A Survey on TFNP and its Applications

Krish Singal

December 30th 2022

Preface

This survey paper is the culmination of my reading course done under Professor Christos Papadimitriou in Fall of 2022. I started by reading Tim Roughgarden's *Twenty Lectures in Algorithmic Game Theory* and pivoted from there to learn more about functional complexity classes, existence theorems, and their applications to cryptography and game theory. Towards the second half of the semester, I focused more on complexity theory and its applications to cryptography through *Computational Complexity: A Modern Approach* by Sanjeev Arora and Boaz Barak. Throughout the paper, I included pedantic examples and theorems that I proved as exercises when learning the material. The hope is for this work to be an overview of the key points in complexity theory and its amazing applications to a wide variety of problems.

Contents

1 Functional Classes and Total Functions

1.1 Functional Classes

Classical complexity theory deals with the notion of "decision problems" that admit a binary answer as to whether the instance is positive or negative with respect to the class being considered. Analog functional complexity classes exist for each canonical decision problem based complexity class and are denoted by **FC** where **C** is a decision problem based class. We formalize the definition using **FNP** as an example below.

Definition 1. Let $R \subset \Sigma^* \times \Sigma^*$ be a binary relation. We call R polynomially decidable if there exists a deterministic polynomial time Turing Machine that decides that language $\{(x, y)|(x, y) \in R$. We call R polynomially balanced if $|y| \leq |x|^k$ for some $k \in \mathbb{Z}^+$.

From this definition, it is easy to see that we can define the class **NP** equivalently as

Definition 2. Language $L \in$ **NP** iff there exists an polynomially decidable and balanced binary relation R such that $L = \{x| \exists y\ (x, y) \in R\}$

Definition 3. Let $L \in$ **NP** and R_L be its corresponding binary relation. We define the corresponding function problem FL. It can be viewed as a function $FL(x) = y$ such that $R_L(x, y)$ or $FL(x) = -1$ if no such y exists.

Definition 4. **FNP** is the set of function problems FL such that $L \in$ **NP**

An analogous definition can be used to define **FP** and, in general, the entire polynomial hierarchy. However, of most interest to us will be **FNP** and its subclasses.

Example 1. Consider the following problems and their functional counterparts

(a) SINGLE SOURCE SHORTEST PATH: The problem of determining whether a given weighted graph G has a path of length $< k$ between source s and node t is in **P** . It follows that the corresponding functional problem to return such a path is in **FP** .

(b) HAMILTONIAN PATH: The problem of determining whether a given graph G has a Hamiltonian path is known to be in **NP** . Then, the corresponding functional problem to return such a path is in **FNP** .

(c) SAT: The problem of determining whether a given boolean formula ϕ has a satisfying assignment is in **NP** . The corresponding functional problem to return a satisfying assignment is in **FNP** .

With this new category of complexity classes, it is natural to reason about how completeness looks in the changed landscape.

Definition 5. Language P reduces to language Q if there exists poly-time computable functions f and g such that $\forall x \in P\ f(x) \in Q$ and if y is a valid output of $f(x)$, then $g(y)$ is a valid output for x.

Theorem 1. FP = **FNP** if and only if P = NP

Proof. Note that the reductions in functional classes follow the same dependency graph as that of the corresponding decision problem class. To see this, let f be a poly-time computable reduction from problem A to B in class C. Then, for functional problems FA and FB, there is a corresponding reduction q between possible outputs of FA and possible outputs of FB. Thus, $g = q^{-1}$, and (f, g) is a functional reduction.

From this, we conclude that FSAT is **FNP** -complete since the reduction graph remains the same. If **FNP** = FP, FSAT $\in$ FP and it trivially follows that SAT $\in$ P since any polynomial time deterministic turing machine that can solve FSAT can also decide SAT by a constructive algorithm.

If P = NP, then SAT $\in$ P. We show, via self-reducibility, that this implies FSAT $\in$ FP. Consider the following constructive algorithm

Algorithm 1 Self-Reduction

```
procedure DTM D(SAT SOLVER, φ)
    if SAT SOLVER(φ) = False then
        return False
    Assignments ← [0, ..., 0]
    for variable x_m ∈ φ do
        if SAT SOLVER(φ_m^T) = True then
            Assignments(m) ← True
        else
            Assignments(m) ← False
     return Assignments
```

Where ϕ_m^T denotes the boolean formula ϕ with x_m set to 1. Notice that if ϕ is satisfiable, it must be the case that some assignment of x_m results in a satisfying assignment. Therefore, if ϕ_m^T is not satisfiable, ϕ must be satisfiable with x_m set to false. Observe that the above self-reduction of SAT from a decision to a search problem requires $O(n \cdot \text{TIME(SAT SOLVER)})$ where TIME(SAT SOLVER) $= poly(n)$. Thus, TIME(D) $= poly(n)$ and it follows that FSAT $\in$ FP.

□

1.2 Total Functions

In some sense, the extension to functional complexity is aligned with how we think about algorithm design in the real world. In many use cases, simply knowing whether a solution exists or not is insufficient. Yet, for a small class of problems, we are guaranteed existence – reducing the task to searching for (one of) the successful construction(s).

Definition 6. Let FL be a function problem with corresponding language L and binary relation R_L. Then FL is *total* if for every x there exists a y such that $(x, y) \in R_L$.

Corollary 1. Notice that by definition 3 and definition 6, we can conclude that for any function problem FL that is total, $L = \Sigma^*$.

Example 2. Consider existence theorems such as the fundamental theorem of arithmetic that show the existence of a unique prime factorization of every integer. Although existence of an answer is guaranteed, constructing it is considered difficult.

We can then define the class **TFNP**

Definition 7. Let FL be a function problem. $FL \in \mathbf{TFNP}$ if $FL \in \mathbf{FNP}$ and FL is total.

Intuitively, we can think of **TFNP** as the class of problems that are guaranteed an answer that is exceedingly difficult to find. As is the topic of sections 2 and 3, the combination of totality and intractability has applications to many familiar problems. For now, we explore some of the characteristics of the class **TFNP** further.

Theorem 2. $TFNP = F(NP \cap \ coNP)$

Proof. We proceed by showing inclusion in both directions. Let FL $\in$ **TFNP** . Then, by definition $FL \in \mathbf{FNP}$ and thus $FL \in F(\mathbf{NP} \cap \mathbf{coNP})$. Therefore, $\mathbf{TFNP} \subseteq F(\mathbf{NP} \cap \mathbf{coNP})$.

Let $FL \in F(\mathbf{NP} \cap \mathbf{coNP})$. Clearly $FL \in FNP$. Note $L \in (\mathbf{NP} \cap \mathbf{coNP})$ and it follows that every instance has either a succinct proof or disqualification. Then, consider the polynomially decidable and balanced binary relation $R_L(x, y)$ (where the DTM used returns an OR of the NP verifier and coNP verifier). Because every instance x has a succinct proof or disqualification, it follows that $\forall x.\ \exists y.\ (x, y) \in R_L$. Thus, the function problem FL is total. □

Theorem 3. If a **TFNP** problem is **NP** -Hard, then **NP** = **coNP**

Proof. Let S be **TFNP** function problem that is **NP** -Hard. By definition, any language $L \in \mathbf{NP}$ can be polynomially reduced to S. Furthermore, since $S \in \mathbf{TFNP}$, there exists a polynomial verifier V for S. A reduction from a decision problem to a search problem is defined such that given a solution y to an instance s of S, there exists a poly time deterministic Turing machine M such that

$$x \in L \iff M(x, f(x), y) = 1$$

In other words, given a solution to $f(x)$ we can efficiently compute whether $x \in L$ (where $f(x)$ is an instance of S). Because S is total, it then follows that

$$x \notin L \iff \exists y.\ V(f(x), y) = 1 \text{ and } M(x, f(x), y) = 0$$

Which implies that $L \in \mathbf{coNP}$. Thus, **NP** = **coNP**.

□

2 Existence Theorems

This section details some of the existence theorems I found to be the most fascinating during my foray into the topic. For the purposes of this paper, I discuss the low dimensional cases for intuition. Corresponding discussion of each in arbitrary dimensions are referenced at the end of the paper.

Theorem 4. (Brouwer's Fixed Point Theorem) Every continuous map $f : B^2 \to B^2$ has a fixed point.

Note that in the one dimensional case, the theorem is a direct corollary of the intermediate value theorem.

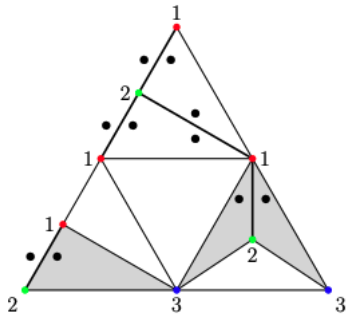
Theorem 5. (Borsuk-Ulam Theorem) Every continuous map $f : S^2 \to \mathbb{R}^2$ has a point $x \in S^2$ such that $f(-x) = -f(x)$.

Corollary 2. (Bisection Theorem) Let P and Q be two bounded polygonal regions in $\mathbb{R}^2$. Then, there exists a line in $\mathbb{R}^2$ that bisects P and Q simultaneously.

The above celebrated theorems require knowledge of basic knowledge of algebraic topology such as homotopy type, retractions, and fundamental groups. While these concepts are outside the scope of this paper, resources for learning more are referenced at the end.

Theorem 6. (Sperner's Lemma) Let triangle ABC lie in $\mathbb{R}^2$ and let T be a triangulation of ABC. Define a Sperner triangulation to consist of three colors (1, 2, 3) such that vertices A, B, C are colored distinctly and vertices (formed by T) that lie along any edge of ABC are colored by only the two colors on the endpoints of its constituent edge. Triangle ABC with arbitrary triangulation and Sperner coloring contains at least one complete intermediate triangle (where complete implies a triangle with distinctly colored vertices).

Proof. We will show a stronger statement – that the number of complete interior triangles is odd. First consider the 1-2 edges comprising of endpoints of color 1 and 2 by placing a dot on either side of the edge. We now wish to count the number of dots inside triangle ABC. Note that a 1-2 edge on the boundary will contribute 1 dot to the interior while a 1-2 edge in the interior will contribute 2. Because triangle ABC is complete and all vertices on the 1-2 edge of ABC must be colored either 1 or 2, it follows that there are an odd number of 1-2 edges on the boundary of ABC. Therefore, there are an odd number of dots in the interior of ABC.



On the other hand, consider interior triangles in the triangulation of T. A complete triangle will house exactly one dot while all other triangles will house 2. Because there are an odd number of dots inside ABC, there must also then be an odd number of complete interior triangles. □

3 TFNP and Cryptography

In the last section, we explored a variety of different existence theorems that gave rise to TFNP problems. This gives us a natural (yet painstaking) method to populate TFNP with problems that have known existence theorems. However, this reveals a glaring hole in the potential existence of complete problems for TFNP. TFNP is known as a *semantic* complexity class wherein the lack of rigid commonalities between languages of this class makes reductions ill-defined. In the case of TFNP, membership in the class is governed by whether a problem has a corresponding existence theorem. As illustrated in section 2, these existence theorems come in all shapes and sizes, so reducing a problem that is guaranteed an answer due to the pigeonhole principle to a problem that is guaranteed an answer due to Brouwer fixed point theorem is not well defined.

TFNP is then naturally partitioned into sub-classes based on the existence theorem that credits membership.

3.1 PPP and PWPP

PPP or polynomial pigeon-hole principle is the complexity class defined by problems that are guaranteed answers due to the pigeon-hole principle.

Example 3. The canonical PPP-complete problem is EQUAL OUTPUTS. Given a boolean circuit C with n input gates and n output gates, find either (1) an input x that outputs 0^n or (2) x, y such that $C(x) = C(y)$. It is easy to see that an answer here is guaranteed by the pigeon-hole principle.

Example 4. EQUAL SUMS: Given n positive integers $a_1, ..., a_n$ such that $\sum_{i=1}^{n} a_i < 2^n - 1$. Find two different sets of such n-tuples that have the same sum.

Here, we show that EQUAL SUMS $\in$ **PPP**. Note that there are

$$\sum_{s=n}^{2^n-2} \binom{s-1}{n-1}$$

many subsets of positive integers with sum $< 2^n - 1$. Using the combinatorial hockey stick identity,

$$= \binom{2^n - 2}{n} \approx \frac{(2^n-2)^n}{n!} = O(\frac{2^{n^2}}{n!}) > 2^n$$

by Stirling's approximation. Thus, two different subsets with the same sum exist by the pigeon-hole principle.

Similar to **PPP**, WEAK EQUAL OUTPUTS is defined to be Given a boolean circuit C with n input gates and $n-1$ output gates, find x, y such that $C(x) = C(y)$. This problem is complete for the subclass **PWPP**(polynomial weak pigeonhole principle).

3.2 Lattice Based Cryptography

Definition 8. Let $B = \{b_1, ..., b_n\}$ be a basis for $\mathbb{R}^n$. Then, $L = \{\sum_{i=1}^{n} z_i b_i | z_i \in \mathbb{Z}\}$ is an n-dimensional lattice.

Lattice-based cryptography is a technique that relies on the apparent computational hardness of lattice problems such as SVP (Shortest Vector Problem) that computes the non-zero vector $v \in L$ with smallest Euclidean length. It is considered a candidate for post-quantum cryptographic schemes based on current research. In particular, one problem in particular – Shortest Integer Solution (SIS) was used to construct a family of one way functions that is secure if SVP is hard in the worst case.

Definition 9. (Shortest Integer Solution Problem) Given a matrix $A \in \mathbb{Z}_q^{r \times t}$ where q is a power of 2 and $t > r \log q$, return $x, x' \in \{0,1\}^t$ such that $Ax = Ax'$

Notice that $t > r \log q \implies 2^t > q^r$. The problem then trivially reduces to WEAK EQUAL OUTPUTS and it follows that $SIS \in \mathbf{PWPP}$. However, it is unknown whether SIS is **PWPP**-complete. A constrained version of the problem known as weakly constrained SIS (wc-SIS) is known to be **PWPP**-complete.

Definition 10. (Weakly Constrained Shortest Integer Solution Problem) Given matrices $A \in \mathbb{Z}_q^{r \times t}$ and $B \in \mathbb{Z}_q^{d \times t}$ where q is a power of 2, $t > (r + d) \log q$, and B is of the form

$$\begin{bmatrix} 2^0 & 2^1 & ... & 2^{l-1} & ... & ... & ... & ... \\ 0 & 2^0 & 2^1 & ... & 2^{l-1} & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & ... & 0 & 2^0 & 2^1 & ... & 2^{l-1} & ... \end{bmatrix}$$

where $l = l \log q$ and row i contains $(i-1)l$ leading zeros followed by the first l powers of 2. Then, return $x, x' \in \{0,1\}^t$ such that $Ax = Ax'$ and $Gx = Gx' = 0$ simultaneously.

Theorem 7. wc-SIS $\in$ **PWPP**

Proof. Notice that we can easily construct a solution x to $Gx = 0$. Setting the last $t - dl$ entries arbitrarily, the first dl entries are uniquely defined by setting each block of l from $(i-1)l$ to il. Naturally, we obtain the following reduction to the WEAK EQUAL OUTPUTS problem.

Construct circuit $C : \{0,1\}^{t-dl} \to \{0,1\}^{rl}$ that given $x \in \{0,1\}^{t-dl}$ computes the unique vector $v = [u, x]$ such that $Gx = 0$ then returns Av. It is clear then that wc-SIS $\in$ **PWPP**. □

Theorem 8. wc-SIS is **PWPP**-complete

The proof of this fact is out of scope for this paper. However, it follows that problems in cryptography are rooted in their use and construction of one way hash functions that inevitably tie the field closely to **TFNP** . It is worth studying this connection more deeply since we do not know whether cryptography truly "exists". In particular, $\mathbf{P} \neq \mathbf{NP} \iff$ cryptography exists, but we do not know whether **NP** being hard in the average case implies that one way functions and other cryptographic primitives exist. As we saw in section 1, **TFNP** problems are unlikely to be **NP** -hard, and therefore provide a source of "easier" intermediate problems whose hardness may be easier to connect with the existence of cryptography.

4 Average Case Complexity

The interaction between TFNP and cryptography acts via average case complexity, a theory devised by Levin. At its core, Levin proposed that

4.1 Distributions

Levin formalized the notions of polynomially sampleable and computable distributions as follows

Definition 11. $D = \{D_n\}$ is a family of distributions, with D_n being a probability density function over $\{0,1\}^n$

Definition 12. A distribution D is **P**-computable if there exists a deterministic polynomial time algorithm A that can compute the cumulative density function on D

$$\mu_{D_n}(x) = \sum_{x' \in \{0,1\}^n | x' \leq x} \mathbb{P}_{D_n}[x']$$

We denote the set of **P**-computable distributions as $\mathbf{P_{COMP}}$.

Corollary 3. Let D be a **P**-computable distribution. Then, there exists a deterministic polynomial time algorithm that can compute the probability density function on D

Proof. Notice that

$$\mathbb{P}_{D_n}(x) = \mu_{D_n}(x) - \mu_{D_n}(x-1)$$

then, construct algorithm A' that runs algorithm A to compute μ_{D_n} and computes adjacent differences. □

Definition 13. A distribution D is **P**-sampleable if there exists a randomized polynomial time algorithm A such that $A(1^n) \sim D_n$ (i.e. the output of A on the all ones string is identically distributed to D_n). We denote the set of **P**-sampleable distributions as $\mathbf{P_{SAMP}}$.

Intuitively, Levin described a scenario where the distributions we care about are those that can be sampled polynomially... We observe the natural relationship between the two classes of distributions.

Theorem 9. $\mathbf{P_{COMP}} \subseteq \mathbf{P_{SAMP}}$

Proof. Let $D \in \mathbf{P_{COMP}}$ where A is a deterministic polynomial time algorithm that can compute μ_{D_n}. We construct randomized polynomial time algorithm A' such that $A(1^n) \sim D_n$. Let $q(n)$ be the polynomial describing the runtime of algorithm A. Then, A' on input 1^n, flips $q(n)$ many random coins to generate a $p \in [0,1]$ with $q(n)$ finite precision.

A' performs binary search on μ_{D_n} to find (and output) $x^* \in \{0,1\}^n$ such that $\mu_{D_n}(x^*) \geq p > \mu_{D_n}(x^* - 1)$. During this binary search, $\mu_{D_n}(y)$ is only computed if y is the current middle element of comparison. Thus, the runtime of A' is $q(n)\log(n)$ and therefore polynomial time. □

Notice the use of $q(n)$ many coins specifically in the above proof. Because D is **P**-computable, the pdf on D_n can be computed in polynomial time, implying that $\mathbb{P}(x)$ for each x has finite precision. To ensure distinguishability between the density function on different inputs, algorithm A' generates random p with $q(n)$ precision.

Theorem 10. If $\mathbf{P}^{\#\mathbf{P}} \neq \mathbf{P}$, then $\mathbf{P_{SAMP}} \not\subseteq \mathbf{P_{COMP}}$

Proof. Assume that $\mathbf{P}^{\#\mathbf{P}} \neq \mathbf{P}$. Since $\mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}$ trivially, it follows that $\mathbf{P}^{\#\mathbf{P}} \not\subseteq \mathbf{P}$. Thus, there exists a language L such that $L \in \mathbf{P}^{\#\mathbf{P}}$ but $L \notin \mathbf{P}$. Let D denote the Turing Machine with oracle access to $\mathbf{P}^{\#\mathbf{P}}$ that decides L. There must exist an $x \in L$ such that D makes a counting query q which cannot be computed in polynomial time. That is, q is a 'hard' counting query.

We show that this implies there exists a distribution that is **P**-sampleablebut not **P**-computable. Consider the probability density function p that assigns probability $\frac{\#q}{2^n}$ to the 0^n state and $1 - \frac{\#q}{2^n}$ where n is chosen such that $2^n > \#q$. Then, note that this function is **P**-sampleablebecause randomized algorithm A can simply choose an input $x \in \{0,1\}^n$ uniformly at random and simulate polynomial verifier M (that exists due to $q \in \#\mathbf{P}$). If $M(x) = 1$, A outputs 0^n, otherwise it outputs 1^n.

To see that this distribution cannot be **P**-computable, we show the contrapositive. That is, if p is **P**-computable, then $\#q$ can be computed in polynomial time. This is trivially true since $\mathbb{P}[0^n] \cdot 2^n = \#q$, so if $\mathbb{P}[0^n]$ can be computed in polynomial time, $\#q$ can be computed in polynomial time. By assumption $\#q$ is 'hard' and cannot be computed in polynomial time, thus p is not **P**-computable.

In all, distribution p is then **P**-sampleablebut not **P**-computable. Thus, $\mathbf{P_{SAMP}} \not\subseteq \mathbf{P_{COMP}}$. □

4.2 Distributional Problems

We define the average case complexity with respect to a distribution on inputs for a particular problem.

Definition 14. $\langle L, D\rangle$ where $L \subseteq \{0,1\}^*$ and $D = \{D_n\}$ is a family of distributions, with D_n being a distribution over $\{0,1\}^n$, is a *distributional problem.*

Naturally, this leads us to define the average case analogs of **P** and **NP**, **distP** and **distNP** respectively.

Definition 15. A distributional problem $\langle L, D\rangle \in \mathbf{distP}$ if there exists a deterministic polynomial time algorithm A such that

$$\exists C, \epsilon \in \mathbb{R}. \forall n \in \mathbb{N}. \mathbb{E}_{x \sim D_n}[\frac{\text{time}_A(x)^\epsilon}{n}] \leq C$$

Talk about why this is the "right" definition for distP.

Definition 16. A distributional problem $\langle L, D\rangle \in \mathbf{distNP}$ if $L \in NP$ and $D \in \mathbf{P_{COMP}}$

4.3 One-Way Functions

Definition 17. (Negligible Function) We call a function $\epsilon : \mathbb{N} \to [0,1]$ negligible if $\epsilon(n) = n^{-\omega(1)}$.

Definition 18. (One way function) We call a function $f : \{0,1\}^* \to \{0,1\}^*$ one-way if it is polynomial time computable and for every randomized polynomial time algorithm there exists a negligible function ϵ such that

$$\mathbb{P}_{x\in\{0,1\}^n, y\in Im(f)}[A(y) = x \text{ where } f(x) = y] < \epsilon(n)$$

Theorem 11. If OWF's exist, then $\mathbf{P} \neq \mathbf{NP}$

Proof. We prove the contrapositive. Suppose $\mathbf{FP} = \mathbf{FNP}$ and f is a one way function. Note that the problem of computing $f^{-1}(y)$ where $y \in \text{Im}(f)$ is clearly in **FNP** (consider the NDTM that branches on all possible $x \in \{0,1\}^n$ and accepts if $f(x) = y$). If $\mathbf{FNP} = \mathbf{FP}$, then there exists polynomial time DTM D that solves f^{-1}. Then,

$$\mathbb{P}_{x\in\{0,1\}^n, y\in Im(f)}[A(y) = x \text{ where } f(x) = y] = 1 > \epsilon(n)$$

for any negligible function $\epsilon(n)$. Thus, f cannot be a one-way function. □

Theorem 12. If OWF's exist, then $\mathbf{distNP} \not\subseteq \mathbf{distP}$

Proof. We prove the contrapositive. Suppose $\mathbf{distNP} \subseteq \mathbf{distP}$ and f is a one way function. Note that the problem of computing $f^{-1}(y)$ where $y \in \text{Im}(f)$ is clearly in **FNP**. Then, $\langle f^{-1}, D\rangle$, where D is a **P**-computabledistribution over $\{0,1\}^n$, is in **distFNP**. By assumption, there then exists deterministic polynomial time algorithm A such that

$$\exists C, \epsilon \in \mathbb{R}.\forall n \in \mathbb{N}.\mathbb{E}_{x\sim D_n}[\frac{\text{time}_A(x)^\epsilon}{n}] \leq C$$

By Markov's inequality, we get that for $k > 1$,

$$\mathbb{P}[\frac{\text{time}_A(x)^\epsilon}{n} < kC] = \mathbb{P}[\text{time}_A(x) < (kCn)^{\frac{1}{\epsilon}}] > 1 - \frac{1}{k}$$

which is a constant. Therefore, there cannot always exist a negligible function that upper bounds the correctness probability of A and we can conclude that f cannot be a one-way function. □

4.4 Connections to TFNP

5 Conclusion

References

[1] Sanjeev Arora and Boaz Barak (2016) *Computational Complexity: A Modern Approach*.

[2] Christos Papadimitriou (1993) *Computational Complexity*, Addison Wesley.

[3] Daniel Mitropolsky (2022) *A Survey of TFNP-Cryptography*.

[4] Christos Papadimitriou (1992) *On The Complexity of the Parity Argument and Other Inefficient Proofs of Existence*.

[5] Katerina Sotiraki, Manolis Zampetakis, Giorgos Zirdelis (2018) *PPP-Completeness with Connections to Cryptography*.

[6] James Munkres (2014) *Topology*.