Two applications of pseudo-random graphs

Andrei Romashchenko, IITP of RAS (Moscow)

Abstract

We discuss two constructions based on pseudo-random graphs: a bit-probe scheme with one-sided error that provides a very compact encoding for small sets from a large universe, and an asymptotically optimal randomized communication protocol that synchronizes remote strings of bits with a small Hamming distance. We show that in both cases rather standard derandomization technique (e.g., Nisan's generator of pseudo-random bits) is enough to get surprisingly strong statements.

1 Introduction

A typical application of graphs in coding theory or computer science can be roughly described by the following abstract scheme. First, we prove that a random graph (from an explicitly defined family of graphs) with high probability enjoys some nice combinatorial property A (e.g., mixing, expansion, high connectivity, etc). Second, based on a randomly chosen graph we build some combinatorial gadget (e.g., a code, a Boolean circuit, a data structure, etc.). Third, we prove that this gadget satisfies some nontrivial properties B (corrects many errors, admits fast decoding procedure, etc.) if the graph used in the constriction satisfies combinatorial property A. From this three-steps argument it follows that there exists a gadget with property B; moreover, we obtain a randomized construction of such gadgets (taking a random graph, we obtain with high probability the required combinatorial gadget). In many applications we need something stronger: we want to get an 'explicit' (in some sense) construction of a gadget. To this end we should derandomize the scheme and provide an explicit construction of a graph that satisfies A.

A beautiful example of a success story of this general abstract scheme is the famous construction of expander codes [3]. The simplest construction of expanders graph exactly follows the scheme explained above. At the first step we notice that a randomly chosen bipartite graph (with an appropriate number of vertices and edges) is an *expander*. Then we use Tanner's construction of a linear code on a bipartite graph. At last, we prove that a code corresponding to some expander (a) has rather large code distance, and (b) there exists a fast error correcting algorithm. We obtain a family of asymptotically good codes with a fast decoding algorithm. These codes are not explicit since they are based on a random graph. To construct such a code effectively, we need an

explicit expander graph with appropriate parameters. This is the most difficult step of all the plan: the know explicit constructions of expanders still do not match the best bounds achievable for random graphs.

We want to stress the difference between two types of arguments involving graphs: some arguments involve a random graph (the constructed object has some nice properties with high probability), and some constructions involve an explicit graph (e.g., see applications of expander graphs in an excellent survey [5]). In this paper we consider an intermediate approach. We use graphs that are not explicitly constructed but are also not 'random' in the conventional sense. It is natural to call them *pseudo-random*. More precisely, we define a tiny family of graphs parameterized by a short parameter (a *seed*). A graph must be effectively constructed when its seed is given. Intuitively these graphs are outputs of a pseudo-random generator that maps a seed to a graph. The idea is to show that for most seeds the corresponding graph enjoys the desired property. Then we can fix a suitable value of the seed and make it a part of the construction (the gadget) based on the corresponding pseudo-random graph.

Actually the difference between 'truly random' and 'pseudo-random' graphs is vague and informal. We talk about 'truly random' family of graphs when we fix only some basic parameters of a graph (e.g., when we consider all graphs of fixed degree with n vertices). We talk about 'pseudo-random' graphs when we make some special efforts to define a "sparse" family of graphs, and these graphs are parameterized by some parameter with a rather small domain.

It is not surprising that in some cases a pseudo-random graph can achieve better bounds than any known explicit construction of graphs. We want to stress another advantage of pseudo-random constructions: in some sense pseudo-random graphs are even better than 'truly random' graphs. Indeed, a pseudo-random graph has a concise description. So, in some applications we can embed a pseudo-random graph (its seed) into a combinatorial gadget, whereas we cannot do the same with large 'truly random' graphs.

In this paper we consider two particular applications of pseudo-random graphs. The first example is a construction of bit-probe scheme that stores a small set S from a large universe, such that reading of a single bit from the database is enough to reply queries "$x \in S$?". We show that with pseudo-random expander graphs we can achieve better results than with any explicit or 'truly random' graphs. In our second example we use pseudo-random bipartite graphs to organize a protocol of synchronization of two remote bit strings with bounded Hamming distance. For this problem we obtain a randomized communication protocol with asymptotically optimal communication complexity (our protocol is a simplified version for the construction from [6]). Similarly to the first example, in this construction a pseudo-random graphs works better than any explicit or 'truly random' graphs.

2 Bit-probe schemes with a single bit query and one-sided error.

In this section we consider the static version of the membership problem suggested in [9]. We construct a data structure to store an n-elements subset S from a universe of size m. We obtain a data structure of size $O(n \log^2 m)$ that allows to query elements of the set with a small one-sided error. That is, for every $x \in S$ with probability 1 we return the answer 'yes', and for every $x \notin S$ with probability 0.99 we return the answer 'no'. We assume that the number of elements in the set $n = |S|$ is much less than size m of the universe (e.g., $n = m^{0.01}$).

We follow the ideas from [4]. We take a bipartite graph $G = (L, R, E)$ such that $|L| = m$ and $|R| = O(n \log^2 m)$. To encode a subset $S \subset L$ we assign to each vertex of R a bit 0 or 1; to answer query "$x \in S$?" we take a random neighbor of the vertex $x \in L$ and return the mark assigned to this vertex. A suitable distribution of marks on R (marks that guarantee small probability of faulty answers) exists provided that the graph G satisfies the following property:

Let $G = (L, R, E)$ be a bipartite graph, and $A \subset L$ be a subset of vertices from the left part. We say that the *ε-reduction property* holds for A in this graph if $|\Gamma(x) \cap \Gamma(A)| \leq \varepsilon d$ for all $x \in L \setminus A$.

This property is true for truly random graphs and even for "pseudo-random" graphs generated by several standard pseudo-random bit generators. So, we can take a suitable seed of a generator and make it a part of the data structure (it becomes the "cache memory" of the bit-probe scheme).

The choice of the graph G in this construction depends on S. The size of the seed (that specifies the choice of the graph) is $\mathrm{poly}(\log m)$. This construction is interesting only if the size of the seed (memory in cache) is much less than the size of the main storage, which is $O(n \log^2 m)$. We notice that a bit-probe scheme with one-sided error of size $O(n \log^2 m)$ cannot be obtained without cached memory (see [4]), so, pseudo-random graphs work here better than any explicit graph. On the other hand, schemes with a cached graph make no sense for 'truly random' graphs ('truly random' graphs are too large); so, pseudo-random graphs work better 'truly random' graphs.

It remains to explain how to choose a pseudo-random bits generator. Several known pseudo-random bits generators are suitable for our construction:

- The desired property of a graph can be tested by a AC^0-circuits. Hence, the generator of Nisan–Wigderson [2] can be used.
- M. Braverman proves that all polylog-independent functions fool AC^0-circuits [7]. Hence, we can take as a generator any $(\log^c n)$-independent function (for large enough constant c), e.g., a polynomial of degree $\log^c n$ (seeds of this 'pseudo-random generator' are coefficients of a polynomial).
- The property of strong ε-reduction can be tested by a Turing machine with logarithmic space. Hence, we can use Nisan's generator [1].

3 Synchronizing remote strings with bounded Hamming distance

In this section we discuss another application of pseudo-random combinatorial objects. We investigate the following communication complexity problem. Alice and Bob hold n-bits strings x and y respectively; they know that Hamming distance between x and y is bounded by αn. The goal is to communicate x to Bob. A simple information-theoretic bound implies that communication complexity of this problem (for deterministic and randomized communication protocols) is greater than $h(\alpha)n$, where $h(\alpha)$ is the binary entropy function. A. Orlitsky proved that there exists a *deterministic* communication protocol for this problem that asymptotically achieves this bound; however the known deterministic protocols require exponential calculations by Alice and Bob. A. Smith [6] constructed for this problem an asymptotically optimal *randomized* protocol with polynomial algorithms for both participants. The construction of Smith is a sophisticated combination of two ideas: Forney's codes and a complicated construction of a pseudo-random permutation. A simplified version of this argument is discussed in [10].

Probability of an error in the construction of Smith is exponentially small. Here we show that the problem can be solved by a much simpler protocol works for this problem if we allow a small *constant* probability of an error. The idea is to split all positions of strings of x and y into $m = cn/\log n$ blocks ($x_1, \ldots, x_m$ and $y_1, \ldots, y_m$ respectively). We can think of this spitting as a bipartite graph with n vertices in one part and m vertices in another part; each vertex in the first part should have degree one. For almost all random splittings of $\{1, \ldots, n\}$ into m groups we have two properties: (a) each group x_i, y_i is of size $\Theta(\log n)$; and (b) each pair of blocks x_i, y_i differ from each other in approximately $\alpha|x_i| = \alpha|y_i|$ bits. If both these properties hold, we can use the exponential-time (exponential in $\Theta(\log n)$) communication protocol and communicate x_i from Alice to Bob with the optimal communication complexity. All computations in this protocol work in time poly(n).

If Alice and Bob have a joint source of randomness, we are done: Alice and Bob randomly split all positions of x and y into m groups, and apply the known exponential protocol for each of the these groups of logarithmic size. In the model with private random bits the things are more complicated: Alice and Bob must choose somehow a common random splitting of positions $\{1, \ldots, n\}$ into m groups. The required property of a splitting is rather simple (it can be tested by a Final State Machine with logarithmic memory). Hence, we can use a *pseudo-random* splitting produced by Nisan's generator (Alice choses a random seed of the generator and communicates it to Bob).

Comparative to the construction in [6] we have (a) much simple algorithms of Alice and Bob; (b) much easier pseudo-random object to generate, but (c) a weaker bound for the probability of an error.

4 Conclusion

In this paper we discussed two applications of pseudo-random graphs. In both cases pseudo-random graphs cannot be substituted by any deterministic construction of graphs or by 'truly random' graphs. In both cases a naive application of classic generators of pseudo-random bits (e.g., Nisan's generator [1] and Nissan–Wigderson's [2] generators) work pretty well. Similar ideas of 'naive derandomization' were recently applied by D. Musatov and M. Zimand in problems of resource bounded Kolmogorov complexity [8, 11]. In all these examples a standard derandomization technique is enough to get rather strong statements. We believe that this method (perhaps, with a more advanced pseudo-random generators) can be applied in other problems of coding theory.

References

[1] N. Nisan. Pseudorandom generators for space-bounded computation. Combinatorica, 12(4), 1992, pp. 449–461.

[2] N. Nisan, A. Wigderson. Hardness vs Randomness. J. Comput. Syst. Sci. 49(2), 1994, pp. 149–167.

[3] D. Spielman and M. Sipser. Expander Codes. IEEE Transactions on Information Theory, Vol 42, No 6, 1996, pp. 1710–1722.

[4] Harry Buhrman, Peter Bro Miltersen, Jaikumar Radhakrishnan, Venkatesh Srinivasan. Are bitvectors optimal? Siam J. on Computing 31(6), 2002, pp. 1723–1744.

[5] S. Hoory, N. Linial, A. Wigderson. Expander graphs and their applications. Bulletin of the American Mathematical Society, 43(4) 2006, pp. 439–561.

[6] A. D. Smith. Scrambling Adversarial Errors Using Few Random Bits. SODA 2007.

[7] M. Braverman. Poly-logarithmic Independence Fools AC^0 Circuits. IEEE Conference on Computational Complexity 2009, pp. 3–8.

[8] D. Musatov. Theorems about space-bounded Kolmogorov complexity obtained by "naive" derandomization. In Proc. Computer Science in Russia, 2011. Preliminary version: arXiv:1009.5108 (2010).

[9] A. Romashchenko. Pseudo-random graphs and bit probe schemes with one-sided error. To appear in Proc. Computer Science in Russia, 2011.

[10] A. Chuklin. Effective protocols for low-distance file synchronization. arXiv:1102.4712 (2011).

[11] M. Zimand. On the optimal compression of sets in PSPACE. arXiv:1104.2816 (2011). To appear in IEEE Conference on Computational Complexity 2011.