Algebraic Fingerprints for faster algorithms

Ioannis Koutis
Computer Science Dept.
U of Puerto Rico-Rio Piedras
ioannis.koutis@upr.edu

Ryan Williams
Computer Science Dept.
Stanford University
rrw@cs.stanford.edu

ABSTRACT

There has recently been impressive progress –after nearly fifty years of stagnation– in algorithms that find exact solutions for certain hard computational problems, including the famous Hamiltonian path problem. This progress has been due to a few core ideas that have found several applications. A unifying theme is algebra: we 'transform' the given problem into a more general algebraic format, then solve the corresponding algebraic problem that arises.

This article walks the reader through some of these exciting developments and the underlying ideas. It also puts them in context with the discovery process that led to them, highlighting the role of *parameterization* as a way of dealing with intractability.

1. INTRODUCTION

It was a major surprise when, in 2010, Andreas Björklund discovered what many previously thought impossible: a significantly improved algorithm for the famous *Hamiltonian path* problem, also known as Hamiltonicity [4]. Hamiltonicity asks if a given graph contains a path that goes through each vertex exactly once, as illustrated in Figure 1.

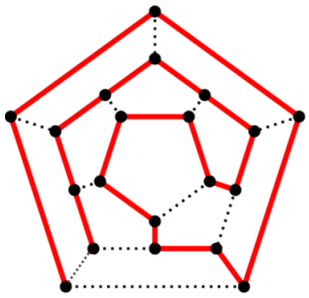

Figure 1: A Hamiltonian Path.

Hamiltonicity was one of the first problems shown to be NP-complete by Karp [20]. The only known algorithms for NP-complete problems require time scaling **exponentially** with the size of the input. It is believed that they can't be solved much faster in general, although the possibility hasn't been ruled out [17]. Given an undirected graph on n vertices, Björklund's algorithm can find a Hamiltonian path or report that no such path exists in $O^*(1.657^n)$ time[1]. The algorithm still runs in exponential time but it is much faster than the $O^*(2^n)$ running time of the previously fastest algorithm, known since the 1960's [3, 19].

Hamiltonicity is a prominent algorithmic problem. Many researchers before Björklund have tried their hand at it, without success. But some of the tools that Björklund used, didn't become available until 2009, thanks to progress in the k-path problem, a related problem in the context of **parameterized algorithms**.

1.1 A parameterized race and a conjecture

The k-path problem is a natural parameterized analogue of Hamiltonicity. The goal now is to find in the given graph a path of length k for some specified value of k, rather than a path of length n. It could be argued that, from a practical standpoint, the k-path problem is better motivated relative to Hamiltonicity. Algorithms designed specifically for the k-path problem have been actually used to detect signaling pathways in large protein interaction networks [29].

Most NP-complete problems have similar parameterizations, where the parameter k measures the **solution size** to the given instance. This is natural because in practice, the solution length to an NP-hard problem is often short relative to the length of the overall instance. Downey and Fellows [12] observed that some parameterized problems appear to be 'easier' than others. To capture that, they defined the class of **Fixed Parameter Tractable** (FPT) problems. The definition of FPT is rather simple. The running time of a parameterized algorithm should clearly depend on the input size n, and the parameter k. But the dependence should be somewhat special: a problem is defined to be FPT if it can be solved by an algorithm in $O^*(f(k))$ time.

The FPT notion has proven to be extremely insightful and has led to a more detailed understanding of problem hardness. It is believed that not all NP-hard problems are FPT; a prominent example is the k-clique problem, where the goal is to detect a k-subset of the network nodes that are all pairwise connected (see Figure 2). The best known algorithm for k-clique is not much better than exhaustive search, which takes time $O(n^k)$. However, experience shows that when a problem is shown to be FPT, algorithmic progress has just begun as it is often the case that the function $f(k)$ in the

[1] $O^*(f(k))$ means a function smaller than $p(n)\cdot f(k)$ for some polynomial $p(n)$.

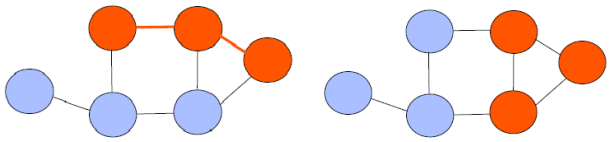

Figure 2: A 3-path and a 3-clique. Finding a k-clique appears to be much harder than finding a k-path.

running time can be improved.

The k-path problem is a prime example. The problem was shown to be FPT by Monien [25] who described an algorithm running in $O^*(k!)$ time. This was later improved to $O^*((2e)^k)$ by Alon et al. [2] and to $O^*(4^k)$ by Chen et al. [10]. Each of these steps represents the introduction of a significant new idea. Similarly, for many FPT problems there are now **races** for faster parameterized algorithms [27], which have enriched the field of algorithm design with new techniques.

However with each improvement, one is faced with the question: is further progress possible? Known algorithms for the 'parent' NP-complete problem can be a valuable guide to answering this question. This is because their running time sets a **clear target** for the parameterized algorithm. In the case of Hamiltonicity, we know of an algorithm that runs in $O^*(2^n)$ time. It is then reasonable to **conjecture** that there is an algorithm for k-path that runs in $O^*(2^k)$ time, 'matching' the Hamiltonian path algorithms for the extreme value of the parameter.

Roadmap. The development of the algorithmic techniques we are about to discuss was largely driven by the k-path conjecture. But for the sake of simplicity we will illustrate them in the context of another important NP-complete problem known as the 3D-matching problem, which has also played a central role. In fact, the Hamiltonicity algorithm was preceded by another record-breaking algorithm for the 3D-matching problem, also due to Björklund [5].

In this article, our first step towards faster algorithms is what we call an **algebraization** of the combinatorial problem, i.e. capturing the problem as a question about the existence of certain monomials in a polynomial. We discuss one algebraization of the 3D-matching problem in Section 2. Given the polynomial, the original combinatorial question becomes now an issue of **algebraic detection** of monomials; that is, extracting information from the polynomial by assigning values into its variables. Looking for assignments that fit our purpose will lead us to calculations **modulo 2**, over an algebra in which sums of pairs cancel out. This in turn will present us with a problem of *unwanted cancelations* of monomials. We will solve it with the method of **fingerprints** which augments the 3D-matching polynomial in order to make sure that no unwanted cancelations occur. The k-path conjecture was answered in the positive via a generalization of algebraic detection with fingerprints, which also yielded many other faster parameterized algorithms. These ideas are reviewed in Section 3. In Section 4, we arrive at Björklund's key twist that helped unlock the faster Hamiltonicity algorithm. While still employing in a relaxed way the method of fingerprints, Björklund treated computation modulo 2 not just as a nuisance, but also as a resource, by presenting algebraizations where many *unwanted monomials* *cancel out*, because they come in pairs. In order to derive these sharper algebraizations that **exploit cancelations**, Björklund tapped into the power of algebraic combinatorics that study graphs via linear algebra. Faster algorithms for several other parameterized problems followed. We highlight one of these advances in Section 5.

2. ALGEBRAIZATION

Consider the following fictional airline scheduling problem: on any given day there is a set X of designated captains, a set Y of designated second officers, and a set Z of destinations. Each captain and second officer declare a number of preferred destinations. The airline would like to accommodate as many preferences as possible; but at the same time, they would like to match on the same flight a pilot and second officer who have flown together before. This creates a number of triples: if captain x has previously flown with second officer y and they would both like to fly to destination z, they define a triple $\{x, y, z\}$. Of course, any given captain, second officer or destination can appear in many triples. But to maximize utility, the airline would like to select a big matching, i.e. a subset of triples that are pairwise disjoint; based on it they can then schedule the personnel. The NP-complete problem asks for a matching of maximum size, while the parameterized problem asks for a k-matching with at least k triples, which we also call a k-3D matching. An example of the problem is shown in Figure 3.

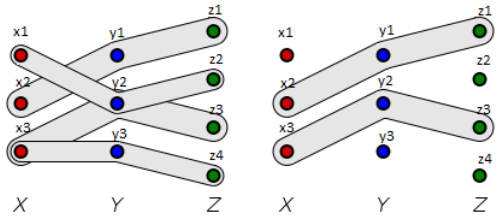

Figure 3: A set of triples and a matching

2.1 A polynomial for 3D-matching

It was probably understood by many before it was made explicit in [21] that we can view the k-3D-matching problem through an algebraic lens. Let us explain how, by means of the example in Figure 3.

First, we view as a **variable** each element of the sets X, Y, Z. That is we have the variables $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3\}$ and $Z = \{z_1, z_2, z_3, z_4\}$. Then for each triple we construct a **monomial**, by taking the product of the corresponding variables. In our example of Figure 3, we have the monomials

$$\{x_1y_2z_2, x_2y_1z_1, x_3y_2z_3, x_3y_3z_4\}.$$

We also define the **instance polynomial** P_1 to be the sum of these monomials. Finally, we set $P_k = P_1^k$; we will call P_k the kth **encoding polynomial**. For instance, for the 2nd encoding polynomial, we have

$$P_2 = (x_1y_2z_2 + x_2y_1z_1 + x_3y_2z_3 + x_3y_3z_4)^2.$$

To see the motivation behind these definitions, consider

the expansion of P_2 into a **sum-product form**.

$$\begin{aligned} P_2 &= (x_1y_2z_2)^2 + (x_2y_1z_1)^2 \\ &\quad +(x_3y_2z_3)^2 + (x_3y_3z_4)^2 \\ &\quad +2x_1x_2y_1y_2z_1z_2 + 2x_1x_3y_2^2z_2z_3 \\ &\quad +2x_1x_3y_2y_3z_2z_4 + 2x_2x_3y_1y_2z_1z_3 \\ &\quad +2x_2x_3y_1y_3z_1z_4 + 2x_3^2y_2y_3z_3z_4. \end{aligned} \tag{2.1}$$

The monomials in the sum-product expansion fall into two basic classes. In the **solution part** of the expansion, monomials that correspond to solutions of the problem are **linear**, i.e. the product does not contain squares or other high powers of variables. As an example, the monomial $x_2x_3y_1y_3z_1z_4$ in equation 2.1 is linear, and corresponds to selecting the second and fourth triples. In the complementary **non-solution part**, each non-linear monomial corresponds to a 'non-solution', i.e. a choice of non-disjoint triples.

This construction can be generalized to any instance of the problem; the only difference would be that if we want to check for a k-matching, we would have to look at the sum-product expansion of the polynomial P_k. Summarizing, we have the following:

Observation. *An instance of the 3D-matching problem contains a matching of size k if and only if the sum-product expansion of the kth encoding polynomial P_k contains a linear monomial.*

Therefore, to solve the 3D matching problem, it would suffice to expand the polynomial P_k into a sum of products and check if the sum contains a linear monomial. However an $O^*(2^N)$ *dynamic programming* algorithm is known, for $N = |X|+|Y|+|Z|$. On the other hand, the number of possible monomials in N variables is much larger than $O^*(2^N)$. Thus, the simple idea of expanding P_k into a sum of products does not give a good algorithm.

2.2 The dynamic programming algebra

Dynamic programming often involves inductive definitions that are tedious to state; the analogue in our algebraic setup is perhaps more intuitive. It can be naturally viewed as a 'truncated' expansion of the polynomial P_k into a sum-product form. More concretely, imagine expanding P_k slowly, one multiplication at a time. We observe that monomials containing squared variables can be thrown away as soon as they are formed, because they don't affect the presence of linear monomials in the full sum-product expansion of P_k. There are 2^N linear monomials in N variables. This implies that the 'truncated' expansion can be carried out in $O^*(2^N)$ time. If there is at least one monomial left in the final truncated sum-product form of P_k, we can conclude that the given instance contains a 3D matching of size k. Otherwise, it doesn't.

More formally, we are computing the sum-product expansion of P_k in an extended *dynamic programming algebra of polynomials* which has all the usual rules for multiplication and addition of polynomials, plus the additional rule that **all squared variables evaluate to zero**. Notice that this implies that all non-linear monomials also evaluate to zero. We can thus recast in algebraic terms our observation regarding linear monomials of P_k:

An instance of the 3D-matching problem contains a matching of size k if and only if the kth encoding polynomial P_k is not identical to zero in the dynamic programming algebra.

In general we can have $N = 3k$, since all elements of $X \cup Y \cup Z$ could participate in a 3D-matching. However, if we are merely looking for a 3D matching containing only k triples, then we might set as our **target** an $O^*(2^{3k})$ time parameterized algorithm.

Testing Polynomials

Testing whether a given arithmetic expression equals the unique polynomial whose coefficients are all zeroes, will be a recurring theme in the sequel.

2.3 Parameterizing via assignments

The goal of this subsection is to describe a first parameterized algorithm for the 3D-matching problem, which will demonstrate the use of assignments the variables of the polynomial in order to extract information from it.

The problem with 'parameterizing' the $O^*(2^N)$ algorithm lies clearly in the number of variables N. Trying to deal with this problem Alon, Yuster and Zwick [2] came up with a method known as color coding. Their solution can be understood as an assignment as follows. Going back to the example from Section 2.1, suppose that we are interested in finding a matching of size 2. Then, as discussed above, all linear monomials in the sum-product expansion of P_2 have degree six, i.e. they are products of six distinct variables. Alon et al. proposed the introduction of a new set W containing exactly six variables (more generally $3k$ variables for a k-matching). So, let

$$W = \{w_1, w_2, w_3, w_4, w_5, w_6\}.$$

We then perform a **random assignment** of the variables in W into the variables in X, Y, Z. We can also think of this assignment as a random coloring of the N variables with k colors. Let us consider one such assignment:

$$\begin{array}{ll} x_1 \leftarrow w_1 & x_2 \leftarrow w_4 \\ x_3 \leftarrow w_1 & y_1 \leftarrow w_3 \\ y_2 \leftarrow w_2 & y_3 \leftarrow w_5 \\ z_1 \leftarrow w_6 & z_2 \leftarrow w_6 \\ z_3 \leftarrow w_1 & z_4 \leftarrow w_2 \end{array}$$

By just substituting in equation 2.1, $P_2(X, Y, Z)$ becomes now a polynomial in the variables W:

$$\begin{aligned} P_2(W) &= (w_1w_2w_6)^2 + (w_4w_3w_6)^2 \\ &\quad +(w_1w_2w_1)^2 + (w_1w_5w_2)^2 \\ &\quad +2w_1w_2w_3w_4w_6^2 + 2w_1^3w_2^2w_6 \\ &\quad +2w_1^2w_2^2w_5w_6 + 2w_1^2w_2w_3w_4w_6 \\ &\quad +2w_1w_2w_3w_4w_5 + 2w_1^3w_2^2w_5. \end{aligned}$$

It can be seen that monomials in $P_k(X, Y, Z)$ get mapped to monomials in $P_k(W)$. If a monomial in $P_k(X, Y, Z)$ is not linear, the same holds for the corresponding monomial in $P_k(W)$. In contrast, a linear monomial of $P_k(X, Y, Z)$ may or may not **survive** as a linear monomial $P_k(W)$. In our example all but one linear monomials are mapped to a linear monomial in $P_2(W)$. So detecting a linear monomial in $P_k(W)$ allows us to infer its presence in $P_k(X, Y, Z)$ as well, and consequently the existence of a k-matching in our original problem. The inverse may be not true, as it may be the case that no linear monomial survives the assignment.

We can evaluate $P_k(W)$ in the dynamic programming algebra as we did with $P_k(X, Y, Z)$. Because now W contains only $3k$ variables, the evaluation takes $O^*(2^{3k})$ time.

However, the **probability** that any given linear monomial survives through a random color coding assignment is very small; it can be calculated to be around e^{-3k} where $e \simeq 2.72$. This means that one has to try $O(e^{3k})$ random assignments in order to have the monomial survive with an acceptable constant probability, independent from k. So, the overall running time is $O^*((2e)^{3k})$. This is still a factor of e^{3k} away from our $O^*(2^{3k})$ target.

Color Coding

Color coding has proven to be an extremely useful tool in the design of parameterized algorithms. It has been applied on a diverse list of parameterized problems, including the k-path problem for which it yields an algorithm that runs in $O^*((2e)^k)$ time.

3. ALGEBRAIC DETECTION

The methods of the previous section for detecting a linear monomial are essentially **combinatorial**, but we presented them in algebraic guise. Can we find an even faster algorithm if we use a genuinely **algebraic** method?

3.1 A matrix assignment

In Section 2.3 we considered the idea of an **assignment** into the variables of the encoding polynomial P and its subsequent **evaluation** in the dynamic programming algebra. The dynamic programming algebra and the color-coding assignment are only one possibility. There are numerous possible algebras and assignments. Matrices in particular seem to offer rich possibilities. An attractive feature of matrix algebra is that the square of a matrix can be 0, offering a genuinely algebraic way of 'implementing' the dynamic programming algebra, and specifically its rule that squares of variables should be zero. As a simple example, we have

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}^2 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

This leads us to the following list of requirements for a set of **random** matrices to be used for detecting linear monomials of degree k, i.e. products of k distinct variables.

(i) To simplify reasoning about monomials, the matrices must be pairwise **commutative**, i.e. the order in which we multiply them should not affect the value of the product.

(ii) The **square** of each matrix must be equal to **zero**. Along with commutativity, this implies that non-linear monomials will evaluate to zero as well.

(iii) Linear monomials of degree k must 'survive' the assignment, i.e. evaluate to a **non-zero** matrix, with constant probability, say at least 1/4. It would be convenient if this non-zero matrix contains only ones in its diagonal.

(iv) Since we're looking for speed, the matrices must be as **small** as possible, in order to support fast evaluation. Ideally, matrix operations should take time $O^*(2^k)$ or less.

In [22] an efficient randomized construction of such matrices was given, showing that it is possible to satisfy all these requirements if we are willing to relax our notion of zero:

By 'zero', we now mean zero modulo 2

We will refer to this as the **The Mod-2 Matrix Fact**.

We won't discuss the technical details of how to prove the Mod-2 Matrix Fact, since all we need here is to understand its algorithmic power and consequences.

Comparing with the color coding method, we see that evaluation of the polynomial takes the same time, $O^*(2^k)$ for detecting a linear monomial with k variables. The gain is in the probability that a linear monomial survives the assignment; from the exponentially small probability e^{-k}, we went to a constant probability of 1/4. This appears to suggest that we got rid of the undesired e^k factor and have reached our target of detecting degree-k linear monomials in $O^*(2^k)$ time and therefore a size k 3D-matching in $O^*(2^{3k})$ time.

Unwanted Cancelations. However, a more careful look reveals that we are not done yet due to our relaxed notion of what is zero. It is clear that the polynomial $P(X)$ evaluates to zero if it doesn't contain linear monomials. However, the Mod-2 Matrix Fact does not guarantee that $P(X)$ evaluates to non-zero, even if some of its linear monomials evaluate to non-zero. The most significant source of this problem is not in the evaluation, but rather in the polynomial P itself: by taking a look at the sum-product expansion in equation 2.1 we see that the solution part of P that consists of the linear monomials is already equal to 0 *mod* 2, just because the monomials are multiplied by 2.

3.2 The method of fingerprints

We tackle this problem of multiple copies of linear monomials by making sure that each monomial is **unique**, through the use of a set of 'fingerprint' variables $A = \{a_1, a_2, \ldots, \}$. We illustrate the idea on the k-3D-matching polynomial P_2 which we will now define as follows:

$$\begin{aligned} P_2 \quad = \quad & (a_1x_1y_2z_2 + a_2x_2y_1z_1 + a_3x_3y_2z_3 + a_4x_3y_3z_4) \\ & *(a_5x_1y_2z_2 + a_6x_2y_1z_1 + a_7x_3y_2z_3 + a_8x_3y_3z_4). \end{aligned}$$

All we are doing here is multiplying each occurrence of a triple in P_2 with a "fresh" variable from the set A. Consider then what happens to the two copies of the monomial $x_2x_3y_1y_3z_1z_4$; they are now replaced by two separate linear monomials: $a_2a_8x_2x_3y_1y_3z_1z_4$ and $a_6a_4x_2x_3y_1y_3z_1z_4$.

It appears though that the introduction of the auxiliary variables comes at the cost of getting linear monomials of higher degree, which would require larger matrices in order to survive through an evaluation, resulting in a slower algorithm.

However, an alternative idea is to use a different assignment for the A variables. For example we can assign random $\{0, 1\}$ values in the variables of A. In essence we wish to create an odd number of copies for the linear monomials, so that the solution part of the encoding polynomial is not trivially zero modulo 2. In [22] it was shown that this idea is enough for linear monomial detection to go through *in the 3D-matching case*. That is, by randomly assigning matrices to the variables X, Y, Z, and $\{0, 1\}$ values to the fingerprint variables, the polynomial will evaluate to non-zero with constant probability if its sum-product expansion contains a linear monomial and always to zero otherwise.

Target reached: We have designed an $O^*(2^{3k})$ time algorithm for the k-3D-matching problem.

3.3 A general algebraic framework

We have come awfully close to a fast parameterized algorithm for a problem much **more general** than the k-3D-matching problem: that is the detection of degree-k linear monomials in the sum-product expansion of **arbitrary** polynomials $P(X)$ that are given to us in some concise, 'unexpanded' representation. The importance of this problem was established in [22, 24] where it was observed that a fast algorithm for it implies faster algorithms for several parameterized problems, including the k-path problem. So, what do we need to do in order to generalize the method?

All steps described above carry over to arbitrary polynomials, including generating unique linear monomials with an appropriate pre-processing of the polynomial and a placement of fingerprint variables in it. However, we're still stuck with the multiple copies problem because the random $\{0,1\}$ assignment to the fingerprint variables is not guaranteed to work for any polynomial. We will thus need to generalize to some other type of assignment that works in arbitrary situations, while still being compact and easy to handle computationally.

Here is the solution proposed in [31]. Imagine evaluating $P(X,A)$ in two stages. First we assign random matrices $\bar{X}$ from the Mod-2 Matrix Fact into the X variables and compute $P(\bar{X},A)$, leaving the A variables unevaluated. The result is a matrix whose diagonal is a polynomial $Q(A)$. The Mod-2 Matrix Fact implies that $Q(A)$ is zero modulo 2 if $P(X)$ does not contain a linear monomial. If $P(X)$ does contain a linear monomial then $Q(A)$ is non-zero with probability at least $1/4$. This consideration crystallizes the problem: we wish to be able to test whether $Q(A)$ is identical to zero modulo 2, and we would like to do so by evaluating $Q(A)$ on some 'compact' assignment to its variables. This problem however has been studied and solved earlier. It is known as **identity testing**.

A solution to identity testing, known as Schwartz-Zippel Lemma [26], is simple and intuitive. The reason we picked the values $\{0,1\}$ in our assignment for the 3D-matching polynomial is that with them we can perform multiplication and addition that respects modulo 2 arithmetic. But there are other algebraic 'fields' that allow us the same kind of operations, with the bonus fact that they are larger, in the sense that they contain more values. If instead of $\{0,1\}$ we pick a field consisting of $O(k)$ values the number of possible assignments to the fingerprint variables by far outnumbers the number of possible roots of $Q(A)$, i.e. the number of assignments that make it evaluate to zero. Hence a **random assignment** from this larger field will result with high probability in $Q(A)$ evaluating to a non-zero value in the field. This allows us to claim the following result.

Fast linear monomial detection. The problem of detecting a degree-k square-free monomial in an arbitrary polynomial $P(X)$ can be solved in $O^*(2^k)$ time.

Linear monomial detection

The parameterized linear monomial detection problem along with the algorithm for its solution provide a general framework for the solution of parameterized problems. Most parameterized algorithms that rely on color coding can be accelerated by a factor of $O^*(e^k)$, by simply applying linear monomial detection, as we saw in the matching problem. In fact the acceleration is precisely e^k in most of these cases, including the k-path problem. This yields an $O^*(2^k)$ time algorithm for the k-path problem.

4. BREAKING BARRIERS

A faster parameterized algorithm for linear monomial detection would have a tremendous impact, as it would imply faster algorithms not only for many parameterized problems, but also for the corresponding non-parameterized NP-hard problems. Thus, an intriguing and natural question is whether the problem can be solved faster by evaluating the input polynomial over a more exotic algebra supporting faster operations. Unfortunately, the question has been answered in the negative; it is **impossible** to find a better algebra [24].

This algebraic barrier suggests that new techniques are required for further progress in the linear detection problem, if progress is possible at all. However, one can be more optimistic about specific problems. Linear monomial detection is very general and completely agnostic to combinatorial properties of the underlying problem, so taking advantage of specific problem properties may sometimes get around the algebraic barrier. On the other hand, attempting to make progress on well-studied NP-complete problems, such as the Hamiltonian path problem or the 3D-matching problem, one faces a perhaps more significant **psychological barrier**: a lack of progress in nearly 50 years.

In brilliant work, Andreas Björklund broke the psychological barrier with an $O^*(2^{N/3})$ time algorithm for the 'exact' X3D-matching problem [5] and an astonishing $O^*(1.657^n)$ time algorithm for the Hamiltonicity [4]. These running times were later almost matched by parameterized algorithms [6]. The reader can find in [14] an excellent exposition of the algorithm for the Hamiltonian path problem for bipartite graphs.

Central to Björklund's work are sharper algebraic tools that draw from the large pool of algebraic combinatorics (e.g.[28]) to produce lower degree polynomials. More crucially, Björklund proposed the idea of relaxing the method of fingerprints in order to **exploit cancelations** of pairs of non-solution monomials. In the rest of this Section we will see how these ideas got applied in the case of the X3D-matching problem.

4.1 Counting weighted matchings mod 2

Consider a restriction of our 3D-matching problem to a 2D-matching problem in which we are given pairs consisting of a captain and a second officer, rather than triples. We will assume that the two sets X and Y are of equal size n.

The problem has a graph representation as shown in Figure 4. A solution of n edges/pairs that covers all vertices of the graph is called a **perfect matching**.

The perfect matching problem can be solved in polyno-

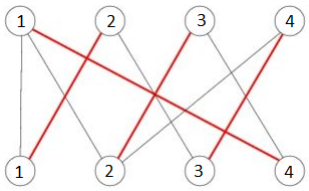

Figure 4: A perfect bipartite matching

mial time. But *counting* the number of perfect matchings is known to be a hard problem; it is as hard as counting the number of solutions for any NP-complete problem [30]. In fact, it is also hard to compute the number of perfect matchings modulo any prime p, except when $p = 2$. If $p = 2$ the number of perfect matchings modulo 2 is equal to the determinant of the **incidence matrix** of the bipartite graph [28]. This key fact is going to play significant role, because the matrix determinant can be computed with a **polynomial number** of arithmetic operations.

To define the incidence matrix A, we arbitrarily number the nodes in X with numbers from 1 to n, and do the same with the nodes in Y, as shown in Figure 4. The entry $A(i, j)$ is 1 if node $i \in X$ is connected to node $j \in Y$, and 0 otherwise.

Given this definition, if N_2 denotes the number of perfect matchings, we have what we will call the **Matching Lemma**:

$$N_2 \ mod\ 2 = Determinant(A)\ mod\ 2.$$

We now consider an extension of the above lemma which will be particularly useful. The perfect matching counting problem has a natural generalization to **edge-labeled** multi-graphs, where (i) we allow multiple 'parallel' edges between any two given nodes and (ii) each edge e is labeled with a unique monomial l_e.

The **signature** of a matching μ is defined to be the monomial formed by taking the product of the labels of the edges in the matching μ. For example, the signature of the matching in Figure 4 is $l_{1,4}l_{2,1}l_{3,2}l_{4,3}$. More generally,

$$sig(\mu) \triangleq \prod_{e \in \mu} l_e.$$

Given a labeling of the edges we can extend N_2 to a **solution polynomial** $N_2(L)$ which is simply the sum of the signatures of all perfect matchings, i.e. if M is the set of perfect matchings in the graph, then

$$N_2(L) \triangleq \sum_{\mu \in M} sig(\mu)$$

Finally, we can extend the incidence matrix A to a matrix A_L over L in a natural way: if nodes i and j are connected by one or more edges, we let $A_L(i, j)$ be the *sum* of their labels. We have the following **Generalized Matching Lemma**:

$$N_2(L)\ mod\ 2 = Determinant(A_L)\ mod\ 2. \tag{4.2}$$

4.2 Monomial detection for X3D-matching

We now return to the X3D-matching problem which is 3D-matching problem augmented with the constraint that $|X| = |Y| = |Z| = n$. In this case we want to decide if there is a solution that consists of exactly $n = N/3$ triples, i.e. a solution that covers all elements of X, Y, Z.

The algorithm involves a **transformation** of the X3D matching problem into a labeled matching instance. The sets X and Y define the two parts of the bipartite graph. In order to determine the labels, we use two sets of variables, Z and U. Each destination corresponds to a distinct variable in Z and each triple corresponds to a distinct variable in U.

To see now how we form the labeled bipartite graph we give an example that illustrates the edges between two given vertices of the graph, along with their labels. Suppose the X3D-matching instance contains the triples

$$\{W,K,\text{SJU}\},\ \{W,K,\text{SFO}\},\ \{W,K,\text{PIT}\}, \tag{4.3}$$

and no other triples with $\{W, K\}$ as a captain and second officer. Also suppose that we have associated with these triples the variables u_1, u_2, u_3, respectively. Then, nodes W and K will be connected by three edges, labeled respectively with the monomials $u_1 z_{\text{SJU}}$, $u_2 z_{\text{SFO}}$, and $u_3 z_{\text{PIT}}$.

Consider now the polynomial $N_2(Z, U)\ mod\ 2$. Because each edge of the bipartite graph corresponds to a triple, any monomial of $N_2(Z, U)\ mod\ 2$ corresponds to the selection of n triples, because it is the signature of a matching. By definition of $N_2(Z, U)\ mod\ 2$ these n triples cover exactly and without overlaps the sets X and Y. In the case there is an overlap in destinations, the monomial **contains a squared variable**, corresponding to that destination. On the other hand if a set of triples forms a 3D matching, the monomial is **linear** with respect to the variables in Z because no destination appears twice. So, assigning the random matrices from the Mod-2 Matrix Fact to the variables in Z we zero-out the the non-linear monomials. In addition we observe that the coefficient of each monomial in $N_2(L)\ mod\ 2$ is 1, because the U variables specify each set of n triples. Thus the U variables are our fingerprint variables and we will assign to them values from an appropriate algebraic field of size $O(n)$. With these assignments, the outcome of the evaluation will be non-zero with a good probability if and only if $N_2(L)$ contains a monomial corresponding to a solution of the instance. Hence we can detect an exact 3D-matching if there exists one. The running time of the algorithm is $O^*(2^n)$ because the degree of $N_2(Z, U)$ in terms of the Z variables is n.

Determinants and Cycles. The monomials that are canceled when the determinant is computed modulo 2 as in equation 4.2 correspond to **cycle covers**. A cycle is a path plus one edge that closes the loop. A cycle cover is a set of cycles in the graph such that each vertex participates in exactly one cycle. In particular, a Hamiltonian cycle, i.e. a cycle containing all the vertices, is a cycle cover. A perfect matching is also a cycle cover albeit consisting only of 'degenerate' cycles, that is edges. Cycle covers that contain at least one non-degenerate cycle come in pairs basically because each such cycle can be traversed in two possible ways, clockwise and counterclockwise. Björklund's Hamiltonicity algorithm, which actually targets Hamiltonian cycles rather than paths, breaks this symmetry by introducing direction to some edges in the graph. His algorithm still relies crucially on the remaining symmetries and cancelations. It remains open whether Hamiltonicity has a faster than $O^*(2^n)$ time algorithm for general directed graphs.

Exploiting Cancelations

The algebraization for X3D-matching is relaxed by design, as not all monomials in the determinant polynomial have unique fingerprints. Instead, many monomials that correspond to graph structures other than matchings happen to come in pairs, and computing the determinant modulo 2 exploits that. The idea of treating modulo 2 arithmetic as a resource rather than as a nuisance was also used for Hamiltonicity and later works.

5. MORE RECENT ADVANCES

Since its appearance, the general framework of algebraic fingerprints with modulo computations has been used in the design of faster algorithms for several parameterized problems. Examples include: (i) finding subgraphs that are more complicated than paths [16, 32], (ii) finding functional motifs in biological networks [18, 23, 8], (iii) finding the shortest cycle through specified nodes in a given graph [7], and (iv) the repetition-free longest common subsequence problem for strings, with applications in computational biology [9]. Among the many examples, we highlight remarkable progress in algorithms for NP-complete problems on graphs, parameterized by the so-called **treewidth** of the input graph.

Several NP-complete problems on graphs are tractable if the graph is a **tree**, i.e. if it contains no cycles. The NP-completeness of graph problems is usually proved via the construction of very intricate graphs that are arguably artificial comparing to graphs arising in practical situations. In particular, real world graphs often have a more tree-like structure.

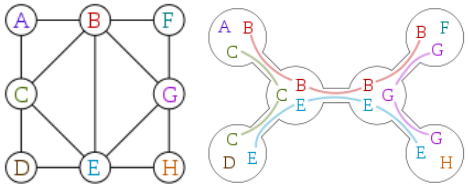

Figure 5: A graph and its treewidth decomposition. [Source:Wikipedia]

The notion of treewidth offers a way to quantify how much a given graph 'deviates' from a being tree. An example is given in Figure 5. The vertices of the graph are arranged in a tree structure. Each tree node lists copies of some of the graph vertices. Each edge of the graph connects two vertices that are listed together at some tree node. In addition, any given graph vertex can appear only in contiguous tree nodes. The treewidth tw of a graph is defined as the maximum number of graph vertices hosted in a tree node; in our example $tw = 3$. As a more general example, the natural class of *planar* graphs, i.e. graphs that can be drawn on the plane without crossings has treewidth $O(\sqrt{n})$, where n is the number of vertices.

Cygan et.al. [11] gave faster algorithms for many graph problems parameterized by treewidth. Previous algorithms had running time of the form $O^*(tw^{tw})$, while the new algorithms have dramatically improved running times of the form $O^*(c^{tw})$ for small constants c; for example the Hamiltonian path cycle can be now solved in $O^*(4^{tw})$ time.

6. OPEN PROBLEMS

The algorithms in this article are all randomized; on any given run they have a small probability of reporting that an instance does not have a solution, when the opposite is true. But running them $O(\log(1/p))$ times will make the overall probability of failure smaller than p, for any $p>0$. The fastest known **deterministic** algorithm for the k-path problem requires $O^*(2.85^k)$ time [15]. Finding a deterministic algorithm that solves the problem in $O^*(2^k)$ time remains an open problem. The way things stand, it would seem to require a **deterministic** version of the Schwartz-Zippel Lemma: a deterministic polynomial-time algorithm for testing whether a polynomial given as an arithmetic circuit is identically zero.

The method of algebraic fingerprints can be adapted to solve the **weighted k-path** problem; here the goal is to find a k-path of minimum total cost, assuming a cost is attached to every edge of the graph. The running time is $O^*(2^k W)$ where W is the largest cost associated with an edge in the graph. On the other hand, dynamic programming can handle the problem in $O^*(2^n \log W)$ time, when $k = n$. It is reasonable to conjecture that k-path has an $O^*(2^k \log W)$ algorithm. It should be noted though that, unlike the algebraic fingerprints method that can be implemented in memory of size polynomial in n, color coding for weighted k-path requires $O^*(2^k)$ memory which can be a very limiting factor in practice.

Dynamic programming can also be used to **count** the number of Hamiltonian paths. Counting the number of k-paths exactly is probably not FPT [13], but color coding can be used to count k-paths *approximately*, in $O^*((2e)^k)$ time [1]. This stands as the fastest known algorithm for this problem, but again it is reasonable to conjecture that there is an $O^*(2^k)$ algorithm for approximate counting.

Solving any of these open questions may require fresh ideas that will start a new cycle of discovery.

Acknowledgements

I. Koutis is supported by NSF CAREER award CCF-1149048. R. Williams is supported by a David Morgenthaler II Faculty Fellowship, NSF CCF-1212372, a Sloan Fellowship, and a Microsoft Research Fellowship. This idea for this article originated from a special session of the 2013 Joint Mathematics Meeting, on "The underpinnings of multivariate complexity theory and algorithm design, and its frontiers, and the field of incrementalization", organized by R. Downey, M. Fellows, A. Nerode, and F. Rosamond. The last stages of this work were carried out at the Simons Institute for the Theory of Computing where both authors were hosted during Fall 2014.

7. REFERENCES

[1] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp. Biomolecular network motif counting and discovery by color coding. In *Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada, July 19-23, 2008*, pages 241–249, 2008.

[2] N. Alon, R. Yuster, and U. Zwick. Color coding. *Journal of the ACM*, 42(4):844–856, 1995.

[3] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9:61–63, 1962.

[4] A. Björklund. Determinant sums for undirected hamiltonicity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 173–182. IEEE Computer Society, 2010.

[5] A. Björklund. Exact covers via determinants. In J. Marion and T. Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, volume 5 of *LIPIcs*, pages 95–106. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[6] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010.

[7] A. Björklund, T. Husfeldt, and N. Taslaman. Shortest cycle through specified elements. In Y. Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1747–1753. SIAM, 2012.

[8] A. Björklund, P. Kaski, and L. Kowalik. Probably optimal graph motifs. In N. Portier and T. Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 20–31. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[9] G. Blin, P. Bonizzoni, R. Dondi, and F. Sikora. On the parameterized complexity of the repetition free longest common subsequence problem. *Information Processing Letters*, 112(7):272 – 276, 2012.

[10] J. Chen, S. Lu, S.-H. Sze, and F. Zhang. Improved algorithms for path, matching, and packing problems. In *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 298–307, 2007.

[11] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In R. Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011.

[12] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[13] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

[14] F. V. Fomin and P. Kaski. Exact exponential algorithms. *Commun. ACM*, 56(3):80–88, Mar. 2013.

[15] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Representative sets of product families. In A. S. Schulz and D. Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2014.

[16] F. V. Fomin, D. Lokshtanov, V. Raman, S. Saurabh, and B. V. R. Rao. Faster algorithms for finding and counting subgraphs. *J. Comput. Syst. Sci.*, 78(3):698–706, 2012.

[17] L. Fortnow. The status of the P versus NP problem. *Commun. ACM*, 52(9):78–86, 2009.

[18] S. Guillemot and F. Sikora. Finding and counting vertex-colored subtrees. *Algorithmica*, 65(4):828–844, 2013.

[19] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *J.Soc. Indust. Appl. Math*, 10:196–210, 1962.

[20] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[21] I. Koutis. A faster parameterized algorithm for set packing. *Information Processing Letters*, 94(1):4–7, 2005.

[22] I. Koutis. Faster algebraic algorithms for path and packing problems. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

[23] I. Koutis. Constrained multilinear detection for faster functional motif discovery. *Inf. Process. Lett.*, 112(22):889–892, 2012.

[24] I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ICALP '09, pages 653–664, Berlin, Heidelberg, 2009. Springer-Verlag.

[25] B. Monien. How to find long paths efficiently. *Annals of Discrete Mathematics*, 25:239–254, 1985.

[26] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[27] F. Rosamond and M. Fellows. Table of FPT Races. `http://http://fpt.wikidot.com/fpt-races/`.

[28] G. Royle and C. Godsil. *Algebraic Graph Theory*. Graduate Texts in Mathematics. Springer Verlag, 1997.

[29] J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of Computational Biology*, 13(2):133–144, 2006.

[30] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[31] R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109:315–318, February 2009.

[32] V. V. Williams, J. R. Wang, R. R. Williams, and H. Yu. Finding four-node subgraphs in triangle time. In P. Indyk, editor, *Proceedings of the Twenty-Sixth*

Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015, pages 1671–1680. SIAM, 2015.