled due to the UPG. We consider two goals, that of producing similar problems and parameterized problems.

5.1 Similar Problems

Definition 8 (Similar Problems) *We say that two problems Q_1 and Q_2 are* similar *if they entail a similar minimal abstract proof tree. Two abstract proof trees are* similar *if they involve exactly the same order of inference rule applications.*

Generating similar problems can help address copyright and plagiarism issues as discussed in Section 1.

Algorithm Let P be a problem with n variables and maximum proposition size s and that needs to be solved using inference rules $\mathcal{I}$. Our algorithm for generating problems that are similar to P involves the following steps.

1. Generate a minimal abstract proof A for P by performing a forward breadth-first search in the $(n, s, \mathcal{I})$-UPG.

Premises	Conclusion	j	Num. of problems	Time min:sec
$x_1 \vee x_2,\ \ x_3,\ \ (x_3 \wedge (x_1 \vee x_2)) \rightarrow \neg x_4,\ \ \neg x_5 \rightarrow x_4$	$\neg\neg x_5$	3	21849	42:42
$x_1,\ \ (x_1 \wedge x_2) \rightarrow (x_3 \wedge x_4),\ \ x_2,\ \ x_4 \rightarrow x_5$	x_5	4	9290	30:28
$\neg x_1,\ \ (\neg x_1 \wedge \neg x_2) \rightarrow (x_3 \rightarrow x_4),\ \ \neg x_2,\ \ x_4 \rightarrow x_5$	$x_3 \rightarrow x_5$	3	5306	17:56
$\neg(x_1 \wedge x_2),\ (x_1 \rightarrow \neg x_2) \rightarrow (\neg x_3 \wedge \neg x_4),\ x_5 \vee (x_3 \vee x_4)$	x_5	2	4001	5:56
$x_1 \rightarrow (x_2 \rightarrow x_3),\ \ \neg(x_2 \rightarrow x_3),\ \ \neg x_4 \vee x_1$	$\neg x_4$	2	2908	0:06
$x_1 \rightarrow (x_2 \rightarrow (x_3 \vee x_4)),\ \ x_2 \wedge x_1,\ \ \neg x_4$	x_3	2	2353	0:14
$\neg x_1 \rightarrow x_2,\ \ x_1 \rightarrow x_3,\ \ \neg x_3,\ \ x_2 \rightarrow x_4$	$x_4 \vee x_5$	4	1248	5:17
$(x_1 \rightarrow x_2) \wedge \neg x_3,\ \ (\neg x_1 \rightarrow x_5) \vee x_3,\ \ (\neg x_2 \rightarrow x_5) \rightarrow x_4$	x_4	5	274	1:18
$(x_3 \vee x_2) \rightarrow \neg x_1,\ \ x_4 \vee x_1,\ \ x_2$	x_4	3	122	0:02
$x_1 \vee (x_2 \wedge x_3),\ \ x_1 \rightarrow x_4,\ \ x_4 \rightarrow x_5$	$x_2 \vee x_5$	3	516	1:52

(a)

n	m	j	$\mathcal{I}$	Num. of problems	Time min:sec
2	2	1	MP	4	0:00
2	2	2	Simp, MP	6	0:02
3	3	2	DS, HS	145	0:00
3	3	3	DD, MT, Conj	188	0:00
3	2	4	Simp, Conj, MP	279	0:09
4	2	3	Simp, MP,	2060	1:52
4	3	3	Conj, MP, HS	6146	1:29
4	4	3	HS, MP	18132	9:52
5	3	2	Simp, MP	5628	8:04
5	3	2	HS, DS	5838	7:39

(b)

Figure 6: Problem Generation Statistics.
(a) Number of similar problems generated and time taken to generate them for a few representative selection of problems (shown along with the number of inference steps j).
(b) Number of $(n, m, s, j, \mathcal{I})$-problems generated and time taken to generate them for few choices of $n, m, j, \mathcal{I}$ (as obtained from a representative selection of problems) and $s = 4$.

Premise 1	Premise 2	Premise 3	Conclusion
$x_1 \equiv x_2$	$x_3 \rightarrow \neg x_2$	$(x_4 \rightarrow x_5) \rightarrow x_3$	$x_1 \rightarrow (x_4 \wedge \neg x_5)$
$x_1 \wedge (x_2 \rightarrow x_3)$	$(x_1 \vee x_4) \rightarrow \neg x_5$	$x_2 \vee x_5$	$(x_1 \vee x_4) \rightarrow x_3$
$(x_1 \vee x_2) \rightarrow x_3$	$x_3 \rightarrow (x_1 \wedge x_4)$	$(x_1 \wedge x_4) \rightarrow x_5$	$x_1 \rightarrow x_5$
$(x_1 \rightarrow x_2) \rightarrow x_3$	$x_3 \rightarrow \neg x_4$	$x_1 \vee (x_5 \vee x_4)$	$x_5 \vee (x_2 \rightarrow x_1)$
$x_1 \rightarrow (x_2 \wedge x_3)$	$x_4 \rightarrow \neg x_2$	$(x_3 \equiv x_5) \rightarrow x_4$	$x_1 \rightarrow (x_3 \equiv \neg x_5)$

(a)

Premise 1	Premise 2	Premise 3	Conclusion
$(x_1 \rightarrow x_3) \rightarrow x_2$	$x_2 \rightarrow x_3$	$\neg x_3$	$x_1 \wedge \neg x_3$
$x_3 \rightarrow x_1$	$(x_3 \equiv x_1) \rightarrow x_2$	$\neg x_2$	$x_1 \wedge \neg x_3$
$(x_1 \equiv x_3) \vee (x_1 \equiv x_2)$	$(x_1 \equiv x_2) \rightarrow x_3$	$\neg x_3$	$x_1 \equiv x_3$
$x_1 \equiv \neg x_3$	$x_2 \vee x_1$	$x_3 \rightarrow \neg x_2$	$x_1 \wedge \neg x_3$
$x_3 \rightarrow x_1$	$x_1 \rightarrow (x_2 \wedge x_3)$	$x_3 \rightarrow \neg x_2$	$\neg x_3$

(b)

Figure 7: Some automatically generated problems.
(a) 5 (out of 516) new similar problems generated from the last problem in Fig. 8(a).
(b) 5 (out of 145) new parameterized problems generated from the parameters in the 3rd row in Fig. 8(b) (namely, $n = 3$, $m = 3$, $s = 4$, $j = 2$, $\mathcal{I} = \{\texttt{DS}, \texttt{HS}\}$).

2. Find matches of A in the $(n, s, \mathcal{I})$-UPG using a backtracking based backward search. Replace each truth-table node in the match by its canonical proposition.

Results Fig. 8(a) presents statistics on the number of similar (but well-defined) problems that we produced from various problems and the time taken to produce them. We removed all trivial replacements that correspond to replacing a variable by any other variable or its negation. In Fig. 7(a), we show 5 (out of 516) new similar problems generated for the last problem in Fig. 8(a), which is the same problem from Example 1. Fig. 2(d) shows a solution for the first new problem in Fig. 7. Observe that the abstract version of this solution (i.e., the bold steps) are similar to the abstract version of the solution for the original problem, which is shown in Fig. 2(a).

5.2 Parameterized Problems

Definition 9 ($(n, m, s, j, \mathcal{I})$-problem) *A $(n, m, s, j, \mathcal{I})$ problem is one that has (i) m premises involving n variables and of size at most s, and (ii) that has a minimal abstract proof that involves j steps and makes use of only and all inference rules from set $\mathcal{I}$.*

Generating parameterized problems can be used to generate personalized workflows as discussed in Section 1.

Algorithm The algorithm for generating $(n, m, s, j, \mathcal{I})$-problems that have n variables and maximum proposition size s involves performing a backtracking based backward search in $(n, s, \mathcal{I})$-UPG to find appropriate matches. To generate a concrete problem, we replace each truth-table node in the match by its canonical proposition.

Results Fig. 8(b) presents statistics on the number of well-defined $(n, m, s, j, \mathcal{I})$-problems produced for certain values of $(n, m, j, \mathcal{I})$ as obtained from some existing problems and $s = 4$. As before, we do not count those duplicate problems that can be obtained by replacing a variable by any other variable or its negation. In Fig. 7(b), we show 5 (out of 145) new problems generated for $n = 3$, $m = 3$, $s = 4$, $j = 2$, and $\mathcal{I} = \{\texttt{DS}, \texttt{HS}\}$, which corresponds to the parameters in the third row in Fig. 7(b).

6 Related Work

Natural Deduction Several proof assistants have been developed for teaching natural deduction: MacLogic [Dyckhoff *et al.*,], Symlog [Frederic D. Portoraro, 1994], Jape [Bornat,], Hyperproof [Barwise and Etchemendy, 1994], Pandora [Broda *et al.*, 2007], Proof Lab [Sieg, 2007], and ProofWeb [Wiedijk and van Raamsdonk, 2007]. These tools basically differ in how proofs are visualized, (notably whether proofs are trees or sequences), whether both forward and backward reasoning are supported, the availability of global, tactical and strategic help and debugging facilities, and proof checking. [Van Ditmarsch, 1998] provides a nice survey of

these tools. We discuss few notable ones below.

Pandora is a Java based tool that allows the user to reason both forwards and backwards, checking the rule application at each stage and providing feedback. Proof Lab makes use of the AProS algorithm and can additionally search for proofs as well using a combination of forward chaining, backward chaining, and contradiction. In contrast, we focus on generating proofs without contradiction, thereby staying true to what the original problem asks for.

ProofWeb makes use of a state-of-the-art proof assistant Coq [Coq Development Team, 2006], which allows encoding of various tactics for proof generation. However, none of these systems present any performance results as we do. More significantly, this paper makes an orthogonal point, namely how to speed up search [1], whether forward/backward and with/without contradiction, by using offline computation (which exploits the small formula sized hypothesis) and a two-staged proof generation strategy that first computes an abstract proof and then the natural proof. These optimizations also pave way for generating fresh problems that is not addressed by any of these existing tools.

Problem Generation [Singh *et al.*, 2012] describes a problem generation technology for generating problems that are similar in structure to a given algebraic identity proof problem. The underlying technology leverages continuity of the underlying domain of algebraic expressions, and uses extension of polynomial identity testing to check the correctness of a generated problem candidate on a random input. In contrast, the domain of Boolean expressions is highly non-continuous or discrete, and hence requires a different technology of checking the correctness or well-formedness of a problem candidate on all inputs. Furthermore, unlike [Singh *et al.*, 2012], our technology also enables solution generation.

[Andersen *et al.*, 2013] describes a problem generation technology for *procedural domain*, which includes problems commonly found in middle-school math curriculum such as subtraction and greatest common divisor computation. The underlying technology leverages test input generation techniques [Tillmann and de Halleux, 2008] to generate problems that explore various paths in the procedure that the student is expected to learn. In contrast, we address problem generation for *conceptual domain*, where there is no step-by-step decision procedure that the student can use to solve a problem, but it requires creative skills such as pattern matching.

Solution Generation [Gulwani *et al.*, 2011] describes a solution generation technology for ruler/compass based geometric construction problems. It is based on search techniques like ours with some interesting similarities and differences. It performs a forward breadth-first search (like ours) by repeatedly applying ruler/compass operations to reach the desired output geometric object from input geometric objects. It represents a geometric object using a concrete representation, which constitutes its *probabilistic* hash, in order to avoid symbolic reasoning. We also avoid symbolic reasoning, but by using an *abstract hash* (i.e., truth-table representation). Since inference rules cannot be directly applied on the abstract hash, we resort to leveraging offline computation (UPG)—this also provides the additional benefit of generating problems with given solution characteristics.

[1] Though we demonstrate this in the context of forward search without contradiction, our methodology is applicable more broadly.

7 Conclusion

Computer-aided instruction can raise the quality of education by making it more interactive and customized. We have provided the core building blocks, namely problem generation and solution generation, for the classic subject domain of natural deduction taught in an introductory logic course. This can free instructors from the burden of creating and generating sample solutions to assignment problems. An instructor can create few interesting seed problems and our tool can automatically generate variants of these problems with similar difficulty level. Our tool is not only indispensable in MOOCs (Massive Open Online Courses) settings, but is also useful for traditional classroom settings.

While this paper focuses on problem generation and solution generation, another important component of computer-aided education is automated feedback generation [Singh *et al.*, 2013] and grading [Alur *et al.*, 2013]. We are working on extending our solution generation technology to complete partial proofs or fix buggy proofs submitted by students. We next plan to deploy our tool in a real course and perform user studies to measure the productivity impact of various workflows around our tool. The data collected can also help improve the effectiveness of problem recommendation and hint generation. For example, once we have statistical data about perceived problem difficulty, we can use machine learning techniques that can learn correlations between problem difficulty features and perceived difficulty, and use it to build a better problem recommendation tool.

The SAT solving and theorem proving communities have focused on solving large-sized problem instances in a reasonable amount of time. In contrast, this paper innovates by developing techniques for solving small-sized instances in real time. The small-sized assumption allows use of offline computation and use of bitvector data-structures to alleviate the cost associated with symbolic reasoning. This paves way for some new applications, namely generation of human-readable proofs, and problem generation. We believe that some of the ideas presented in the paper might also be more broadly applicable. Breaking proof search in two parts (abstract proof and its refinement) is a general concept that might apply to other proof generation settings. Our modeling of problem generation as reverse of solution generation is a general concept that might apply to problem generation in other subject domains.

Acknowledgments

Umair Z. Ahmed was partially supported by a grant from Microsoft Research, India. We would like to thank Adam Smith and the anonymous reviewers for their valuable feedback on previous drafts of this paper.

References

[Alur *et al.*, 2013] Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. Automated grading of dfa constructions. In *IJCAI*, 2013.

[Andersen *et al.*, 2013] Erik Andersen, Sumit Gulwani, and Zoran Popovic. A trace-based framework for analyzing and synthesizing educational progressions. In *CHI*, 2013.

[Barwise and Etchemendy, 1994] Jon Barwise and John Etchemendy. *Hyperproof*. CSLI Publications, 1994.

[Bergmann, 2008] Merrie Bergmann. *The Logic Book*. McGraw-Hill, 5th edition, 2008.

[Bornat,] Richard Bornat. Jape. http://www.cs.ox.ac.uk/people/bernard.sufrin/personal/jape.org.

[Broda *et al.*, 2007] K. Broda, J. Ma, G. Sinnadurai, and A. Summers. Pandora: A reasoning toolbox using natural deduction style. *Logic Journal of IGPL*, 15(4):293–304, 2007.

[Cederblom and Paulsen, 2011] Jerry Cederblom and David Paulsen. *Critical Reasoning*. Wadsworth Publishing, 7th edition, 2011.

[Coq Development Team, 2006] Coq Development Team. The Coq proof assistant reference manual: Version 8.1. 2006.

[Coursera,] Coursera. Introduction to Logic. https://www.coursera.org/course/intrologic.

[Dyckhoff *et al.*,] Roy Dyckhoff, Neil Leslie, Tom Peillon, Brenda Rapley, Luis Pinto, Andrew Adams, Christian Urban, Jacob Howe, and Others. MacLogic. http://www.cs.st-andrews.ac.uk/ rd/logic/mac.

[Frederic D. Portoraro, 1994] Robert E. Tully Frederic D. Portoraro. *Logic with Symlog; Learning Symbolic Logic by Computer*. Prentice Hall, Englewood Cliffs USA, 1994.

[Gentzen, 1964] G. Gentzen. Investigations into logical deduction. *American philosophical quarterly*, 1(4):288–306, 1964.

[Gulwani *et al.*, 2011] Sumit Gulwani, Vijay Anand Korthikanti, and Ashish Tiwari. Synthesizing geometry constructions. In *PLDI*, 2011.

[Howard-Snyder *et al.*, 2008] Frances Howard-Snyder, Daniel Howard-Snyder, and Ryan Wasserman. *The Power of Logic*. McGraw-Hill, 4th edition, 2008.

[Hurley, 2011] Patrick J Hurley. *A Concise Introduction to Logic*. Wadsworth Publishing, 11th edition, 2011.

[kha,] Khan Academy. Logical Reasoning. https://www.khanacademy.org/math/geometry/logical-reasoning/.

[Knuth, 2011] Donald E. Knuth. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 2011.

[oli,] Open Learning Initiative. Logic & Proofs. http://oli.cmu.edu/courses/free-open/logic-proofs-course-details.

[Sieg, 2007] W. Sieg. The apros project: Strategic thinking & computational logic. *Logic Journal of IGPL*, 15(4):359–368, 2007.

[Singh *et al.*, 2012] Rohit Singh, Sumit Gulwani, and Sriram Rajamani. Automatically generating algebra problems. In *AAAI*, 2012.

[Singh *et al.*, 2013] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. In *PLDI*, 2013.

[Tillmann and de Halleux, 2008] Nikolai Tillmann and Jonathan de Halleux. Pex-white box test generation for .NET. In *TAP*, pages 134–153, 2008.

[Van Ditmarsch, 1998] H. Van Ditmarsch. User interfaces in natural deduction programs. *User Interfaces*, 98:87, 1998.

[vander Nat, 2010] Arnold vander Nat. *Simple formal logic. With common-sense symbolic techniques.* London: Routledge, 2010.

[Wiedijk and van Raamsdonk, 2007] C.K.F. Wiedijk and M.H.F. van Raamsdonk. Teaching logic using a state-of-the-art proof assistant. *PATE*, page 33, 2007.