Research Proposal

Overview and Objectives. My research area is Computer Algebra. Three important computational problems in Computer Algebra are

- computing multivariate polynomial greatest common divisors (GCDs),
- multiplying and factoring multivariate polynomials, and
- solving systems of polynomial equations.

They are important because the overall speed and capability of a Computer Algebra System like Maple critically depends on them. Algorithms for these problems depend on the coefficients of the polynomials. In practice, integer coefficients are the most common. Algebraic numbers e.g. $\sqrt{3}$, and algebraic functions, e.g. $\sqrt{1-t^2}$, also occur frequently. My **long term objectives** are to

- design and implement efficient algorithms for these and related problems,
- design and implement parallel algorithms for these problems for multi-core computers,
- develop and apply sparse interpolation tools and black box algorithms, and
- integrate the software into Maple where it gets used, automatically, by scientists and engineers.

A challenge in Computer Algebra is getting implementations of good algorithms integrated well into Computer Algebra Systems so they can be easily applied to real problems. Another challenge is how to utilize our multi-core computers with their vector processors. We need to think beyond just designing new algorithms if we want to have an impact outside the Computer Algebra community.

I propose the following four **short-term objectives**. Each involves algorithm design, implementation, analysis (complexity and failure probability), and application. Each is suitable for graduate student training and research. Objectives 3 and 4 are accessible to undergraduate students.

1 Black box multivariate polynomial GCD and polynomial factorization algorithms.
2 Computing GCDs and factoring multivariate polynomials with algebraic function coefficients.
3 Factoring multilinear polynomials over GF(2).
4 To build a library of tools for black boxes and sparse interpolation.

A central theme of my research program is sparsity and the black box representation. Let me explain what these are and why they are important.

The **sparse representation** of a polynomial $f = \sum_{i=1}^{t} a_i M_i(x_1, \ldots, x_n)$ is a list of t non-zero coefficients a_i in a ring R and exponent vectors for the monomials M_i. We say a polynomial of degree d in n variables is **sparse** if the number of terms t is much smaller than $\binom{n+d}{d}$, the maximum possible. We want algorithms whose complexity is polynomial in n, d, t and not $\binom{n+d}{d}$. The main reason this is important is that in most applications, polynomials in many variables are sparse.

The **black box representation** for f is a computer program $\mathbf{B}$ that evaluates f at a point $\alpha \in R^n$, that is, $\mathbf{B}(\alpha)$ computes $f(\alpha)$. We seek algorithms which minimize the number of probes (evaluations) of the black box. The main reason to consider the black box representation is it can be exponentially more compact than the sparse representation.

Since the factors of a polynomial f are usually much smaller in size than f, computing the factors of a polynomial given by a black box is especially important. Consider the polynomial

$$f = (x_1 - x_2)(x_1 - x_3)(x_1 - x_4) \times \cdots \times (x_1 - x_n).$$

This factorization has $n-1$ factors of size $O(1)$ bits, yet if we interpolate f in the sparse (expanded) representation it has 2^{n-1} terms of size $O(n)$ bits! Computing such a factorization from a black box has obvious practical value.

Literature Review. In Computer Algebra, the black box representation was first investigated by Kaltofen and Trager [22] in 1990 who gave the first algorithms for factoring polynomials and computing GCDs of polynomials which are represented by black boxes. In [8, 9] Kaltofen and Diaz improved the GCD algorithm and developed a C^{++} implementation called FOXBOX. In [35] Rubinfeld and Zippel give a black box factorization algorithm for $\mathbb{Z}[x_1, \ldots, x_n]$ that factors many univariate polynomials in $\mathbb{Z}[x_1]$. In [6] Chen and I give a new black box factorization algorithm that does a lot fewer probes to the black box than Rubinfeld and Zippel; it needs only one univariate factorization in $\mathbb{Z}[x_1]$. Very little other work has been done and, as far as I know, the black box representation is not used in any Computer Algebra System. One reason for this is the lack of infrastructure needed to implement black box algorithms. Objective 4 addresses this.

Users of black box algorithms will need to recover the sparse representation of f from it's black box representation, that is, interpolate f. There is a large body of literature for sparse polynomial interpolation. Early algorithms for polynomials with integer coefficients include Zippel [39] and Ben-Or and Tiwari [3] whose bit complexity is polynomial in n, d, t and $\log h$ where $h = \max_{i=1}^{t} |a_i|$. In [12], Garg and Schost initiated a different approach that is polynomial in $\log d$. This line of research culminated in 2022 with Giorgi et. al. [14] which is softly linear in $t(n \log d + \log h)$, that is, the bitsize of f.

For objective 2 I need to interpolate rational functions. A first sparse rational function interpolation algorithm was given by Kaltofen and Trager in [22]. In [26], De Kleine, Wittkopf and I gave an algorithm based on Zippel's sparse polynomial interpolation. The number of function evaluations needed was subsequently reduced by Kaltofen and Yang [23] and further reduced by Cuyt and Lee [7]. In [18], van der Hoeven and Lecerf present a variation of Cuyt and Lee. In [21], Jinadu and I modified Cuyt and Lee to use a Kronecker substitution and applied our method to interpolate Dixon resultants which we used to solve parametric polynomial systems from [24, 25, 27, 28, 29, 37].

1: Black box multivariate polynomial GCD and factorization algorithms [MSc1, PhD1]

Recent Progress: To factor a polynomial f in $\mathbb{Z}[x_1, ..., x_n]$ given by a black box, instead of interpolating f with sparse interpolation then factoring f, the technology I have developed with my PhD student T. Chen in [4, 6] recovers the variables $x_2, \ldots, x_n$ in the factors of f, in the sparse representation, one at a time, from bivariate images obtained using my bivariate Hensel lifting from [32, 33]. We are using our software to compute the factors of determinants of structured matrices of polynomials.

One advantage of our approach is that we know exactly how many bivariate images (hence black box probes) we need to recover each variable which simplifies parallelization. I would like to do a parallel implementation of the algorithm for multi-core computers. A second advantage is we can easily omit computation of the content(f, x_1). An important application of this advantage is computing Dixon resultants. Let me describe this application.

Let $f_1, ..., f_n$ be polynomials in n variables $x_1, ..., x_n$ and m parameters $y_1, ..., y_m$ with integer coefficients. The computational problem is to eliminate $x_2, x_3, \ldots, x_n$ from the parametric polynomial system $\{f_1 = 0, \ldots, f_n = 0\}$ to obtain a polynomial in x_1 only which we then solve for x_1. Three general approaches to do this are Groebner bases, Triangular sets and resultants.

In [27, 28, 29] Lewis presents many real problems where Groebner bases and Triangular sets fail. The approach I tried with Jinadu in [21] was to compute the Dixon resultant $R = \det(D)$ where

D is the Dixon matrix, a matrix of polynomials in $x_1, y_1, \ldots, y_m$. See Kapur et. al. [24, 25]. Let $C = \text{content}(R, x_1)$ and $M = R/C$. Often C is large and M is much smaller than R. We modified Cuyt and Lee's sparse rational function interpolation to interpolate S, the square-free part of M, from monic images of R in x_1. Thus to solve $R = 0$ for x_1 we solve $S = 0$ for x_1. The advantage? Often S is much smaller than R in size. On one of Lewis' problems S is over 10,000 times smaller than R. We were able to solve all the application problems in Lewis' papers.

I want to try a different approach to handle larger problems. The idea is to try to compute the factors of $R = \det(D)$ in x_1 only. *Methodology:* Treat R as a black box. Step 1: pick integers $b_1, \ldots, b_m$ randomly from a large set and interpolate $R(x_1, b_1, \ldots, b_m)$ in $\mathbb{Z}[x_1]$. Step 2: factor $R(x_1, b_1, \ldots, b_m)$ and use our new Hensel lifting algorithm to recover $y_1, \ldots, y_m$ for the factors in x_1 only, thus avoiding the content C. The Hensel lifting increases the cost by a factor of m but we get to lift the factors of S which, depending on the application, can be much smaller than S, and we avoid rational function interpolation which saves a factor of $\deg(S, x_1)$. An alternative approach would be to use Groebner bases to compute the many images of $R(x_1, y_1, \ldots, y_m)$ modulo primes needed in Steps 1 and 2. I will try both approaches.

With my PhD student G. Paluck we have just started designing black-box GCD algorithms. *Methodology:* Let $A, B \in \mathbb{Z}[x_1, \ldots, x_n]$ be given by black boxes. There are many approaches to compute $G = \gcd(A, B)$ using our black box Hensel lifting technology. One is to Hensel lift a full factorization of the GCD $G(x_1, a_2, ..., a_n)$ in $\mathbb{Z}[x_1]$. To avoid this factorization, we can instead lift the "cheap to compute" square-free factorization of G modulo a prime p. Let $C = A/G$ and $D = B/G$ be the cofactors. A third approach, useful if we want also to also compute C and/or D, would be to lift the factorization of $A = GC$ and/or $B = GD$.

2: Computing GCDs and factoring polynomials over function fields [MSc2, PhD2]

Let $\alpha_1, \ldots, \alpha_n$ be algebraic numbers and $F = \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ be an algebraic number field of degree d over $\mathbb{Q}$. For GCD computation in $F[x]$, for $n = 1$, Encarnacion [11] developed a modular algorithm that uses Chinese remaindering and rational number reconstruction to recover the rational coefficients in the monic gcd. In [16], van Hoeij and I generalized Encarnacion's method to $n > 1$. For large d, Li, Moreno Maza and Schost designed fast arithmetic for F in [30].

Recent Progress: In [1, 2] my PhD student M. Ansari and I designed new modular GCD and resultant algorithms for polynomials in $A, B \in F[x_1, \ldots, x_k]$ which use dense interpolation for $x_2, \ldots, x_k$. We employ a primitive element γ, computed modulo a prime p, to map a computation over $F \mod p$ into $\mathbb{Q}(\gamma) \mod p$ to speed up arithmetic in $F \mod p$. We are trying to complete a failure probability analysis which is difficult. As an application of our work we are trying to factor four polynomials in $F[x]$ given to us by Grasl [15]. The four problems are for $n = 4, 7, 6, 4$, $d = 136, 128, 624, 744$ and $\deg(f, x) = 136, 1024, 624, 744$, respectively.

For the algebraic function field case, Dr. J. Gerhard of Maplesoft has asked me for help with GCD and factorization problems from users which do not terminate in Maple. Let $T = t_1, \ldots, t_m$ be parameters and $\alpha_1(T), \ldots, \alpha_n(T)$ be algebraic functions and $L = \mathbb{Q}(\alpha_1(T), \ldots, \alpha_n(T))$ be an algebraic function field over $\mathbb{Q}(T)$.

For GCD computation in $L[x_1, \ldots, x_k]$, in previous work [17], van Hoeij and I tried evaluation and dense rational function interpolation for the parameters and variables. *Methodology:* Two possible sparse approaches are (i) to compute monic univariate images of the gcd in x_1 and use sparse *polynomial* interpolation to recover $x_2, \ldots, x_k$ then sparse *rational function* interpolation to recover the parameters $t_1, \ldots, t_m$ in the monic gcd, and (ii) to compute monic bivariate images of the gcd in x_1, x_j and x_1, t_j and recover $x_2, \ldots, x_n, t_1, \ldots, t_m$ sequentially using our sparse bivariate

Hensel lifting from objective 1. This will be complicated to implement so I think it wise to first design a data structure for $L[x_1, \ldots, x_n]$ and code arithmetic and support tools to facilitate a clean implementation. To do a failure probability analysis we will need degree bounds and coefficient bounds. We will try to express all quantities as determinants of matrices of polynomials.

For the factorization problem, Maple currently uses Trager's algorithm from [38]. For $f \in L[x_1]$, Trager's algorithm picks a random shift $s \in \mathbb{Z}^n$, computes the norm h of $f(x_1 = x_1 + \sum_{i=1}^{n} s_i \alpha_i)$ using a sequence of resultants to obtain h a polynomial in $\mathbb{Q}(T)[x]$, then factors h over $\mathbb{Q}(T)$ to get $h = \prod_{i=1}^{l} h_i$. The factors of f are then obtained from $\gcd(h_i, f)$. The problem is that the norm h is, in general, a factor of d^{m+2} times larger than f.

Methodology: I propose to first try a black box solution, which, in principle, can avoid computing h explicitly. To compute h implicitly we will need to design a black box resultant algorithm. Then we would apply a black factorization algorithm to factor h. The final step, computing $\gcd(h_i, f)$, is an application of our black box GCD from objective 1. It is not clear that this will be fast enough in practice because we would be composing black box resultant, factorization and GCD algorithms which multiplies costs.

3: Factoring multilinear polynomials over GF(2). [PhD3]

A polynomial f is multilinear if it is degree 1 in every variable. For example $f = (x_1+x_2)(x_3x_4+1)$ is multilinear. This restriction means the factors must have distinct variables. My reason for studying this problem is firstly the application to boolean circuit optimization [10], and secondly, to try to pin down the theoretical complexity of this very restricted factorization problem.

Recent Progress: In 2010 Shpilka and Volkovich [36] showed that factoring polynomials whose factors have distinct variables is polynomial time equivalent to identity testing. In 2018 Emelyanov and Ponomaryov [10] gave a deterministic algorithm based on such an identity test for factoring a multilinear polynomial f in $GF(2)[x_1, \ldots, x_n]$ (a boolean polynomial) which has bit complexity $O(n^2t^2 \log t)$ where t is the number of terms of f. Note the bitsize of f is $O(nt)$.

My first goal is to remove the quadratic dependence on t. I can see how to get a Monte Carlo algorithm for the sparse representation with algebraic complexity $O(n^2t)$ multiplications in $GF(2^k)$ for suitable k. This should be faster in practice if multiplication in $GF(2^k)$ is efficient.

Emelyanov and Ponomaryov gave a second algorithm for the sparse representation based on GCD computation. For $f = ax_1 + b$ where $a, b \in GF(2)[x_2, \ldots, x_n]$, $a \neq 0$, they compute $g = \gcd(a, b)$ which gives the factorization $f = gh$ where $h = f/g$ is irreducible. Emelyanov and Ponomaryov assume Zippel's probabilistic algorithm is used for the GCD and state the complexity of Zippel's algorithm at $O(t^3)$ arithmetic operations in $GF(2^k)$ based on the work of de Kleine, Wittkopf and myself [26]. I will instead try recursive content computations to compute g and its factorization.

If f has m factors, each with 2 terms in distinct variables, then f has $t = 2^m$ terms in expanded form which limits the size of the problem which can be tackled in the sparse representation. I propose to design a black box factoring algorithm to avoid the 2^m size.

Methodology: Let $p = f/g$ be the primitive part of f in x_1, which is irreducible. A first approach would be to interpolate p using a black box GCD algorithm from objective 1 and then construct a black box for $g = f/p$ and factor it recursively.

A black box algorithm must be able to choose evaluation points from a large set but for boolean polynomials, we have only two values, 0 and 1. So we must choose values from a field extension, $GF(2^k)$, for suitably large k. This poses an unavoidable software design problem; how to allow the code for the black box to execute over a field extension. If the black box is a Maple procedure (Maple procedures are white boxes) I will try evaluating the code using an interpreter.

4: A black box library. [MSc3, PhD3, PDF1]

I propose to build a library of tools for black box algorithm development so that researchers who want to implement black box algorithms don't have to start from scratch. Part of the motivation is that Computer Algebra Systems have very little support for black box algorithms. Let f be a polynomial represented by a black box. A minimal list of tools would include

1. a test for $f = c$ for a constant c,
2. computation of $\deg(f)$ and $\deg(f, x_i)$,
3. a black box factorization algorithm for $\mathbb{Q}[x_1, \ldots, x_n]$,
4. a black box factorization algorithm for $\mathbb{F}_q[x_1, \ldots, x_n]$,
5. sparse polynomial interpolation of a black box
6. sparse rational function interpolation of a black box
7. a black box for $\partial f/\partial x_i$ and for ∇f the gradient of f

Methodology: For flexibility and ease of use, I will first try Maple procedures as the representation for black boxes. For 3, for efficiency, the black box model needs to allow us to compute f modulo a prime p. For 4, for randomization, the black box model needs to allow us to evaluate f over an extension field. For 5 I propose to use a Kronecker substitution plus Ben-Or/Tiwari interpolation as I did in [19] and [21]. For 6 I propose to modify Kaltofen and Yang's method from [23] to use a random dilation (see Giesbrecht and Roche [13]) so that it does not require randomized evaluation points. Although Kaltofen and Yang is slightly less efficient than Cuyt and Lee, it is much easier to parallelize and analyse.

A challenge is the gradient operation $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ which cannot be done efficiently in the black box model. However, since Maple procedures are white boxes, we could use Automatic Differentiation (see Corliss et. al. [5]) to compute ∇f. In the white box representation, if f does m arithmetic operations, ∇f can be computed in $O(m + n)$ arithmetic operations whereas the black box representation needs $O(mn)$ arithmetic operations. As the author of Maple's toolbox for Automatic Differentiation [31], I have relevant expertise in this.

Another challenge is how the user creates efficient code for the black box because the speed of a black box algorithm depends on how fast we can evaluate the black box. For example, if $\mathbf{B}$ is a Maple procedure that constructs a matrix then computes its determinant modulo a prime p, that is, $\mathbf{B} : (\mathbb{Z}^n, p) \to \mathbb{Z}_p$, we may need to translate $\mathbf{B}$ into C/C^{++} code and compile it for speed. The current Maple C compiler has limitations. One is that the prime $p < 2^{31.5}$.

Overall Impact. The primary goal of my research program is to be able to solve large instances of problems that are of central importance in Computer Algebra, in a reasonable time, and to see high performance implementations of the new algorithms installed in a Computer Algebra System like Maple so that scientists and engineers can easily access them. I also believe objectives 1,2,3, and 4 offer HQP training opportunities in algebraic computation of the highest quality. As well as the HQP trained, the benefit to Canada will be new capabilities of Maple, a Canadian product.

Objective 1 will make the fastest black box factorizer even faster by parallelizing it. That combined with the new approach to computing the factors of the Dixon resultant will increase the range of parametric polynomial systems that can be solved. Objective 2, if successful, could dramatically increase the size of computations involving algebraic functions that can be tackled. Objective 4, if easy to use, will stimulate the development and application of black box tools to other areas of Science and Engineering.

References

[1] Mahsa Ansari and Michael Monagan. Computing GCDs of Multivariate Polynomials over Algebraic Number Fields Presented with Multiple Extensions. *Proceedings of CASC 2023*, LNCS **14139**:1–20, Springer, 2023.

[2] Mahsa Ansari and Michael Monagan. A Modular Algorithm to Compute the Resultant of Multivariate Polynomials over Algebraic Number Fields Presented with Multiple Extensions. *Proceedings of CASC 2024*, LNCS **14938**:27−46, Springer, 2024.

[3] Michael Ben-Or and Prasson Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. *Proceedings of STOC '20*, pp. 301–309, ACM, 1988.

[4] Tian Chen and Michael Monagan. Factoring Multivariate Polynomials Represented by Black Boxes. *Math. in Comp. Sci.* **16**:(2-3), article 18, Springer, 2022.

[5] George Corliss, Christele Faure, Andreas Griewank, Laurent Hasoet, Uwe Naunann. *Automatic Differentiation of Algorithms: From Simulation to Optimization.* Springer, 2002.

[6] Tian Chen and Michael Monagan. A new black box factorization algorithm - the non-monic case. *Proceedings of ISSAC 2023*, pp. 173-181, ACM, 2023

[7] Annie Cuyt and Wen-shin Lee. Sparse interpolation of multivariate rational functions. *J. Theoretical Comp. Sci.* **412**:1445–1456, Elsevier, 2011.

[8] Angel Díaz and Erich Kaltofen. On computing greatest common divisors with polynomials given by black boxes for their evaluations. *Proceedings of ISSAC 1995*, pp. 232–239, ACM 1995.

[9] Angel Díaz and Erich Kaltofen. FOXBOX: a system for manipulating symbolic objects in black box representation. *Proceedings of ISSAC 1998*, pp. 30–37, ACM, 1998.

[10] Pavel Emelyanov and Denis Pononmaryov. On a polynomial factorization algorithm for multilinear polynomials over $\mathbb{F}_2$. *Proceedings of CASC 2018*, LNCS **11077**:164–176, 2018.

[11] Mark Encarnacion. Computing GCDs of polynomials over algebraic number fields. *J. Symbolic Computation*, **20**:299−313, 1995.

[12] Sanchit Garg and Erich Schost. Interpolation of polynomials given by straight-line programs. *J. Theoretical Comp. Sci.* **410**:2659–2662, 2009.

[13] Mark Giesbrecht and Daniel Roche. Diversification improves interpolation. *Proceedings of ISSAC '11*, pp. 123–130, ACM, 2011.

[14] Pascal Giorgi, Bruno Grenet, Armelle Perret du Cray and Daniel Roche. Sparse polynomial interpolation and division in soft-linear time. *Proceedings of ISSAC 2022*, pp. 459-468, ACM, 2022.

[15] Markus Grasl. Personal communication, 2023.

[16] Mark van Hoeij and Michael Monagan. A modular GCD algorithm over number fields presented with multiple field extensions. *Proceedings of ISSAC 2002*, pp. 109–116, ACM, 2002.

[17] M. van Hoeij, M. B. Monagan. Algorithms for Polynomial GCD Computation over Algebraic Function Fields. *Proceedings of ISSAC 2004*, pp. 297−304, ACM, 2004.

[18] Joris van der Hoven and Gregoire Lecerf. On sparse interpolation of rational functions and gcds. *Communications in Computer Algebra* **55**(1):1−12, ACM, 2021.

[19] Jiaxiong Hu and Michael Monagan. A fast parallel sparse polynomial GCD algorithm. *J. Symb. Cmpt.* **105**:(1) 28–63, Springer, 2021. https://doi.org/10.1016/j.jsc.2020.06.001

[20] Qiao-Long Huang and Michael Monagan. A New Sparse Polynomial GCD by Separating Terms. *Proceedings of ISSAC 2024*, pp. 134-142, ACM, 2024

[21] Ayoola Jinadu and Michael Monagan An interpolation algorithm for computing Dixon resultants. *Proceedings of CASC 2022*, LNCS **13366**:185−205, Springer 2022

[22] Eric Kaltofen and Barry Trager. Computing with polynomials given by black boxes for their evaluations. Greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Cmpt.*, **9**(3):301−320, Springer, 1990.

[23] Erich Kaltofen and Zhengfeng Yang. On exact and approximate interpolation of sparse rational functions. *Proceedings of ISSAC 2007*, pp. 203−210, ACM 2007.

[24] Deepak Kapur, Tushar Saxena and Lu Yang. Algebraic and Geometric Reasoning using Dixon Resultants. *Proceedings of ISSAC '94*, pp. 99–107, ACM, 1994.

[25] Deepak Kapur and Tusha Saxena. Comparison of Various Multivariate Resultant Formulations. *Proceedings of ISSAC '95*, pp. 187–194, ACM, 1995.

[26] Jennifer de Kleine, Michael Monagan and Allan Wittkopf. Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm. *Proceedings of ISSAC 2005*, pp. 124–131, ACM, 2005.

[27] Robert Lewis. Dixon-EDF: The Premier Method for Solution of Parametric Polynomial Systems. *Applications of Computer Algebra 2015*, Proceedings in Mathematics and Statistics, **198**, pp. 238–256, Springer, 2017.

[28] Robert Lewis. Resultants, Implicit Parameterizations, and Intersections of Surfaces. *Proceedings of ICMS 2018*, LNCS **10931**:310−318, 2018.

[29] Robert Lewis. Image Analysis: Identification of Objects via Polynomial Systems, *Proceedings of ICMS 2018*, LNCS **10931**:305−309, 2018.

[30] Xin Li, Marc Moreno Maza, Eric Schost. Fast arithmetic for triangular sets: From theory to practice. *J. Symbolic Computation,* **44**:891−907, 2009.

[31] Michael Monagan and Gladys Monagan. A Maple toolbox for program manipulation and efficient code generation with an application to a problem in computer vision. *Proceedings of ISSAC '97*, pp. 257−264, ACM, 1997.

[32] Michael Monagan. Linear Hensel Lifting for $\mathbb{F}_p[x,y]$ and $\mathbb{Z}[x]$ with Cubic Cost. *Proceedings of ISSAC 2019*, pp. 299−306, ACM, 2019.

[33] Michael Monagan and Garrett Paluck. Linear Hensel lifting for $\mathbb{Z}_p[x,y]$ for n factors with cubic cost. *Proceedings of ISSAC 2022*, pp. 159−166, ACM, 2022.

[34] Michael Monagan and Baris Tuncer. The complexity of sparse Hensel lifting and sparse polynomial factorization. *J. Symb. Cmpt.* **99**:189–230, Springer, 2020.

[35] Ronnitt Rubinfeld and Richard Zippel. A new modular interpolation algorithm for factoring multivariate polynomials. *Proceedings of ANTS-I '94*, pp. 93–107, Springer, 1994.

[36] Amir Shpilka and Ilya Volkovich. On the relation between polynomial identity testing and finding Variable disjoint factors. Proceedings of ICALP 2010, LNCS **6198**:408–419, 2010.

[37] Duane Storti. Algebraic Skeleton Transform: A Symbolic Computation Challenge. ISSAC 2020 Poster Session, ISSAC 2020.

[38] Barry Trager. Algebraic factoring and rational function integration. *Proceedings of SYMSAC '76*, pp. 219–226, ACM, 1976.

[39] Richard Zippel. Probabilistic algorithms for sparse polynomials. *Proceedings of EUROSAM '79*, pp. 216–226. Springer, 1979.