

Research Statement

Joshua Buresh-Oppenheim

December 9, 2007

In this time of extremely large data sets that come from, say, the internet or biological research, the need for efficient algorithms is particularly important. Of course this means that we need very talented people working on algorithms for individual problems, but these conditions also underscore the need for a theory of algorithms. The era of simple, elegant and efficient algorithms is largely over; modern algorithms often use very powerful machinery, are sometimes highly heuristic, and may guarantee only approximately optimal solutions. All the more reason to continue to introduce science into the art and engineering of algorithm design, so that we can negotiate this expanded view of what constitutes a viable algorithm.

The first goal of such a theory should be to describe which problems can be solved by efficient algorithms, and, therefore, which cannot. Unfortunately there is currently a great divide between our ability to design algorithms and our ability to explain the limitations of algorithms. Part of this state of affairs is due to the inherent difficulty of the question, but another part is surely due to methodology. For one thing, the two sides of the coin have been decoupled and relegated to two different research communities: algorithm analysis and complexity theory. Not surprisingly, these communities have diverged and lost a lot of their compatibility. My version of the story goes a little like this: in the beginning, there were simple algorithms and lots of real-world problems people couldn't solve efficiently. Elegant complexity classes, such as P and NP, were defined, and it seemed reasonable to show that there were natural problems that could not be computed in polynomial time. Many people began working on this, but it eventually became clear that it would be quite some time before a satisfactory conclusion could be reached. In the meantime, complexity theorists began defining natural subclasses of "efficient computation" and achieved some impressive lower bounds for these subclasses. At the same time, however, other people were designing more and more sophisticated algorithms. The problem is that the subclasses that complexity theorists defined that are susceptible to lower bounds bear little resemblance to the modes of computation that algorithms people were using.

One way to reunify the communities is suggested by the way we learn algorithms. Almost every introductory course on algorithms divides the world up into algorithmic paradigms such as greedy algorithms, dynamic programming, network flow, linear and semidefinite programming, etc. And in fact most real-world algorithms fall into one, or maybe a combination of two, of these paradigms. Therefore, instead of starting with arbitrary efficient computation and restricting it, why not formalize modes of computation around these algorithmic paradigms, engendering a sort of *algorithmic complexity theory*? Not only would this give new hope for our first goal, in the sense that we might be able to prove that, say, satisfiability cannot be computed efficiently by the types of algorithms that people actually use, but it could also allow practitioners of algorithm design to find methods for tractible problems more quickly and easily. More precisely, it could (1) guide the creation of new algorithms by making the ingredients of each algorithmic paradigm clearer; and (2) allow the algorithm designer to determine when a certain paradigm will not work on a given problem by

enabling lower bound proofs and impossibility results.

In what follows, I will try to use my research and related research to illustrate the benefits of this idea of algorithmic complexity theory. In fact, just like the introductory algorithms course, I will proceed by paradigm, describing formal models and resulting upper bounds (including new, useful algorithms that arose from this theory) and lower bounds in the realms of (1) linear and semidefinite programming, (2) greedy algorithms and dynamic programming, and (3) backtracking algorithms. Finally, I will conclude with a separate line of my research that illustrates how lower bounds for traditional complexity classes can lead to useful algorithms.

Linear and Semidefinite Programming

Linear and semidefinite programming constitute a sophisticated and extremely successful algorithmic technique. For simplicity, I'll describe the basic framework for linear programming (for semi-definite it is similar, but more technical). The solutions to an optimization problem can often be described as integer points in, say, n dimensions that satisfy a set of linear inequalities and that maximize a linear optimization function. To try to find such a solution, we can relax the constraint that the points be integer-valued and find the best valid real-valued point, which can be done using well-known efficient algorithms (the simplex method, for example). The question, then, is how well, and in what sense, this best real-valued point approximates the best integer-valued point. For one thing, the value that the real-valued point achieves under the optimization function may be higher than the value the integer point achieves, but how much higher? For the sake of a good approximation to the value of the optimal solution, we hope to minimize this gap. One recourse the algorithm designer has in this setup is to add extra linear inequalities that hold for every valid integer-valued point, but not for every valid real-valued point. Finding such useful inequalities sometimes seems like an art, rather than a science, however. Fortunately, Lovász and Schrijver [29], Sherali and Adams [33] and Lasserre [26] (among others) have described general-purpose, systematic procedures for introducing such extra inequalities, thereby formalizing, in a sense, subclasses of linear and semi-definite programming. I'll focus primarily on the first of these systems and make a few comments about the others.

Lovász and Schrijver used their system to create a new algorithm for finding the maximum independent set in t -perfect graphs, a generalization of perfect graphs. More spectacularly, however, as [1] points out, two of the most important approximation algorithms in recent years, those for Maxcut [21] and Sparsest-cut [6], can be seen as applications of the LS system. Historically, these algorithms were discovered independently of LS, but the question is clear: what other major new algorithms can be discovered by applying the LS system? Or, put the opposite way, what problems cannot be solved efficiently by such a powerful system? While these are really the same question, partial progress tends to fall into either the upper or lower bound categories. I'll describe some strong advances in the latter category that I helped establish. As for the former, I think this is one of the most promising directions in algorithmic design and that the time is ripe, both in terms of motivation and accessibility, for new results. [18] has recently shown some nice new results here using the Lasserre hierarchy.

To find limitations on the LS system, we consider a very hard problem (satisfiability), but we also get a very strong result. To explain it, though, I'll need to introduce a few more details. All variants of the LS system described above operate in rounds. That is, given a polytope that is not the convex hull of the integer points it contains, one round of LS produces a tighter polytope that still contains the same integer points. After n rounds (n is the number of dimensions), the integral hull is achieved. The magic of the system is that, assuming we could optimize over the initial polytope in polynomial time, then after r rounds, we can still optimize over the resulting polytope in time $n^{O(r)}$. The maxcut and sparsest-cut SDP's are implied by,

respectively, one and three rounds of the strongest of the LS systems: LS^+ . In [12], we show that for a random 5CNF formula, it is impossible for LS^+ to achieve any non-trivial approximation to the size of a maximum satisfiable subset of the formula even with linearly many rounds. This lower bound is saying that a wide class of algorithms that is one of our best hopes for approximating other NP-hard optimization problems cannot achieve either heuristic or approximate success here in any subexponential time! This result is one of the first of its kind (after [5]), and is the first exponential lower bound for an interesting optimization problem. Subsequently there has been a lot of interesting work in this area (see, for example, [1, 32, 31, 20, 19, 17]).

One of my current research projects focuses on lower bounds for approximating the vertex expansion of graphs using these systems. Vertex expansion is a particularly intriguing property of graphs in that it can serve as both a boon, such as in the construction of superconcentrators, which are useful for distributed networks, and as an obstacle (indeed, our lower bound in [12] relies on the vertex expansion of graphs representing the hard CNFs). Therefore, determining whether a graph has a certain measure of vertex expansion is of great interest and it is one of many such problems for which more general hardness of approximation results are not known.

Greedy Algorithms and Dynamic Programming

Greedy algorithms and dynamic programming are two of the most fundamental algorithmic techniques we know. While simple and classical, these techniques are ubiquitous in science and industry. But what exactly are they? They are usually taught by example and so far I think most people¹ would view them the way Supreme Court Justice Potter Stewart viewed obscenity: I can't define it, but I know it when I see it. Borodin, Nielsen and Rackoff [9] define a syntactic model that captures many algorithms which are generally considered to be "greedy algorithms." To illustrate the model, consider the very simple greedy algorithm for unit-profit interval scheduling: given n intervals in, say, $[0, 1]$ given by start time and finish time, find a maximum subset of non-overlapping intervals. The algorithm sorts the input intervals by non-decreasing finish time and then goes through one-by-one, accepting an interval into the subset if and only if it does not overlap with the previously accepted intervals. To model this process, imagine the algorithm is solving an arbitrary problem where the input is broken up into items, each of which describes a small part of the input. The algorithm begins by ordering the items according to some general rule (like non-decreasing finishing time in the case of intervals) and then it steps through the items in this order and makes an irrevocable decision (accept or reject in the case of intervals) about each one based on the decisions it has made about previous ones. This assignment of decisions to items defines the algorithm's output. In [2, 11], we extend this model to capture many algorithms generally considered to be dynamic programming algorithms. We show many lower bounds in these models which I'll describe and interpret momentarily, but here I'll try to illustrate how they can guide the search for new algorithms in a positive sense. For one thing, the model points out very clearly what is needed to construct an algorithm. In the case of greedy algorithms, you plug in an ordering and a decision-making procedure, and you have a greedy algorithm. Dynamic Programming can be broken down into similar simple requirements. This intuition guided us to find a new algorithm for 2SAT which fits into one of our models [2]. On the other hand, it is sometimes possible to move backwards from an algorithm in the model to a more intuitive dynamic programming understanding of the algorithm. In this way, the 2SAT result inspired my work in [13, 14], where we use dynamic programming (in the usual, informal sense) to give, as far as we know, the first polynomial-time algorithms for finding a minimum unsatisfiable subset and, more intricately, a minimum Resolution refutation of a set of 2CNF

¹Excluding, of course, those people who have worked on formalizing them before.

clauses. These algorithms contrast with known hardness results for finding a maximum satisfiable subset of a set of 2CNF clauses and a minimum unsatisfiable subset of a set of Horn clauses. Finally, the formalization of Borodin, Nielsen and Rackoff allowed [27] to search for new algorithms in a very interesting way: they begin with a greedy algorithm (in the formal sense) and define a distribution of algorithms which are defined as probabilistic modifications of the original algorithm. They show empirical evidence that this process can improve the original algorithm. I think that extending their methods to our stronger model of dynamic programming could yield a very powerful framework for finding algorithms.

One of the remarkable aspects about our model for dynamic programming is that it allows us to prove lower bounds on problems that are tractable for some other algorithmic technique, but do not seem to have dynamic programming solutions. First, however, a little more detail on the model. The way we extend the [9] model for greedy algorithms is by allowing an algorithm to branch on several decisions for an item, rather than just one. Hence an execution of an algorithm creates a tree whose width corresponds to the number of partial solutions we need to consider at any given time, or, to the size of the table in a dynamic programming solution to the problem. For example, the m -machine interval scheduling problem with proportional profit is traditionally solved using a dynamic programming algorithm with a table of size (and hence a running time of at least) n^m . This algorithm fits into our weakest model of dynamic programming with width $O(n^m)$. On the other hand, we can prove that no algorithm in this model can improve on this width, suggesting that all dynamic programming solutions to this problem require a table of size $O(n^m)$. There is also a more complicated, network-flow-based algorithm for the problem due to [4] that achieves a running time of $O(n^2 \log n)$ for constant m . So, this shift of algorithmic technique appears to be necessary for improved performance! Another such example is the problem of maximum bipartite matching. The classic Ford-Fulkerson result tells us that we can solve this problem by network flow (or by linear programming with zero rounds of LS). On the other hand, our strongest model of dynamic programming requires exponential width to solve the problem.

As an example of how these very issues can arise in interesting places, consider the bioinformatics problem of homology search: that is, given two genomes, find similar regions in them. Essentially, this is a version of the classic problem of computing edit distance, which, of course, has a very natural dynamic programming algorithm. The standard dynamic programming algorithm is much too slow, however, for homology search applications. Instead, people have moved to more heuristic methods. Wouldn't it be nice to know that we can't get the required performance out of any dynamic programming algorithm (or, even better, that we can)? The above model offers hope towards this goal.

Backtracking Algorithms

The last class of algorithms I'll discuss are backtracking algorithms, specifically as applied to the problem of satisfiability for formulas in conjunctive normal form. Such algorithms are often called DPLL algorithms (for Davis, Putnam, Logemann, Loveland). They are currently by far the most popular and useful algorithms for solving satisfiability and doing automated theorem proving in practice. In fact, they are so successful in the satisfiability domain, that a common approach to solving other NP-hard problems is to reduce them to SAT and use these algorithms to solve the resulting instance.² Intuitively, such algorithms work by selecting a variable of the formula, selecting a value (true or false) for that variable, and then recursing on the simplified formula where that variable is set to the chosen value. When the recursive call returns, if no

²At Simon Fraser University, David Mitchell and Eugenia Ternovska have initiated the MX-Project to harness the power of SAT algorithms to solve NP-hard search problems, as well as decision problems, in a seamless way by using reductions guaranteed by Fagin's Theorem. The emphasis here is to make it easy on the user by allowing her to model her problem at a very high level.

satisfying assignment has been found, the algorithm tries the other value for that variable.

When run on unsatisfiable CNF formulas, as is often the case in applications such as automated theorem proving, DPLL algorithms produce various types of Resolution proofs of the unsatisfiability of the formula. Hence, one can look at DPLL algorithms as algorithms for searching for Resolution proofs. There is a spectrum of restricted types of Resolution proofs, starting with tree-like Resolution (the weakest, and the one produced by the more classical DPLL algorithms), and ending with general Resolution. Of course there is a trade-off here: the more we restrict the form of the proof, the easier the search problem may become, but the longer the proofs will be. Abstracting DPLL algorithms to this level and analyzing the structure of tree-like Resolution proofs led Ben-Sasson and Wigderson [7] to an algorithm which finds a tree-like Resolution proof of an unsatisfiable 3CNF formula in time $S^{O(\log n)}$, where n is the number of variables in the formula and S is the length of its shortest tree-like Resolution proof (so, almost polynomial in S). Unfortunately, there are hardness results against searching for general Resolution proofs efficiently [3], but in [15] we analyze many restricted types of Resolution and conclude that some may be better to search for than others. In particular, we show that negative resolution, which is used in constraint-satisfaction solvers, is strictly stronger than tree-like Resolution, but strictly weaker than general Resolution. Hence, there may be an algorithm that searches for negative Resolution proofs that gets better performance than those which search for tree-like Resolution proofs. On the other hand, we show that linear Resolution proofs, a popular search strategy related to the implementation of Prolog, is just as hard to search for as general Resolution proofs, so it may be more beneficial to search for more restricted types of proofs.

In [2] we prove a lower bound on a large class of DPLL algorithms when run on *satisfiable* formulas (that is, our model for dynamic programming amazingly also captures many algorithms considered to be backtracking). Such a lower bound is rare because we cannot appeal to the Resolution proof machinery, and therefore it required introducing many new techniques. Of course, no one expected this class of algorithms, or any given class of algorithms, to solve every SAT problem efficiently, but another benefit of being able to prove lower bounds is that it allows us to identify, in a formal sense, exactly which instances of a problem are hard for a particular type of algorithm. In this case, the hard examples we use are efficiently solvable by Gaussian elimination.

From Lower Bounds to Upper Bounds

So far I have been arguing that this so-called algorithmic complexity theory may be more relevant to algorithm designers than traditional complexity theory. I also want to point out, again using my research and related research as examples, that lower bounds for traditional complexity classes are sometimes intertwined with the existence of efficient algorithms. In particular, sometimes proving that a certain problem is hard for a certain mode of computation implies that a different problem is easy for a different mode of computation. Also, sometimes the proof of a lower bound requires the introduction of new and useful algorithms.

As a first example, let us consider cryptography, which, I think it's fair to say, is an important subject for both computer scientists and civilians. Many essential cryptographic protocols (RSA, for example) are predicated on the existence of one-way functions. One-way functions, in turn, imply the existence of problems that are solvable in NP, but that are hard to solve in the average case for polynomial-time probabilistic algorithms. So, a lower bound against such algorithms would not immediately imply, but is a necessary step towards, truly viable cryptography. Of course, we do not even know if there is a problem in NP that is hard in the worst-case for probabilistic algorithms, but there has been a long line of research showing that if there is

a problem in NP that is mildly average-case hard, then there is one that is very average-case hard³. Such results are called *hardness amplification* results. In [10] we continue this line of research by giving an alternate proof of the best-known hardness amplification result against uniform algorithms, due to Trevisan [35, 36]. The advantage of our proof is that it makes clearer the role of error-correcting codes in such a result, which is good because error-correcting codes have proven extremely useful in this domain before (e.g. [34]) and seem to avoid certain inherent limitations in other techniques. This brings me to another algorithmic benefit of hardness amplification. Efficient and new constructions of error-correcting codes have been inspired by the problem of hardness amplification and the techniques that have been developed for it (e.g. [35]). In our case, we study how efficient codes can be when the encoding function is monotone; such codes may be of some independent interest. But it doesn't end with error correcting codes! One of the seminal papers in hardness amplification by Impagliazzo [24] features the discovery of a boosting algorithm which has had a major impact on machine learning [25].

The second example is the famous hardness-randomness tradeoff first noted by Blum and Micali, and Yao [8, 37] and exploited in many subsequent papers ([30, 23] to name just a couple). Very roughly, if there are families of boolean functions that can be computed even in exponential time by uniform algorithms, but which cannot be computed by small boolean circuits, then randomized algorithms can be derandomized. Many computer scientists challenge the importance of derandomizing algorithms when there are so many randomized algorithms in use that seem to work perfectly well and very efficiently. The reality remains, however, that generating bits which we think are close to random is expensive and often involves observing various random-looking natural processes. Typically, in practice we try to get by with bits that are generated by pseudorandom processes where the emphasis is on the *pseudo*; that is, there is no real justification of their randomness and certainly no reason to trust mission-critical executions of randomized algorithms to them. Another benefit of the hardness-randomness tradeoff is that, depending on how strong a lower bound we can prove against boolean circuits, this tradeoff could be very important for constructing random-looking combinatorial objects, such as optimal expander graphs, which have proven exceptionally useful in many aspects of computer science. Of course, proving lower bounds for explicit functions against boolean circuits is one of the major open questions in theoretical computer science. In [16], however, we look at the problem of improving circuit lower bounds. More specifically, if we are given a boolean function requiring circuits of size s , can we compress it efficiently into a smaller function that still requires circuits of size s ? Under certain conditions, we show that such a transformation can be achieved, although we also give strong evidence that it cannot be done in general. One of our algorithms for such a compression uses a new construction of a certain type of covering code, analogously to the way that hardness amplification uses constructions of error correcting codes.

References

- [1] M. Alekhnovich, S. Arora, and I. Turlakis. Towards strong approximability results in the Lovász-Schrijver hierarchy. In *FOCS: IEEE Symposium on Foundations of Computer Science*, 2005.
- [2] M. Alekhnovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi. Toward a Model for Backtracking and Dynamic Programming. In *Proceedings of the 20th IEEE Conference on Computational Complexity (CCC)*, 2005.

³There are too many impressive papers in this category for me to name. Let me just say that the first seminal result is Yao's XOR Lemma [37, 28, 22, 24]

- [3] M. Alekhovich and A. A. Razborov. Resolution is not automatizable unless W[P] is tractable. In IEEE, editor, *42nd IEEE Symposium on Foundations of Computer Science: proceedings: October 14–17, 2001, Las Vegas, Nevada, USA*, pages 210–219, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2001. IEEE Computer Society Press.
- [4] E. M. Arkin and E. L. Silverberg. Scheduling jobs with fixed start and end times. *Disc. Appl. Math*, 18:1–8, 1987.
- [5] S. Arora, B. Bollobás, L. Lovász, and I. Tourlakis. Proving integrality gaps without knowing the linear program. Preliminary version appeared in FOCS 2002, 2006.
- [6] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC: ACM Symposium on Theory of Computing*, 2004.
- [7] E. Ben-Sasson and A. Wigderson. Short proofs are narrow – Resolution made simple. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 517–526, Atlanta, GA, May 1999.
- [8] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.
- [9] A. Borodin, M. Nielsen, and C. Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37:295–326, 2003.
- [10] J. Buresh-Oppenheim, V. Kabanets, and R. Santhanam. Uniform hardness amplification for NP. Technical Report TR06–154, Electronic Colloquium on Computational Complexity (ECCC), 2006.
- [11] J. Buresh-Oppenheim, S. Davis, and R. Impagliazzo. Stronger models of dynamic programming algorithms. Manuscript in preparation, 2007.
- [12] J. Buresh-Oppenheim, N. Galesi, S. Hoory, A. Magen, and T. Pitassi. Rank bounds and integrality gaps for cutting planes procedures. *Theory of Computing*, 2:65–90, 2006. A preliminary version appeared in FOCS 2003.
- [13] J. Buresh-Oppenheim and D. Mitchell. Minimum witnesses for unsatisfiable 2CNFs. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2006.
- [14] J. Buresh-Oppenheim and D. Mitchell. Minimum 2CNF resolution refutations in polynomial time. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2007.
- [15] J. Buresh-Oppenheim and T. Pitassi. The relative complexity of resolution refinements. *Journal of Symbolic Logic*, 72(4), 2007. A preliminary version appeared in LICS 2003.
- [16] J. Buresh-Oppenheim and R. Santhanam. Making hard problems harder. In *Proceedings of the 21st IEEE Conference on Computational Complexity (CCC)*, 2006.
- [17] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Integrality gaps for sherali-adams relaxations. manuscript, 2007.

- [18] E. Chlamtac. Approximation algorithms using hierarchies of semidefinite programming relaxations. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007.
- [19] Wenceslas Fernandez de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of maxcut. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2007.
- [20] K. Georgiou, A. Magen, T. Pitassi, and I. Tourlakis. Tight integrality gaps for vertex cover SDPs in the lovasz-schrijver hierarchy. In *FOCS*, 2007.
- [21] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.
- [22] Goldreich, Nisan, and Wigderson. On yao’s XOR-lemma. In *ECCC’TR: Electronic Colloquium on Computational Complexity, technical reports*, 1995.
- [23] R. Impagliazzo and A. Wigderson. $P = BPP$ unless E has sub-exponential circuits. In *ACM Symposium on Theory of Computing*, 1997.
- [24] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *FOCS*, pages 538–545, 1995.
- [25] A. Klivans and R. Servedio. Boosting and hard-core sets. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.
- [26] Jean B. Lasserre. An explicit exact SDP relaxation for nonlinear 0 – 1 programs. *Lecture Notes in Computer Science*, 2081:293–303, 2001.
- [27] N. Lesh and M. Mitzenmacher. Bubblesearch: A simple heuristic for improving priority-based greedy algorithms. *IPL: Information Processing Letters*, 97, 2006.
- [28] L.A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.
- [29] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM J. Optimization*, 1(2):166–190, 1991.
- [30] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.
- [31] Grant Schoenebeck, Luca Trevisan, and Madhur Tulsiani. A linear round lower bound for lovasz-schrijver SDP relaxations of vertex cover. In *IEEE Conference on Computational Complexity*. IEEE Computer Society, 2007.
- [32] Grant Schoenebeck, Luca Trevisan, and Madhur Tulsiani. Tight integrality gaps for lovasz-schrijver LP relaxations of vertex cover and max cut. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, 2007.
- [33] H. D. Sherali and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3:411–430, 1990.

- [34] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma (extended abstract). In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1999.
- [35] L. Trevisan. List decoding using the XOR lemma. In *In proceedings of the 44th IEEE FOCS*, 2003.
- [36] L. Trevisan. On uniform amplification of hardness in NP. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2005.
- [37] A. C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, IL, November 1982. IEEE.