Combinatorial PCPs with efficient verifiers

Or Meir

The original proof of the PCP theorem by Arora et al. (J. ACM 1998) was based on algebraic techniques. While those techniques are very important and useful, they seem to give little intuition as to why the theorem holds. In her seminal paper, Dinur (J. ACM 2007) gave an alternative proof of the PCP theorem using a combinatorial approach. Her proof is not only considerably simpler than the original proof, but also seems to shed more light on the intuitions that underly the theorem. However, her PCP construction is still inferior to the algebraic constructions in few aspects. We believe that it is important to try to come up with a combinatorial construction of PCPs that matches the algebraic constructions in those aspects, as this will hopefully advance our understanding of the PCP theorem.

In this work we present a combinatorial construction of PCPs that matches the algebraic constructions in the aspect of the running time of the PCP verifiers. In order to verify a claim, the verifier must run in time which is at least linear in the length of the *claim*, but the effect of the *proof length* on the verifier's running time may be much smaller. Using the algebraic techniques, one can construct PCP verifiers whose running time depends only poly-logarithmically on the proof length. On the other hand, the verifiers obtained from Dinur's proof of the PCP theorem are not as efficient, and their running time depends polynomially on the proof length. While this difference has relatively small effect in the context of standard PCPs for **NP**, it is very significant in two related setting: The first setting is the extension of the PCP theorem to **NEXP**, i.e., proving that $\mathbf{NEXP} = \mathbf{PCP}[\text{poly}(n), O(1)]$. The second setting is the construction of super-fast PCPs of Proximity (PCPPs), which are a variant of PCPs that allows verifying that a claim is *close to a valid claim* while running in time which is only poly-logarithmic in the length of the claim.

As mentioned, in this work we present a combinatorial construction of PCPs whose verifiers' running time is poly-logarithmic in the proof length, thus matching the verifiers of the algebraic constructions.

Overview of our construction

Dinur's construction of PCPs is an iterative construction: In order to construct a PCP for claims of size n, the construction performs $O(\log n)$ iterations. Each iteration increases the verifiers' running time by a constant factor, thus resulting in verifiers that run in polynomial time. The key to achieve poly-logarithmic running time is therefore reducing the number of iterations. To this end, we consider an earlier combinatorial construction of PCPs due to Dinur and Reingold (SICOMP 2006), to which we refer as the "DR construction". Like Dinur's constrution, the DR construction is iterative, but it performs only $O(\log\log n)$ iterations, as we desire. Unfortunately, the DR construction yields PCPs whose proof length is super-polynomial and whose verifiers run in time which is not poly-logarithmic. Thus, our focus is on strengthening the DR construction such that it yields PCPs that have proofs of polynomial length and verifiers that run in poly-logarithmic time.

Following Dinur and Reingold, it is more convinient to describe our construction in terms of "assignment testers" (ATs). Assignment testers are PCPPs that allow verifying that an assignment is close to a satisfying assignment of a given circuit. Any construction of ATs yields a construction of PCPs and PCPPs, and therefore our goal is to construct ATs whose running time is poly-logarithmic in the size of the given circuit.

The crux of the DR construction is a reduction that transforms an AT that acts on circuits of size k to an AT that acts on circuits of size k^c (for some constant $c > 0$). Using such a reduction, it is possible to construct an AT that works on circuits of size n by starting from an AT that works on circuits of constant size and applying the reduction for $O(\log \log n)$ times. However, the DR reduction also increases the proof length from ℓ to $\ell^{(1+\varepsilon)c}$ (for some constant $\varepsilon > 0$), which causes the final ATs to have proof length $n^{\mathrm{poly}\log n}$. Moreover, the reduction runs in time that is polynomial in the given circuit, rather than poly-logarithmic.

The main contribution of this work is showing how to modify the DR reduction so that it increases the proof length from ℓ to only $\tilde{O}(\ell^c)$ while increasing the running time of the AT by only a constant factor. Using the modified reduction indeed yields a construction with the desired parameters. Below we discuss the aspects of the proof length and running time separately, but of course, in the actual construction these aspects need to be combined.

The proof length A close examination of the DR reduction shows that its superfluous blow-up stems from two sources. The first source is the use of a "parallel repetition"-like error reduction technique, which yields a polynomial blow-up to the proof length. This blow-up can be easily reduced by using the more efficient error reduction technique that was developed in Dinur's proof of the PCP theorem.

The second source of the blow-up is the use of a particular circuit decomposition technique. The DR reduction uses a procedure that decomposes a circuit into an "equivalent" set of smaller circuits. This part of the reduction yields a blow-up that is determined by the parameters of the decomposition. The DR reduction uses a straightforward method of decomposition that incurs a polynomial blow-up. In our work, we present an alternative decomposition method that is based on packet-routing ideas and incurs a blow-up of only a poly-logarithmic factor, as required. We mention that in order to analyze our circuit decomposition method we prove a lemma on sorting networks, which may be of independent interest.

The running time For the rest of the discussion, it would be convinient to view the DR reduction as constructing a "big" AT that acts on "big" circuits from a "small" AT that acts on "small" circuits. The big AT works roughly by decomposing the given circuit to an equivalent set of smaller circuits, invoking the small AT on each of the smaller circuits, and combining the resulting residual tests in a sophisticated way. However, if we wish the big AT to run in time which is linear in the running time of the small AT, we can not afford invoking the small AT on each of the smaller circuits since the the number of those circuits is super-constant. We therefore modify the reduction so that it does not invoke the small AT on each of the smaller circuits, but rather invoke it once on the "universal circuit", and use this single invocation for testing the smaller circuits. When designed carefully, the modified reduction behaves like the original reduction, but has the desired poly-logarithmic running time.

In addition to the foregoing issue, we must also show that our decomposition method and Dinur's error reduction method have sufficiently efficient implementations. This is easily done for the decomposition. However, implementing Dinur's error reduction is non-trivial, and can be done only for PCPs that possess a certain property. The efficient implementation of Dinur's error reduction method is an additional contribution of our work.