Optimal automatizable heuristic proof systems

Alexander V. Smal
St. Petersburg Department of Steklov Mathematical Institute,
Laboratory of Mathematical Logic
smal@logic.pdmi.ras.ru

Given a specific problem, does there exist the "fastest" algorithm for it? Does there exist a proof system possessing the "shortest" proofs of the positive solutions to the problem? Although the first result in this direction was obtained by Levin in 1970s, these important questions are still open for most interesting languages, for example, the language of propositional tautologies.

1 Classical proof complexity

1.1 Proof systems

Cook and Reckhow introduced notion of a proof system in 1979.

Definition 1.1. *Proof system* for a language L is a polynomial-time mapping of all strings ("proofs") onto "theorems" (elements of L). If $L = \mathbf{TAUT}$ is the language of all propositional tautologies, the system is called a *propositional proof system*.

The existence of a polynomially bounded propositional proof system (that is, a system that has a polynomial-size proof for every tautology) is equivalent to $\mathbf{NP} = \mathbf{co{-}NP}$. In the context of polynomial boundedness a proof system can be equivalently viewed as a function that given a formula and a "proof", verifies in polynomial time that a formula is a tautology: it must accept at least one "proof" for each tautology (*completeness*) and reject all proofs for non-tautologies (*soundness*).

Resolution. There is a lot of proof systems that are studied in proof complexity. The most well-known example of propositional proof systems is *Resolution proof system.* This proof system is based on a *resolution rule*: let A and B be disjunctions of literals and x be a propositional variable, then

$$\frac{A \vee x, \quad B \vee \neg x}{A \vee B}.$$

It's not hard to show that using the resolution rule one can deduce an empty clause from a set of clauses iff this set is unsatisfiable. Given a formula Φ in DNF, *a resolution proof* for it is a sequence of the resolution rule applications (it can also be a tree or a DAG) to clauses of $\neg\Phi$ (in CNF). It is proved by Haken in 1985 that some tautologies (e.g., pigeonhole principle) have only exponentially long proofs in the Resolution proof system.

1.2 Optimal proof systems

How to compare two proof systems? Which proof system is the best one? To answer these questions it is necessary to establish a partial order on a set of proof systems that reflects shortest proofs.

Definition 1.2. One proof system Π_w is *simulated* by another one Π_s if the shortest proofs for every tautology in Π_s are at most polynomially longer than the shortest proofs in Π_w . The notion of p-simulation is similar, but requires also a polynomial-time computable function for translating the proofs from Π_w to Π_s.

Definition 1.3. A (p-)optimal propositional proof system is one that (p-)simulates all other propositional proof systems.

The existence of an optimal (or p-optimal) proof system is a major open question for many languages including $\mathbf{TAUT}$. Optimality would imply p-optimality for any language if and only if there is a p-optimal proof system for SAT. If an optimal system for $\mathbf{TAUT}$ would exist, it would allow to reduce the $\mathbf{NP}$ vs. $\mathbf{co{-}NP}$ question to proving proof size bounds for just one proof system. The existence of an optimal system for quantified Boolean formulas would imply a complete language in $\mathbf{NP} \cap \mathbf{co{-}NP}$. Unfortunately, no concise widely believed conjectures (like $\mathbf{NP} \neq \mathbf{co{-}NP}$) are known to imply the (non-)existence of (p-)optimal proof systems.

1.3 Acceptors and automatizable proof systems

Definition 1.4. An *acceptor* for language L is a semidecision procedure, i.e., an algorithm that answers 1 for $x \in L$, and does not answer 1 for $x \notin L$. It is optimal if it takes for it at most polynomially longer to stop on every $x \in L$ than for any other correct algorithm on the same x. (Note that Levin showed that an optimal algorithm does exist for finding witnesses to non-tautologies. However, it gives an optimal acceptor neither for $\mathbf{TAUT}$ nor for $\mathbf{SAT}$.)

Definition 1.5. An automatizable proof system is one that has an automatization procedure that given a tautology, outputs its proof of length polynomially bounded by the length of the shortest proof in time bounded by a polynomial in the output length.

It is easy to see that such proof system can be easily turned into acceptor with the running time polynomially related to the proof size. Vice versa, an acceptor can be converted into an automatizable proof system, where the proof is just the number of steps (written in unary) that the acceptor makes before accepting its input. Thus, in the classical case there is no difference between acceptors and automatizable proof systems.

2 Heuristic proof complexity

2.1 What does "heuristic" mean

Hirsch and Itsykson introduced heuristic acceptors and heuristic proof systems. Heuristic algorithms are algorithms that make errors for a small amount of inputs. Similarly, heuristic proof systems claim a small amount of wrong theorems.

What does "small amount" formally mean is defined with respect to a probability distribution concentrated on non-theorems, and it is required that the probability of sampling a non-theorem accepted by an algorithm or validated by a proof system is small.

Definition 2.1. A pair (D, L) is called a *distributional proving problem* if D is a collection of probability distributions D_n concentrated on $\overline{L} \cap \{0,1\}^n$.

In what follows $\Pr_{x \leftarrow D_n}$ denotes the probability taken over x from such distribution, while $\Pr_A$ denotes the probability taken over internal random coins used by algorithm A.

2.2 Heuristic automatizer

Definition 2.2. A heuristic acceptor for distributional proving problem (D, L) is a randomized algorithm A with two inputs $x \in \{0,1\}^*$ and $d \in \mathbb{N}$ that satisfies the following conditions:

1. A either outputs 1 or does not halt at all;
2. for every $x \in L$ and $d \in \mathbb{N}$, $A(x, d) = 1$;
3. for every $n, d \in \mathbb{N}$, $\Pr_{r \leftarrow D_n}\left\{\Pr_A\{A(r, d) = 1\} > \frac{1}{8}\right\} < \frac{1}{d}$.

Remark. For recursively enumerable L, conditions 1 and 2 can be easily enforced at the cost of a slight overhead in time by running L's semidecision procedure in parallel.

Definition 2.3. The *time* spent by automatizer A on input (x, d) is defined as the median time

$$t_A(x, d) = \min\left\{t \in \mathbb{N} \;\middle|\; \Pr_A\{A(x, d) \text{ stops in time at most } t\} \geq \frac{1}{2}\right\}.$$

Definition 2.4. Automatizer S simulates automatizer W if there are polynomials p and q such that for every $x \in L$ and $d \in \mathbb{N}$,

$$t_S(x, d) \leq \max_{d' \leq q(d \cdot |x|)} p(t_W(x, d') \cdot |x| \cdot d).$$

Definition 2.5. An *optimal* automatizer is one that simulates every other automatizer.

Theorem 2.1 (optimal automatizer)**.** Let (D, L) be a distributional proving problem, where L is recursively enumerable and D is polynomial-time samplable, i.e., there is a polynomial-time randomized Turing machine that given 1^n on input outputs x with probability $D_n(x)$ for every $x \in \{0,1\}^n$. Then there exists an optimal automatizer for (D, L).

2.3 Heuristic proof systems

Definition 2.6. Randomized Turing machine Π is a *heuristic proof system* for distributional proving problem (D, L) if it satisfies the following conditions.

1. The running time of $\Pi(x, w, d)$ is bounded by a polynomial in d, $|x|$, and $|w|$.
2. (Completeness) For every $x \in L$ and every $d \in \mathbb{N}$, there exists a string w such that $\Pr\{\Pi(x, w, d) = 1\} \geq \frac{1}{2}$. Every such string w is called a *correct $\Pi^{(d)}$-proof of x*.
3. (Soundness) $\Pr_{x \leftarrow D_n}\{\exists w : \Pr\{\Pi(x, w, d) = 1\} > \frac{1}{8}\} < \frac{1}{d}$.

Definition 2.7. Heuristic proof system is *automatizable* if there is a randomized Turing machine A satisfying the following conditions.

1. For every $x \in L$ and every $d \in \mathbb{N}$, there is a polynomial p such that

   $$\Pr_{w \leftarrow A(x,d)}\left\{|w| \leq p(d \cdot |x| \cdot |w_*|) \wedge \Pr\{\Pi(x, w, d) = 1\} \geq \tfrac{1}{4}\right\} \geq \tfrac{1}{4},$$

   where w_* is the shortest correct $\Pi^{(d)}$-proof of x.
2. The running time of $A(x, d)$ is bounded by a polynomial in $|x|$, d, and the size of its own output.

Remark. It is not required that the algorithm A generates correct proofs. It suffices to generate "quasi-correct" (such that $\Pr\{\Pi(x, w, d) = 1\} \geq \frac{1}{4}$) with probability $\frac{1}{4}$.

Definition 2.8. Heuristic proof system Π_1 *simulates* heuristic proof system Π_2 if there exist polynomials p and q such that for every $x \in L$, the shortest correct $\Pi_1^{(d)}$-proof of x has size at most

$$p(d \cdot |x| \cdot \max_{d' \leq q(d \cdot |x|)}\{\text{the size of the shortest correct } \Pi_2^{(d')}\text{-proof of } x\}). \tag{1}$$

Heuristic proof system Π_1 *p-simulates* heuristic proof system Π_2 if Π_1 simulates Π_2 and there is a polynomial-time (deterministic) algorithm that converts a $\Pi_2^{(q(d \cdot |x|))}$-proof into a $\Pi_1^{(d)}$-proof of size at most (1) such that the probability to accept a new proof is no smaller than the probability to accept the original one.

Theorem 2.2. Automatizable heuristic proof systems are equivalent to heuristic automatizers, e.g.:

1. every such system defines a heuristic automatizer taking time at most polynomially larger than the length of the shortest proof in the initial system;
2. every heuristic automatizer defines such a system that has proof of size at most polynomially larger than the time spent by automatizer.

Corollary 2.1. There is an optimal automatizable heuristic proof system.