The Computational Complexity Column

Eric Allender
Rutgers University, Department of Computer Science
Piscataway, NJ 08855 USA
allender@cs.rutgers.edu

Are there too many complexity classes? Merely trying to understand one aspect of computation, such as the power of randomness, leads to a whole range of complexity classes, such as ZPP, RP, and BPP, to name but a few. Do we really need all of these classes?

One of the most exciting developments in complexity theory in the past few years is the growing body of evidence that all of the aforementioned classes are merely pseudonyms for P. Our guest column this issue gives an overview of this area.

Recent Advances Towards Proving P = BPP

Andrea E. F. Clementi[1] José D. P. Rolim[2] Luca Trevisan[3]

Abstract

Two independent techniques have been developed recently that yield sufficient conditions for P = BPP in terms of worst-case circuit complexity of functions computable in exponential time. Andreev, Clementi and Rolim proved that P = BPP provided that a sparse "efficiently enumerable" language exists of sufficiently high circuit complexity. This result has been subsequently improved by Impagliazzo and Wigderson by showing that either P = BPP or all the decision problems solvable in time $2^{O(n)}$ are solvable by circuits of size $2^{o(n)}$. In this column we discuss these results and their relation with previously known sufficient conditions for P = BPP.

1 Introduction

Randomness is very useful in the design of efficient algorithms for several important problems. Probabilistic algorithms are often the simpler ones to solve a given problem, or the most efficient, or the only efficiently parallelizable ones (see [9]). For some problems, including primality testing and approximation of # P-complete counting problems, only randomized solutions are known. In computational geometry, problems such as the approximate computation of the volume of a convex body *only* admit randomized solutions.

There are several classes of efficient probabilistic algorithms (see e.g. [15]) that differ on the adopted acceptance criteria. In particular, we will consider two classes of algorithms for decision problems: *two-sided bounded error* polynomial time algorithms (BPP algorithms) and *one-sided bounded error* polynomial time algorithms (RP algorithms). A BPP algorithm is required to give a correct answer with probability not smaller than 2/3 for any input; an RP algorithm is a BPP algorithm with the additional guarantee that if the input is a NO instance the algorithm will give the right answer with probability 1. It is not known whether it is possible to efficiently transform any BPP algorithm into an RP algorithm. Furthermore, the question of whether BPP (or RP) is contained in P (here we refer

[1]Dipartimento di Scienze dell'Informazione, University of Rome "La Sapienza", clementi@dsi.uniroma1.it
[2]Centre Universitaire d'Informatique, University of Geneva, rolim@cui.unige.ch
[3]Laboratory for Computer Science, MIT, luca@theory.lcs.mit.edu

to BPP and RP as the class of languages decided by BPP and RP algorithms respectively) is not even a generally accepted conjecture, as opposed to other questions in complexity theory. The results that are described in this column support the conjecture $\mathsf{P} = \mathsf{RP} = \mathsf{BPP}$.

There is a rich area of research in complexity theory dealing with the use of randomness in computations, and in particular on how to efficiently simulate randomized algorithms by means of deterministic ones. The first foundational results for this line of research can be found in the seminal works of Blum and Micali [7] and Yao [19], motivated by cryptographic applications. These papers introduced a formal definition of *Pseudo-Random Generator* (PRG) as a function

$$G \;=\; \{\; G_n \;:\; \{0,1\}^{k(n)} \rightarrow \{0,1\}^n \;,\; n > 0 \;\},$$

denoted by $G : k(n) \rightarrow n$ that "stretches" $k(n)$ truly random bits into n pseudo-random bits ($k(n)$ is commonly said the *price* of G). More formally, G is a PRG if for any sufficiently large n and for any Boolean circuit $C : \{0,1\}^n \rightarrow \{0,1\}$ whose size is at most n we have:

$$|\mathbf{Pr}\,(C(\vec{y}) = 1) - \mathbf{Pr}\,(C(G_n(\vec{x})) = 1)| \leq \frac{1}{n}$$

where $\vec{y}$ is chosen uniformly at random in $\{0,1\}^n$, and $\vec{x}$ in $\{0,1\}^{k(n)}$. A Boolean operator is said to be *quick* if it is computable in time polynomial in the length of its *output* [14]. In particular, a PRG G is quick if $G_n : \{0,1\}^{k(n)} \rightarrow \{0,1\}^n$ is computable in time polynomial in n.

The output set $\{G_n(x) : x \in \{0,1\}^{k(n)}\}$ of a PRG G for a fixed n is also called a *discrepancy set* for the class of circuits of size n[4]. The computation of a BPP algorithm on a fixed input is an easy-to-compute function C of the outcomes of the random coins. It follows that a pseudo-random generator can be used to approximate the fraction of random coin outcomes that make C accept, and thus allows one to decide whether the algorithm accepts the input or not. PRG's can then be considered the natural general method to de-randomize BPP algorithms. In particular, we note that the following simple implication holds.

Theorem 1.1 *If a quick PRG of logarithmic price exists then* $\mathsf{P} = \mathsf{BPP}$.

In Section 2 we will see sufficient conditions for the existence of PRG's of logarithmic price. The above discussion shows that any such condition implies $\mathsf{P} = \mathsf{BPP}$.

The "one-sided" version of a PRG is a *Hitting Set Generator* (HSG): a family of functions $H = \{H_n : \{0,1\}^{k(n)} \rightarrow \{0,1\}^n, n > 0\}$ (denoted by $H : k(n) \rightarrow n$) that, for any sufficiently large n and for any n-input Boolean circuit C with size at most n such that

$$\mathbf{Pr}\,(C(\vec{y}) = 1) \;\geq\; \frac{1}{n}\;,$$

it holds

$$\text{there exists } \;\vec{x} \in \{0,1\}^{k(n)} \;\text{ such that } C(H_n(\vec{x})) = 1\;.$$

While a PRG of price $k(n)$ converts $k(n)$ bit of randomness into a pseudo-random string of length n to be used in a BPP algorithm, a HSG converts $k(n)$ bits of *non-determinism* into a string of length n to be used in an RP algorithm. The following result is a consequence of the definition of HSG's.

[4] The restriction to circuits with n inputs and of size n is done without loss of generality, since we can see a generic circuit with k inputs and of size n as a circuit having n inputs, $n-k$ of them are not used.

Theorem 1.2 *If a quick HSG of logarithmic price exists, then* $\mathsf{P} = \mathsf{RP}$

Indeed, given an RP algorithm and an input, one can run the algorithm on each pseudo-random sequence generated by the HSG and accept the input if and only if one of these sequences make the algorithm accept. It is apparent from the definition that a PRG is also a HSG but the converse is not necessarily true. In Section 3 we will see sufficient conditions for the existence of HSG's of logarithmic price.

2 Proving $\mathsf{P} = \mathsf{BPP}$ Under Circuit Complexity Assumptions

2.1 Using Average-Case Circuit Complexity

The main results in the theory of de-randomization for general classes of probabilistic algorithms can be seen as general techniques to construct PRG's that rely on unproven hardness conditions. In particular, Nisan and Wigderson [14] presented a method to construct *quick* PRG's based on the existence of Boolean functions in EXP that have exponential *hardness* [14]. The hardness condition used by Nisan and Wigderson requires the existence of a function in EXP that has hard *average-case* circuit complexity. More formally, a function $f : \{0,1\}^n \rightarrow \{0,1\}$ is (ϵ, L)-*hard* if, for any circuit C of size at most L,

$$|\mathbf{Pr}\,(C(\vec{x}) = f(\vec{x})) - 1/2| \leq \epsilon/2.$$

Given a Boolean function $F = \{F_n : \{0,1\}^n \rightarrow \{0,1\},\ n > 0\}$, the *hardness* at n of F (denoted as $H_F(n)$) is defined as the maximum integer h_n such that F_n is $(1/h_n, h_n)$-hard. Then, F has exponential hardness if $H_F(n) \geq 2^{\Omega(n)}$. Nisan and Wigderson showed a fundamental "Hardness vs Randomness" result.

Theorem 2.1 ([14]) *If a Boolean function F exists such that* i) $F \in \mathsf{EXP}$, *and* ii) *F has exponential* hardness, *then there exists a quick PRG $G : k(n) \rightarrow n$ where $k(n) = O(\log n)$, and consequently* $\mathsf{P} = \mathsf{BPP}$.

A more general form of the above theorem states that if BPP is not contained in $\mathsf{DTIME}(n^{\mathrm{poly}\,log(n)})$ then any EXP-complete function has rather "low" hardness. This possible shape of the "complexity world" would be quite different from what most complexity theorists expect. Nisan and Wigderson's work thus had a tremendous impact in the Complexity Community: people started to think that the gap between BPP and P might be very small. In [6], another Hardness-vs Randomness trade off has been obtained that states that if there is a function in EXP having hardness $2^{n^{\Omega(1)}}$ then $\mathsf{BPP} \subseteq \mathsf{DTIME}(n^{\mathrm{poly}\log n})$. We thus have that under the hardness assumption above, the use of randomness can be just slightly helpful in speeding computation.

Further research on the "Hardness vs Randomness" problem has been focused on the following aspect of Nisan and Wigderson's work. The hardness required by Nisan and Wigderson's construction of quick PRG's refers to average-case circuit complexity. Then a consequent and natural question is the following: Does any "worst-case" hardness assumption on the circuit complexity of Boolean functions computable in time exponential in the input size exist which allows one to derive an efficient derandomization method (in particular, to obtain $\mathsf{P} = \mathsf{BPP}$)? To motivate this question, consider the potential new insights that would flow from the knowledge of the precise relationship between the true computational power of randomness and the problem of finding lower bounds for the worst-case circuit complexity of classes of recursive Boolean functions (this latter being one of the most studied problems in complexity theory). This argument will be the subject of Sections 3.2 and 3.3.

3 HSG's: How to Use and Construct Them

3.1 Hitting Set Generators and BPP

Another interesting question is that concerning the real relationship between RP and BPP. In the context of de-randomization, this question leads us to a deeper comparison between the real power of PRG's and that of HSG's: can the latter replace the former to de-randomize BPP algorithms? As we have observed above, an HSG is not in general a PRG and since the acceptance criterion of BPP requires the presence of a two-sided random space, the intuition here is that HSG's do not correctly work for BPP. Furthermore, in several cases the construction of combinatorial objects having one-sided random (i.e. hitting) properties has turned out to be more efficient than that of combinatorial objects having two-sided random (i.e. discrepancy) properties (for a survey of these cases see Appendix C of [10]). This is for instance the case for *extractors* and *OR-dispersers* ([13]). Another case in which one-sided randomness seems to be easier to achieve is in the case of *"small" linear subspaces* of $\{0,1\}^n$ [4]. It is indeed possible to construct small hitting sets for this class of subsets (and, so, for the corresponding characteristic functions) that imply some explicit, exponential lower bounds for the branching program model [4], but no construction of non-trivial discrepancy sets for this class is known.

One general reason for the fact that one-sided random objects seem to be easier to construct is that they have a *monotonicity* property not satisfied by discrepancy sets: if $H = \{H_n, n > 0\}$ is a family of hitting objects then any other family $H' = \{H'_n, n > 0\}$, such that for any $n > 0$ H'_n "contains" H_n, has (at least) the same same hitting properties.

In a rather surprising way, Andreev *et al* showed that the above intuition about the real relationship between PRG's and HSG's is somewhat false.

Theorem 3.1 ([1]) *Let $k(n) = O(\log n)$. If there exists a quick HSG $H : k(n) \to n$ then* BPP = P.

The proof of the above result relies on the following lemma which is of independent interest.

Lemma 3.1 ([1]) *Let $q(n)$ be any positive function such that $n \le q(n) \le 2^n$. There is a deterministic algorithm A that, given access to a quick HSG $H : k(n) \to n$ (with $k(n) = \Omega(\log n)$), and given in input any circuit $C(x_1, \ldots, x_n)$ of size at most $q(n)$, computes in time polynomial in*

$$2^{k(q(n)^{O(1)})}$$

a value $A(C)$ such that

$$|\mathbf{Pr}(C = 1) - A(C)| \le \frac{1}{q(n)}.$$

The algorithm A uses the hitting set generated by the HSG to construct a discrepancy set for C. The main novelty is that the obtained discrepancy set depends on C and thus on the BPP algorithm that we want to de-randomize. This implies that the de-randomization method is not *oblivious*. Roughly speaking, the algorithm trade-off "uniformity" versus "two-sidedness".

Actually, Theorem 3.1 gives a more general consequence: by considering the "price" $k(n)$ (with $k(n) = \Omega(\log n)$) of the HSG as a parameter, we have the following

Corollary 3.1 ([1]) *If a quick HSG $H : k(n) \to n$ exists, then for any time-bound $t(n)$, we have*

$$\mathsf{BPTIME}(t) \subseteq \mathsf{DTIME}\left(2^{O(k(t^{O(1)}))}\right),$$

where $\mathsf{BPTIME}(t)$ *is the class of languages accepted by probabilistic, two-sided error Turing machines running in time t.*

Notice that this result is comparable to the one in [12, 14] stating that the existence of a quick PRG $G : k(n) \to n$ implies $\mathsf{BPTIME}(t) \subseteq \mathsf{DTIME}(2^{O(k(t^2))})$.

We emphasize that the same technique used to prove Lemma 3.1 has been recently used to solve the "one-sided vs two-sided" problem in other frameworks (see Section 4).

3.2 Using Worst-Case Hardness to construct HSG's

Let's return now to the question of finding worst-case hardness conditions sufficient to de-randomize BPP-algorithms. Theorem 3.1 shows a different way to achieve this goal. This is used in Theorem 3.2 to show that it is indeed sufficient to to use a *worst-case* hard Boolean function (as opposed to *average-case* hardness) to construct a HSG.

Intuitively speaking, the worst-case hardness of a Boolean function seems to be much closer to its hitting properties than to its discrepancy properties. In [2] and successively in [3], Andreev *et al* indeed gave the first worst-case hardness condition which is sufficient to construct quick HSG's that satisfy Theorem 3.1 thus obtaining $\mathsf{P} = \mathsf{BPP}$. A more formal description of this result follows. The circuit complexity of a Boolean operator H will be denoted as $L^{op}(H)$. Observe that if $L^{op}(k, n)$ denotes the worst-case circuit complexity of Boolean operators $H : k(n) \to n$, then it is known [17] that, for any $\log n \leq k \leq n$,

$$L^{op}(k, n) = (1 + o(1))(2^k n)/(k + \log n).$$

Furthermore, for almost every Boolean operator $H : k \to n$, we have

$$L^{op}(H) = \Theta((2^k n)/(k + \log n)) .$$

The following theorem gives a sufficient condition for $\mathsf{P} = \mathsf{BPP}$ in terms of the worst-case circuit complexity of characteristic functions of sets generated by Boolean operators (as defined above).

Theorem 3.2 ([3]) *Let $k(n) = (2+O(1)) \log n$. A constant $0 < c_0 < 1$ exists such that if there exists a quick operator $H : k(n) \to n$ such that the characteristic function of its output sets*

$$F^H = \{F_n^H : \{0,1\}^n \to \{0,1\} \ , \ \textit{where} \ \ F_n^H(\vec{x}) = 1 \ \ \textit{iff} \ \exists \ \vec{y} \in \{0,1\}^{k(n)} \textit{s.t.} \ H_n(\vec{y}) = \vec{x}, \ n > 0\}$$

satisfies

$$L(F_n^H) \geq 2^{k(n)} n^{c_0} \ ,$$

then it is possible to construct a quick HSG $H' : k'(n) \to n$ where $k'(n) = \Theta(\log n)$, thus obtaining $\mathsf{P} = \mathsf{BPP}$.

Another way to state the above theorem is the following. Assume that there exists a sparse language $S = \{S_n \subseteq \{0,1\}^n, n > 0\}$ that can be generated by a uniform algorithm that runs in time polynomial in n (so in time polynomial in the length of its output), and such that the worst-case circuit complexity of deciding S is not much smaller (up to some polynomial factor) than the worst-case circuit complexity of generating S. Then $\mathsf{P} = \mathsf{BPP}$.

The proof of Theorem 3.2 relies on the following fact. There is a precise trade-off between the worst-case circuit complexity of partial Boolean functions and the number of 1's in their output table. In particular, we give a precise mathematical form of the intuitive fact that a partial Boolean function having a hard worst-case circuit complexity cannot return 0 for a "large" number of inputs. So, according to the definition of HSG's, any function that has a hard worst-case circuit complexity turns out to have also a good hitting property. This property is used to construct the preliminary version of the HSG which is then combined with a convenient use of *OR Dispersers* [16], a family of particular *expander* graphs.

3.3 A Stronger Result Via the De-randomization of the XOR Lemma

Theorem 2.1 requires the existence of a Boolean function f in EXP such that, for some $\epsilon > 0$, any circuit C of size $2^{\epsilon n}$ can only achieve success probability $1/2 + 2^{-\epsilon n}$ while trying to predict g on a random input of length n. Babai et al. [6] later proved that if a function f in EXP exists having circuit complexity $2^{\Omega(n)}$ then there exists another function g in EXP such that, for some fixed $\epsilon > 0$, any circuit of size $2^{-\epsilon n}$ can only achieve success probability $1 - 1/n^2$ while trying to predict f on a random input of size n. The difficulty of predicting g on a random input can be increased by defining a function $h(x_1, \ldots, x_k) = g(x_1) \oplus \ldots \oplus g(x_k)$: from Yao's Xor Lemma [19] it follows that the success probability of a circuit for h of size $2^{\epsilon n}$ goes down to $1/2$ exponentially fast in k. If we take $k = O(n)$ the success probability will be as low as $1 - 2^{-\Omega(n)}$, however h does not yet satisfy the conditions of Theorem 2.1 since it has $O(n^2)$ inputs and the hardness is "only" $2^{\Omega(n)}$. From this assumption it is only possible to infer from the techniques of [14, 6] that BPP can be deterministically simulated in time $n^{\text{poly} \log n}$.

Impagliazzo and Wigderson [8] recently made further progress by showing how to de-randomize the Xor Lemma. Their main result is a procedure that, given a function g with n inputs and a certain unpredictability, constructs another function h with $O(n+k)$ inputs whose unpredictability decreases exponentially in k. Thus, assuming that a function f in EXP exists with circuit complexity $2^{\Omega(n)}$, one can deduce that a function g exists in EXP such that circuits of size $2^{\Omega(n)}$ only have success probability $1 - 1/n^2$ on g. Then, using a result of Impagliazzo [11] it follows that a function g' in EXP exists such that circuits of size $2^{\Omega(n)}$ have success probability at most $2/3$, and, eventually, that a function h exists in EXP such that circuits of size $2^{\epsilon n}$ only have success probability $2^{-\epsilon n}$. In turn, the latter statement implies the existence of PRG of logarithmic price and thus $\mathsf{P} = \mathsf{BPP}$.

Theorem 3.3 *[8] If a function in f exists having circuit complexity $2^{\Omega(n)}$ then* $\mathsf{P} = \mathsf{BPP}$.

4 Related Results and Conclusion

De-randomization is not the only research direction that has been explored about the use of randomness in computation. An alternative approach deals with the use of *weak sources of randomness* (see [13]). Even in this case there is a difference between one-sided pseudorandom structures and two-sided pseudorandom structures and Theorem 3.1 is a useful tool [5].

A natural question to ask is whether the results described in this column (or at least part of them) extend to parallel and space-bounded classes. The issue of parallelization is not addressed by Impagliazzo and Wigderson [8], and some steps in their construction appear to be hard to parallelize. On the other hand, a bottleneck for the parallelization of the techniques of Andreev et al. [3] was the use of Theorem 3.1, whose proof in [1] was inherently sequential. A new proof of Theorem 3.1 appeared in [5] extends to parallel and space-bounded classes, and so do, to a certain extent, the techniques of [3]. The current state of the art on this topic is that a reasonable worst-case sufficient condition exists implying $\mathsf{BPNC} = \mathsf{NC}$ but no worst-case circuit complexity condition is known to imply $\mathsf{BPL} = \mathsf{L}$ (or even $\mathsf{RL} = \mathsf{L}$).

While the main goal of de-randomization theory is to prove $\mathsf{P} = \mathsf{BPP}$, some weaker, but still extremely interesting, results may be within reach. Theorem 3.1 already states that BPP is not much more powerful than RP. A recent result due to Fortnow establishes that any BPP problem is solvable by an RP algorithm making one oracle query to a promise-RP oracle (promise-RP is the extension of RP to *promise problems*). It is an open question to prove the same result without using promise problems. Solving this question may be an important intermediate step towards proving $\mathsf{BPP} = \mathsf{RP}$, a result that would have several complexity-theoretic consequences (e.g. RP would be closed under complement, BPP would be contained in NP and so on).

References

[1] Andreev A., Clementi A., and Rolim J. (1996), "A New General De-Randomization Method", *J. of the ACM.* to appear. Extended Abstract in *23-th Annual International Colloquium on Algorithms, Logic and Programming (ICALP'96)*, LNCS, 1099, pp. 357-368.

[2] Andreev A., Clementi A., and Rolim J. (1996) "Hitting Properties of Hard Boolean Operators and Their Consequences on BPP, *ECCC Report TR96-055.*

[3] Andreev A., Clementi A., and Rolim J. (1997), "Worst-case Hardness Suffices for Derandomization: a New method for Hardness-Randomness Trade-Offs", in *24-th Annual International Colloquium on Algorithms, Logic and Programming (ICALP'97)*, LNCS, 1256, pp. 177-187.

[4] Andreev A., Baskakov J., Clementi A., and Rolim J. (1997), "Efficient Construction of ϵ-Biased Sample Spaces for Systems of Linear Tests and Applications", *Technical Report in ECCC - TR-97-053.*

[5] Andreev A., Clementi A., Rolim J. and Trevisan L. (1997), "Weak Random Sources, Hitting Sets, and BPP Simulation", *SIAM J. of Comput.*, to appear. Extended abstract in *38-th Annual IEEE Symposium on Foundations on Computer Science*, pp. 264-273.

[6] Babai L., Fortnow L., Nisan N. and Wigderson A. (1993) "BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs" *Computational Complexity*, 3, pp. 307-318.

[7] Blum M., and Micali S. (1984), "How to generate cryptographically strong sequences of pseudo-random bits", *SIAM J. of Computing*, 13(4), pp. 850-864.

[8] Impagliazzo R., and Wigderson A. (1997), "P= BPP if E requires exponential circuits: Derandomizing the XOR lemma" In *29-th Annual ACM Symposium on Theory of Computing*, pp. 220-229.

[9] Motwani R., and Raghavan P. (1995), *Randomized Algorithms*, Cambridge University Press.

[10] Goldreich O. (1997), "A Sample of Samplers: A Computational Perspective on Sampling", *ECCC*, TR97-20.

[11] Impagliazzo, R. (1995), "Hard-core distributions for somewhat hard problems", in *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pp. 538–545.

[12] Nisan N. (1990), *Using Hard Problems to Create Pseudorandom Generators, ACM Distinguished Dissertation*, MIT Press.

[13] Nisan N. (1996), "Extracting randomness: How and why", In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, pp. 44-58.

[14] Nisan N., and Wigderson A. (1994), "Hardness vs Randomness", *J. Comput. System Sci.* 49, pp. 149-167.

[15] Papadimitriou C.H. (1993), *Computational Complexity,* Addison-Wesley.

[16] Saks M., Srinivasan A., and Zhou S. (1995). "Explicit dispersers with polylog degree", In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pp. 479-488, 1995.

[17] Wegener, I. (1987), *The complexity of finite Boolean functions, Wiley-Teubner Series in Computer Science.*

[18] Ta-Shma A. (1996), "On extracting randomness from weak random sources", In *28th Annual ACM Symposium on Theory of Computing*, pp. 276-285.

[19] Yao A. (1982), "Theory and applications of trapdoor functions", in *23th Annual IEEE Symposium on Foundations on Computer Science*, pp. 80-91.