The Computational Complexity Column

by

Vikraman Arvind

Institute of Mathematical Sciences, CIT Campus, Taramani
Chennai 600113, India
arvind@imsc.res.in
http://www.imsc.res.in/~arvind

The mid-1960's to the early 1980's marked the early epoch in the field of computational complexity. Several ideas and research directions which shaped the field at that time originated in recursive function theory. Some of these made a lasting impact and still have interesting open questions. The notion of lowness for complexity classes is a prominent example.

Uwe Schöning was the first to study lowness for complexity classes and proved some striking results. The present article by Johannes Köbler and Jacobo Torán, written on the occasion of Uwe Schöning's soon-to-come 60^{th} birthday, is a nice introductory survey on this intriguing topic. It explains how lowness gives a unifying perspective on many complexity results, especially about problems that are of intermediate complexity. The survey also points to the several questions that still remain open.

LOWNESS RESULTS: THE NEXT GENERATION

Johannes Köbler
Humboldt Universität zu Berlin
koebler@informatik.hu-berlin.de

Jacobo Torán
Universität Ulm
jacobo.toran@uni-ulm.de

Abstract

Our colleague and friend Uwe Schöning, who has helped to shape the area of Complexity Theory in many decisive ways is turning 60 this year. As a little birthday present we survey in this column some of the newer results related with the concept of lowness, an idea that Uwe translated from the area of Recursion Theory in the early eighties. Originally this concept was applied to the complexity classes in the polynomial time hierarchy. An overview of the many results inspired by the lowness idea was written by the first author in [25]. We review here the lowness scene 20 years later, focusing mainly in the classes out of PH.

1 Introduction

The concept of lowness originated in the area of Recursion Theory (see [30]). Intuitively a language is low for a computation model (like the Turing machine) or an operator if it is powerless when used as oracle for the model. Lowness indicates that a set has low complexity or low information content since it behaves like the empty set with respect to a certain operator. Uwe Schöning [37] introduced the notion of lowness to the area of complexity where this concept really flourished and found many new applications. Schöning also found several natural examples of low sets, most notably the Graph Isomorphism problem, being low for different complexity classes. Initially the notion of lowness was applied to the classes in the polynomial hierarchy but it was soon realized that it can be very useful to consider lowness for other complexity classes like $\oplus\mathsf{P}$ or PP as well.

Let C be a complexity class. For a language L, we denote by $C(L)$ the class of languages computed by a computation model of the kind defining C with ad-

ditional access to an oracle for L. We say that L is low for C if the computation model for the class does not get any extra power when querying L. More formally:

Definition 1. *Let C be a complexity class (with a reasonable relativized version). A language L is said to be low for C (in symbols: $L \in \mathsf{Low}(C)$) if $C(L) = C$.*

An easy observation is that the class of sets that are low for P is P itself. The class of low sets for NP is exactly $\mathsf{NP} \cap \mathsf{coNP}$. To see this, observe that if L is low for any complexity class C, then L as well as $\overline{L}$ belong to C. This also shows that a lowness result is in principle stronger than a containment result. For the other direction, a set L in $\mathsf{NP} \cap \mathsf{coNP}$ does not provide any additional power to an NP computation since the nondeterministic machine can simulate any query q to L by guessing the answer to q as well as a certificate for $q \in L$ or for $q \notin L$.

Another complexity class whose low sets can be completely characterized is $\oplus\mathsf{P}$ (parity P), the class of languages computed by polynomial time nondeterministic Turing machines that have an odd number of accepting paths if and only if the input belongs to the language. Papadimitriou and Zachos [33] showed that the class of low sets for $\oplus\mathsf{P}$ coincides with $\oplus\mathsf{P}$ itself. For many other complexity classes, an exact characterization of the low sets for the class is not known. In fact, the characterization of the low sets for Σ_2^p was already posted as an open question in Schöning's original paper [37]. There are however many interesting lowness results for a large variety of complexity classes. Some of these results strengthen existing containment results or show that certain sets cannot be complete unless unexpected collapses happen. The most well-known among results of this kind is the fact that the Graph Isomorphism problem is low for Σ_2^p [38] implying that GI cannot be NP-complete unless the polynomial hierarchy collapses. Some other lowness results offer a unified explanation of complexity upper bounds. For example, Toda's Theorem [40] showing that PH is included in P^{PP} follows directly from several lowness results: First Toda extends the randomized reduction from SAT to the set of formulas with at most one satisfying assignment from Valiant and Vazirani [41] (which can be written as $\mathsf{NP} \subseteq \mathsf{BPP}(\oplus\mathsf{P})$) to a lowness result stating that NP is low for $\mathsf{BPP}(\oplus\mathsf{P})$. This implies that the whole polynomial time hierarchy is included in $\mathsf{BPP}(\oplus\mathsf{P})$. Then Toda shows that $\oplus\mathsf{P}$ is low for P^{PP}. Since BPP is also low for PP [26] we have that in fact PH is low for P^{PP}.

The body of results inspired by the concept of lowness is quite large. We refer the interested reader to the exhaustive overview of the state of the art regarding lowness at the time, written by the first author in [25]. Twenty years later, we review in this column some of the lowness results that have been published since then. If originally the lowness result dealt mostly with the polynomial time hierarchy, most of the later results are related to probabilistic or to counting complexity classes. We use this fact for the organisation of the overview in two main sections. To improve readability, we do not include full proofs and just sketch the

ideas behind the main results. We also skip the definitions of the better known complexity classes. For this, we refer the interested reader to the books in the area, like [7, 34, 1]. Depending on the context, a computational model C with access to an oracle A, sometimes will be denoted by $C(A)$ and sometimes by C^A.

2 Probabilistic complexity classes

As already explained in the introduction, lowness properties provide a meaningful explanation for many conditional collapse results. An important example is the Karp-Lipton Theorem [24] which says that NP is not contained in P/poly unless PH collapses to its second level Σ_2^P. The lowness property implying this theorem was revealed by Balczar, Book and Sch¶ning [6] who proved that all self-reducible sets in P/poly are low for the class Σ_2^P. In [29], the lowness of self-reducible sets in P/poly has been improved to the probabilistic class ZPP(NP) (we note that it is not hard to show that $\mathsf{Low}(\mathsf{ZPP}(\mathsf{NP})) \subseteq \mathsf{Low}(\Sigma_2^P)$). There are many different types of self-reducibility. For our purpose, the following one (called also Turing-self-reducibility) suits best. A language L is *self-reducible* if there is a polynomial-time oracle machine M such that $L = L(M^L)$ and M on input x asks only queries y whose length is smaller than that of x.

Theorem 1. [29] *Every self-reducible set* $L \in \mathsf{P/poly}$ *is low for* $\mathsf{ZPP(NP)}$.

The idea behind the proof is to compute a collection of polynomially many advice strings such that the majority of these strings correctly decides all oracle queries to L in a given $\mathsf{ZPP}^{\mathsf{NP}^L}$ computation on input x. To be more specific, let A be a set in P such that for some polynomial p and all m there exists a string $w \in \{0,1\}^{p(m)}$ with the property that for all strings y of length at most m,

$$y \in L \quad \Leftrightarrow \quad \langle y, w \rangle \in A.$$

Since L is self-reducible, with the help of an NP oracle it is possible to check for a given collection $W = (w_1, \ldots, w_k)$ of advice strings whether their majority gives the correct answer to all strings y of length up to m. Moreover, in case there exists a *counterexample* y for which the majority of W gives the wrong answer, such a string y can also be efficiently computed by asking an NP oracle. In the latter case, the string y together with the information whether it belongs to L or not is stored in a set S of all counterexamples encountered so far.

More precisely, the $\mathsf{ZPP}^{\mathsf{NP}^L}$ computation on a given input x is simulated as follows by a ZPP(NP) algorithm M. Let m be large enough such that the length of all queries to L is bounded by m. Then, starting with the empty set $S = \emptyset$, M repeatedly samples a collection $W = (w_1, \ldots, w_k)$ of advice strings, where each

w_i is chosen uniformly at random from the set *Correct*(S) of all advice strings $w \in \{0,1\}^{p(m)}$ that for all strings $y \in S$ correctly decide their membership to L. As long as there is a counterexample y for W, M adds it to S and repeats the loop. Otherwise M uses the collection W sampled in the last round to simulate the $\mathsf{ZPP}^{\mathsf{NP}^L}$ computation by answering all oracle queries to L according to the majority vote of W.

It remains to argue that M is indeed a ZPP(NP) algorithm. Call a string y *bad* for S if including it into S does not reduce the size of *Correct*(S) by a factor less than $3/4$. The important observation is that a string y can only be bad for S if at least $3/4$ of all advice strings in *Correct*(S) decide y correctly. Hence, the probability that there exists a bad string y for which the majority of $w_1, \ldots, w_k$ takes the wrong decision, becomes exponentially small when k is chosen large enough. Since adding a counterexample shrinks the size of *Correct*(S) in each round with high probability by a factor less than $3/4$, the expected number of rounds is polynomially bounded.

The ZPP(NP) algorithm described in the preceding paragraph is very similar to the ZPP algorithm of [10] for learning boolean circuits by using equivalence queries and the help of an NP oracle. A subtle point in both algorithms is how the sampling from the set *Correct*(S) can be implemented. At the time when Bshouty et al. presented their learning algorithm, it was only known [22] how to achieve *almost* uniform sampling from an NP witness set with the help of an NP oracle (which suffices for this application, although the analysis is more complicated). In the meanwhile, also uniform sampling from an NP witness set can be performed with an NP oracle [8]. On the other hand, the proof given in [29] neither uses uniform nor almost uniform sampling from an NP witness set.

As observed by Sengupta (see [11]), the proof of the Karp-Lipton Theorem proposed by Hopcroft [21] actually shows a collapse of PH to the symmetric alternation class S_2^P. This class was introduced in [13, 35] and shown to lie between the two classes P^{NP} and $\Sigma_2^P \cap \Pi_2^P$. Later, Cai [11] showed that S_2^P is in fact a subclass of ZPP(NP) and hence the collapse of PH to S_2^P might be deeper than to ZPP(NP). This raised the question whether all self-reducible sets in P/poly are even low for the class S_2^P, which was answered affirmatively in [12]. Later, it was proved by Chakaravarthy and Roy [14] that self-reducible sets in P/poly are also low for the oblivious symmetric alternation class $\mathsf{O}_2^P \subseteq \mathsf{S}_2^P$ which led to a further improvement of the Karp-Lipton Theorem.

Another interesting class of problems that are low for ZPP(NP) is AM∩coAM. The class AM consists of all languages that have a two-round interactive proof system with public coins and has been introduced by Babai; see [4, 5]. We state the definition of the class AM with one-sided error which is known to be equivalent to AM with two-sided error.

Definition 2. *A language L is in* AM *if there is a set $A \in$ NP and a polynomial p such that for every string x and a randomly chosen string $y \in_R \{0,1\}^{p(|x|)}$,*

$$\begin{aligned} x \in L &\Rightarrow \text{Prob}[\langle x, y\rangle \in A] = 1, \\ x \notin L &\Rightarrow \text{Prob}[\langle x, y\rangle \in A] \leq 1/2. \end{aligned}$$

The class AM ∩ coAM plays an important role in the context of classifying several group-theoretic problems. Schöning proved [36] that all sets in AM∩coAM are low for Σ_2^p. Since Graph Isomorphism also belongs to this class [20, 19] it follows that GI cannot be NP-complete unless the polynomial hierarchy collapses [9]. Later it was shown that AM ∩ coAM is also low for the classes AM [28] and ZPP(NP) [3].

Theorem 2. [3] AM ∩ coAM *is low for* ZPP(NP).

The proof of this lowness result provides a way to decide queries to an AM ∩ coAM oracle by using a random string in a coNP set as advice for an NP ∩ coNP computation. More specifically, let $L \in$ AM ∩ coAM be the oracle in a given ZPP^{NP^L} computation. By applying standard probability amplification techniques (cf. [36]) we can assume that there are NP sets A and B such that for some polynomial p and all m there exists a subset $S_m \subseteq \{0,1\}^{p(m)}$ of size at least $2^{p(m)-1}$ with the property that for all strings y of length at most m,

$$\begin{aligned} y \in L &\Rightarrow \forall w : \langle y, w\rangle \in A \text{ and } \forall w \in S_m : \langle y, w\rangle \notin B, \\ y \notin L &\Rightarrow \forall w : \langle y, w\rangle \in B \text{ and } \forall w \in S_m : \langle y, w\rangle \notin A. \end{aligned}$$

Call a string $w \in \{0,1\}^{p(m)}$ *bad*, if it fulfills the following property:

$$\exists y \in \{0,1\}^{\leq m} : \langle y, w\rangle \in A \cap B.$$

Since S_m does not contain bad strings, most strings in $\{0,1\}^{p(m)}$ are good. Now, in order to simulate the ZPP^{NP^L} computation on a given input x by a ZPP oracle machine M with the help of an NP oracle, let m be large enough such that all the queries to L have length at most m. To decide x, M first chooses a random string $w \in \{0,1\}^{p(m)}$ and uses its oracle to check that it is not bad. Using such a w as advice, M can replace each query to the oracle L with an NP ∩ coNP computation.

As a consequence we get the following inclusions between lowness classes:

$$\text{NP}\cap\text{coNP} = \text{Low(NP)} \subseteq \text{AM}\cap\text{coAM} = \text{Low(AM)} \subseteq \text{Low(ZPP(NP))} \subseteq \text{Low}(\Sigma_2^p).$$

3 Counting classes

Extending the definition of Valiant's #P counting functions, Fenner et al. [17] defined the class GapP of functions that express the difference between accepting and rejecting computations of a nondeterministic Turing machine.

Definition 3. *For a nondeterministic machine M and an input x, let $acc_M(x)$ ($rej_M(x)$) be the number of accepting (rejecting) paths of M on input x. A function $f : \Sigma^* \to \mathbb{Z}$ belongs to the class* GapP *if there is a polynomial-time nondeterministic Turing machine M such that for every input $x \in \Sigma^*$,*

$$f(x) = \mathrm{acc}_M(x) - \mathrm{rej}_M(x).$$

The class of GapP functions allows us to define many counting complexity classes in a uniform way. For example, PP is the class of languages L for which there is a GapP function f such that for every x, $x \in L \Leftrightarrow f(x) > 0$, and $L \in \oplus\mathsf{P}$ if there is a function $f \in$ GapP such that for every x, $x \in L \Leftrightarrow f(x)$ is odd. Also, most of the lowness results for counting classes use in their proofs the GapP function machinery.

We already mentioned that $\oplus\mathsf{P}$ is low for itself. Regarding PP there are several classes of problems, like for example, BPP or the class of sparse sets in NP, that are known to be low for this class [26]. Also the Graph Isomorphism problem is low for PP [27]. It is interesting to observe that all these examples are also low for Σ_2^p. As we will see, there are however some other interesting classes of problems that are low for PP and not known to be low for any class in the polynomial time hierarchy. These results can be best explained introducing some new counting classes.

Definition 4. [17] *A language L is in* SPP *if there is a* GapP *function f satisfying that for every x,*

$$\begin{aligned} x \in L &\Rightarrow f(x) = 1 \\ x \notin L &\Rightarrow f(x) = 0 \end{aligned}$$

A language L is in LWPP *if there are a* GapP *function f and polynomial time computable function g such that $0 \notin \mathrm{range}(g)$ and for every x*

$$\begin{aligned} x \in L &\Rightarrow f(x) = g(1^{|x|}) \\ x \notin L &\Rightarrow f(x) = 0 \end{aligned}$$

In other words, the languages in SPP and LWPP can be computed by nondeterministic polynomial time machines in which the difference between accepting

and rejecting configurations in the cases of $x \in L$ and $x \notin L$ are known. Clearly $\mathsf{SPP} \subseteq \mathsf{LWPP}$.

Fenner et al. [17] proved that SPP is low for PP and that in fact SPP is low for itself, $\mathsf{SPP}(\mathsf{SPP}) = \mathsf{SPP}$. They also show (citing a private communication from Toda) that LWPP is low for PP. A method to prove that a problem is low for PP, based in these results, is to show that it is included in LWPP. This is exactly what was done in [27] with Graph Isomorphism and some other related group problems. Carefully computing a generator set for the automorphism group of a graph by making controlled queries to GI, the authors proved that GI is contained in LWPP, while the seemingly easier problem to decide whether a graph has some non-trivial automorphism, Graph Automorphism, is contained in SPP. Arvind and Kurur [2] improved the result for GI proving that the problem belongs to the tighter class SPP.

Theorem 3. [2] GI *belongs to* SPP.

A consequence of this is that GI is also low for $\oplus\mathsf{P}$. They extended the result to the Hidden Subgroup problem (HSP) which plays an important role in quantum computation.

Definition 5. HSP: *On input a permutation group G (given by a generator set), and a function $f : G \to F$ for some finite set F, (f given in the form of a "black box") with the property that there exists a subgroup $H < G$ such that f is constant in the right cosets defined by H and different in the distinct right cosets of H, find a generator set for the hidden subgroup H.*

It is not hard to see that the problem of finding a generator set for the automorphism group of a given graph (a problem that is hard for Graph Isomorphism) is a particular case of HSP. For a graph X with n vertices, the group G is in this case the set S_n of permutations acting on $\{1, \ldots, n\}$. F is the set of graphs with n vertices and the function $f : S_n \to F$ is defined as $f(\pi) = \pi(X)$. If H is the automorphism group of X, then clearly f is constant and distinct on the different cosets defined by H.

Observe that HSP is a functional problem and therefore it cannot be included in a class of decisional problems like SPP. Arvind and Kurur show that HSP can be computed in polynomial time by an algorithm making queries to SPP.

HSP has received much attention because of its relation to quantum computation. BQP is the class of problems that can be solved in polynomial time with a quantum computer model with bounded error probability. Several problems proven to be in BQP but not known to be in P, are concrete instances of HSP. In fact it is known that the version of HSP with the additional condition that the given group G is commutative, can be solved in BQP [23]. Unfortunately, for the case

of GI, G is the group of all permutations, which is not commutative. Nevertheless the fact that GI is an instance of HSP gives some hope to find an efficient quantum algorithm for the graph isomorphism problem.

HSP being low for PP is not the only connection between lowness and quantum computation. Fortnow and Rogers [18] proved that the whole class BQP is low for PP. This is done by proving that BQP is included in yet another counting class, AWPP, introduced in [39]. AWPP is low for PP as was shown in [31]. AWPP is the best upper bound known for BQP so far. We give here a definition of this class, more elegant than the original, due to Fenner [16]:

Definition 6. *A language L is in* AWPP *if there is a* GapP *function f and a polynomial p satisfying that for every x,*

$$\begin{aligned} x \in L &\Rightarrow \frac{2}{3} \leq \frac{f(x)}{2^{p(|x|)}} \leq 1, \\ x \notin L &\Rightarrow 0 \leq \frac{f(x)}{2^{p(|x|)}} \leq \frac{1}{3}. \end{aligned}$$

It follows directly from the definition that AWPP generalizes both SPP and BPP.

Theorem 4. [18] BQP *is low for* PP.

This result is proven by showing that the probability of acceptance of a polynomial time quantum machine M on an input x can be exactly computed using GapP functions. For this the quantum machine is normalized so that it has a unique accepting computation and uses only unitary transformations from a finite basis set. As a consequence, the amplitudes of the quantum states during the computation range only over a finite set of values. The graph of all possible reachable configurations of $M(x)$ can be exponentially large, but the probability of reaching a particular configuration can be computed from the set of computation paths that can reach this configuration and the probability of the machine taking one of this paths. This can all be computed with GapP functions.

A different lowness result for the class PP comes from the area of games with unique winning strategies. Generalizing the class UP of NP problems with at most one accepting path, Niedermeier and Rossmanith [32] introduced an unambiguous version of an alternating Turing machine and defined the class UAP.

Definition 7. *An alternating Turing machine is unambiguous if every accepting existential configuration has exactly one move to an accepting configuration, and every rejecting universal configuration has exactly one move to a rejecting configuration. The class* UAP *consists of all languages accepted by polynomial-time unambiguous alternating machines.*

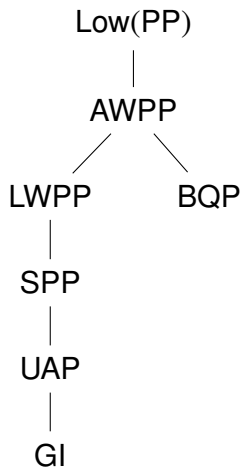

Figure 1: Inclusion structure of classes known to be low for PP.

In [32] it is shown that UAP is contained in SPP and therefore low for PP. Later it was proved [15] that UAP is low for itself, UAP(UAP) = UAP, and also that Graph Isomorphism is contained in UAP. This improves the SPP upper bound for GI proven in [2].

The inclusion relations among the low for PP complexity classes discussed here can be seen in the diagram of Figure 1.

We have cited several results proving that some complexity class or problem is low for PP. Is there an exact characterization of the class of sets that are low for PP? Does this class coincide with AWPP? SPP and UAP contain GI and are both low for themselves, an interesting question is whether these two classes coincide.

References

[1] S. Arora and B. Barak. *Computational complexity*. Cambridge University Press, 2009.

[2] V. Arvind and P. P. Kurur. Graph isomorphism is in SPP. *Information and Computation*, 204(5):835–852, 2006.

[3] V. Arvind and J. Köbler. New lowness results for ZPP^{NP} and other complexity classes. *Journal of Computer and System Sciences*, 65(2):257–277, 2002.

[4] L. Babai. Trading group theory for randomness. In *Proceedings of 17th Annual ACM Symposium on Theory of Computing (STOC)*, pages 421–429, 1985.

[5] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[6] J. L. Balcázar, R. Book, and U. Schöning. Sparse sets, lowness and highness. *SIAM Journal on Computing*, 23:679–688, 1986.

[7] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. EATCS Monographs on Theoretical Computer Science. Springer, second edition, 1995.

[8] M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of NP-witnesses using an NP-oracle. *Information and Computation*, 163:510–526, 2000.

[9] R. B. Boppana, J. Hstad, and S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.

[10] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52:421–433, 1996.

[11] J.-Y. Cai. $S_2^p \subseteq ZPP^{NP}$. In *Proceedings of 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 620–628, 2001.

[12] J.-Y. Cai, V. T. Chakaravarthy, L. A. Hemaspaandra, and M. Ogihara. Competing provers yield improved Karp-Lipton collapse results. *Information and Computation*, 198(1):1–23, 2005.

[13] R. Canetti. More on BPP and the polynomial-time hierarchy. *Information Processing Letters*, 57(5):237–241, 1996.

[14] V. T. Chakaravarthy and S. Roy. Oblivious symmetric alternation. In *Proceedings of 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, number 3884 in LNCS, pages 230–241, Berlin, 2006. Springer.

[15] M. Crasmaru, C. Glaer, K. W. Regan, and S. Sengupta. A protocol for serializing unique strategies. In *Proceedings of 29th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, number 3153 in LNCS, pages 660–672. Springer, 2004.

[16] S. Fenner. PP-lowness and a simple definition of AWPP. *Theory of Computing Systems*, 36:199–212, 2003.

[17] S. A. Fenner, L. Fortnow, and S. A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[18] L. Fortnow and J. Rogers. Complexity limitations on quantum computation. *Journal of Computer and System Sciences*, 59(2):240–252, 1999.

[19] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991.

[20] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 73–90. JAI Press, 1989.

[21] J. E. Hopcroft. Recent directions in algorithmic research. In P. Deussen, editor, *Proceedings 5th Conference on Theoretical Computer Science*, volume 104 of *LNCS*, pages 123–134. Springer, 1981.

[22] M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[23] R. Jozsa. Quantum factoring, discrete logarithms and the hidden subgroup problem. *Journal of Computing in Science and Engineering*, 3:34–43, 2001.

[24] R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of 12th Annual ACM Symposium on Theory of Computing (STOC)*, pages 302–309, 1980.

[25] J. Köbler. On the structure of low sets. In *Proceedings of 8th IEEE Structure in Complexity Theory Conference (Structures)*, pages 246–261. IEEE Computer Society Press, 1993.

[26] J. Köbler, U. Schöning, S. Toda, and J. Torín. Turing machines with few accepting computations and low sets for PP. *Journal of Computer and System Sciences*, 44(2):272–286, 1992.

[27] J. Köbler, U. Schöning, and J. Torín. Graph isomorphism is low for PP. *Computational Complexity*, 2(4):301–330, 1992.

[28] J. Köbler, U. Schöning, and J. Torín. *The graph isomorphism problem:*. Progress in Theoretical Computer Science. Birkhuser, Boston and others, 1993.

[29] J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM Journal on Computing*, 28(1):311–324, 1998.

[30] M. Lerman. *Degrees of Unsolvability*. Springer, 1983.

[31] L. Li. *On the counting functions*. PhD thesis, University of Chicago, 1993.

[32] R. Niedermeier and P. Rossmanith. Unambiguous computations and locally definable acceptance types. *Theoretical Computer Science*, 194:137–161, 1998.

[33] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI-Conference on Theoretical Computer Science*, volume 145 of *LNCS*, pages 269–276. Springer, 1983.

[34] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Mass., 1995.

[35] A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998.

[36] U. Schöning. Probabilistic complexity classes and lowness. *Journal of Computer and System Sciences*, 39:84–100, 1989.

[37] U. Schöning. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, 27:14–28, 1983.

[38] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):321–323, 1988.

[39] S. K. Stephen Fenner, Lance Fortnow and L. Li. An oracles builder's toolkit. *Journal of Information and Computation*, 182:95–136, 2003.

[40] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[41] L. G. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.