Electronic Colloquium on Computational Complexity, Report No. 125 (2019)

Hardness Amplification of Optimization Problems

Elazar Goldenberg
The Academic College of Tel Aviv-Yaffo
elazargo@mta.ac.il

Karthik C. S.*
Weizmann Institute of Science
karthik.srikanta@weizmann.ac.il

August 27, 2019

Abstract

In this paper, we prove a general hardness amplification scheme for optimization problems based on the technique of direct products.

We say that an optimization problem Π is direct product feasible if it is possible to efficiently aggregate any k instances of Π and form one large instance of Π such that given an optimal feasible solution to the larger instance, we can efficiently find optimal feasible solutions to all the k smaller instances. Given a direct product feasible optimization problem Π, our hardness amplification theorem may be informally stated as follows:

> If there is a distribution $\mathcal{D}$ over instances of Π of size n such that every randomized algorithm running in time $t(n)$ fails to solve Π on $\frac{1}{\alpha(n)}$ fraction of inputs sampled from $\mathcal{D}$,
> then, assuming some relationships on $\alpha(n)$ and $t(n)$,
> there is a distribution $\mathcal{D}'$ over instances of Π of size $O(n \cdot \alpha(n))$ such that every randomized algorithm running in time $\frac{t(n)}{poly(\alpha(n))}$ fails to solve Π on $99/100$ fraction of inputs sampled from $\mathcal{D}'$.

As a consequence of the above theorem, we show hardness amplification of problems in various classes such as NP-hard problems like Max-Clique, Knapsack, and Max-SAT, problems in P such as Longest Common Subsequence, Edit Distance, Matrix Multiplication, and even problems in TFNP such as Factoring and computing Nash equilibrium.

*This work was partially supported by Irit Dinur's ERC-CoG grant 772839.

ISSN 1433-8092

1 Introduction

The widely believed conjecture $\mathsf{P} \neq \mathsf{NP}$ asserts that the class NP cannot be decided efficiently on the worst-case. That is, no polynomial time algorithm can decide the satisfiability of a CNF formula on *every* instance. However, the worst case hardness of NP still does not clarify its average-case hardness: how hard is to decide the satisfiability on a uniformly random instance.

Studying the average-case hardness of NP has a two-fold motivation. First, it may provide a more meaningful explanation than worst-case complexity about the intractability of NP-hard instances actually encountered in practice. In other words, if NP is hard only on the worst-case, then the theory of worst-case complexity that has been extensively developed over the last fifty years, might be not be a good reflection of reality. Second, hardness on average is the cornerstone of modern cryptography as the security of any nontrivial cryptosystem requires some computational problem to be average-case hard (for some nice distribution). Additionally, showing average-case hardness for functions is a stepping stone towards proving strong derandomization results and the construction of pseudorandom generators.

The study of hardness amplification is the task of connecting the worst-case and average-case hardness. More specifically, based on a worst-case hardness (assumption) one would like to prove the average-case hardness of the problem.

1.1 Utopic Theorem of Hardness Amplification

A utopic theorem in the context of hardness amplification would assert that if a function is hard in the worst-case then it implies the average-case hardness for the same function against algorithms with essentially the same running time complexity. More formally it would look as follows:

Utopic Theorem 1 (A Utopic Hardness Amplification Theorem)**.** *Let $\{f_n\}_{n\in\mathbb{N}}$ be a family of functions. Assume that every algorithm running in time $t(n)$, fails to compute f_n on at least $\gamma(n)$ fraction of inputs. Then there exists a family $\{g_n\}_{n\in\mathbb{N}}$ of functions, such that every algorithm running in time $t'(n)$, fails to compute g_n on at least $\gamma'(n)$ fraction of inputs.*

Ideally, we would like to achieve the above amplification for the following parameters[1]

1. $\gamma(n) = O(1/2^n)$ *and* $\gamma'(n) = 1/2 - O(1/2^n)$,
2. $t'(n) \approx t(n)$,
3. $\{f_n\}_{n\in\mathbb{N}} = \{g_n\}_{n\in\mathbb{N}}$.

We briefly elaborate here why we would like the above three setting of parameters in our utopic hardness amplification theorem. Item 1 would yield a worst-case to average-case reduction, and therefore extend all the lower bounds and hardness results that have been achieved in the theory of worst-case complexity for f to the average-case

[1] In order to succinctly specify the desirable parameters of a hardness amplification theorem, we assume here that f_n and g_n are Boolean functions.

complexity of g. In fact, achieving $\gamma'(n) = 1/2 - O(1/2^n)$ would imply that no algorithm running in time $t'(n)$ can do much better than randomly guessing the output. Item 2 would imply that our worst-case complexity lower bounds meaningfully translate to lower bounds in the average-case. Item 3 expresses the notion of self-reducibility: if we are interested in understanding the average complexity of a problem, our hardness amplification theorem should enable us to do so by analyzing the worst-case complexity of the *same* problem. In summary, obtaining a hardness amplification result satisfying the three items is in a sense an attempt to bridge the gap between theory and practice. Finally, we remark that our utopic theorems would gain more importance if the family of functions for which we show hardness amplification are natural (in some broad sense).

Specifically, if we prove such a theorem for the family of deciding satisfiability of CNF formulas, then we get that the assumption that P $\neq$ NP implies that every polynomial time algorithm fails to decide satisfiability on slightly more that half of the CNF formulas – a highly non-trivial and very desirable result that would pave the way for the construction of one-way functions from (weak) worst-case assumptions. However, as we wake up from the dream of a utopia, one may wonder if such a result can even be achieved [BT06b].

Remarkably, nearly three decades ago, Lipton [Lip89, CPS99] proved the above type of (utopic) theorem for the function of computing the permanent (a #P-complete problem) against probabilistic polynomial time algorithms. Trevisan and Vadhan [TV07] following a line of works [BFNW93, IW97, STV01] were almost able to prove such an amplification result for the class EXP (they couldn't achieve Item 3). For the class NP we are far from proving a strong hardness amplification result, and there are some known barriers while trying to convert worst-case NP-hardness into average-case hardness (see e.g. [BT06a]). More recently, strong hardness amplification results have been proved for functions in P [BRSV17, GR17, GR18]. We also note that hardness amplification results have also been shown for one-way functions [Yao82, GIL+90, BR13].

Given the above state-of-the-art picture, we raise a few natural questions and address them in this paper. There are many problems that are hard in the worst-case but easy on average. For example, 3-coloring is a well-known NP-hard problem, but it is an easy exercise to show that it can be solved in linear time with high probability on a random graph. This motivates us to distinguish within worst-case hard problems as to which of them remain hard on average. One way to go about this task is to identify which worst-case hard problem admits a hardness amplification theorem.

For which problems can we amplify hardness?
Can we identify a mathematical structure that allows us to amplify hardness?

The latter question has been implicitly addressed in literature (for example, if the problem has algebraic structure like in the case of computing permanent [Lip89] or counting k-cliques [GR18]), but are quite specific and not broad enough to capture the class of problems that we believe are hard on average. In Section 1.2.1 we address the above two questions.

Next, we turn our attention to NP-hard problems. In a beautiful paper, O'Donnell [O'D04] initiated the study of (non-uniform) hardness amplification in NP. His result was improved by [HVV06] who showed that if for some function in NP (with n inputs)

we have that any $s(n)$ size circuit fails to compute the function correctly on $1/poly(n)$ fraction of the inputs then, the hardness can be amplified to show that there is some function in NP such that any $s(\sqrt{n})^{\Omega(1)}$ size circuit fails to compute the function on $1/2 - 1/s(\sqrt{n})^{\Omega(1)}$ fraction of the inputs. However, the best uniform hardness amplification results (against algorithms as opposed to circuits) that have been achieved do not match the parameters of [HVV06]: Trevisan [Tre05, BKS06] improving on his previous work [Tre03] showed that we can amplify hardness from $1/poly(n)$ to $1/2 - 1/polylog(n)$ for NP against randomized polynomial time algorithms (later extended to deterministic algorithms in [GG11]). However, it is important to note that all these hardness amplification results are for decision problems, and this leads us to our next question, do we gain anything by moving to search problems, or more precisely to the focus of this paper, to optimization problems?

Can we improve our hardness amplification results for optimization problems?
Can we prove stronger uniform hardness amplification results for MaxSAT*?*

Arguably, optimization problems are as natural as decision problems, but are strictly harder from the point of view of computational complexity. Does this mean we can either give simpler proofs of hardness amplification for optimization problems or prove stronger results? We address the above questions in Section 1.2.2.

We now shift our focus to the class P. As mentioned earlier, we have strong worst-case to average-case results established for problems in P [BRSV17, GR18]. The drawback however is that they are all for counting problems. This is indeed inherent as the underlying technique these works use are the same as the one used to show worst-case to average-case reduction for the permanent problem. While counting the number of k-cliques (the problem considered in [GR18]) is a natural problem, and therefore hardness amplification for that problem is interesting, it still leaves the door open for proving hardness amplification for the search problem of just finding one k-clique in a graph (an easier problem and thus harder to amplify hardness).

Can we prove hardness amplification results for natural search problems in P*?*

Moreover, there is a barrier [Abb19] to using the algebraic techniques of [BRSV17, GR18] to obtain hardness amplification for important problems studied in fine-grained complexity such as computing the Longest Common Subsequence (LCS) and Edit Distance (Edit-Distance) for a pair of strings. In particular, if these string problems can be represented using low-degree polynomials, then we could obtain small speedups by using the polynomial method [CW16], which would imply new circuit lower bounds [AHWW16]. This suggests we might need to look beyond these algebraic techniques for proving hardness amplification for these string problems. Is there a different technique to prove hardness amplification in P? We address these aforestated questions in Section 1.2.3.

1.2 Our Results

Our main contribution is a general hardness amplification theorem for optimization problems which we state in Section 1.2.1. Next, we apply our main theorem to various problems. In Section 1.2.2 we state our hardness amplification theorems for various

NP-hard problems such as Knapsack and MaxSAT. In Section 1.2.3 we state our hardness amplification theorems for various string problems in P such as LCS and Edit-Distance Finally, in Section 1.2.4 we state our hardness amplification theorems for various problems in TFNP (believed to not be in P) such as Factoring and computing Nash equilibrium.

1.2.1 Hardness Amplification of Optimization Problems

Aggregation is a key tool in the field of hardness amplification. If a function f is hard to compute on a tiny fraction of the domain, then, intuitively, computing multiple instances of f in one shot should be hard on a larger fraction of the inputs. More formally, for a function $f : [N] \to \Sigma$ and $k \in \mathbb{N}$, its k-direct product encoding is defined as a function $f^{(k)} : [N]^k \to \Sigma^k$ mapping each tuple $(x_1, \ldots, x_k)$ into $(f(x_1), \ldots, f(x_k))$. Using standard techniques one can show a 'direct product theorem' stating that if f is hard against $t(n)$ running-time algorithms on $\alpha(n)$-fraction of the domain, then $f^{(k)}$ is hard against $t'(n)$ running-time algorithms on $\approx k \cdot \alpha(n)$-fraction of its domain. But in order to utilize such a direct product result, we need to be able to stitch k-instances into a single (larger) instance. To address this task we introduce the following notion of direct product feasibility.

Definition 1.1 (Direct Product Feasibility; Informal statement of Definition 3.3)**.** *Let Π be an optimization problem. We say that Π is (S, T)-direct product feasible*[2] *if the exists a pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ satisfying the following:*

- Gen *takes as input k instances $(I_1, \ldots, I_k)$ of Π each of size n and outputs an instance I' of Π of size $S(n, k)$.*
- Dec *gets as input $(I_1, \ldots, I_k)$, the instance I' which is the output of Gen on input $(I_1, \ldots, I_k)$, an optimal solution for I', and $i \in [k]$. It outputs an optimal solution for the instance I_i.*
- *The running time of Gen and Dec is bounded by $T(n, k)$.*

Our main theorem is about hardness amplification for an arbitrary direct product feasible problem Π. In particular we show that if Π is hard against $t(n)$ running time randomized algorithms on a tiny fraction of the domain, then Π is hard on a much larger fraction of the domain against randomized algorithms with a similar running time.

Theorem 1.2 (Informal Statement of Theorem 3.4)**.** *Let Π be (S, T)-direct product feasible. Let $\mathcal{D}(n)$ be an efficiently samplable distribution over the instances of Π of size n. Assume the following:*

- *Any $t(n)$ running-time algorithm with success probability at least $2/3$ fails to compute an optimal solution on at least $\alpha(n)$-fraction of the inputs sampled from $\mathcal{D}$.*
- *Fix $k = poly((\alpha(n))^{-1})$. Then we have $T(n, k) = o(t(n))$.*

[2] In the formal definition of direct product feasibility, it is defined for a pair of optimization problems (Π, Λ) for technical reasons which will be addressed later in Section 1.2.3. In the case $\Pi = \Lambda$ we formally call it as self direct product feasible and this notion coincides with the informal definition given here. For most of the applications given in this paper, self direct product feasibility notion suffices.

- *We can (deterministically) decide the optimality of a given solution to any instance in $o(t(n))$ time.*

Then there exists an efficiently samplable distribution $\mathcal{D}'(S(n,k))$ over instances of Π of size $S(n,k)$ such that every $t(n)$ running-time algorithm with success probability at least $2/3$ fails to compute an optimal solution on at least 99% of the inputs sampled from $\mathcal{D}'$.

Naturally, the distribution $\mathcal{D}'$ is defined as follows: Draw k independent samples $I_1, \ldots, I_k$ from $\mathcal{D}$, and output $\mathsf{Gen}(I_1, \ldots, I_k)$. The proof of our main theorem is based on a reduction using an oracle access to an algorithm that solves $\mathcal{D}'$ on 99% of the inputs, we convert it into an algorithm solving $\mathcal{D}$ on greater than $1 - \alpha(n)$ fraction of the inputs. The reduction is uniform, so in case that the algorithms $(\mathsf{Gen}, \mathsf{Dec})$ are uniform we get a uniform hardness amplification result.

Another key point is that our hardness amplification is a self-reduction, i.e., if a problem is somewhat hard against one distribution $\mathcal{D}$, then the same problem is much harder against a different distribution $\mathcal{D}'$.

To the best of our knowledge, this is the first result to study hardness amplification for optimization problems. It opens avenues to prove results in various subclasses as we will see in subsequent subsections.

1.2.2 Hardness Amplification for NP-hard Problems

In the NP world, we generalize the results of [O'D04, Tre05] to optimization problems. In particular we show that if MaxSAT is hard to solve on $1/poly(n)$ fraction of the inputs of samples drawn from some samplable distribution $\mathcal{D}$. Then there exists a samplable distribution $\mathcal{D}'$ such that solving MaxSAT on $\mathcal{D}'$ is hard on at least $99/100$-fraction of the samples (See Corollary 4.3).

Theorem 1.3 (Informal Statement of Corollary 4.3.)**.** *Let $\mathcal{D}(n)$ be a distribution over 3-CNF formulas with n variables and $poly(n)$ clauses, such that for every randomized algorithm $\mathcal{A}$ running in time $poly(n)$, we have:*

$$\Pr_{\Psi \sim \mathcal{D}} [\mathcal{A} \text{ finds a optimal assignment for } \Psi \text{ w.p. } \geq 2/3] \leq 1 - 1/poly(n).$$

Then there exists a distribution $\mathcal{D}'(n')$ over 3-CNF formulas with n' variables $poly(n')$ clauses, such that for every randomized algorithm $\mathcal{A}'$ running in time $poly(n')$, we have:

$$\Pr_{\Psi' \sim \mathcal{D}'} \left[\mathcal{A}' \text{ finds a optimal assignment for } \Psi' \text{ w.p. } \geq 2/3\right] \leq 0.01.$$

Moreover, if $\mathcal{D}(n)$ is $poly(n)$-samplable then $\mathcal{D}'(n')$ is $poly(n')$-samplable.

Observe that the failure probability on $\mathcal{D}'$ is much larger than in [O'D04, Tre05] and can even tend to 0 for a proper choice of our parameters. This can be achieved since we deal with optimization problems instead of decision problems.

We also remark that our reduction and the proof correctness are much simpler, and in particular we do not rely the hard core set lemma [Imp95], a powerful and non-trivial key tool in the previous known proofs.

Our result easily extends into other NP-hard problems such as finding the largest clique in a graph, or finding smallest dominating set or vertex cover of a graph, etc (see Remark 4.4).

However, there are other NP-hard problems for which establishing a hardness amplification result through Theorem 1.2 is not easy. A special highlight is that of proving such a result for the Knapsack problem, as it isn't immediately clear if it's direct product feasible for reasonable range of parameters. This is because, for the Knapsack problem, when we aggregate instances in the natural way, optimal solutions of one instance may interfere with other instances (see Section 4 for details). Nonetheless, with some care, the direct product feasibility of Knapsack problem was established (see Lemma 4.7).

The Exponential Time Hypothesis (ETH) [IP01, IPZ01, CIP06] asserts that that we cannot decide whether a given 3-CNF is satisfiable in time which is sub-exponential in the number of variables. That is a worst case assumption, and it raises a natural question arises: Can we prove stronger hardness amplification result based on ETH? In fact, can we prove a worst case to an average case hardness amplification based on ETH?

Our next theorem is a step towards proving such a worst-case to an average case reduction for MaxSAT.

Theorem 1.4 (Informal Statement of Corollary 4.5.)**.** *Let $\mathcal{D}(n)$ be a distribution over* 3-CNF *instances with n variables and $O(n)$-clauses, such that for every randomized algorithm $\mathcal{A}$ running in time $2^{o(n)}$, we have:*

$$\Pr_{\Psi\sim\mathcal{D}}\left[\mathcal{A} \textit{ finds an optimal assignment for } \Psi \textit{ w.p. } \geq 2/3\right] \leq 1 - \frac{1}{2^{o(n)}}.$$

Then there exists a distribution $\mathcal{D}'(n')$ over 3-CNF *instances with n' variables and $2^{o(n')}$ clauses, such that for every polynomial time randomized algorithm $\mathcal{A}'$, we have:*

$$\Pr_{\Psi'\sim\mathcal{D}'}\left[\mathcal{A}' \textit{ finds an optimal assignment for } \Psi' \textit{ w.p. } \geq 2/3\right] \leq 0.01.$$

Healy et al. [HVV06] proved a similar result for the non-uniform case. Our result is stronger in the sense that we use a weaker assumption: we rely on the ETH that is assuming that every *uniform* algorithm fails on $1/2^{o(n)}$-fraction of inputs. While Healy et al. use similar assumption against non-uniform algorithms.

1.2.3 Hardness Amplification in P

We investigate hardness amplification in P and can show results for string problems, such as LCS and Edit-Distance, which were not possible in previous works.

Theorem 1.5 (Informal statement; see Corollaries 5.4 and 5.10)**.** *Fix $\varepsilon > 0$. Let $\mathcal{D}(n)$ be an efficiently samplable distribution over the instances of* LCS/Edit-Distance *of length n. Assume that any $n^{2-\varepsilon}$ running-time algorithm with success probability at least $2/3$ fails to compute an optimal alignment on at least $1/n^{o(1)}$-fraction of the inputs sampled from $\mathcal{D}$. Then for some $\varepsilon' > 0$ there exists an efficiently samplable distribution $\mathcal{D}'(n^{1+o(1)})$ over instances of* LCS/Edit-Distance *of size $n^{1+o(1)}$ such that every $n^{2-\varepsilon'}$ running-time algorithm with success probability at least $2/3$ fails to compute an optimal solution on at least 99% of the inputs sampled from $\mathcal{D}'$.*

Recall from earlier in this section that Abboud [Abb19] had pointed out a barrier to obtaining a result such as above, through algebraic techniques. Another similarity search problem that is studied along with LCS and Edit-Distance, is the problem of computing the Fréchet distance between two (discrete) curves. Strangely, this problem resists all natural approaches to show that it is direct product feasible (see Remark 5.11). Therefore, it is an interesting question as to whether it is possible to show that it is direct product feasible (for relevant range of parameters) or whether it is a candidate for a problem that is not direct product feasible.

Additionally, we show hardness amplification for a very different kind of problem, that of computing the product of two matrices (see Corollary 5.14). We highlight this problem, as it does not directly follow from our main theorem (i.e., Theorem 1.2). Elaborating, a detail that was brushed under the carpet while discussing Theorem 1.2 was that, given an instance of an optimization problem and a candidate solution, we need to able to efficiently compute the value of the objective of the candidate solution for that instance. This naturally holds for all the problems considered in this paper except the task of computing the product of two matrices, i.e., we do not know a way to *deterministically* verify if the product of two matrices is equal to a given third matrix, which is significantly faster than actually multiplying the two given matrices and checking if it's equal to the third matrix [Kün18, WW18]. Nonetheless, we modify the proof of Theorem 1.2 to handle this issue.

1.2.4 Hardness Amplification in TFNP

Total problems (with not necessarily efficient verification of totality) are essentially equivalent to Optimization problems (see Remark 3.7). The class TFNP is special as it is in an informal sense the intersection of Search NP and Optimization problems. Problems in TFNP capture problems in various areas such as game theory, cryptography, computational geometry, etc. We show that our general theorem can be applied to TFNP problems as well, and as an example show it for the Factoring problem (see Corollary 6.4) and the End of a Line problem (see Corollary 6.8). The latter hardness amplification result directly implies the hardness amplification of various problems in game theory such as computing an approximate Nash equilibrium (see Section 6.2 for details).

1.3 Open Problems

Our work leaves open several questions. We state a few of them below.

1.3.1 Stronger Hardness Amplification

In Theorem 1.4 we showed that if MaxSAT is hard to compute on $1 - 1/2^{o(n)}$-fraction of inputs for sub-exponential time algorithm, then there exists a distribution on which it is hard on a constant fraction of inputs for algorithms running in time $n^{\omega(1)}$. A natural open question is the following:

Can we improve Theorem 1.4 and get hardness amplified against sub-exponential time algorithms (instead of super-polynomial time algorithms)?

It seems to us that derandomized direct product theorems may serve as the key tool to address the above question (for example, see [IKW12]). In particular, if one can prove a (strongly) derandomized version of [FK00] then it might be possible to both aggregate sub-exponentially many instances succinctly and sample from the (derandomized) direct product distribution efficiently.

1.3.2 Direct Product Feasibility

In this paper, we were able to show direct product feasibility for certain problems quite easily (for example, see Theorems 1.3 and 1.5), but had to work harder to prove them for some other problems (for example, see Lemmas 4.7 and 5.13), and in some problem(s) were unable to establish the property of direct product feasibility (see Remark 5.11). This leads us to the following question.

Can we pinpoint what property of a problem makes it possible to establish direct product feasibility?

1.3.3 Gap Amplification versus Hardness Amplification

Direct Product theorems are key ingredients for both gap amplification and hardness amplification. Also, there are many philosophical similarities in the techniques known in literature of the aforementioned two kinds of amplifications. Thus we can ask the following (ambitious) question:

Can we obtain a trade-off between gap amplification and hardness amplification?

In particular, can we show that if one problem is hard to approximate on worst case within some factor $\alpha > 0$, then it is hard to approximate within a factor $\alpha/100$ on average? We note here that Feige [Fei02], did answer the converse of this question, i.e., he used average case hardness assumptions to prove hardness of approximation results for various problems in NP.

It seems to us that analyzing the operation of performing a small perturbation on the given instance may be the right direction to proceed. Elaborating, consider a (worst case) hard distribution over gap instances of some problem. If we build a new distribution, which samples from the aforementioned distribution, then performs a small perturbation on the sampled gap instance, and outputs the perturbed instance, then we would still retain most of the gap in the instance sampled from the new distribution, but on the other hand, the fraction of instances on which it is hard to solve the problem should increase significantly. It would be interesting if this intuition/approach could be made to work.

A related question is to ask if we can improve our result in Theorem 1.4 (for example, by making progress on the question detailed in Section 1.3.1) using Gap-ETH [Din16, MR16] (instead of ETH)?

1.3.4 Average Case Hard Problems in P

In this paper, we looked at average case hardness of some problems in P against some efficiently sampleable distribution but one can ask if we can achieve more.

Can we show for some natural problem in P *that it is hard to solve for the uniform distribution?*

Another important question stemming from cryptography [BRSV17] is whether we can construct a *fine-grained* one way function from worst case assumptions?

1.4 Organization of Paper

In Section 2, we provide the proof overview of our main theorem (Theorem 1.2). In Section 3, we formally state and prove Theorem 1.2. In Section 4, we prove hardness amplification results for various problems in NP. In Section 5, we prove hardness amplification results for various problems in P. Finally, in Section 6, we prove hardness amplification results for various problems in TFNP.

2 Proof Overview

We provide a proof overview for our hardness amplification result for the problem of finding the maximum clique in a graph and then in the subsequent section we will show how our general result (i.e., Theorem 1.2) would follow.

2.1 Hardness Amplification for Max Clique

To illustrate the main ideas behind our scheme let us focus on MaxCLIQUE, the problem of finding the largest clique in a given graph G.

Assume the existence of a distribution $\mathcal{D}$ over graphs on n vertices which is somewhat hard to compute. That is for every *randomized* algorithm $\mathcal{A}$ running in time $poly(n)$, we have

$$\Pr_{G\sim\mathcal{D}}\left[\mathcal{A} \text{ finds max-clique in } G \text{ w.p. } \geq 2/3\right] \leq 1 - 1/n. \tag{1}$$

We would like to prove the existence of a new distribution $\mathcal{D}'$ over graphs on $poly(n)$ vertices which is much harder to compute. That is, for every randomized algorithm $\mathcal{A}'$ running in time $poly(n')$, we have:

$$\Pr_{G'\sim\mathcal{D}'}\left[\mathcal{A}' \text{ finds max-clique in } G' \text{ w.p. } \geq 2/3\right] \leq 0.01. \tag{2}$$

Moreover if $\mathcal{D}$ is $poly(n)$-time samplable, then so is $\mathcal{D}'$.

Construction of New Distribution: $\mathcal{D}'$ samples a graph H as follows:

1. Independently sample $G_1, \ldots, G_k$ from $\mathcal{D}$, where $k = poly(n)$.
2. Define $V(H) = V(G_1) \dot{\cup} \cdots \dot{\cup} V(G_k)$.
3. For every $i \in [k]$, connect the vertices in $V(G_i)$ using the original edges in G_i.
4. For every $i, j \in [k]$ such that $i \neq j$, insert all the possible edges between G_i and G_j.
5. Output H.

Clearly, if $\mathcal{D}$ is $poly(n)$-time samplable, then so is $\mathcal{D}'$. Now assume for sake of contradiction, that there exists $\mathcal{A}'$ running in time $poly(n')$, violating Equation (2). We show the existence of an algorithm $\mathcal{A}$ running in time $poly(n)$ violating Equation (1).

The algorithm $\mathcal{A}$ on input graph G with n vertices is defined as follows:

1. Let $\mathcal{S}$ be an empty set.
2. Repeat following $O(n)$ times.
   (a) Pick randomly $i \in [k]$.
   (b) Independently sample $G_1, \ldots, G_{i-1}, G_{i+1}, \ldots G_k$ from $\mathcal{D}$.
   (c) Construct H setting G_i to be G.
   (d) Find clique in H using $\mathcal{A}'$.
   (e) Restrict clique in H to the vertices of G and add it to $\mathcal{S}$.
3. Output the largest clique in $\mathcal{S}$.

Clearly, the running time of $\mathcal{A}$ is $poly(n)$, as $n' = poly(n)$ and the running time of $\mathcal{A}'$ is $poly(n')$. Our first observation is that for any graph H constructed by $\mathcal{A}$, and for every $i \in [k]$ the restriction of a maximal clique in H into G_i, is a maximal clique for G_i.

Let $\mathcal{A}_0$ be one iteration of step 2 of $\mathcal{A}$. If we show that $\mathcal{A}_0$ outputs maximum clique w.p. $\Omega(1/n)$ on $1 - 1/n$ fraction of samples from $\mathcal{D}$ then, $\mathcal{A}$ outputs maximum clique w.p. $2/3$ on $1 - 1/n$ fraction of samples from $\mathcal{D}$.

Now, observe that if instead of planting the given input graph G as the i-th subgraph of H, we were planting a uniformly random sample of $\mathcal{D}$, then we get a graph H which is drawn according to $\mathcal{D}'$. Consequently, if that was the case, then the success probability of $\mathcal{A}_0$ was equal the probability of $\mathcal{A}'$ and we were done.

Let $\mathcal{D}'_G$ denote the marginal distribution over H, where the graph G is planted at a random coordinate $i \in [k]$. We conclude the proof by showing that for $1 - 1/n$-fraction of instances G drawn from $\mathcal{D}$ we have:

$$\Pr_{G' \sim \mathcal{D}'_G} \left[\mathcal{A}' \text{ finds max-clique in } G' \text{ w.p. } \geq 2/3 \right] \geq \frac{1}{2} \Pr_{G' \sim \mathcal{D}'} \left[\mathcal{A}' \text{ finds max-clique in } G' \text{ w.p. } \geq 2/3 \right].$$

Towards this goal we use a result by Feige and Kilian [FK00] that was proven in the context of parallel repetition. Under minor manipulations their result can be stated as follows:

Let X be a universe and $\mathcal{T}$ be a distribution over X. Let $f : X^k \to \{0,1\}$. Define

$$\mu = \mathop{\mathbb{E}}_{x^k \sim \mathcal{T}^k} \left[f\left(x^k\right) \right],$$

$$\mu_x = \mathop{\mathbb{E}}_{i \in [k], x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k \sim \mathcal{T}} \left[f(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots x_k) \right].$$

$$\Pr_{x \sim \mathcal{T}} \left[|\mu_x - \mu| \geq k^{-1/6} \right] \leq k^{-1/6}, \tag{3}$$

To conclude the result, set X as the set of graphs with n vertices, and $\mathcal{T}$ be the distribution $\mathcal{D}$. We have $\mathcal{D}' = \mathcal{D}^k$. Define $f : X^k \to \{0,1\}$ by:

$$f(G') = 1 \iff \mathcal{A}' \text{ finds a maximal clique in } G \text{ w.p. } \geq 2/3.$$

In these notations,

$$\begin{aligned} \mu &= \Pr_{G' \sim \mathcal{D}'} \left[\mathcal{A}' \text{ finds max-clique in } G' \text{ w.p. } \geq 2/3 \right] \\ \mu_x &= \Pr_{G' \sim \mathcal{D}'_G} \left[\mathcal{A}' \text{ finds max-clique in } G' \text{ w.p. } \geq 2/3 \right]. \end{aligned}$$

By an application of (3), and a proper choice of k, we get that for all but at most $k^{1/6}$-fraction of graphs G drawn according to $\mathcal{D}$, the success probability of $\mathcal{A}'$ on ${\mathcal{D}'}_G$ is $\Omega(n)$, as claimed.

2.2 Abstraction

In the previous subsection, we showed the main ingredients used for proving hardness amplification for the task of finding a maximal clique in a given graph. What were the properties of MaxCLIQUE that we utilized to prove the result?

One property that we used was that if we are given k input graphs $G_1, \ldots, G_k$, there exists an efficient way to construct a large graph H such that a maximal clique in H induces a maximal clique on each of the graphs G_i. The second property was that given a maximal clique in H there exists an efficient algorithm to construct a maximal clique on each of the graphs G_i.

These two properties are captured in Definition 1.1: The first property of a problem Π being Direct Product feasible is the existence of an efficient algorithm Gen stitching k instances $I_1, \ldots, I_k$ of Π into a larger instance I' of Π, such that: an optimal solution for I' induces an optimal solution for each of the instances I_i. The second property the existence of an efficient algorithm Dec converting an optimal solution for I' into an optimal solution of I'.

Once we show Π is Direct Product feasible then the rest of the proof goes through. Indeed, assuming the existence of a distribution $\mathcal{D}$ on instances of Π for which any efficient algorithm fails to compute on $1 - 1/n$ fraction of inputs, we define the distribution $\mathcal{D}', {\mathcal{D}'}_I$ as follows:

- $\mathcal{D}'$ is the k-product distribution of $\mathcal{D}$, where we pick k random samples from $\mathcal{D}'$

independently.

- $\mathcal{D}'_I$ is the distribution where we pick uniformly at random $i \in [k]$, and independently sample $I_1, \ldots, I_{i-1}, I_{i+1}, \ldots I_k$ from $\mathcal{D}$. Finally, we construct I' by setting I_i to be I.

Now we can use [FK00] to show that for most instances $I \sim \mathcal{D}$ to connect the success probability of $\mathcal{A}'$ on $\mathcal{D}'$ and $\mathcal{D}'_I$, to conclude the proof.

Remark about Direct Product results and Hardness Amplification. The direct product lemma at the heart of most hardness amplification results is the XOR lemma [Yao82]. But here we critically use the fact the problem is total, so at the surface at least, our results are incomparable to the hardness amplification results for NP and EXP obtained via XOR lemmas.

3 Hardness Amplification of Optimization Problems

In this section, we prove our main result. First, we define some notations. We use the definition of optimization problems given in [ACG+99] with additional formalism.

Definition 3.1 (Optimization Problem)**.** *An optimization problem Π is charaterized by the following quadruple of objects $(I_\Pi, \mathsf{Sol}_\Pi, \Delta_\Pi, \mathsf{goal}_\Pi)$, where:*

- *I_Π is the set of instances of Π. In particular for every $d \in \mathbb{N}$, $I_\Pi(d)$ is the set of instance of Π of input size at most d (bits);*
- *Sol_Π is a function that associates to any input instance $x \in I_\Pi$ the set of feasible solutions of x;*
- *Δ_Π is the measure function[3], defined for pairs (x, y) such that $x \in I_\Pi$ and $y \in \mathsf{Sol}_\Pi(x)$. For every such pair (x, y), $\Delta_\Pi(x, y)$ provides a non-negative integer which is the value of the feasible solution y;*
- *$\mathsf{goal}_\Pi \in \{\min, \max\}$ specifies whether Π is a maximization or minimization problem.*

We would like to identify a subset of our solution space which are optimal with respect to our measure function. To this effect, we define a notion of optimal feasible solution.

Definition 3.2 (Optimal Feasible Solution)**.** *Let $\Pi(I_\Pi, \mathsf{Sol}_\Pi, \Delta_\Pi, \mathsf{goal}_\Pi)$ be an optimization problem. For every $x \in I_\Pi$ and $y \in \mathsf{Sol}_\Pi(x)$ we say that y is an optimal feasible solution of x if for every $y' \in \mathsf{Sol}_\Pi(x)$ we have $\Delta_\Pi(x, y) \geq \Delta_\Pi(x, y')$ if $\mathsf{goal}_\Pi = \max$ and $\Delta_\Pi(x, y) \leq \Delta_\Pi(x, y')$ if $\mathsf{goal}_\Pi = \min$.*

Now, we can formally define the notion of direct product feasibility of a pair of optimization problems.

[3] We define the measure function only for feasible solutions of an instance. Indeed if an algorithm solving the optimization problem outputs a non-feasible solution then, the measure just evaluates to -1 in case of maximization problems and ∞ in case of minimization problems.

Definition 3.3. *Let $\Pi(I_\Pi, \mathsf{Sol}_\Pi, \Delta_\Pi, \mathsf{goal}_\Pi)$ and $\Lambda(I_\Lambda, \mathsf{Sol}_\Lambda, \Delta_\Lambda, \mathsf{goal}_\Lambda)$ be two optimization problems. Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. We say that the pair (Π, Λ) is (S, T)-direct product feasible if there exists a pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ such that for every $k, d \in \mathbb{N}$ the following holds:*

- Gen *takes as input $x_1, \ldots, x_k \in I_\Pi(d)$ and outputs $x' \in I_\Lambda(S(d,k))$.*
- *For any feasible solution $y' \in \mathsf{Sol}_\Lambda(x')$, Dec takes as input $i \in [k]$, $x_1, \ldots, x_k \in I_\Pi(d)$, and y', and outputs $y \in \mathsf{Sol}_\Pi(x_i)$. Moreover if $y' \in \mathsf{Sol}_\Lambda(x')$ is an optimal feasible solution then so is $y \in \mathsf{Sol}_\Pi(x_i)$.*
- Gen *and* Dec *run in $T(d,k)$ time.*

Moreover, if $\Pi = \Lambda$ then we say that Π is (S, T)-self direct product feasible

All but two results in this paper use the notion of self direct product feasibility. Only the problems of computing the longest common subsequence and computing the edit distance between two strings require us to define direct product feasibility for a pair of problems (instead of a single problem). Even for the aforementioned two problems, direct product feasibility is shown for a pair of essentially *same* problems (see Lemmas 5.3 and 5.8 for more details).

We now show our main theorem, that direct product feasibility implies hardness amplification.

Theorem 3.4 (Formal version of Theorem 1.2)**.** *Let $p \in (0,1)$. Let $\Pi(I_\Pi, \mathsf{Sol}_\Pi, \Delta_\Pi, \mathsf{goal}_\Pi)$ and $\Lambda(I_\Lambda, \mathsf{Sol}_\Lambda, \Delta_\Lambda, \mathsf{goal}_\Lambda)$ be two optimization problems. Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be such that (Π, Λ) is (S, T)-direct product feasible. Let $v : \mathbb{N} \to \mathbb{N}$ be such that there is a deterministic algorithm $\mathcal{V}$ running in time $v(d)$ which on input $x \in I_\Pi(d)$ and $y \in \mathsf{Sol}_\Pi(x)$ always correctly computes $\Delta_\Pi(x, y)$. Let $s, t : \mathbb{N} \to \mathbb{N}$, $\mathsf{fail} : \mathbb{N} \to (0,1]$, and $\mathcal{D} = \{D_d\}_{d \in \mathbb{N}}$ be a family of distributions such that for every randomized algorithm $\mathcal{A}$ running in time $t(d)$, the following holds for large $d \in \mathbb{N}$.*

- *D_d is a distribution over $I_\Pi(d)$ where an instance of $I_\Pi(d)$ can be sampled from D_d in $s(d)$ time.*
- $\Pr_{x \sim D_d}[\mathcal{A}$ *finds an optimal feasible solution for x with probability at least p*$] \leq 1 - \mathsf{fail}(d)$.

Let $k := 64 \cdot (\mathsf{fail}(d))^{-6}$ and $c := \frac{200 \ln(1/1-p)}{p}$. If $k \cdot s(d) + T(d,k) + v(d) \leq \frac{t(d)}{2c}$, then there is a distribution family $\mathcal{D}' = \{D'_d\}_{d \in \mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $\frac{t(d)}{2c}$, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_\Lambda(m)$ where $m = S(d,k)$ and an instance in $I_\Lambda(m)$ can be sampled from D'_d in $O(s(d)k + T(d,k))$ time.*
- $\Pr_{x' \sim D'_d}[\mathcal{A}'$ *finds an optimal feasible solution for x' with probability at least p*$] \leq 0.01$.

A key ingredient in the proof of the above theorem is the following direct product lemma, which may be seen as a combinatorial analogue of a special case of the celebrated parallel repetition theorem [Raz98]:

Lemma 3.5 (Feige-Kilian Direct Product Lemma [FK00]). *Let X be a finite set and $\mathcal{D}$ a probability distribution on X. Let $k \in \mathbb{N}$ and $f : X^k \to \{0,1\}$. We define the following two measures:*

$$\mu = \mathop{\mathbb{E}}_{x_1,\ldots,x_k \sim \mathcal{D}} [f(x_1,\ldots,x_k)] = \Pr_{x_1,\ldots,x_k \sim \mathcal{D}} [f(x_1,\ldots,x_k) = 1],$$

where $x_1,\ldots,x_k$ are sampled independently, and for every $i \in [k]$ and $x \in X$ we define

$$\mu_{i,x} = \mathop{\mathbb{E}}_{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_k \sim \mathcal{D}} [f(x_1,\ldots.x_{i-1},x,x_{i+1},\ldots,x_k)],$$

where $x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_k$ are sampled independently. Then, we have the following:

$$\Pr_{\substack{x \sim \mathcal{D} \\ i \sim [k]}} \left[|\mu_{i,x} - \mu| \geq \frac{1}{k^{1/3}} \right] \leq \frac{1}{k^{1/3}},$$

where we sample i from $[k]$ uniformly at random.

The proof of the above lemma as stated above may be found in [GO05] and we provide it in Appendix A for completeness. We also note that variants of the above lemma have previously appeared in direct product testing literature [DG08, IKW12, DS14].

Proof of Theorem 3.4. First we describe the construction of $\mathcal{D}'$ and then show the claims made in the theorem statement.

Construction of $\mathcal{D}'$

Let $(\mathsf{Gen}, \mathsf{Dec})$ be the pair of algorithms guaranteed by Definition 3.3 for the pair (Π, Λ). Fix $d, k \in \mathbb{N}$. We construct D'_d from D_d as follows. Independently sample k pairs of instances, say $x_1,\ldots,x_k$ from D_d and feed it as input to Gen. The sampling algorithm of the distribution D'_d then outputs the output of Gen.

Therefore, the m in the theorem statement, the size of the instances outputted by D'_d is equal to $S(d,k)$. The time needed to sample from D'_d is the time needed to sample k independent samples from D_d which is $k \cdot s(d)$ time, plus the running time of Gen which is $T(d,k)$.

Correctness of the Claim

We will show that if there is a randomized algorithm $\mathcal{A}'$ with success probability p running in time $t(d)/2^c$ that finds an optimal feasible solution of an instance sampled from D'_d with probability (over the sampling) greater than 0.01 then, there is a randomized algorithm $\mathcal{A}$ with success probability p running in time $t(d)$ that finds an optimal feasible solution for an instance sampled from D_d with probability (over the sampling) greater than $1 - \mathsf{fail}(d)$, reaching a contradiction. First we describe below the algorithm[4] $\mathcal{A}$.

[4]We remark here that the simulation of instances sampled from D'_d from an instance of D_d is similar to the simulation described in the (textbook) proof of showing existence of weak one-way functions imply existence of strong one-way functions [Yao82, Gol08].

Algorithm $\mathcal{A}$:

Input: An instance $x \in I_\Pi(d)$.

Output: A feasible solution $y \in \mathsf{Sol}_\Pi(x)$.

Procedure:

1. Let $\mathcal{S} \subseteq \mathsf{Sol}_\Pi(x)$ be a subset of feasible solutions initialized to $\emptyset$.
2. Repeat the below procedure c times.
   - 2.1. Pick $i \in [k]$ uniformly at random.
   - 2.2. Independently sample $k-1$ pairs from D_d say $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k$.
   - 2.3. Define $x_i = x$.
   - 2.4. Feed $x_1, \ldots, x_k$ as input to Gen. Let $x' \in I_\Lambda(m)$ be the output of Gen.
   - 2.5. Feed x' as input to $\mathcal{A}'$. Let $y' \in \mathsf{Sol}_\Lambda(x')$ be the output of $\mathcal{A}'$.
   - 2.6. Feed $i, x_1, \ldots, x_k$, and y' as input to Dec. Let y be the output of Dec.
   - 2.7. Include y in $\mathcal{S}$.
3. Run $\mathcal{V}$ on (x, y) for each $y \in \mathcal{S}$ and output the feasible solution in $\mathcal{S}$ which optimizes Π (depends on goal_Π).

Let us first analyze the running time of $\mathcal{A}$. Since we repeat Step 2, c times, it suffices to analyze the time needed for one iteration. Step 2.2 needs $(k-1) \cdot s(d)$ time, Step 2.4 needs $T(d,k)$ time, Step 2.5 needs time $t(d)/2c$ time, and finally Step 2.6 needs time $T(d,k)$. Therefore, the total running time of $\mathcal{A}$ is less than $c(k \cdot s(d) + T(d,k) + t(d)/2c + v(d)) \leq t(d)$.

Finally, we argue on the correctness probability of $\mathcal{A}$. Let $\mathcal{A}_0$ be the same algorithm as $\mathcal{A}$ except Step 2 has only one iteration. Therefore it suffices to show that $\mathcal{A}_0$ outputs an optimal feasible solution with probability at least $\frac{\ln(1/1-p)}{c}$ on $1 - \mathsf{fail}(d)$ fraction of instances sampled from D_d, as this implies that $\mathcal{A}$ outputs an optimal feasible solution with probability at least $\left(1 - \left(1 - \frac{\ln(1/1-p)}{c}\right)^c\right) \geq 1 - e^{\ln(1-p)} = p$ on greater than $1 - \mathsf{fail}(d)$ fraction of instances sampled from D_d. Notice that if y' is an optimal solution to x' then Dec always outputs an optimal solution to x. Therefore, it suffices to show that on input x', $\mathcal{A}'$ in Step 2.5. outputs an optimal feasible solution with probability at least $\frac{\ln(1/1-p)}{c}$ on greater than $1 - \mathsf{fail}(d)$ fraction of instances sampled from D_d.

Consider the Boolean function $f : I_\Lambda(m) \to \{0,1\}$ where $f(x') = 1$ if and only if $\mathcal{A}'$ outputs an optimal feasible solution of x' with probability at least p. From assumption on the fraction of sampled inputs that $\mathcal{A}'$ outputs an optimal feasible solution on, we have that $\mu := \mathop{\mathbb{E}}_{x' \sim D'_d}[f(x')] > 0.01$. Define $\mu_{x,i}$ as follows:

$$\mu_{x,i} := \Pr_{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_k \sim D_d}[f(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_k) = 1].$$

From Lemma 3.5, we have the following,

$$\Pr_{\substack{x\sim D_d \\ i\in[k]}}\left[|\mu_{x,i}-\mu|\geq k^{-1/3}\right]\leq k^{-1/3}.$$

Call an instance $x \in I_\Pi(d)$ "bad" if $\Pr_{i\in[k]}\left[\mu_{x,i} < \mu - k^{-1/6}\right] \leq k^{-1/6}$; otherwise it is called "good". From Markov inequality we have that:

$$\Pr_{x\sim D_d}[x \text{ is good}] \geq 1 - k^{-1/6}.$$

Take any good x then with probability at least $1 - k^{-1/6}$ over $i \in [k]$ we have that $\mu_{x,i} \geq \mu - k^{-1/6} \geq 0.01 - k^{-1/6}$. Next, conditioned on picking $i \in [k]$ such that $\mu_{x,i} > 0.01 - k^{-1/6}$, then with probability at least $0.01 - k^{-1/6}$ we have $f(x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_k) = 1$. Conditioned on $f(x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_k) = 1$, we get that $\mathcal{A}'$ outputs an optimal feasible solution of x' with probability at least p. Summarizing, if we sample a good x then with probability at least $(0.01 - k^{-1/6}) \cdot p \geq 0.005 \cdot p = \frac{\ln(1/1-p)}{c}$ the algorithm $\mathcal{A}'$ in Step 2.5 outputs an optimal feasible solution of x'. The proof concludes by noting that a good x is sampled with probability at least $1 - k^{-1/6} = 1 - \frac{\mathsf{fail}(d)}{2} > 1 - \mathsf{fail}(d)$. □

We conclude this section by providing a couple of remarks on the above theorem and proof.

Remark 3.6 (Amplification factor)**.** *We would like to note that the amplified hardness for the optimization problem Λ (which is shown in Theorem 3.4 to be 0.01) can be further amplified to any arbitrarily small positive constant*[5] *close to 0 by adjusting the parameters in the proof.*

Remark 3.7 (Total Problems)**.** *One may observe that the class of optimization problems are indeed equivalent to the class of* total *problems. For some finite alphabet Σ, we call a relation $R \subseteq \Sigma^* \times \Sigma^*$ to be total if for every $x \in \Sigma^*$ there is always a $y \in \Sigma^*$ such that $(x, y) \in R$. To see that every optimization problem Π is also a total problem, it suffices to note that the range of the measure function Δ_Π is bounded and therefore a maximum/minimum always exists. And to see that every total problem R is also an optimization problem Π_R, it suffices to note we can define $\Delta_{\Pi_R}(x, y)$ to be 1 if $(x, y) \in R$ and 0 otherwise, and set Π_R to be a maximization problem (i.e., $\mathsf{goal}_{\Pi_R} = \max$).*

Given this new outlook at optimization problems, one may see the existence of solutions as a crucial requirement in the proof of Theorem 3.4. In particular, our algorithm $\mathcal{A}$ (design and analysis) would be meaningless without the existence of solutions for any instance of the optimization problem.

[5] Actually, it can be even amplified to sub-constant, but we would have to pay in the running time lower bound for Λ.

4 Almost Worst Case to Average Case for Problems in NP

In this section, we show hardness amplification results for various NP-hard problems, with a focus on MaxSAT and Knapsack.

4.1 Maximum Satisfiability Problem

We recall below the Maximum satisfiability problem in our formalism for optimization problems.

Definition 4.1 (MaxSAT problem)**.** *The Maximum Satisfiability problem (*MaxSAT*) is an optimization problem characterized by the following quadruple of objects* $(I_{\mathsf{SAT}}, \mathsf{Sol}_{\mathsf{SAT}}, \Delta_{\mathsf{SAT}}, \max)$*, where:*

- *For every* $d \in \mathbb{N}$*,* $I_{\mathsf{SAT}}(d)$ *is the set of all* CNF *formulas on* $n = \Omega(d)$ *variables and* $O(n)$ *clauses;*
- *For every* $\phi \in I_{\mathsf{SAT}}$ *we have* $\mathsf{Sol}_{\mathsf{SAT}}(\phi) = \{0,1\}^n$*;*
- *For every* $\phi \in I_{\mathsf{SAT}}$ *and every* $x \in \{0,1\}^n$ *we define* $\Delta_{\mathsf{SAT}}(\phi, x)$ *to be the number of clauses that are satisfied by the assignment* x *to the variables of* ϕ*.*

We now show that MaxSAT is self direct product feasible (in a rather naive way).

Lemma 4.2. *Let* $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$*, where* $S(d,k) = dk$ *and* $T(d,k) = O(dk)$*. Then we have that* MaxSAT *is* (S,T)*-self direct product feasible.*

Proof. We define the pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ below.

For every $\phi_1, \ldots, \phi_k \in I_{\mathsf{SAT}}(d)$ given as input to Gen, it outputs the instance ϕ' in $I_{\mathsf{SAT}}(d')$ defined as:

$$\phi' := \phi'_1 \wedge \cdots \wedge \phi'_k$$

Where ϕ'_i is the formula obtained by replacing each literal l_j in ϕ_i by $l_{(i-1)n+j}$. It is clear that the running time of Gen is $O(d')$.

Next, for every $i^* \in [k]$, $\phi_1, \ldots, \phi_k \in I_{\mathsf{SAT}}(d)$, and a SAT assignment $x' \in \{0,1\}^{n'}$ for ϕ' given as input to Dec, the algorithm first runs Gen to compute ϕ' and then outputs $x \in \{0,1\}^n$ which is computed as follows: $x_j = x'_{(i^*-1)n+j}$.

We next show that if x' is an optimal SAT assignment for ϕ' then x is an optimal assignment for ϕ_{i^*}. This follows easily by the fact that the ϕ'_is are defined on disjoint sets of variables, hence an optimal solution for ϕ' induces an optimal solution for each of the ϕ_is.

The running times of Gen and Dec trivially follow. □

The above rather simple self direct product feasibility has a rather strong consequence when combined with our generic hardness amplification theorem.

Corollary 4.3. *Let* $a \geq 8$*. Let* $\mathcal{D} = \{D_d\}_{d \in \mathbb{N}}$ *be a family of distributions such that for every randomized algorithm* $\mathcal{A}$ *running in time* $O(d^a)$ *over inputs of size* d*, the following holds for large* $d \in \mathbb{N}$*.*

- *D_d is a distribution over $I_{\mathsf{SAT}}(d)$ where an instance of $I_{\mathsf{SAT}}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.*
- $\Pr_{\phi \sim D_d}[\mathcal{A}$ *finds maximizing assignment for* ϕ *with probability at least* $2/3] \leq 1 - \frac{1}{d}$.

Then for $m := \Theta(d^7)$, there is a distribution family $\mathcal{D}' = \{D'_d\}_{d \in \mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $O(m^{a/7})$ over inputs of size m, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_{\mathsf{SAT}}(m)$ and an instance in I_{SAT} can be sampled from D'_d in $\widetilde{O}(m)$ time.*
- $\Pr_{\phi' \sim D'_d}[\mathcal{A}'$ *finds maximizing assignment for* ϕ' *with probability at least* $2/3] \leq 0.01$.

Proof. We apply Theorem 3.4 by setting, $\Pi = \Lambda = \mathsf{MaxSAT}$, $p = 2/3$, $S(d,k) = dk$, $T(d,k) = w \cdot dk$ (for some $w \in \mathbb{N}$), $v(d) = w' \cdot d$ (for some $w' \in \mathbb{N}$), $s(d) = \widetilde{O}(d)$, $t(d) = d^a$, and $\mathsf{fail}(d) = 1/d$. Then we have that $k := \Theta(d^6)$ and $c := 300 \ln 3$. We verify that $k \cdot s(d) + T(d,k) + v(d) \leq \frac{t(d)}{2c}$ holds by noting that $k \cdot s(d) + T(d,k) + v(d) = \widetilde{O}(d^7)$ and $\frac{t(d)}{2c} = \Omega(d^a) = \Omega(d^8)$ (because $a \geq 8$). Therefore, we have that the sampling time from D'_d is $\widetilde{O}(d^7) = \widetilde{O}(m)$ and that the theorem statement holds for any randomized algorithm $\mathcal{A}'$ running in time $\frac{t(d)}{2c} = \Theta(d^a) = \Theta(m^{a/7})$. □

Remark 4.4. *The idea of taking k instances disjointly as in Lemma 4.2 can be extended to many* NP*-hard* covering *problems such as Vertex Cover, Dominating Set, etc. Consequently, we obtain hardness amplification results similar to Corollary 4.3 for these problems as well.*

Now we consider the same hardness amplification result of MaxSAT but against subexponential time algorithms. We provide below a slightly informal statement (using asymptotic notations as opposed to providing specific constants) for clarity.

Corollary 4.5. *Let $\mathcal{D} = \{D_d\}_{d \in \mathbb{N}}$ be a family of distributions such that for every randomized algorithm $\mathcal{A}$ running in time $2^{o(d)}$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.*

- *D_d is a distribution over $I_{\mathsf{SAT}}(d)$ where an instance of $I_{\mathsf{SAT}}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.*
- $\Pr_{\phi \sim D_d}[\mathcal{A}$ *finds maximizing assignment for* ϕ *with probability at least* $2/3] \leq 1 - \frac{1}{2^{o(d)}}$.

Then for $m := 2^{o(d)}$, there is a distribution family $\mathcal{D}' = \{D'_d\}_{d \in \mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $m^{\omega(1)}$ over inputs of size m, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_{\mathsf{SAT}}(m)$ and an instance in I_{SAT} can be sampled from D'_d in $\widetilde{O}(m)$ time.*
- $\Pr_{\phi' \sim D'_d}[\mathcal{A}'$ *finds maximizing assignment for* ϕ' *with probability at least* $2/3] \leq 0.01$.

Proof. Let $h : \mathbb{N} \to \mathbb{N}$ be some slowly increasing function such that $\lim_{x\to\infty} h(x) = \infty$ and let $h := h(d)$. We apply Theorem 3.4 by setting, $\Pi = \Lambda = \mathsf{MaxSAT}$, $p = 2/3$, $S(d,k) = dk$, $T(d,k) = w \cdot dk$ (for some $w \in \mathbb{N}$), $v(d) = w' \cdot d$ (for some $w' \in \mathbb{N}$), $s(d) = \widetilde{O}(d)$, $t(d) = 2^{d/h}$, and $\mathsf{fail}(d) = 1/2^{d/(6\cdot h^2)}$. Then we have that $k := \Theta(2^{d/h^2})$ and $c := 300 \ln 3$. We verify that $k \cdot s(d) + T(d,k) + v(d) \leq \frac{t(d)}{2c}$ holds by noting that $k \cdot s(d) + T(d,k) + v(d) = 2^{(1+o(1))\cdot d/h^2}$ and $\frac{t(d)}{2c} = \Omega(2^{d/h})$. Therefore, we have that the sampling time from D'_d is $2^{(1+o(1))\cdot d/h^2} = O(m)$ and that the theorem statement holds for any randomized algorithm $\mathcal{A}'$ running in time $\frac{t(d)}{2c} = O(2^{d/h}) = \Theta(m^{h(d)}) = m^{\omega(1)}$. □

Note that if we assume the (randomized) Exponential Time Hypothesis (ETH) for 3-SAT [IP01, IPZ01, CIP06] then after applying the Sparsification lemma, we obtain for some $\varepsilon > 0$, a family of distributions $\mathcal{D} = \{D_d\}_{d\in\mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}$ running in time $2^{\varepsilon d}$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.

- D_d is a distribution over $I_{\mathsf{SAT}}(d)$ where an instance of $I_{\mathsf{SAT}}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.
- $\Pr_{\phi\sim D_d}[\mathcal{A} \text{ finds maximizing assignment for } \phi \text{ with probability at least } 2/3] \leq 1 - \frac{1}{2^{\widetilde{O}(d)}}$.

Therefore, our amplification in Corollary 4.5 is an *almost* worst-case to average-case reduction for MaxSAT (under subexponential time reductions).

4.2 Knapsack Problem

In this subsection, we study the direct product feasibility of the Knapsack problem and as we will see, showing that its direct product feasible for reasonable parameters is significantly more non-trivial than was with the case for MaxSAT.

In the Knapsack problem we are given a target *sack weight* W, and set of items via pairs (w, v) where w is the weight of the item and v is the value of the item, and the goal is to pick a subset of items which maximizes the sum of the values of the picked items given the constraint that their total weight is at most W. More formally, we describe it as follows.

Definition 4.6 (KS problem)**.** *The Knapsack problem (*KS*) is an optimization problem characterized by the following quadruple of objects* $(I_{\mathsf{KS}}, \mathsf{Sol}_{\mathsf{KS}}, \Delta_{\mathsf{KS}}, \max)$*, where:*

- *For every* $d \in \mathbb{N}$, $I_{\mathsf{KS}}(d) = (W, \{(v_i, w_i)\}_{i=1}^n)$ *where* $W, n, v_i, w_i \in \mathbb{N}$ *such that* $d = \log W + \sum_{i=1}^n (\log v_i + \log w_i)$*;*
- *For every* $(W, \{(v_i, w_i)\}_{i=1}^n) \in I_{\mathsf{KS}}$ *we have* $\mathsf{Sol}_{\mathsf{KS}}((W, \{(v_i, w_i)\}_{i=1}^n))$ *is the set of all subsets* S *of* $[n]$ *satisfying* $\sum_{i\in S} w_i \leq W$*;*
- *For every* $S \in 2^{[n]}$ *satisfying* $\sum_{i\in S} w_i \leq W$ *we define* $\Delta_{\mathsf{KS}}(S)$ *to be* $\sum_{i\in S} v_i$*.*

It is not trivial to show the self direct problem feasibility for Knapsack. To see that, consider the case $k = 2$. Naively, if the sacks weights are W_1, W_2 then one may define a new sack of weight $W_1 + W_2$, and then take the union of the item sets while leaving their

weights and values untouched. However, in this simple reduction, we may use some of the target sack weight of one instance against another instance. Nonetheless we show with some care, a direct product feasibility result can be obtained.

Lemma 4.7. *Let* $\mathbb{T} := \{2^\ell \mid \ell \in \mathbb{N}\}$. *Let* $S, T : \mathbb{N} \times \mathbb{T} \to \mathbb{N}$, *where* $S(d,k) = k^{O(1)} \cdot d$ *and* $T(d,k) = k^{O(1)} \cdot d$. *Then we have that* KS *is* (S,T)*-self direct product feasible.*

Proof. We first show that there exists a pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ such that the conditions of (S,T)-self direct product feasibility in Definition 3.3 are met for the Knapsack problem for every $d \in \mathbb{N}$ but when k is fixed to be 2. Using that, we prove the lemma statement for any value of k which is a power of 2. The proof proceeds by first creating (recursively) two instances: the first corresponds to the first $k/2$ instances and the second to the last $k/2$ ones. Then we use the result for $k = 2$ to create a single instance.

Base Case $k = 2$**.** Let us first present the algorithm Gen. Let $I_1, I_2 \in I_{\mathsf{KS}}(d)$ be the input to Gen where for every $j \in \{1,2\}$, we have $I_j = (W_j, \{(v_i^{(j)}, w_i^{(j)})\}_{i \in [n_j]})$. We first normalize the weights (by multiplying the weight of both the sack and each item by the same factor c) so that $W_1 = W/2, W_2 = W$, for some $W \in \mathbb{N}$ (note that we can achieve this normalization for $W = 2 \cdot W_1 \cdot W_2$).

The output of Gen is a new instance $I' := (W', \{(v_i', w_i')\}_{i \in [n']}) \in I_{\mathsf{KS}}(d')$, where $d' = O(d^2)$, $n' := n_1 + n_2 + \log W$, and $W' = W^2 + W/2$. We define $N_1 = \{1, \ldots, n_1\}, N_2 = \{n_1 + 1, \ldots, n_2\}$, and $D = \{n_1 + n_2 + 1, \ldots, n_1 + n_2 + \log W\}$.

Now we define the items (v_i', w_i') for all $i \in [n']$. The first n_1 items correspond to the items of I_1, the next two n_2 items correspond to the items in I_2, and the last $\log W$ are dummy items. Elaborating, we have

$$\forall i \in [n'],\ v_i' = \begin{cases} v_i^{(1)} \text{ if } i \le n_1 \\ v_{i-n_1}^{(2)} \cdot (m+1)W + 1 \text{ if } n_1 < i \le n_1 + n_2 \\ (m+1) \cdot 2^{i-n_1-n_2-1} \text{ if } i > n_1 + n_2 \end{cases},$$

$$\forall i \in [n'],\ w_i' = \begin{cases} w_i^{(1)} \text{ if } i \le n_1 \\ w_{i-n_1}^{(2)} \cdot W \text{ if } n_1 < i \le n_1 + n_2 \\ W \cdot 2^{i-n_1-n_2-1} \text{ if } i > n_1 + n_2 \end{cases},$$

where $m := \sum_{i \in [n_1]} v_i^{(1)}$. Note that the size of I' is indeed $O(d)$.

Now we define Dec: It gets as input an index $j \in \{1,2\}$, an instance I' which was generated by Gen and an optimal solution S' for I'. It is required to produce an optimal solution for the instance I_j. The algorithm Dec returns $S' \cap N_1$ if $j = 1$, otherwise it outputs $S' \cap N_2$ (where each element is translated by $-n_1$ so that the final output resides in $[n_2]$).

The correctness of Dec follows by the following claim.

Claim 4.8. *Let* S' *be an optimal solution for* I'*, then:*

1. *$S'_1 := S' \cap N_1$ is an optimal solution for I_1.*
2. *Let $S_2 := S' \cap N_2$ and define S'_2 as the set of elements in S_2 where each element is translated by $-n_1$, then S'_2 is an optimal solution for I_2.*

Proof. 1. We first show that for S' we have

$$\mathbf{w} := \sum_{\{i' \in S' | i' \in N_2 \cup D\}} w'_i = W^2.$$

Assume not, then either we have $\mathbf{w} > W^2$ or $\mathbf{w} < W^2$. If it is the former then notice that since w'_i is a multiple of W for all $i > n_1$, we arrive at a contradiction as $\mathbf{w} \leq W' = W^2 + W/2$. Therefore let us assume that it is the latter. We define a 'better' solution S'' as follows.

First include into S'' all elements in $S' \cap N_2$. Let $\rho = W^2 - \sum_{i' \in S' \cap N_2} w'_i$, denote the remaining slack in the sack after inserting the elements in $N_2 \cap S'$. Next, for each $i \in D$ we insert i into S'' if in the binary representation of ρ, the i-th bit equals 1. We show that $\Delta(S'') > \Delta(S')$, contradicting the optimality of S'.

Let $D' = S' \cap D$, and let $\rho' = \sum_{i \in D'} 2^{i-(n_1+n_2)-1}$ be the number obtained by the binary representation of the elements in D'. Observe that since by our assumption $\sum_{\{i' \in S' | i' \in N_2 \cup D\}} w'_i < W^2$ we get $\rho \geq \rho' + 1$ and hence:

$$\Delta(\left|S'' \cap D\right|) - \Delta(\left|S' \cap D\right|) = (\rho - \rho') \cdot (m+1) \geq m+1,$$

and we conclude:

$$\begin{aligned} \Delta(S'') - \Delta(S') &= \Delta(\left|S'' \cap D\right|) - \Delta(\left|S' \cap D\right|) + \Delta(\left|S'' \cap N_1\right|) - \Delta(\left|S' \cap N_1\right|) \\ &\geq m+1 - \sum_{i \in S' \cap N_1} {v_i}' \\ &\geq 1, \end{aligned}$$

where the last inequality follows since $\sum_{i \in S' \cap N_1} v'_i \leq m = \sum_{i \in [n_1]} v_i^{(1)}$, contradicting the optimality of S'.

Now, clearly, if $\sum_{\{i' \in S' | i' \in N_2 \cup D\}} w_i = W^2$, then $S' \cap N_1$ is an optimal solution for I_1, since otherwise we can improve over the solution S' (by taking the same items from N_2 and D and add the optimal solution for I_1).

2. Assume for sake of contradiction that, S'_2 defined in the claim, is not an optimal solution for I_2. Let $\widetilde{S_2}$ be an optimal solution for I_2. Let us define S'' as follows: First we include the items from $\widetilde{S_2}$, then add items in D until the total weight reaches W^2. Finally, we include the items from $S' \cap N_1$.

Observe that by the previous item, the set S'' is a feasible solution (as the weight of the elements in $S' \cap N_1$ does not exceeds $W/2$). Since in I' for each $i \in N_2$ we set: $v'_i = v^{(2)}_{i-n_1} \cdot (m+1)W + 1$ and by the optimality of $\widetilde{S_2}$ we have:

$$\Delta(S'' \cap N_2) - \Delta(S' \cap N_2) \geq (m+1)W + 1.$$

Observe that S' may contain at most $\log W$ more elements from D than S'' contains. However their value is bounded by $(m+1)W$, and hence:

$$\begin{aligned}\Delta(S'') - \Delta(S') &= (\Delta(S'' \cap N_2) - \Delta(S' \cap N_2)) + (\Delta(S'' \cap D) - \Delta(S' \cap D)) \\ &\geq (m+1)W + 1 - (m+1)W \\ &\geq 1.\end{aligned}$$

Contradicting the optimality of S'. □

Finally, it is easy to see that running time of Dec is at most $O(d')$.

General Case $k = 2^\ell$, for some $\ell \in \mathbb{N}$. We will use $(\mathsf{Gen}, \mathsf{Dec})$ given in the previous case to show that there exists a pair of deterministic algorithms $(\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Dec}})$ such that the conditions of (S,T)-self direct product feasibility in Definition 3.3 are met for the Knapsack problem for every $d \in \mathbb{N}, k \in \mathbb{T}$.

Let us first present the algorithm $\widetilde{\mathsf{Gen}}$. Let $I_1, \ldots I_k \in I_{\mathsf{KS}}(d)$ be the input to Gen where for every $j \in [k]$, we have $I_j = (W_j, \{(v_i^{(j)}, w_i^{(j)})\}_{i \in [n_j]})$. We arbitrarily pair up the k instances, and feed each pair of instances to Gen (described previously in the proof). We obtain $k/2$ instances of Knapsack problem of size $O(d)$. We repeat the process of arbitrarily pairing up the instances and feeding it to Gen. After doing this process $\log k$ times, we will have a single instance of Knapsack as the output of Gen which will be of size at most $k^{O(1)} \cdot d$. This is the output of $\widetilde{\mathsf{Gen}}$.

Finally, it suffices to note that $\widetilde{\mathsf{Dec}}$ simply does a restriction of the solution to the coordinates of the instance of interest as in the previous case where k was set to 2. □

Like MaxSAT, Knapsack too admits a hardness amplification result but we skip writing it here for the sake of non-repetitiveness. Also, notice that the above lemma is proven for functions S, T on domain $\mathbb{N} \times \mathbb{T}$ instead of $\mathbb{N} \times \mathbb{N}$. This is done for the sake of clear presentation. The above proof can be extended to prove the direct product feasibility for the general case as well.

Remark 4.9 (Adopting above proof to maximization version of other covering problems)**.** *The idea of taking k instances with appropriate scaling as in Lemma 4.7 can be extended to many* maximization *versions of covering problems (that are* NP*-hard) such as Max-coverage, Clustering etc. Consequently, we obtain hardness amplification results for these problems as well.*

5 Mild Average Case to Sharp Average Case for Problems in P

In this section, we look at hardness amplification for two natural and important string problems which have been at the center stage of fine-grained complexity in the last few years. We also look at the problem of matrix multiplication, which does *not* fit into our scheme of hardness amplification given in Section 3 as it is not known to admit efficient deterministic verification. We also propose the problem of computing Frechet distance as a natural problem which might not be direct product feasible.

5.1 Longest Common Subsequence

In the Longest Common Subsequence problem we are given two strings and the goal is to find the subsequence of maximum length that is in both the strings. It is indeed a natural maximization problem and fits smoothly into our formalism as follows.

Definition 5.1 (LCS alignment)**.** *Let Σ be a finite non-empty set and $n \in \mathbb{N}$. For every pair of strings $(a, b) \in \Sigma^n \times \Sigma^n$ and every function $\sigma : [n] \to [n] \cup \{\perp\}$, we say that σ is an* LCS *alignment for (a, b) if the following holds.*

- ***Monotonicity****: For every $i, j \in [n]$, $i < j$ we have that if $\sigma(i) \neq \perp$ and $\sigma(j) \neq \perp$ then $\sigma(i) < \sigma(j)$.*
- ***Matching****: For every $i \in [n]$, if $\sigma(i) \neq \perp$ then we have $a_i = b_{\sigma(i)}$.*

The length of an LCS *alignment σ is defined as the cardinality of the preimage of $[n]$, i.e.,*

$$|\{i \in [n] \mid \sigma(i) \in [n]\}|\,.$$

Definition 5.2 (LCS problem)**.** *Let Σ be a finite non-empty set. The Longest Common Subsequence (LCS_Σ) is an optimization problem charaterized by the following quadruple of objects $(I_{\mathsf{LCS}_\Sigma}, \mathsf{Sol}_{\mathsf{LCS}_\Sigma}, \Delta_{\mathsf{LCS}_\Sigma}, \max)$, where:*

- *For every $d \in \mathbb{N}$, $I_{\mathsf{LCS}_\Sigma}(d) = \Sigma^n \times \Sigma^n$ where[6] $n = d/(2|\Sigma|)$;*
- *For every $(a, b) \in I_{\mathsf{LCS}_\Sigma}$ we have $\mathsf{Sol}_{\mathsf{LCS}_\Sigma}(a, b)$ is the set of all* LCS *alignments for (a, b);*
- *For every $(a, b) \in I_{\mathsf{LCS}_\Sigma}$ and every* LCS *alignment σ for (a, b) we define $\Delta_{\mathsf{LCS}_\Sigma}(a, b, \sigma)$ to be the length of σ.*

Now we show that LCS on alphabets of some size are direct product feasible with LCS on alphabets with an additional character.

Lemma 5.3. *Let Σ be a finite non-empty set. Let Ξ be a superset of Σ of cardinality $|\Sigma| + 1$. Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $S(d, k) = 2|\Xi|(dk^2 + k - 1)$ and $T(d, k) = 2 \cdot S(d, k)$. Then we have that $(\mathsf{LCS}_\Sigma, \mathsf{LCS}_\Xi)$ are (S, T)-direct product feasible.*

Proof. Let $\Xi = \Sigma \cup \{\xi\}$, where $\xi \notin \Sigma$. We define the pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ below.

Fix $k, d \in \mathbb{N}$ (and consequently $n \in \mathbb{N}$). Let $\ell := 1 + nk$, $m := nk + \ell(k - 1)$, and $d' = 2m|\Xi|$. Let $\mathbf{z} \in \Xi^\ell$ be the concatenation of ℓ copies of ξ, i.e.,

$$\mathbf{z} := \underbrace{\xi \circ \cdots \circ \xi}_{\ell \text{ copies}}.$$

[6]Here we use the unary encoding to encode a symbol in Σ for the ease of presentation, as it circumvents rounding issues. This does not affect the results in this paper as we think of Σ as some small universal constant (like $\Sigma = \{0, 1\}$). The results in this paper also hold for larger alphabets, but more care needs to taken in the rounding of parameters in the forthcoming proofs while using the binary encoding.

For every $(a_1, b_1), \ldots, (a_k, b_k) \in I_{\mathsf{LCS}_\Sigma}(d)$ given as input to Gen, it outputs the instance $(\mathbf{a}, \mathbf{b})$ in $I_{\mathsf{LCS}_\Xi}(d')$ where,

$$\mathbf{a} := a_1 \circ \mathbf{z} \circ a_2 \circ \mathbf{z} \circ \cdots \circ \mathbf{z} \circ a_k \in \Xi^m, \;\; \mathbf{b} := b_1 \circ \mathbf{z} \circ b_2 \circ \mathbf{z} \circ \cdots \circ \mathbf{z} \circ b_k \in \Xi^m.$$

It is clear that the running time of Gen is d'.

Next, for every $i \in [k]$, $(a_1, b_1), \ldots, (a_k, b_k) \in I_{\mathsf{LCS}_\Sigma}(d)$, and an LCS alignment $\widetilde{\sigma} : [m] \to [m] \cup \{\perp\}$ for $(\mathbf{a}, \mathbf{b})$ given as input to Dec, the algorithm first runs Gen to compute $(\mathbf{a}, \mathbf{b})$ and then outputs $\sigma : [n] \to [n] \cup \{\perp\}$ which is computed as follows:

$$\forall j \in [n],\ \sigma(j) = \begin{cases} \widetilde{\sigma}(j + \mathtt{loc}) \text{ if } \mathtt{loc} < \widetilde{\sigma}(j + \mathtt{loc}) \leq n + \mathtt{loc}, \\ \perp \text{ otherwise,} \end{cases}$$

where $\mathtt{loc} = (n + \ell)(i - 1)$. It is easy to see that σ is an LCS alignment for (a_i, b_i). Also, it is easy to see that the running time of Dec is running time of Gen plus d' (needed to compute σ).

We next show that if $\widetilde{\sigma}$ is an optimal LCS alignment for $(\mathbf{a}, \mathbf{b})$ then σ is an optimal alignment for (a, b). To show this we first show that if $\widetilde{\sigma}$ is an optimal LCS alignment then it has a certain "block structure". Consider the following LCS alignment $\widetilde{\tau}$ for $(\mathbf{a}, \mathbf{b})$:

$$\forall j \in [m],\ \widetilde{\tau}(j) = \begin{cases} j \text{ if } \mathbf{a}_j = \xi, \\ \perp \text{ otherwise.} \end{cases}$$

$\widetilde{\tau}$ is an LCS alignment because of our construction of $(\mathbf{a}, \mathbf{b})$, where we have that for all $j \in [m]$ if $\mathbf{a}_j = \xi$ then $\mathbf{b}_j = \xi$ as well. We have that $\Delta_{\mathsf{LCS}_\Xi}(\boldsymbol{a}, \boldsymbol{b}, \widetilde{\tau}) = \ell(k-1)$. Therefore if $\widetilde{\sigma}$ is an optimal LCS alignment then we should have $\Delta_{\mathsf{LCS}_\Xi}(\boldsymbol{a}, \boldsymbol{b}, \widetilde{\sigma}) \geq \ell(k-1)$.

Suppose there exists $j \in [n]$ such that $\widetilde{\sigma}(j + \mathtt{loc}) \neq \perp$ and is strictly greater than $n + \mathtt{loc}$. If there is more than one such j then we pick the smallest one. Then we have that $\widetilde{\sigma}(j + \mathtt{loc}) > n + \mathtt{loc} + \ell$ as $\mathbf{a}_{j+\mathtt{loc}} \in \Sigma$ but $\mathbf{b}_{n+\mathtt{loc}+r} = \xi \notin \Sigma$ for all $r \in [\ell]$. Also we have that for every $j' \in [n]$ that is strictly less than j either $\widetilde{\sigma}(j + \mathtt{loc}) = \perp$ or is at most $n + \mathtt{loc}$. In this case, we have that the length of $\widetilde{\sigma}$ is at most $m - \ell = nk + \ell(k-2) = \ell(k-1) - 1$, a contradiction as we showed earlier that $\Delta_{\mathsf{LCS}_\Xi}(\boldsymbol{a}, \boldsymbol{b}, \widetilde{\sigma}) \geq \ell(k-1)$. By following a similar argument we can show that there does not exist $j \in [n]$ such that $\widetilde{\sigma}(j + \mathtt{loc}) \neq \perp$ and is less than or equal to $\mathtt{loc}$. This implies that $\widetilde{\sigma}$ restricted to the coordinates in the interval $[\mathtt{loc} + 1, \mathtt{loc} + n]$ provides an optimal LCS alignment for pair of contiguous substring of $\mathbf{a}$ and $\mathbf{b}$ restricted to the coordinates in the interval $[\mathtt{loc} + 1, \mathtt{loc} + n]$. Thus σ is an optimal LCS alignment for (a_i, b_i).

Finally, we note that the bound on the functions S and T hold as $d' = (nk + (nk + 1)(k-1))2|\Xi| = 2|\Xi|(nk^2 + k - 1) \leq 2|\Xi|(dk^2 + k - 1)$. □

LCS problem has been of special interest in the last few years thanks to the advancements in fine grained complexity [ABW15, AHWW16, Wil15, Wil16, Wil18]. The above direct product feasibility immediately implies the below hardness amplification result.

Corollary 5.4. *Let $\varepsilon > 0$. Let Σ be a finite non-empty set. Let Ξ be a superset of Σ of cardinality $|\Sigma| + 1$. Let $\mathcal{D} = \{D_d\}_{d \in \mathbb{N}}$ be a family of distributions such that for every randomized algorithm $\mathcal{A}$ running in time $d^{1+\varepsilon}$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.*

- *D_d is a distribution over $I_{\mathsf{LCS}_\Sigma}(d)$ where an instance of $I_{\mathsf{LCS}_\Sigma}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.*
- $\Pr_{(a,b)\sim D_d}[\mathcal{A}$ *finds optimal* LCS *alignment for* (a,b) *with probability at least* $2/3] \leq 1 - d^{-o(1)}$.

Then there is some $\varepsilon' > 0$ such that there is a distribution family $\mathcal{D}' = \{D'_d\}_{d\in\mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $d^{1+\varepsilon'}$, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_{\mathsf{LCS}_\Xi}(d^{1+o(1)})$ and an instance in I_{LCS_Ξ} can be sampled from D'_d in $d^{1+o(1)}$ time.*
- $\Pr_{(a',b')\sim D'_d}[\mathcal{A}'$ *finds optimal* LCS *alignment for* (a',b') *with probability at least* $2/3] \leq 0.01$.

Proof. We apply Theorem 3.4 by setting, $\Pi = I_{\mathsf{LCS}_\Sigma}$, $\Lambda = I_{\mathsf{LCS}_\Xi}$, $p = 2/3$, $S(d,k) = 2|\Xi|(dk^2 + k - 1)$, $T(d,k) = 2 \cdot S(d,k)$, $v(d) = w \cdot d$ (for some $w \in \mathbb{N}$), $s(d) = \widetilde{O}(d)$, $t(d) = d^{1+\varepsilon}$, and $\mathsf{fail}(d) = 1/d^{o(1)}$. Then we have that $k := d^{o(1)}$ and $c := 300 \ln 3$. We verify that $k \cdot s(d) + T(d,k) + v(d) \leq \frac{t(d)}{2c}$ holds by noting that $k \cdot s(d) + T(d,k) + v(d) = d^{1+o(1)}$ and $\frac{t(d)}{2c} = \Omega(d^{1+\varepsilon})$. Therefore, we have that the sampling time from D'_d is $d^{1+o(1)}$ and that the theorem statement holds for any randomized algorithm $\mathcal{A}'$ running in time $\frac{t(d)}{2c} = \Theta(d^{1+\varepsilon}) = \Theta\left(\left(d^{1+o(1)}\right)^{1+\varepsilon-\delta}\right)$, for any $\delta > 0$. □

Remark 5.5 (Hardness Amplification of k-LCS and other parameterized complexity problems)**.** *We remark here that the above proof strategy can be extended to show a hardness amplification result for the k-*LCS *problem (for fixed k) which is of interest in parameterized complexity. In fact, following Remark 4.4, we can obtain hardness amplification theorems for fundamental problems in fixed parameter complexity such k-clique and k-set cover.*

5.2 Edit Distance

Recall that the edit distance between a pair of strings $a, b \in \Sigma^n$ is defined as the minimal number of edit operations needed to convert x into y, where edit operations are character insertions/ deletions and substitutions.

Definition 5.6 (ED alignment)**.** *Let Σ be a finite non-empty set and $n \in \mathbb{N}$. For every pair of strings $(a,b) \in \Sigma^n \times \Sigma^n$ and every function $\sigma : [n] \to [n] \cup \{\perp\}$, we say that σ is an* LCS *alignment for (a,b) if the following holds.*

***Monotonicity**: For every $i, j \in [n]$, $i < j$ we have that if $\sigma(i) \neq \perp$ and $\sigma(j) \neq \perp$ then $\sigma(i) < \sigma(j)$.*

The cost of an ED *alignment σ is defined as twice the cardinality of the preimage of $\perp$ plus the number of mismatches, i.e.,*

$$2\left|\{i \in [n] \mid \sigma(i) = \perp\}\right| + \left|\{i \in [n] \mid y_\sigma(i) \neq x_i\}\right|.$$

We interpret an alignment σ is as follows: For every $i \in [n]$ if $\sigma(i) \neq \perp$ then σ matches each symbol x_i into $y_{\sigma(i)}$ (while paying an edit operation in case of substitution). In case $\sigma(i) = \perp$, then it means that σ deletes the i-th character of x. In case we

have $\sigma(i), \sigma(i+1) \neq \perp$ but $\sigma(i+1) > \sigma(i)+1$ then it means σ inserts the characters $y_{\sigma(i)+1}, \ldots, y_{\sigma(i+1)-1}$. The bound on the edit cost of σ simply follows by the fact that the number of characters insertions and deletions is equal.

Definition 5.7 (ED problem)**.** *Let Σ be a finite non-empty set. The Edit Distance (ED_Σ) is an optimization problem characterized by the following quadruple of objects $(I_{\mathsf{ED}_\Sigma}, \mathsf{Sol}_{\mathsf{ED}_\Sigma}, \Delta_{\mathsf{ED}_\Sigma}, \min)$, where:*

- *For every $d \in \mathbb{N}$, $I_{\mathsf{ED}_\Sigma}(d) = \Sigma^n \times \Sigma^n$ where[7] $n = d/(2|\Sigma|)$;*
- *For every $(a,b) \in I_{\mathsf{ED}_\Sigma}$ we have $\mathsf{Sol}_{\mathsf{ED}_\Sigma}(a,b)$ is the set of all ED alignments for (a,b);*
- *For every $(a,b) \in I_{\mathsf{ED}_\Sigma}$ and every ED alignment σ for (a,b) we define $\Delta_{\mathsf{ED}_\Sigma}(a,b,\sigma)$ to be the cost of σ.*

Lemma 5.8. *Let Σ be a finite non-empty set. Let Ξ be a superset of Σ of cardinality $|\Sigma|+1$. Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $S(d,k) = 3|\Xi|dk^2$ and $T(d,k) = 2 \cdot S(d,k)$. Then we have that $(\mathsf{ED}_\Sigma, \mathsf{ED}_\Xi)$ are (S,T)-direct product feasible.*

Proof. Let $\Xi = \Sigma \cup \{\xi\}$, where $\xi \notin \Sigma$. We define the pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ below.

Fix $k, d \in \mathbb{N}$ (and consequently $n \in \mathbb{N}$). Let $\ell := 1 + nk$, $m := (n+\ell)k$, and $d' = 2m|\Xi|$. Let $\mathbf{z} \in \Xi^\ell$ be the concatenation of ℓ copies of ξ, i.e.,

$$\mathbf{z} := \underbrace{\xi \circ \cdots \circ \xi}_{\ell \text{ copies}}.$$

For every $(a_1, b_1), \ldots, (a_k, b_k) \in I_{\mathsf{ED}_\Sigma}(d)$ given as input to Gen, it outputs the instance $(\mathbf{a}, \mathbf{b})$ in $I_{\mathsf{ED}_\Xi}(d')$ where,

$$\mathbf{a} := a_1 \circ \mathbf{z} \circ a_2 \circ \mathbf{z} \circ \cdots \circ \mathbf{z} \circ a_k \circ \mathbf{z} \in \Xi^m, \;\; \mathbf{b} := b_1 \circ \mathbf{z} \circ b_2 \circ \mathbf{z} \circ \cdots \circ \mathbf{z} \circ b_k \circ \mathbf{z} \in \Xi^m.$$

It is clear that the running time of Gen is d'.

Next, for every $i \in [k]$, $(a_1, b_1), \ldots, (a_k, b_k) \in I_{\mathsf{ED}_\Sigma}(d)$, and an ED alignment $\widetilde{\sigma} : [m] \to [m] \cup \{\perp\}$ for $(\mathbf{a}, \mathbf{b})$ given as input to Dec, the algorithm first runs Gen to compute $(\mathbf{a}, \mathbf{b})$ and then outputs $\sigma : [n] \to [n] \cup \{\perp\}$ which is computed as follows:

$$\forall j \in [n],\ \sigma(j) = \begin{cases} \widetilde{\sigma}(j + \mathtt{loc}) \text{ if } \mathtt{loc} < \widetilde{\sigma}(j+\mathtt{loc}) \leq n + \mathtt{loc}, \\ \perp \text{ otherwise,} \end{cases}$$

where $\mathtt{loc} = (n+\ell)(i-1)$. It is easy to see that σ is an ED alignment for (a_i, b_i). Also, it is easy to see that the running time of Dec is running time of Gen plus d' (needed to compute σ).

We next show that if $\widetilde{\sigma}$ is an optimal ED alignment for $(\mathbf{a}, \mathbf{b})$ then σ is an optimal alignment for (a, b).

To show this we first prove that there exists an optimal alignment $\widetilde{\sigma}'$ which is an optimal ED alignment for $(\mathbf{a}, \mathbf{b})$ and it is 'block-consistent'. Similarly to the LCS case,

[7] Here again we use the unary encoding to encode a symbol in Σ for the ease of presentation.

every optimal alignment must satisfy: $\widetilde{\sigma}(i) \in \{i - \ell/2, \ldots, i + \ell/2\}$ (as otherwise its cost exceeds the identity alignment cost). To simplify the notations we define for each $j \in [k]$ the starting and the end indices of each block a_i, specifically: $s(a_j) = (\ell - 1 + n)(j-1) + 1, f(a_j) = (\ell - 1 + n)(j-1) + n$.

We denote by $\Delta(\widetilde{\sigma})_j$ the number of edit operations made by $\widetilde{\sigma}$ on the j^{th} block of $(\mathbf{a}, \mathbf{b})$, i.e., it equals twice the cardinality of the preimage of $\perp$ plus the number of mismatches for indices in $j \in \{s(a_j), \ldots s(a_j) + n + \ell\}$

Claim 5.9. *There exists an optimal alignment $\widetilde{\sigma}'$ which is an optimal* ED *alignment for $(\mathbf{a}, \mathbf{b})$ which is 'block-consistent', that is: For all $j \in [k]$, if $j > 1$ then,*

$$\widetilde{\sigma}'(s(a_j) - 1) = s(a_j) - 1 \text{ and } \widetilde{\sigma}'(f(a_j) + 1) = f(a_j) + 1.$$

Moreover,

$$\Delta(\widetilde{\sigma}')_j = \Delta_e(a_j, b_j).$$

Proof of Claim 5.9. We take any optimal alignment $\widetilde{\sigma}$ and gradually change it to be 'block-consistent' while preserving its cost. This is done as follows: We define $\widetilde{\sigma}^0 = \widetilde{\sigma}$. For each $j \in [k]$ we define $\widetilde{\sigma}^j = \widetilde{\sigma}^{j-1}$ and then convert it to be consistent on the j^{th} block by:

- Deleting all the characters in a_j that were mapped into $\mathbf{z}$ and then matching all the characters of $\mathbf{z}$ (formally, for each $i \in \{s(a_j), \ldots, f(a_j)\}$ if $\widetilde{\sigma}^{j-1}(i) > f(a_j)$ set $\widetilde{\sigma}^j(i) = \perp$);
- Matching all the characters in j^{th} block of $\mathbf{z}$ (formally, for each $i \in \{f(a_j) + 1, \ldots, s(a_{j+1}) - 1\}$ set $\widetilde{\sigma}^j(i) = i$);
- Each character in the prefix of a_{j+1} that was mapped into $\mathbf{z}$ in $\widetilde{\sigma}^{j-1}$ is deleted (formally for for each $i \in \{s(a_{j+1}), \ldots, f(a_{j+1})\}$ if $\widetilde{\sigma}^{j-1}(i) < s(a_{j+1})$ set $\widetilde{\sigma}^j(i) = \perp$).

Finally set $\widetilde{\sigma}' = \widetilde{\sigma}^k$.

We next claim that $\Delta(\widetilde{\sigma}^j) \leq \Delta(\widetilde{\sigma}^{j-1})$ (and by the optimality of $\widetilde{\sigma}^0$ we get an optimality of $\widetilde{\sigma}^j$): Let $\text{disp}_{\widetilde{\sigma}^j}(i) = i - \widetilde{\sigma}^j(i)$. The proof proceeds by a case analysis:

Case 1 – $\text{disp}_{\widetilde{\sigma}^{j-1}}(f(a_j) + 1) > 0$: Let us compare $\Delta(\widetilde{\sigma}^{j-1})$ and $\Delta(\widetilde{\sigma}^j)$: In $\widetilde{\sigma}^j$ we delete $\text{disp}_{\widetilde{\sigma}^{j-1}}(f(a_j) + 1)$ symbols that were not deleted in $\widetilde{\sigma}^{j-1}$. However, on each such a deletion in $\widetilde{\sigma}^{j-1}$ we payed for a mismatch. Next, in both $\widetilde{\sigma}^{j-1}, \widetilde{\sigma}^j$ we pay no edit operations as long as we match between $\mathbf{z}$ characters and then:

If $\text{disp}_{\widetilde{\sigma}^{j-1}}(s(a_{j+1})) > 0$: In $\widetilde{\sigma}^j$ we pay $\text{disp}_{\widetilde{\sigma}^{j-1}}(s(a_{j+1}))$ additions the prefix of b_{j+1}. On the other hand, in $\widetilde{\sigma}^{j-1}$ we pay this much of substitutions caused by matching $\mathbf{z}$ characters into b_{j+1}.

If $\text{disp}(\widetilde{\sigma}^{j-1}(s(a_{j+1})) < 0$: In this case in $\widetilde{\sigma}^{j-1}$ we pay at least $\text{disp}(\widetilde{\sigma}^{j-1}(s(a_{j+1}))$ mismatches and characters insertions till we reach an index i $\text{disp}(\widetilde{\sigma}^{j-1}(i) \geq s(a_{j+1})$. On the other hand in $\widetilde{\sigma}^j$ we pay this much of characters deletions to delete the prefix of a_{j+1}. Overall, the total costs of the alignments are equal.

Case 2 – disp$(\widetilde{\sigma}^{j-1}(f(a_j)+1)) < 0$: We handle similarly using the same arguments.

To prove the moreover part, observe that $\widetilde{\sigma}'$ matches all the $\mathbf{z}$ blocks to themselves. Therefore, if there exists a block $j \in [k]$ which $\widetilde{\sigma}'$ does not align optimally a_j into b_j then there exists a better alignment than $\widetilde{\sigma}'$. Contradicting to the optimality of $\widetilde{\sigma}'$. □

To conclude the correctness of our algorithm Dec, observe that it mimics the behavior $\widetilde{\sigma}'$ on the j^{th} block of $(\mathbf{a}, \mathbf{b})$. Since $\Delta(\widetilde{\sigma}')_j = \Delta_e(a_j, b_j)$ the proof follows. Finally, we note that the bound on the functions S and T hold as $d' = (nk + nk^2)2|\Xi| \leq 3|\Xi|dk^2$. □

ED problem has also been of special interest in the last few years thanks to the advancements in fine grained complexity [BI18, AHWW16, Wil15, Wil16, Wil18]. The above direct product feasibility immediately implies the below hardness amplification result.

Corollary 5.10. *Let $\varepsilon > 0$. Let Σ be a finite non-empty set. Let Ξ be a superset of Σ of cardinality $|\Sigma| + 1$. Let $\mathcal{D} = \{D_d\}_{d\in\mathbb{N}}$ be a family of distributions such that for every randomized algorithm $\mathcal{A}$ running in time $d^{1+\varepsilon}$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.*

- *D_d is a distribution over $I_{\mathsf{ED}_\Sigma}(d)$ where an instance of $I_{\mathsf{ED}_\Sigma}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.*
- $\Pr_{(a,b)\sim D_d}[\mathcal{A}$ *finds optimal* ED *alignment for (a, b) with probability at least $2/3$*$] \leq 1 - d^{-o(1)}$.

Then there is some $\varepsilon' > 0$ such that there is a distribution family $\mathcal{D}' = \{D'_d\}_{d\in\mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $d^{1+\varepsilon'}$, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_{\mathsf{ED}_\Xi}(d^{1+o(1)})$ and an instance in I_{ED_Ξ} can be sampled from D'_d in $d^{1+o(1)}$ time.*
- $\Pr_{(a',b')\sim D'_d}[\mathcal{A}'$ *finds optimal* ED *alignment for (a', b') with probability at least $2/3$*$] \leq 0.01$.

Proof. We apply Theorem 3.4 by setting, $\Pi = I_{\mathsf{ED}_\Sigma}$, $\Lambda = I_{\mathsf{ED}_\Xi}$, $p = 2/3$, $S(d,k) = 3|\Xi|dk^2$, $T(d,k) = 2 \cdot S(d,k)$, $v(d) = w \cdot d$ (for some $w \in \mathbb{N}$), $s(d) = \widetilde{O}(d)$, $t(d) = d^{1+\varepsilon}$, and $\mathsf{fail}(d) = 1/d^{o(1)}$. Then we have that $k := d^{o(1)}$ and $c := 300 \ln 3$. We verify that $k \cdot s(d) + T(d,k) + v(d) \leq \frac{t(d)}{2c}$ holds by noting that $k \cdot s(d) + T(d,k) + v(d) = d^{1+o(1)}$ and $\frac{t(d)}{2c} = \Omega(d^{1+\varepsilon})$. Therefore, we have that the sampling time from D'_d is $d^{1+o(1)}$ and that the theorem statement holds for any randomized algorithm $\mathcal{A}'$ running in time $\frac{t(d)}{2c} = \Theta(d^{1+\varepsilon}) = \Theta\left(\left(d^{1+o(1)}\right)^{1+\varepsilon-\delta}\right)$, for any $\delta > 0$. □

Strangely, our technique fails to immediately extend to another similarity search problem, namely computing the Fréchet distance, which is studied together with edit distance and LCS in literature.

Remark 5.11 (Fréchet Distance)**.** *Note that another well studied subquadratic hard* [Bri14, AHWW16] *problem of computing Fréchet Distance does not seem to admit direct product feasibility (at least the natural way to aggregate fails).*

5.3 Matrix Multiplication

Now we consider the matrix multiplication problem. It may be written as an optimization problem in the following rather mundane way:

Definition 5.12 (Matrix Multiplication problem)**.** *Let q be some large prime (universal constant). The Matrix Multiplication (*Mult*) is an optimization problem charaterized by the following quadruple of objects $(I_{\mathsf{Mult}}, \mathsf{Sol}_{\mathsf{Mult}}, \Delta_{\mathsf{Mult}}, \min)$, where:*

- *For every $d \in \mathbb{N}$, $I_{\mathsf{Mult}}(d)$ is the set of all pairs of $n \times n$ matrices with entries in $\mathbb{F}_q$;*
- *For every $(A, B) \in I_{\mathsf{Mult}}$ we have $\mathsf{Sol}_{\mathsf{Mult}}(A, B)$ is the set of all $n \times n$ matrices with entries in $\mathbb{F}_q$;*
- *For every $(A, B) \in I_{\mathsf{Mult}}$ and every $n \times n$ matrix C we define $\Delta_{\mathsf{Mult}}(A, B, C)$ to be the number of entries in C which differ from AB.*

Given the above formalism, we show that it is self direct product feasible.

Lemma 5.13. *Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $S(d, k) = (dk)^2$ and $T(d, k) = O(d^2k^2)$. Then we have that* Mult *is (S, T)-self direct product feasible.*

Proof. We define the pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ below.

For every $(A_1, B_1), \ldots, (A_k, B_k) \in I_{\mathsf{Mult}}(d)$ given as input to Gen, it outputs the instance (A', B') in $I_{\mathsf{Mult}}(d')$ defined as:

$$A' := \begin{bmatrix} A_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & A_k \end{bmatrix}$$

and:

$$B' := \begin{bmatrix} B_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & B_k \end{bmatrix}$$

It is clear that the running time of Gen is $O(d')$.

Next, for every $i \in [k]$, $(A_1, B_1), \ldots, (A_k, B_k) \in I_{\mathsf{Mult}}(d)$, and a matrix $C' \in \mathsf{Sol}_{\mathsf{Mult}}(A', B')$ given as input to Dec, the algorithm first runs Gen to compute (A', B') and then outputs $C \in \mathsf{Mult}(A_i, B_i)$ which is the i-th block of C'.

We next show that if C' is an optimal Mult matrix for (A', B') then C is an optimal alignment for (A_i, B_i). An optimal solution for (A', B') is the matrix $C' = A' \cdot B'$. By matrix multiplication definition its i-th block of C' equals $A_i \cdot B_i$ which is indeed the optimal solution for (A_i, B_i).

Clearly the running time of Dec is $O(d^2k^2)$, the proof follows. □

It is a well-known open problem [Kün18, WW18] to find an efficient way to check if the product of two matrices is equal to the third matrix. However, it can be checked in linear time (in the input size) if we allow randomness [Fre77]. But Theorem 3.4 needs

the verification to be deterministic so we cannot invoke it directly. Let us briefly explain how to modify the proof so it generalizes to the case of matrix multiplication.

First, observe that using [Fre77] result, given $n \times n$ matrices A, B, and C in $\mathbb{F}_q$ one can check whether $C = A \cdot B$ with failure probability less than 2^{-m} in time mn^2.

Recall the proof of Theorem 3.4 and let us restate it in matrix multiplication terminology: Given input matrices A, B, the algorithm repeatedly define matrices A', B' such that A, B are nested as a random sub-block of A', B' as defined in the proof of Lemma 5.13. This above process is repeated for c-times. The matrices A', B' are fed into algorithm $\mathcal{A}'$, and the output of $\mathcal{A}'$ is the product $A' \cdot B'$.

We modify the algorithm so that it iterates over its main loop for $2c$ iterations. In such a way, using the same argument described in the proof of Theorem 3.4, we are guaranteed that, on at least $1 - \mathsf{fail}(d)$ fraction of instances sampled from original distribution, the probability that none of the matrices output by $\mathcal{A}'$ equals $A' \cdot B'$ is at most:

$$\left(1 - \frac{\ln\left(1/1-p\right)}{c}\right)^{2c} \leq 1 - (e^{\ln(1-p)})^2 \leq 1 - 2p.$$

Now suppose that on at least one of the iterations $\mathcal{A}'$ outputs a matrix C' that equals $A' \cdot B'$ (which now happen with probability at least $2p$). We run [Fre77] algorithm on each matrix output by $\mathcal{A}'$, so that we are guaranteed that with probability at least $1 - p$ for each such a matrix, the algorithm outputs 'equal' iff $C' = A' \cdot B'$. In total, the success probability of our modified algorithm is at least $2p - p = p$.

Summarizing, on at least $1 - \mathsf{fail}(d)$ fraction of inputs sampled from the original distribution, our modified algorithm outputs $A \cdot B$ with probability at least p, as claimed. Thus, with a setting of parameters as in the proofs of Corollaries 5.4 and 5.10, we have the following.

Corollary 5.14. *Let $\varepsilon > 0$. Let $\mathcal{D} = \{D_d\}_{d \in \mathbb{N}}$ be a family of distributions such that for every randomized algorithm $\mathcal{A}$ running in time $d^{1+\varepsilon}$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.*

- *D_d is a distribution over $I_{\mathsf{Mult}}(d)$ where an instance of $I_{\mathsf{Mult}}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.*
- $\Pr_{(A,B)\sim D_d}[\mathcal{A}$ *finds correct product of* (A, B) *with probability at least* $2/3] \leq 1 - d^{-o(1)}$.

Then there is some $\varepsilon' > 0$ such that there is a distribution family $\mathcal{D}' = \{D'_d\}_{d \in \mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $d^{1+\varepsilon'}$, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_{\mathsf{Mult}}(d^{1+o(1)})$ and an instance in I_{Mult} can be sampled from D'_d in $d^{1+o(1)}$ time.*
- $\Pr_{(A',B')\sim D'_d}[\mathcal{A}'$ *finds correct product of* (A', B') *with probability at least* $2/3] \leq 0.01$.

6 Almost Worst Case to Average Case for Problems in TFNP

In this section, we look at two problems in TFNP: Factoring and End of a Line. We also point towards future directions of research in the intersection of hardness amplification and TFNP.

6.1 Factoring

Factoring is the problem of given a number as input, the task of determining all its prime factors. Given a candidate set of prime factors for a number it is possible to efficiently check that they are prime numbers, factors of the input number, and exhaustive, i.e., there are no more prime factors of the input number. This is captured in our formalism for optimization problems as follows.

Definition 6.1 (Prime factor set of a number)**.** *For every positive integer $n > 1$ and every set $\Gamma \subseteq [n]$, we say that Γ is a prime factor set of n if the following holds.*

- ***Divisor**: For every $a \in \Gamma$, we have that a divides n.*
- ***Prime**: For every $a \in \Gamma$, we have that a is prime (greater than 1).*

Definition 6.2 (Factoring problem)**.** *The Factoring problem (*Factor*) is an optimization problem charaterized by the following quadruple of objects $(I_{\mathsf{Factor}}, \mathsf{Sol}_{\mathsf{Factor}}, \Delta_{\mathsf{Factor}}, \max)$, where:*

- *For every $d \in \mathbb{N}$, $I_{\mathsf{Factor}}(d) = \{2, \ldots, 2^d + 1\}$;*
- *For every $n \in I_{\mathsf{Factor}}$ we have $\mathsf{Sol}_{\mathsf{Factor}}(n)$ is the set of all prime factor sets of n;*
- *For every $n \in I_{\mathsf{Factor}}$ and every prime factor set Γ of n, we define $\Delta_{\mathsf{Factor}}(n, \Gamma)$ to be the cardinality of Γ.*

Now, we show that factoring is self direct product feasible.

Lemma 6.3. *Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $S(d, k) = dk$ and $T(d, k) = \widetilde{O}(dk)$. Then we have that* Factor *is (S, T)-self direct product feasible.*

Proof. We define the pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ below.

For every $n_1, \ldots, n_k \in I_{\mathsf{Factor}}(d)$ given as input to Gen, it outputs the instance n' in $I_{\mathsf{Factor}}(d')$ (where $d' \leq dk$) defined as:

$$n' := \prod_{i \in k} n_i.$$

It is clear that the running time of Gen is $\widetilde{O}(d')$.

Next, for every $i \in [k]$, $n_1, \ldots, n_k \in I_{\mathsf{Factor}}(d)$, and a Factor assignment $\Gamma' \in \mathsf{Sol}_{\mathsf{Factor}}(n)$ for n' given as input to Dec, the algorithm first runs Gen to compute n' and then outputs $\Gamma \in \mathsf{Sol}_{\mathsf{Factor}}(n)$ which is computed as follows: We initialize $\Gamma = \emptyset$. Then we scan Γ' and for each $a' \in \Gamma'$ we insert a' in Γ if a' divides n_i.

We next show that if Γ' is an optimal Factor factor set for n' then Γ is an optimal factor set for n_i. Indeed if Γ' is an optimal solution for n' then it equals the set of *all* prime factors of n', which include all prime factors of n_i. Hence each prime factor a of n will be inserted into Γ during the scan of Γ'.

Clearly the running time of Gen is $\widetilde{O}(dk)$, since the check whether each $a \in \Gamma'$ divides n_i takes $\widetilde{O}(d)$-time, the proof follows. □

The factoring problem is known to be in the class PPA (under randomized reductions) [Jer16]. It's also the basis of many cryptosystems. Therefore the following hardness amplification result is of interest.

Corollary 6.4. *Let $a \geq 8$. Let $\mathcal{D} = \{D_d\}_{d \in \mathbb{N}}$ be a family of distributions such that for every randomized algorithm $\mathcal{A}$ running in time $O(d^a)$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.*

- *D_d is a distribution over $I_{\mathsf{Factor}}(d)$ where an instance of $I_{\mathsf{Factor}}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.*
- $\Pr_{n \sim D_d}[\mathcal{A}$ *finds largest prime factor set of n with probability at least* $2/3] \leq 1 - \frac{1}{d}$.

Then for $m := \Theta(d^7)$, there is a distribution family $\mathcal{D}' = \{D'_d\}_{d \in \mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $O(m^{a/7})$ over inputs of size m, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_{\mathsf{Factor}}(m)$ and an instance in I_{Factor} can be sampled from D'_d in $\widetilde{O}(m)$ time.*
- $\Pr_{n' \sim D'_d}[\mathcal{A}'$ *finds largest prime factor set of n' with probability at least* $2/3] \leq 0.01$.

Proof. We apply Theorem 3.4 by setting, $\Pi = \Lambda =$ Factor , $p = 2/3$, $S(d,k) = dk$, $T(d,k) = (dk)^{1+o(1)}$, $v(d) = d^{1+o(1)}$, $s(d) = \widetilde{O}(d)$, $t(d) = d^a$, and $\mathsf{fail}(d) = 1/d$. Then we have that $k := \Theta(d^6)$ and $c := 300 \ln 3$. We verify that $k \cdot s(d) + T(d,k) + v(d) \leq \frac{t(d)}{2c}$ holds by noting that $k \cdot s(d) + T(d,k) + v(d) = \widetilde{O}(d^7)$ and $\frac{t(d)}{2c} = \Omega(d^a) = \Omega(d^8)$ (because $a \geq 8$). Therefore, we have that the sampling time from D'_d is $\widetilde{O}(d^7) = \widetilde{O}(m)$ and that the theorem statement holds for any randomized algorithm $\mathcal{A}'$ running in time $\frac{t(d)}{2c} = \Theta(d^a) = \Theta(m^{a/7})$. □

We wonder if the above result can have any meaningful, concrete applications in cryptography.

6.2 End of a Line and Nash Equilibrium

The End of a Line problem [Pap94, DGP09] is the canonical PPAD-complete problem. Informally it captures the handshaking lemma in directed graphs on 2^n vertices given as input through predecessor and successor circuits of $poly(n)$ size. It may be written as an optimization problem under our formalism in the following (mundane) way.

Definition 6.5 (Size of a Circuit)**.** *For every positive integer n and every circuit $C : \{0,1\}^n \to \{0,1\}^n$, the size of C denoted by $|C|$ is the number of bits needed to describe C.*

Definition 6.6 (End of a Line problem)**.** *The End of a Line problem (*EoL*) is an optimization problem charaterized by the following quadruple of objects $(I_{\mathsf{EoL}}, \mathsf{Sol}_{\mathsf{EoL}}, \Delta_{\mathsf{EoL}}, \max)$, where:*

- *For every $d \in \mathbb{N}$, $I_{\mathsf{EoL}}(d) = \{(A,B) \mid A, B : \{0,1\}^n \to \{0,1\}^n$ and $|A| + |B| \leq d\}$;*
- *For every $(A,B) \in I_{\mathsf{EoL}}$ we have $\mathsf{Sol}_{\mathsf{EoL}}(A,B) = \{0,1\}^n$;*
- *For every $(A,B) \in I_{\mathsf{EoL}}$ and every $x \in \{0,1\}^n$, we define $\Delta_{\mathsf{EoL}}(A,B,x)$ as follows:*

$$\Delta_{\mathsf{EoL}}(A,B,x) = \begin{cases} 1 \text{ if } A(B(x)) \neq x \text{ or } B(A(x)) \neq x \neq 0^n \\ 0 \text{ otherwise} \end{cases}$$

Lemma 6.7. *Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $S(d,k) = dk$ and $T(d,k) = O(dk)$. Then we have that* Factor *is (S,T)-self direct product feasible.*

Proof. We define the pair of deterministic algorithms $(\mathsf{Gen}, \mathsf{Dec})$ below.

Fix $k, d \in \mathbb{N}$. For every $(A_1,B_1),\ldots,(A_k,B_k) \in I_{\mathsf{EoL}}(d)$ given as input to Gen, it outputs the instance (A',B') in $I_{\mathsf{EoL}}(d')$ where $d' = dk$, and $A', B' : \{0,1\}^{nk} \to \{0,1\}^{nk}$ are defined as follows. For all $x' = (x'_1,\ldots,x'_k) \in \{0,1\}^{nk}$, where $\forall i \in [k]$, we have $x'_i \in \{0,1\}^n$, and define $A'(x')$ and $B'(x')$ as follows.

$$A'(x') = (A_1(x'_1),\ldots,A_k(x'_k))$$
$$B'(x') = (B_1(x'_1),\ldots,B_k(x'_k))$$

It is clear that the running time of Gen is $O(d')$.

Next, for every $i^* \in [k]$, $(A_1,B_1),\ldots,(A_k,B_k) \in I_{\mathsf{EoL}}(d)$, and a EoL solution $x' \in \{0,1\}^{nk}$ for (A',B') given as input to Dec, the algorithm first runs Gen to compute (A',B') and then outputs $x \in \{0,1\}^n$ which is computed as follows: $x_j = x'_{(i^*-1)n+j}$.

We next show that if x' is an optimal EoL solution for (A',B') then x is an optimal solution for (A_{i^*},B_{i^*}). This follows easily by the fact that the (A_i,B_i)s are defined on disjoint union of circuits, hence an optimal solution for (A',B') induces an optimal solution for each of the (A_i,B_i)s.

The running times of Gen and Dec trivially follow. □

Corollary 6.8. *Let $a \geq 8$. Let $\mathcal{D} = \{D_d\}_{d\in\mathbb{N}}$ be a family of distributions such that for every randomized algorithm $\mathcal{A}$ running in time $O(d^a)$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.*

- *D_d is a distribution over $I_{\mathsf{EoL}}(d)$ where an instance of $I_{\mathsf{EoL}}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.*
- $\Pr_{(A,B)\sim D_d}[\mathcal{A} \text{ finds a solution of } (A,B) \text{ with probability at least } 2/3] \leq 1 - \frac{1}{d}$.

Then for $m := \Theta(d^7)$, there is a distribution family $\mathcal{D}' = \{D'_d\}_{d\in\mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $O(m^{a/7})$ over inputs of size m, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_{\mathsf{EoL}}(m)$ and an instance in I_{EoL} can be sampled from D'_d in $\widetilde{O}(m)$ time.*
- $\Pr_{(A',B')\sim D'_d}[\mathcal{A}' \text{ finds a solution of } (A',B') \text{ with probability at least } 2/3] \leq 0.01.$

Proof. We apply Theorem 3.4 by setting, $\Pi = \Lambda = \mathsf{EoL}$, $p = 2/3$, $S(d,k) = dk$, $T(d,k) = w \cdot dk$ (for some $w \in \mathbb{N}$), $v(d) = w' \cdot d$ (for some $w' \in \mathbb{N}$), $s(d) = \widetilde{O}(d)$, $t(d) = d^a$, and $\mathsf{fail}(d) = 1/d$. Then we have that $k := \Theta(d^6)$ and $c := 300 \ln 3$. We verify that $k \cdot s(d) + T(d,k) + v(d) \leq \frac{t(d)}{2c}$ holds by noting that $k \cdot s(d) + T(d,k) + v(d) = \widetilde{O}(d^7)$ and $\frac{t(d)}{2c} = \Omega(d^a) = \Omega(d^8)$ (because $a \geq 8$). Therefore, we have that the sampling time from D'_d is $\widetilde{O}(d^7) = \widetilde{O}(m)$ and that the theorem statement holds for any randomized algorithm $\mathcal{A}'$ running in time $\frac{t(d)}{2c} = \Theta(d^a) = \Theta(m^{a/7})$. □

Since there is a direct many-one reduction from EoL to the problem of computing approximate Nash equilibrium in various kinds of games [CDT09, Rub18, Rub16], the above corollary extends to give hardness amplification results for computing Nash equilibrium as well.

Remark 6.9. *The proof of Lemma 6.7 can be mimicked to get self direct product feasibility of canonical complete problems of other* TFNP *classes like Leaf (complete for* PPA*) [Pap94], LocalOPT (complete for* PLS*) [JPY88, DP11], End of Metered Line (equivalent to EoPL)[HY17], and Unique EOPL [FGMS18].*

Now we consider the same hardness amplification result of EoL but against subexponential time algorithms. We provide below a slightly informal statement (using asymptotic notations as opposed to providing specific constants) for clarity.

Corollary 6.10. *Let $\mathcal{D} = \{D_d\}_{d\in\mathbb{N}}$ be a family of distributions such that for every randomized algorithm $\mathcal{A}$ running in time $2^{o(d)}$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.*

- *D_d is a distribution over $I_{\mathsf{EoL}}(d)$ where an instance of $I_{\mathsf{EoL}}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.*
- $\Pr_{(A,B)\sim D_d}[\mathcal{A} \text{ finds a solution of } (A,B) \text{ with probability at least } 2/3] \leq 1 - \frac{1}{2^{o(d)}}.$

Then for $m := 2^{o(d)}$, there is a distribution family $\mathcal{D}' = \{D'_d\}_{d\in\mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}'$ running in time $m^{\omega(1)}$ over inputs of size m, the following holds for all large enough $d \in \mathbb{N}$.

- *D'_d is a distribution over $I_{\mathsf{EoL}}(m)$ and an instance in I_{EoL} can be sampled from D'_d in $\widetilde{O}(m)$ time.*
- $\Pr_{(A',B')\sim D'_d}[\mathcal{A}' \text{ finds a solution of } (A',B') \text{ with probability at least } 2/3] \leq 0.01.$

Proof. Let $h : \mathbb{N} \to \mathbb{N}$ be some slowly increasing function such that $\lim_{x\to\infty} h(x) = \infty$ and let $h := h(d)$. We apply Theorem 3.4 by setting, $\Pi = \Lambda = \mathsf{EoL}$, $p = 2/3$, $S(d,k) = dk$, $T(d,k) = w \cdot dk$ (for some $w \in \mathbb{N}$), $v(d) = w' \cdot d$ (for some $w' \in \mathbb{N}$), $s(d) = \widetilde{O}(d)$, $t(d) = 2^{d/h}$, and $\mathsf{fail}(d) = 1/2^{d/(6\cdot h^2)}$. Then we have that $k := \Theta(2^{d/h^2})$ and $c := 300 \ln 3$. We verify that $k \cdot s(d) + T(d,k) + v(d) \leq \frac{t(d)}{2c}$ holds by noting that $k \cdot s(d) + T(d,k) + v(d) = 2^{(1+o(1))\cdot d/h^2}$ and $\frac{t(d)}{2c} = \Omega(2^{d/h})$. Therefore, we have that the sampling time from D'_d is $2^{(1+o(1))\cdot d/h^2} = O(m)$ and that the theorem statement holds for any randomized algorithm $\mathcal{A}'$ running in time $\frac{t(d)}{2c} = O(2^{d/h}) = \Theta(m^{h(d)}) = m^{\omega(1)}$. □

Note that if we assume the (randomized) Exponential Time Hypothesis (ETH) for PPAD [BPR16] then we obtain a family of distributions $\mathcal{D} = \{D_d\}_{d\in\mathbb{N}}$ such that for every randomized algorithm $\mathcal{A}$ running in time $2^{d^{1-o(1)}}$ over inputs of size d, the following holds for large $d \in \mathbb{N}$.

- D_d is a distribution over $I_{\mathsf{EoL}}(d)$ where an instance of $I_{\mathsf{EoL}}(d)$ can be sampled from D_d in $\widetilde{O}(d)$ time.
- $\Pr_{\phi\sim D_d}[\mathcal{A} \text{ finds maximizing assignment for } \phi \text{ with probability at least } 2/3] \leq 1 - \frac{1}{2^{d^2}}$.

Therefore, our amplification in Corollary 6.10 is an *almost* worst-case to average-case reduction for EoL (under subexponential time reductions).

Acknowledgements

We would like to thank Amir Abboud, Irit Dinur, and Eylon Yogev for discussions and comments.

References

[Abb19] Amir Abboud. Personal communication, 2019.

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag Berlin Heidelberg, 1999.

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388, 2016.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[BI18] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018.

[BKS06] Joshua Buresh-Oppenheim, Valentine Kabanets, and Rahul Santhanam. Uniform hardness amplification in NP via monotone codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(154), 2006.

[BPR16] Yakov Babichenko, Christos H. Papadimitriou, and Aviad Rubinstein. Can almost everybody be almost happy? In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 1–9, 2016.

[BR13] Andrej Bogdanov and Alon Rosen. Input locality and hardness amplification. *J. Cryptology*, 26(1):144–171, 2013.

[Bri14] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014.

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 483–496, New York, NY, USA, 2017. ACM.

[BT06a] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1), 2006.

[BT06b] Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for NP problems. *SIAM J. Comput.*, 36(4):1119–1159, 2006.

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *J. ACM*, 56(3), 2009.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260, 2006.

[CPS99] Jin-yi Cai, Aduri Pavan, and D. Sivakumar. On the hardness of permanent. In *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, pages 90–99, 1999.

[CW16] Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255, 2016.

[DG08] Irit Dinur and Elazar Goldenberg. Locally testing direct product in the low error range. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 613–622, 2008.

[DGP09] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009.

[Din16] Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *ECCC*, 23:128, 2016.

[DP11] Constantinos Daskalakis and Christos H. Papadimitriou. Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 790–804, 2011.

[DS14] Irit Dinur and David Steurer. Direct product testing. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 188–196, 2014.

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 534–543, 2002.

[FGMS18] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *CoRR*, abs/1811.03841, 2018.

[FK00] Uriel Feige and Joe Kilian. Two-prover protocols - low error at affordable rates. *SIAM J. Comput.*, 30(1):324–346, 2000.

[Fre77] Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP Congress*, pages 839–842, 1977.

[GG11] Parikshit Gopalan and Venkatesan Guruswami. Hardness amplification within NP against deterministic algorithms. *J. Comput. Syst. Sci.*, 77(1):107–121, 2011.

[GIL+90] Oded Goldreich, Russell Impagliazzo, Leonid A. Levin, Ramarathnam Venkatesan, and David Zuckerman. Security preserving amplification of hardness. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 318–326, 1990.

[GO05] Venkatesan Guruswami and Ryan O'Donnell. *Lecture 12: Feige-Kilian "confuse/match" games (Part 1)*. Lecture notes for CSE 533: The PCP Theorem and Hardness of Approximation. University of Washington, 2005.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[GR17] Oded Goldreich and Guy N. Rothblum. Worst-case to average-case reductions for subclasses of P. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:130, 2017.

[GR18] Oded Goldreich and Guy N. Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 77–88, 2018.

[HVV06] Alexander Healy, Salil P. Vadhan, and Emanuele Viola. Using nondeterminism to amplify hardness. *SIAM J. Comput.*, 35(4):903–931, 2006.

[HY17] Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1352–1371, 2017.

[IKW12] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. New direct-product testers and 2-query pcps. *SIAM J. Comput.*, 41(6):1722–1768, 2012.

[Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 538–545, 1995.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. Preliminary version in CCC'99.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. Preliminary version in FOCS'98.

[IW97] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 220–229, 1997.

[Jer16] Emil Jerábek. Integer factoring and modular square roots. *J. Comput. Syst. Sci.*, 82(2):380–394, 2016.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[Kün18] Marvin Künnemann. On nondeterministic derandomization of freivalds' algorithm: Consequences, avenues and algorithmic progress. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 56:1–56:16, 2018.

[Lip89] Richard J. Lipton. New directions in testing. In *Distributed Computing And Cryptography, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, October 4-6, 1989*, pages 191–202, 1989.

[MR16] Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense CSPs. *CoRR*, abs/1607.02986, 2016.

[O'D04] Ryan O'Donnell. Hardness amplification within np. *J. Comput. Syst. Sci.*, 69(1):68–94, 2004.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[Raz98] Ran Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998.

[Rub16] Aviad Rubinstein. Settling the complexity of computing approximate two-player nash equilibria. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 258–265, 2016.

[Rub18] Aviad Rubinstein. Inapproximability of nash equilibrium. *SIAM J. Comput.*, 47(3):917–959, 2018.

[STV01] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001.

[Tre03] Luca Trevisan. List-decoding using the XOR lemma. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 126–135, 2003.

[Tre05] Luca Trevisan. On uniform amplification of hardness in NP. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 31–38, 2005.

[TV07] Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[Wil15] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *IPEC*, pages 17–29, 2015.

[Wil16] Virginia Vassilevska Williams. Fine-grained algorithms and complexity (invited talk). In *STACS*, pages 3:1–3:1, 2016.

[Wil18] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proc. Int. Cong. of Math.*, volume 3, pages 3431–3472, 2018.

[WW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91, 1982.

A Missing Proofs

An elementary proof of Lemma 3.5 may be found in the lecture notes of Guruswami and O'Donnell [GO05] and we reproduce it below.

Proof of Lemma 3.5. We first show that the following holds:

$$\mathop{\mathbb{E}}_{\substack{x\sim\mathcal{D}\\ i\sim[k]}}[(\mu_{i,x}-\mu)^2] \leq \frac{1}{k}. \tag{4}$$

In order to prove (4), we introduce some notations:

- We denote elements of X^k by $\bar{x}$, and denote by $\mathcal{D}^k$ the k-wise product distribution of $\mathcal{D}$.
- Fix $i \in [k], x \in X$, let $\mathcal{D}^{k,-i,x}$ be the distribution over $\bar{x} \in X^k$ obtained by picking $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k$ from $\mathcal{D}$ (independently) and setting $\bar{x} = (x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_k)$.
- Fix $i \in [k]$, define $F^i : X \to \mathbb{R}$ and $\sigma_i \in \mathbb{R}$ as follows:

$$\forall x \in X,\ F^i(x) := \mathop{\mathbb{E}}_{\bar{x}\sim\mathcal{D}^{k,-i,x}}[f(\bar{x}) - \mu] \text{ and } \sigma_i := \mathop{\mathbb{E}}_{x\sim\mathcal{D}}[F^i(x)^2].$$

- Fix $i \in [k]$ define $f^i : X^k \to \mathbb{R}$ by setting for all $\bar{x} \in X^k$, $f^i(\bar{x}) := F^i(\bar{x}_i)$.

With these notations setup, we have:

$$\mathop{\mathbb{E}}_{\substack{x\sim\mathcal{D}\\ i\sim[k]}}[(\mu_{i,x}-\mu)^2] = \mathop{\mathbb{E}}_{i\sim[k]}\left[\mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[F^i(x)^2\right]\right] = \frac{1}{k}\sum_{i\in[k]}\sigma_i.$$

Therefore, in order to show (4), it suffices to show:

$$\sum_{i\in[k]}\sigma_i \leq 1. \tag{5}$$

We observe that from linearity of expectation, the following holds:

$$\forall i \in [k],\ \mathop{\mathbb{E}}_{x\sim\mathcal{D}}[F^i(x)] = 0. \tag{6}$$

Also from the independent choice of coordinates in $\bar{x}$ and (6), we have for all $i, j \in [k]$ such that $i \neq j$ we have:

$$\mathop{\mathbb{E}}_{\bar{x}\sim\mathcal{D}^k}\left[f^i(\bar{x})f^j(\bar{x})\right] = 0. \tag{7}$$

Finally we define $g : X^k \to \mathbb{R}$ as follows:

$$\forall \bar{x} \in X^k,\ g(\bar{x}) = \sum_{i \in [k]} f^i(\bar{x}).$$

Notice that

$$0 \leq \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[(f(\bar{x}) - g(\bar{x}))^2 \right] = \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[f^2(\bar{x}) \right] - 2 \cdot \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[f(\bar{x}) g(\bar{x}) \right] + \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[g^2(\bar{x}) \right]. \tag{8}$$

We bound each of the above terms separately. First, we have that $\mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[f^2(\bar{x}) \right] = \mu$ since f is a boolean valued function. Second, we have

$$\begin{aligned}
\mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[f(\bar{x}) g(\bar{x}) \right] &= \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[f(\bar{x}) \cdot \sum_{i \in [k]} f^i(\bar{x}) \right] \\
&= \sum_{i \in [k]} \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[f(\bar{x}) \cdot f^i(\bar{x}) \right] \\
&= \sum_{i \in [k]} \mathop{\mathbb{E}}_{x \sim \mathcal{D}} \left[F^i(x) \cdot \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^{k,-i,x}} \left[f(\bar{x}) \right] \right] \\
&= \sum_{i \in [k]} \mathop{\mathbb{E}}_{x \sim \mathcal{D}} \left[F^i(x) \cdot (\mu + F^i(x)) \right] && \left(\text{because } F^i(x) = \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^{k,-i,x}} [f(\bar{x}) - \mu] \right) \\
&= \sum_{i \in [k]} \mu \cdot \mathop{\mathbb{E}}_{x \sim \mathcal{D}} \left[F^i(x) \right] + \sum_{i \in [k]} \mathop{\mathbb{E}}_{x \sim \mathcal{D}} \left[\left(F^i(x) \right)^2 \right] \\
&= \sum_{i \in [k]} \mathop{\mathbb{E}}_{x \sim \mathcal{D}} \left[\left(F^i(x) \right)^2 \right] && \text{(from (6))} \\
&= \sum_{i \in [k]} \sigma_i.
\end{aligned}$$

Third, we have

$$\begin{aligned}
\mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[g^2(\bar{x}) \right] &= \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[\left(\sum_{i \in [k]} f^i(\bar{x}) \right)^2 \right] \\
&= \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[\sum_{i,j \in [k]} f^i(\bar{x}) \cdot f^j(\bar{x}) \right] \\
&= \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[\sum_{\substack{i,j \in [k] \\ i \neq j}} f^i(\bar{x}) \cdot f^j(\bar{x}) \right] + \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[\sum_{i \in [k]} \left(f^i(\bar{x}) \right)^2 \right] \\
&= \mathop{\mathbb{E}}_{\bar{x} \sim \mathcal{D}^k} \left[\sum_{i \in [k]} \left(f^i(\bar{x}) \right)^2 \right] && \text{(from (7))} \\
&= \sum_{i \in [k]} \sigma_i.
\end{aligned}$$

Plugging all together in (8), we get:

$$0 \leq \mu - 2 \cdot \sum_{i \in [k]} \sigma_i + \sum_{i \in [k]} \sigma_i.$$

Since $\mu \leq 1$ and we get $\sum_{i \in [k]} \sigma_i \leq 1$, as claimed. Thus we have shown (5) holds and consequently (4) holds. The proof of the lemma then follows from a simple application of Markov inequality:

$$\begin{aligned}
\Pr_{\substack{x \sim \mathcal{D} \\ i \sim [k]}} \left[|\mu_{i,x} - \mu| \geq \frac{1}{k^{1/3}} \right] &= \Pr_{\substack{x \sim \mathcal{D} \\ i \sim [k]}} \left[(\mu_{i,x} - \mu)^2 \geq \frac{1}{k^{2/3}} \right] \\
&\leq k^{2/3} \cdot \mathop{\mathbb{E}}_{\substack{x \sim \mathcal{D} \\ i \sim [k]}} [(\mu_{i,x} - \mu)^2] && \text{(from Markov inequality)} \\
&\leq \frac{k^{2/3}}{k} && \text{(from (4))} \\
&= \frac{1}{k^{1/3}} && \square
\end{aligned}$$

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il