Lecture notes on OPP algorithms [Preliminary Draft]

Jesper Nederlof

June 13, 2016

These lecture notes were quickly assembled and probably contain many errors. Use at your own risk! Moreover, especially the exposition of Drucker's proof is probably still not detailed enough, and seemingly features a very rude oversimplification.

1 Brief introduction to OPP-algorithms

In this lecture we study randomized polynomial time algorithms for NP-complete problems in the worst-case setting. Of course we do not expect these algorithms to solve the problem completely: the catch is that we allow the probabilistic guarantees to be exponentially small.

Definition 1 (OPP)**.** *A* One-sided Probabilistic Polynomial time (OPP) algorithm *is a polynomial time algorithm with one-sided error probability. An algorithm* A solves a decision problem L with success probability $f(x)$ *if for every instance* $x \in \{0,1\}^n$

- $A(x) = NO$*, if* $x \notin L$*,*
- $\Pr[A(x) = YES] \geq f(x)$*, if* $x \in L$*.*

Two simple observations are

Observation 2. *Every problem in NP can be solved with success probability* $2^{-\texttt{poly}(|x|)}$*: simply guess the certificate.*

Observation 3. *If a problem can be solved with success probability* $f(x)$*, there is a Monte Carlo algorithm (e.g., randomized algorithm with one-sided constant error probability) solving the problem in* $\texttt{poly}(|x|)/f(x)$ *time algorithm: simply perform* $1/f(x)$ *independent runs of the OPP-algorithm and return YES whenever YES is returned in some run.*

The latter observation tells us that OPP-algorithms imply certain exponential time algorithms. We'll often call the exponential time algorithms one obtains in this way OPP-algorithms as well. Moreover, if one sees an exponential time algorithm running in time $f(x)\texttt{poly}(|x|)$ somewhere in the literature, chances are actually reasonable that it implies an OPP algorithm with success probability $\log_2(1/f(x))$. This is in particular true for many 'branching algorithms'. Take for example the following algorithm for the Independent Set problem:

Algorithm $\texttt{is}(G = (V, E), k)$
Output: Whether G has an independent set of size at least k.
1: **if** $\exists u \in V : \deg(u) \geq 2$ **then**
2: **return** $\texttt{is}(G[V \setminus u], k) \vee \texttt{is}(G[V \setminus N[u]], k - 1)$
3: **else**
4: Solve problem in polynomial time — Graph is matching plus isolated edges

Algorithm 1: $1.62^n\texttt{poly}(n)$ time algorithm for independent set.

Since in the recursive step we have two sub-problems in which $|V|$ is decreased by either 1 or 2, the running time $T(n)$ as a function of $|V|$ satisfies $T(n) \leq (T(n-1) + T(n-2))\texttt{poly}(n) \leq 1.62^n\texttt{poly}(n)$. We can rewrite this into a polynomial time algorithm that finds a independent set of size at least k with probability 1.62^{-k} if it exists as follows:

Algorithm $\texttt{is2}(G = (V, E), k)$
Output: Whether G has an independent set of size at least k.
1: **if** $\exists u \in V : \deg(u) \geq 2$ **then**
2: Pick $p \in \{0, 1\}$ with $\Pr[p = 1] = 1/1.61$, $\Pr[p = 0] = 1/1.61^2$.
3: **if** $p = 1$ **then return** $\texttt{is}(G[V \setminus u], k)$ **else return** $\texttt{is}(G[V \setminus N[u]], k - 1)$
4: **else**
5: Solve problem in polynomial time — Graph is matching plus isolated edges

Algorithm 2: Polynomial time algorithm finding an independnet set of size at least k with probability 1.61^{-n} if it exists.

Note that intuitively, this algorithm attempts to uniformly sample leaves of a hypothetical (according to the analysis) maximum-sized recursion tree of the precious algorithm. This observation is due to Eppstein. Let us remark that this rewriting applies to a surprisingly big amount of other branching algorithms, but if for example the $\vee$ in the recursive step of Algorithm 2 is replaced with a $\wedge$ it would be hard to mimick the behaviour of the algorithm in a similar way.

Moreover, also many not so typical 'branching algorithm' can also be 'simulated' by OPP algorithms. A nice other example being Schönings algorithm for k-SAT (see e.g. `http://www.win.tue.nl/~jnederlo/2MMD30/lecturenotes/L11/L11.pdf`). Some algorithm are even often stated as (repeated application of) OPP-algorithms, for example for finding a simple path on at least k-vertices:

Algorithm $\texttt{kpath}(G, k)$ — G is directed
Output: Whether G has a simple path on k-vertices, with constant one-sided error probability.
1: **for** $i = 1 \dots k^k$ **do**
2: Pick for every $v \in V$ a number $c(v) \in \{1, \dots, k\}$ uniformly and independently at random
3: Let $G' = (V, E')$ where $E' = \{(u, v) \in E : c(v) = c(u) + 1\}\}$
4: **if** $\texttt{kpathDAG}(G', k)$ **then return true**
5: **return false**

Algorithm 3: $k^k\texttt{poly}(n)$ time randomized algorithm for detecting a k-path. Note here that `kpathDAG` is a polynomial time algorithm that finds long paths in directed acyclic graphs (which is an easy exercise).

In exponential time algorithms we are quite desperate to find algorithm for problems with running times better than trivial algorithms. Unfortunately, for many problems this leads to seemingly too hard questions. Given the apparent power of OPP-algorithms, it thus would be interesting to see how far we can go with OPP-algorithms. The study of lower bounds for certain specific classes of algorithms is something done several times before in computer science (a recent example being 'extension complexity', and for exponential time algorithms 'resolution').

In this lecture we'll mainly discuss two (conditional) lower bounds for OPP algorithms: one by Paturi&Pudlak (STOC'10) and the other one being by Drucker (FOCS'13).

2 Standard stuff

A concentration inequality.

Lemma 4 (Chebyshev's inequality)**.**

$$\Pr[|X - \mathbb{E}[X]| \geq a] \leq \frac{\mathbb{E}[(X - \mathbb{E}[X])^2]}{a^2}.$$

Hashing using a random matrix.

Lemma 5. *Suppose $k, l > 0$, $X \subseteq \mathbb{F}_2^l$ and $|X| = \theta \cdot 2^k$ for some $\theta > 0$. Select $(A, v) \in_R \mathbb{F}_2^{k \times l} \times \mathbb{F}_2^k$. Then*

$$1 - 1/\theta \leq \Pr_{A,v}[\exists x \in X : Ax = v] \leq \theta.$$

Proof. The upper bound follows by taking a union bound over all $x \in X$ since for fixed x (and fixed A) we have $\Pr_v[Ax = v] = 2^{-k}$.

For the lower bound, let $X = \{x^1, \ldots, x^m\}$ and define $P_i = [Ax^i = v]$ and $S = \sum_{i=1}^m P_i$.

Using Chebyshev + pair-wise independence of the P_i's we see

$$\Pr[S \leq 0] \leq \frac{\mathbb{E}[(S - \mathbb{E}[S])^2]}{\mathbb{E}[S]^2} = \frac{\sum_{i \in [m]} \mathbb{E}[(P_i - 2^{-k})^2]}{\theta^2} = \frac{\theta 2^k}{\theta^2}(2^{-k}(1 - 2^{-k})) < 1/\theta.$$

□

The Min-Max Theorem. Consider the following 2-player zero-sum game: given is a fixed $m \times n$ matrix A with values $a_{i,j}$. The while the *maximizer* chooses a *row* index $i \in [m]$, while the *minimizer* chooses a *column* index $j \in [n]$. The outcome of the game is that the column player pays $a_{i,j}$ to the row player. Clearly, the order in which the players play is important in this game (for example, pick A to be the identity matrix).

However, let us now consider the setting where both players may pick a distribution over the rows and columns, e.g., the row player picks a distribution p over $[m]$ while the column player picks a distribution over $[n]$. The Min-Max theorem states that the order of playing does not matter anymore:

Theorem 6 (Min-Max Theorem)**.** *For every $m \times n$ matrix the following holds:*

$$\min_{\substack{p \in [0,1]^m \\ \sum_i p_i = 1}} \max_{\substack{q \in [0,1]^n \\ \sum_i q_i = 1}} p^T A q = \min_{\substack{q \in [0,1]^n \\ \sum_i q_i = 1}} \max_{\substack{p \in [0,1]^m \\ \sum_i p_i = 1}} p^T A q$$

Proof sketch. For a fixed mixed strategy of the row player, there is always an optimal strategy for the column player that is deterministic (i.e. $q_i = 1$ for some i). Therefore p wants to find a distribution that maximizes its revenue on any column. This can be written as LP; the dual of this LP turns out to be symmetric LP you would write for the column player. □

3 Small witnesses and OPP-algorithms.

Given Observation 2, we see there is a direct connection between *witness size* and success probability: if the witness size is $f(x)$, we can solve the problem with success probability $2^{-f(x)}$. Here witness size is formally defined as follows:

Definition 7. *A* verifier *of a language L is a polynomial time algorithm V such that $x \in L$ if and only if $\exists w : V(x, w) = true$. A problem has witness size $f(x)$ if there is a verifier accepting instances x and witnesses of size $f(x)$ as input.*

A natural question is: is there a converse of the relation between success probability and witness size? Can you convince somebody that an n-vertex graph has a k-path by communicating only $\texttt{poly}(k)$ bits using public randomness? The answer turns out to be YES if we change the definition of witness a bit:

Definition 8. *A* probabilistic verifier *of a language L is a polynomial time algorithm that accepts as input an instance x, bitstring w and random string w' such that*

$$\Pr_{w'}[\exists w : A(x, w, w') = true] \geq 1/2, \text{ if } x \in L, \qquad \text{and } A(x, w, w') = false, \text{ otherwise.}$$

A problem has probabilistic witness size $f(x)$ if there is a probabilistic verifier accepting triples (x, w, w') as input with $|w| = f(x)$.

Lemma 9. *A problem has probabilistic witnesses of size $f(x)$ if and only if it can be solved with success probability $2^{-f(x)}$.*

Proof. The forward direction is trivial since we can simply guess a probabilistic witness and run the probabilistic verifier on it.

For the backward direction we use a hashing technique as observed by Paturi and Pudlak: suppose we have an OPP algorithm accepting instance x and a random string $w \in \{0,1\}^l$ as input and the success probability is $s := f(x)$. Suppose that x is a YES-instance. Then the set of random strings $\{w \subseteq \{0,1\}^r : A(x, w) = 1\}$ is of size at least $g = 2^l/s$. Let $k = \lfloor \log_2 g/4 \rfloor$ so that $g \geq 2 \cdot 2^k$. Now Lemma 5 tells us that with

$$\Pr_{A,v \in \mathbb{F}_2^{k \times l} \times \mathbb{F}_2^k}[\exists w \in \{0,1\}^l : A(x, w) = 1 \wedge Ax = v] \geq 1 - 1/4.$$

Thus, interpreting A as a verifier, we may restrict our attention searching for witnesses w such that $Aw = v$. The expected number of such x is $2^l/2^k = 2^{l-k} + O(1)$ (and hence there are at most this many solution with high probability by Markov's inequality). The set $\{\ w \in \{0,1\}^r : Aw = v\}$ can written as $\{p + \sum_{i=1}^{z} b^i w_i : w \in \{0,1\}^z\}$ where p is a solution to $Ap = v$ and $b^1, \ldots, b^z$ a basis of the null-space of A. Now the verifier accepts x and $w \in \{0,1\}^z$ as input and outputs $A(x, p + \sum_{i=1}^{z} b^i w_i)$. We have that $z = l - k + O(1)$ which is $\log_2(s) + O(1)$ (and the constant can be suppressed using brute-force). □

4 Paturi & Pudlak boosting

One very fundamental question is whether it can be determined whether a Boolean circuit with n input variables can be satisfied in $1.99^n\texttt{poly}(n)$ time (the so-called 'Circuit Sat' problem). Note that any polynomial time algorithm accepting $x \in \{0,1\}^n$ as input can be rewritten as a equivalent polynomial sized circuit with n Boolean circuits, so such a result would say that for any polynomial time computable function we manage to find out whether it is satisfiable while avoiding a huge portion of the search space. Besides that such an algorithm would imply (conjectured) circuit lower bounds, no significant consequences of such a result are known.

Using that our polynomial verifier from 9 can be transformed in a circuit we get the following:

Observation 10. *If a problem can be solved with success probability $f(x)$, it admits a Monte Carlo reduction to Circuit Sat with $\log_2(f(x))$ variables.*

Here 'Monte Carlo' means that if the original instance was a YES-instance the circuit is satisfiable with probability at least $1/2$.

This actually means that if Circuit Sat *itself* could be solved with success probability $2^{00.99n}$, we can somehow boost the induced OPP-algorithm to an extent that seems hardly plausible:

Theorem 11 (Paturi & Pudlak)**.** *If Circuit Sat on n input gates can be solved with success probability $2^{-0.99n}$, then Circuit Sat can be solved in $2^{o(n)}\texttt{poly}(|C|)$ time.*

Proof. Given an input circuit C, use Observation 10 to transform it to circuit C' on $0.99n$ input gates with $|C'| = \texttt{poly}(|C|)$ which is equivalent with probability $1/2$. Repeat this $\log_{0.99}(\epsilon)$ times until ϵn input gates are left. This gives a circuit of size $\texttt{poly}(|C|)^{2^{O(\log(1/\epsilon))}}$ which has a solution with constant probability which can be solved in $2^{o(\epsilon n)}$. □

Note: This analysis is by no means tight and Paturi and Pudlak obtain better tradeoffs and also deterministic algorithms.

5 A non-deterministic direct product theorem and its consequences

An interesting question is how fast we can solve many instances of the same problem simultaneously. A nice example is matrix vector multiplication: given one $n \times n$ matrix and one n-dimensional vectors we definitely need $O(n^2)$ time; however if we are given $n \times n$ matrix A and n vectors $b^1, \ldots, b^n$ of dimension n this is simply matrix multiplication which can be done in $n^{2.376}$ time.

For sufficiently hard problems, one might expect there is nothing substantially more clever than using the same algorithm separately for each instance. Direct product theorems formalize this intuition. Put in the contrapositive, we get that if we have a problem for which we solve many instances jointly faster as separately, the problem would be easy in some sense:

Theorem 12 (Drucker)**.** *Let $L \subseteq \{0,1\}^*$, $n \in \mathbb{N}$ and $t := t(n) \leq \texttt{poly}(n)$. Suppose there is a circuit $C : \{0,1\}^{n \times t + r} \to \{0,1\}^t$ such that for every $x_1, \ldots, x_t \in \{0,1\}^t$*

$$\Pr_{w \in_R \{0,1\}^r}[C(x_1, \ldots, x_t, w) = ([x_1 \in L], [x_2 \in L], \ldots, [x_t \in L])] \geq 2^{-t/1000}. \tag{1}$$

Then there exists a circuit $C^\dagger : \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ *of size* $\mathtt{poly}(|C|)$ *such that for every* $x \in \{0,1\}^n$

$$x \in L \leftrightarrow \forall w \in \{0,1\}^r : C^\dagger(x,w) = 1. \tag{2}$$

In the conclusion we recognize co-non-determinism so we actually get the following corollary:

Corollary 13. *Suppose that for every* n *there exist some* $t(n) \leq \mathtt{poly}(n))$ *and circuit* C *satisfying* (1) *and* $|C| \leq \mathtt{poly}(n)$*. Then* $L \in coNP/poly$*.*

The following was recently observed by Drucker,Fortnow and Nederlof (to appear at ESA'16):

Theorem 14. *If that for some* $\epsilon > 0$*, there is an algorithm that given an* v*-vertex planar graph* G *outputs with probability at least* $2^{v^{1-\epsilon}}$ *a maximum independent set of* G*, then* $NP \subseteq coNP/poly$*.*

Proof. Apply Theorem 12 with L being Independent Set on planar graphs. Suppose we are given $t := t(n)$ instances $x_1, \ldots, x_t$ instances of L defined by graphs $(G_1, k), \ldots, (G_t, k)$ where each gaph is on v vertices. Let G be the disjoint union of these graphs (e.g each graphs shows up in different connected components). Note that if we have a maximum independent set of G we can determine in polynomial time for each instance whether it is a yes instance or not since a maximum independent set is also maximum in each connected component. Since G has $t(n)v$ vertices, we find such a set with probability at least $2^{-(tv)^{1-\epsilon}}$. Setting $t(n) = v^c$ such that $(c+1)(1-\epsilon) < c$ does the job. □

Similar consequences holds for more problems, e.g., the same proof gives

Theorem 15. *If there is an algorithm that given an* v*-vertex planar graph with tree-decomposition of* tw *outputs with probability at least* $2^{\mathtt{poly}(w)v^{1-\epsilon}}$ *a maximum independent set of* G*, then* $NP \subseteq coNP/poly$*.*

6 Proof sketch of Theorem 12

The high level idea. A key notion behind the proof is the following:

Definition 16. *An execution of* C *is called* j-valid *if it correctly estimates whether* $x_i \in L$ *for every* $i \leq j$*.*

Note that by (1) and the chain rule, we have for every $x_1, \ldots, x_t$:

$$\prod_{j=1}^{t} \Pr_{w \in \{0,1\}^r}[C(x_1, \ldots, x_t, w) \text{ is } j\text{-valid } | C(x_1, \ldots, x_t, w) \text{ is } (j-1)\text{-valid }] \geq 2^{-t/1000}. \tag{3}$$

Thus (3) implies that for a random choice of j:

$$\Pr_{w \in \{0,1\}^r}[C(x_1, \ldots, x_t, w) \text{ is } j\text{-valid } | C(x_1, \ldots, x_t, w) \text{ is } (j-1)\text{-valid }] \approx 1. \tag{4}$$

Motivated by this observation, the strategy for $C^\dagger$ on a high level will be as follows:

1. Pick $x_1, \ldots, x_t$ in a clever way,
2. Pick $j \in_R [t]$,

3. Denote $\vec{x} = (x_1, \ldots, x_{j-1}, x, x_{j+1}, \ldots, x_t)$,

4. Uniformly sample w from the set $\{w \in \{0,1\}^r : C(\vec{x}, w) \text{ is } (j-1)\text{-valid}\}$,

5. Return the estimate of $C(\vec{x}, w)$ of $[x \in L]$.

There are three issues that immediately come up:

Issue 1: Probabilistic Guarantee. Note that if our whole plan would work, this still only gives a probabilistic guarantee on the estimation of $x \in L$. Fortunately, this is a minor issue that we can overcome with standard techniques as follows: suppose we have a probabilistic circuit $C^\dagger$ that outputs a correct answer with good probability, say, 0.51. Then we can use $O(n)$ independent runs of $C^\dagger$ and take the majority vote to boost this probability to $1 - 2^{-\Omega(n)}$. A union bound over all possible $x \in \{0,1\}^n$ then tells us that there is some random string we could use so our estimate is always correct. Now we can just hard-wire this random string into the circuit $C^\dagger$.

In fact we'll actually use some extension of this idea when dealing with issue 3.

Issue 2: How to get a representative sample? In general it seems to be hard to implement the uniform sampling in step 4 efficiently: even finding a w such that $C(\vec{x}, w)$ is $(j-1)$-valid is NP-hard! However, note that we can use the non-uniformity and non-determinism (as formalized in (2)) to find such a w: in particular, if we fix $x_1, \ldots, x_t$ a priori we may hard-wire the values $[x_1 \in L], \ldots, [x_{j-1} \in L]$ into the circuit $C^\dagger$ and let $C^\dagger(\vec{x}, w)$ output 1 if it does not give a $(j-1)$-valid computation and let it output its estimate of whether $x \in L$ otherwise. However, the w we obtain in this way might of course not be a representative sample.

To extended this idea to get sample the set of $(j-1)$-valid executions we use hashing. In particular let

$$B = \{w \in \{0,1\}^r : C(\vec{x}, w) \text{ is } (j-1)\text{-valid but not } j\text{-valid}\},$$
$$G = \{w \in \{0,1\}^r : C(\vec{x}, w) \text{ is } j\text{-valid}\}.$$

Thus intuitively, B and G refer to the set of random strings $w \in 2^r$ such $C(x, w)$ are 'bad', respectively 'good', executions. Let k be some integer that we will choose later, and let $|G| = \theta 2^k$ and $|B| = \theta' 2^k$. Then from Lemma 5 we obtain that

$$\Pr_{A, v \in_R \mathbb{F}_2^{k \times r} \times \mathbb{F}_2^k} [\exists w \in G : Aw = v \wedge ! \exists w \in B : Aw = v] \geq 1 - 1/\theta - \theta'. \tag{5}$$

Thus if we follow (4) by supposing $|G| \geq 100|B|$, and pick $k = \lfloor \frac{1}{10} \log_2 |G| \rfloor$, then $\theta \geq 10$ and $\theta' \leq 1/5$. Thus, with probability at least 0.7 the event described in (5) happens. Using the above idea, we can now construct a circuit that always outputs a correct answer conditioned on this: the circuit outputs 1 if the execution is not $(j-1)$-valid or $Ax \neq v$, and outputs its estimate of whether $x \in L$ otherwise. Thus, picking $A, v \in_R \mathbb{F}_2^{k \times r} \times \mathbb{F}_2^k$ uniformly at random and then using the above we actually have a circuit that satisfies(2) with probability at least 0.7 (and as mentioned in issue 1 we can get rid of the randomization).

Note this strategy does require $|G| >> |B|$ and to have a know an approximation of $|B|$ (or $|G|$). We'll address this in a moment.

Issue 3: The insertion of x may influence the behavior of C drastically. Though (4) holds indeed for every $x_1, \ldots, x_t$ and random choice j, we actually need to say something about the computation $C(\vec{x}, w)$. In particular, we need to deal with malicious circuits that try to break our proof idea as follows:

1. figure out j,
2. output the correct values $[x_i \in L]$ for every $i \neq j$ and a random coin flip for $x = x_j \in L$.

Moreover, the adversary may pick the instance x as he wishes, so we need to win the following game:

- Adversary chooses a circuit C,
- We choose a circuit $C^\dagger$,
- Adversary chooses an instance x,
- We win if (2) holds.

Thus if our proof idea is going to work at all, we need to make sure the input circuit is not able to distinguish x from the other inputs $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_t$ hardwired into the circuit for any possible x. It is not immediately clear how to win this game since we have no idea how the adversary is going to pick x (and in particular, his choice of x may depend on our choices of instances hard-wired into the circuit).

However, we will now show how to turn around the table using (a standard application of) the Min-Max principle:

Lemma 17. *Suppose that for every distribution $\mathcal{D}$ over $\{0,1\}^n$ there exists a circuit $C_\mathcal{D}$ of size $\mathtt{poly}(|C|)$ such that (2) holds with probability at least 0.51, where the probability is taken over $x \sim \mathcal{D}$. Then there exists a circuit $C^\dagger$ such that (2) holds for every $x \in \{0,1\}^n$.*

Proof. Define the matrix A with rows indexed by the elements of $\{\text{circuit } C^\dagger : \{0,1\}^n \times \{0,1\}^r \to \{0,1\} \text{ s.t. } |C^\dagger| \leq \mathtt{poly}(n)\}$ and columns indexed by the possible inputs $x \in \{0,1\}^n$. Define the payoff at a particular entry $A[C^\dagger, x]$ to be 1 if (2) holds and 0 otherwise.

Since for every distribution $\mathcal{D}$ over $\{0,1\}^n$ there is a circuit[1] $C_\mathcal{D}$ receiving expected payoff 0.51, by Theorem 6 there is a distribution $\mathcal{C}$ over the set of circuits of size $\mathtt{poly}(|C|)$ that receives payoff 0.51 for every[1] $x \in \{0,1\}^n$.

Now we repeat the idea from issue 1: if sample $H = \Theta(n)$ circuits from $\mathcal{C}$ and output the majority vote of whether $x \in L$, we get the incorrect answer with probability at most $2^{-\Omega(n)}$. Since there are only 2^n values for x, there must be some H-tuple of circuits $(C_1, \ldots, C_H)$ whose majority vote always is correct. So we can take $C^\dagger$ to compute this majority vote. □

Summary of what we have so far. Lemma 17 shows we may assume $x \sim \mathcal{D}$ and only need a probabilistic guarantee where probability is taken over $x \sim \mathcal{D}$. Our approach now will be as follows:

1. For every $\mathcal{D}$, construct a circuit as in the assumption of Lemma 17 as follows:

[1] Note that this in fact is a deterministic strategy, e.g., only one row or column has probability 1.

(a) Show there exist an integer $j \in [t]$, and instances $x^1, \ldots, x^t \in \{0,1\}^n$ such that with high probability over $x \sim \mathcal{D}$ we have

$$\Pr[j - \text{valid} | x^1, \ldots, x^{j-1}, x] \geq 0.99 \Pr[(j-1) - \text{valid} | x^1, \ldots, x^{j-1}, x] \quad (6)$$

$$\Pr[(j-1) - \text{valid} | x^1, \ldots, x^{j-1}, x] \approx (1 \pm 0.1) \Pr[(j-1) - \text{valid} | x^1, \ldots, x^{j-1}], \quad (7)$$

where the probabilities are to be taken over the randomness of C and the validness refers to executions $C(\vec{x}, w)$ (where as before, $x_j = x$ in $\vec{x}$).

(b) Hard-wire $\alpha = \Pr[(j-1)-\text{valid} | x^1, \ldots, x^{j-1}]$ into the circuit and pick k to be $\lfloor 10 \log_2(\alpha 2^r) \rfloor$.

(c) Let $C_{\mathcal{D}}$ output 0 if and only if there is a $(j-1)$-valid execution of $C(x_1, \ldots, x_t)$ such that $Hr = v$ where $(H, v) \in_R \mathbb{F}_2^{k \times r} \times \mathbb{F}_2^k$.

2. Use Lemma 17 to construct $C^\dagger$ as promised in Theorem 12

Thus the only part remaining is $1(a)$, which we will focus on now.

Finding the inputs $x^1, \ldots, x^t$ and integer j from $1(a)$ Suppose we pick $x^1, \ldots, x^t \sim \mathcal{D}$, $j \in_R [t]$ independently at random and let $\vec{x} = (x^1, \ldots, x^{j-1}, x, x^{j+1}, \ldots, x^t)$. The nice thing is that since $x \sim \mathcal{D}$, $\vec{x}$ is independent of j. Define

$$\alpha_i := \Pr_{w \in \{0,1\}^r}[C(\vec{x}, w) \text{ is } i\text{-valid} | x^1, \ldots, x^i], \quad (8)$$

$$\beta_i := \Pr_{w \in \{0,1\}^r}[C(\vec{x}, w) \text{ is } i\text{-valid} | x^1, \ldots, x^{i+1}]. \quad (9)$$

We will now heuristically[1] argue that with good probability taken over the unconditioned x^i's there is a j such that

$$\frac{\beta_{j-1}}{\alpha_{j-1}} \approx 1 \pm 0.01 \text{ and } \alpha_j \geq 0.99 \beta_{j-1}. \quad (10)$$

This would finish the proof since this would imply that even a random t-tuple $x^1, \ldots, x^t$ and integer j satisfy the conditions (6) and (7), so in particular there exist such a pair.

The motivation of (10) is very similar to the motivation used in(3) and (4):

For $i \in \{1, \ldots, 2t\}$ define

$$Y_i = \begin{cases} \beta_k / \alpha_k & \text{if } i = 2k+1 \\ \alpha_k / \beta_{k-1} & \text{otherwise.} \end{cases}$$

This is well defined since all values $\alpha_1, \ldots, \alpha_t, \beta_1, \ldots, \beta_t$ are positive by assumption. Let us also define

[1] Drucker's construction is very similar to what we do here; except he considers an experiment where for every $i = 1, \ldots, t$ not a single instance is sampled but a (multi-)set with a constant number of samples. Then he argues similarly as below that there exist a t-tuple of constant sized-sets such that if we sample the instances we feed to C from these sets, then the required property on the conditional probabilities still hold. I (Jesper) at the moment don't really see why this is required, but on an intuitive level it is very similar to what we outline here (but technically more involved). However, note that it is very plausible that I greatly oversimplified some technical issue in this exposition!

$$Y_{prod} := \prod_{i=1}^{2t} Y_i = \left(\frac{\beta_0}{\alpha_0}\right) \cdot \left(\frac{\alpha_1}{\beta_0}\right) \cdot \left(\frac{\beta_1}{\alpha_1}\right) \cdots \left(\frac{\alpha_t}{\beta_{t-1}}\right) \geq 2^{-t/1000}. \tag{11}$$

Note that $Y_i \leq 1$ for every i since $\alpha_k \leq \beta_{k-1}$ as the first equals the probability of a more specific event. For i odd, observe that $\mathbb{E}[\beta_i|\alpha_i] = \alpha_i$ and thus $\mathbb{E}[Y_i] = 1$.

Now we may use the following lemma which has our setting as special case:

Lemma 18. *Let $Y_1, \ldots, Y_{2t}$ be (possibly dependent) nonnegative random variables satisfying $\mathbb{E}[Y_i] \leq 1$, such that $Y_{prod} = \prod_{i=1}^{2t} Y_i \geq q$. Then*

$$\Pr_{i \in_R [2t]}[Y_i \in [0.99, 1.01]] \geq 1 - \frac{2^{16} \ln(1/q)}{2t}.$$

We skip the proof of this lemma, as it mainly uses standard calculus. We get that with high probability $Y_i \in [0.99, 1.01]$ for uniform random $i \in_R [2t]$. By a union bound, there must also exist a j such that $Y_{2j-1}, Y_{2j} \in [0.99, 1.01]$. This is the j as required in (10).

7 Open questions:

- Is there an algorithm that given an instance of subset sum consisting of $n+1$ integers $a_1, \ldots, a_n, t$ each represented by at most b bits, finds a solution with probability at least $2^{-\texttt{poly}(b)n^{1-\epsilon}}$? Here a solution is a subset of the integers $a_1, \ldots, a_n$ summing to t.
- (First posed by Paturi) Can Hamiltonian cycle be solved with success probability $2^{-O(n)}$?
- (Possibly very hard but would be great) Suppose CNF-SAT can be solved with success probability $2^{-0.99n}$; does this have non-trivial consequences? Maybe one could construct sub-exponential co-non-deterministic algorithms for, say CNF-SAT?
- Get rid of the 'constructivity' requirement of the lower bounds.
- Many more..(!)