

Lecture 24: PCP: Alphabet reduction.

Lecturer: Kristoffer Arnsfelt Hansen

Scribe: Kristoffer Arnsfelt Hansen

In this lecture we will complete the third step of the iterative procedure that collectively amplifies the gap of a 2CSP instance, keeping the alphabet size constant and only increasing the number of constraints by a fixed constant factor.

In the previous lecture we covered the gap amplification step. That step can greatly increase the gap, but as an inherent consequence of the construction it will also greatly increase the size of the alphabet.

We will next show how to bring down the alphabet size to a fixed constant size again at a small expense of decrease in the size of the gap. It is local construction on the constraint graph, replacing each edge by a number of constraints. This will involve an extension of the $\mathbf{PCP}(n^{O(1)}, O(1))$ verifier from lecture 21. It is useful to phrase this extension in the notion of PCP of proximity, PCPP (or assignment tester).

1 PCP of proximity

A PCP of proximity verifier has access to two different inputs x and y as well as random access to a written proof π . We denote the inputs as follows.

- The standard input x , with sequential access.
- The auxiliary input y , with random access

As parameters we have

- $r(n)$, the number of random bits the verifier uses.
- $q(n)$, the number of bits that are accesses in *both* y and π .
- δ , a proximity parameter.
- c , the completeness parameter.
- s , the soundness parameter.

Let L be a language. To say that the verifier V accepts L we require that it satisfies the following properties.

Completeness: If $(x, y) \in L$ then there exist a proof π such that

$$\Pr[V^\pi(x, y) \text{ accepts}] \geq c$$

Soundness: If (x, y) is such that for any $(x, y') \in L$ we have that y and y' are δ -far, then for *any* purported proof π ,

$$\Pr[V^\pi(x, y) \text{ accepts}] \leq s$$

The setting we will need is as follows. We have a 2CSP instance φ over an alphabet of size W . Let $k = \log W$. We encode the alphabet by bitstrings in $\{0, 1\}^k$. We think of the constraints as being computed by Boolean circuits on $2k$ inputs. These require circuits of size at most $l = 2^{2 \log W} = W^2$. The language we will construct a PCPP verifier for will be the language $L_{\text{AT}} = \{(C, (\text{WH}(u_1), \text{WH}(u_2))) \mid C(u_1, u_2) = 1\}$, where C is a Boolean circuit of size l with $2k$ inputs and $u = (u_1, u_2) \in \{0, 1\}^{2k}$.

Theorem 1 *There is a PCPP verifier for L_{AT} with completeness $c = 1$, soundness $s = 1/2$ and proximity parameter $\delta = 0.01$, using $O(k)$ random bits and makes $O(1)$ queries.*

Proof We will do a modification of the reduction from Circuit-SAT to QUADEQ. This reduction took a circuit C with l wires and transformed into an instance of QUADEQ with a variable for every wire. We will expect the written proof π to contain $\text{WH}(w)$ and $\text{WH}(w \otimes w)$, where w is the satisfying assignment to the QUADEQ instance corresponding to the evaluation of the circuit on input u . Let π_w and $\pi_{w \otimes w}$ be the part of the written proof we expect these. Note that the first $2k$ bits of w will be the assignment u to the input variables of the circuit C . We expect the auxillary input to be $\text{WH}(u_1)$ and $\text{WH}(u_2)$. Let π_{u_1} and π_{u_2} denote these auxillary inputs. We will do a linearity check to verify that these are 0.99 close to linear functions $g_1 : \{0, 1\}^k \rightarrow \{0, 1\}$ and $g_2 : \{0, 1\}^k \rightarrow \{0, 1\}$. In the QUADEQ verifier we have already established that π_w is 0.99-close to a linear function $f : \{0, 1\}^l \rightarrow \{0, 1\}$. We can evaluate these correctly with only access to π using local decoding.

After running the QUADEQ verifier all that is left is to test that u is a prefix of w . We do the following test 10 times, say. Pick $x_1, x_2 \in \{0, 1\}^k$ at random and test that

$$f(x \circ x_2 \circ 0^{l-2k}) = g_1(x_1) + g_2(x_2)$$

Note that $f(x_1 \circ x_2 \circ 0^{l-2k}) = \langle w, x_1 \circ x_2 \circ 0^{l-2k} \rangle = \langle w_{[1..2k]}, x_1 \circ x_2 \rangle$, $g_1(x_1) = \langle u_1, x_1 \rangle$ and $g_2(x_2) = \langle u_2, x_2 \rangle$. Hence if the u is not a prefix of w we will detect this with probability $1/2 - 0.1$ in each trial.

□

2 Alphabet reduction

With the construction of the PCPP verifier the alphabet reduction is now relatively straightforward. For every constraint in φ we construct the PCPP verifier given above. We convert this verifier into a CSP instance, by running over all possible random input strings. This gives us $2^{O(k)} = W^{O(1)}$ constraints of a fixed constant arity over the binary alphabet. We do this for every constraint in φ and let ψ be the union of all the produced constraints. Note that these will share variables whenever the constraints in φ share variables.

Clearly, if we can satisfy all constraints of φ we can also satisfy all constraints of ψ . Suppose now that $\text{val}(\varphi) \leq 1 - \epsilon$ and consider any assignment to the variables of ψ . For every variable u in φ we have a bitstring $U \in \{0, 1\}^W$ in φ . We decode the assignment to ψ in the following way: If U is 0.99-close to $\text{WH}(a)$ for some $a \in \{0, 1\}^{\log W}$ we assign the variable u the value a . Otherwise we give an arbitrary value. By assumption, this decoded assignment must violate an ϵ fraction of the constraints.

Consider now a given constraint in φ . If the decoding do not satisfy this then at least $1/2$ of the constraint generated by the PCPP verifier for this constraint are violated. This means that at least an $\epsilon/2$ fraction of the constraints are violated in ψ .

We still have one more step to do in order to be able to do the iterative procedure again. We no longer have a 2CSP instance but a q CSP instance for some fixed constant q . For the transformation back to a 2CSP we can use a simple local reduction. Assume that we have variables x_1, \dots, x_n with m q -ary constraints. For the 2CSP instance we add variables $y_1, \dots, y_m \in \{0, 1\}^{2^q}$. The old constraints becomes unary constraints on the variables y_1, \dots, y_m and we add binary consistency constraints as follows. Let $1 \leq i \leq m$ and assume the i th constraint depends on the variables x_{i_1}, \dots, x_{i_q} . Then add the q constraints “ $x_{i_j} = (y_i)_j$ ”, $j = 1, \dots, q$.

This final construction will decrease the gap by at most a factor $1/q$.