An Analysis of some Algorithms for Polynomial Decomposition

Bradley Hedges

April 3,2020

1 Introduction

Throughout this paper, we will be investigating the following problem: Given a polynomial f in $R[x]$, R a ring, can we find polynomials g, h in $R[x]$ such that $f(x) = g(h(x))$? We will call this problem polynomial decomposition. We will extend the investigation to cover the following problems: is a nontrivial polynomial decomposition possible, and what algorithms exist to find a decomposition? We will also look into the runtimes of the algorithms, and some of the history of their construction.

One might wonder why we want to find polynomial decompositions. Take the following illustrative example, inspired by Barton and Zippel [1]:

Example 1.1. Suppose we want to find the roots of

$$f = x^6 - 12x^5 + 64x^4 - 192x^3 + 333x^2 - 308x + 116,$$

in $\mathbb{Z}$, which seems nontrivial. Notice that f decomposes into $g \circ h$ where

$$g = x^3 - 4x^2 + 3x \text{ and } h = x^2 - 4x + 4.$$

Now to find the roots of f, we can find the roots a of g, and solve for the roots of $h - a$. Clearly, g has roots at $x = 0, 1, 3$ so we find the roots of:

$$h - 0 = x^2 - 4x + 4$$
$$h - 1 = x^2 - 4x + 3$$
$$h - 3 = x^2 - 4x + 1.$$

We see that $h - 0$ has a root at $x = 2$, $h - 1$ has roots at $x = 1, 3$ and $h - 3$ has no integer roots. This is consistent with f, which finds its roots similarly at $x = 1, 2, 3$.

As an aside, this method of root finding also maintains the order of the root. In this example, $x = 2$ is a second order root of f, and a second order root of $h - 0$. We simplified the problem of finding roots of a degree six polynomial into finding the roots of a degree three polynomial and three degree 2 polynomials. We were able to find roots that otherwise would have been very difficult to find. We now wonder: how do we find a convenient decomposition?

2 Theorems and Definitions

Before we get into the algorithms, we will first define some terms. The *decomposition* of a polynomial f in $R[x]$ is the set of polynomials $g_1, ..., g_k$ such that

$$f = g_1 \circ g_2 \circ ... \circ g_k.$$

We call g_i the *components* of the decomposition. A component is called *indecomposable* if it has degree greater than one and does not have a nontrivial decomposition in $R[x]$. In the next section, we will see that to decompose f into $g \circ h$, we first find a potential component h and from that attempt to compute g. For convenience, we will call this potential h a *candidate* for the decomposition of f. Since g is dependent on our choice of h, we give it no special treatment.

Next we note that a decomposition in not necessarily unique:

Example 2.1. Take the example from earlier:

$$\begin{aligned} f &= x^6 - 12x^5 + 64x^4 - 192x^3 + 333x^2 - 308x + 116 \\ &= (x^3 - 4x + 3x) \circ (x^2 - 4x + 4) \qquad (1) \\ &= (x^3 - 4x + 3x) \circ (x + 5) \circ (x - 5) \circ (x^2 - 4x + 4) \\ &= (x^3 + 11x^2 + 38x + 40) \circ (x^2 - 4x - 1) \qquad (2) \end{aligned}$$

Despite being completely different polynomials, equations (1) and (2) give the same f and are polynomials of the same degree. We will see in the next section that our algorithms can give different decomposition depending on choices we make.

With these definitions, we can cite a theorem that will be useful in the construction of our first algorithm.

Theorem 2.1 (Fried & MacRae [1])**.** *Let R be a field, f, g, f_1, $g_1 \in R[x]$. Then $f_1(x) - g_1(z)$ divides $f(x) - g(z)$ if and only if there exists $F(x) \in R[x]$ such that $f(x) = F(f_1(x))$ and $g(t) = F(g_1(t))$.*

Example 2.2. Consider the following polynomials:

$$\begin{aligned} f(x) &= -3x^3 - 15x^2 - 20x - 8, \\ g(z) &= -24z^3 + 84z^2 - 88z + 24, \\ f_1(x) &= x + 2\ ,\ g_1(z) = 2z - 2. \end{aligned}$$

Then for $F(x) = -3x^3 + 3x^2 + 4x - 4$, we have $f(x) = F(f_1(x))$ and $g(z) = F(g_1(z))$. According to the theorem, we would expect that $f_1(x) - g_1(z)$ divides $f(x) - g(z)$ and sure enough, $f(x) - g(z) = (-3x^2 - 6xz - 12z^2 - 3x + 18z - 8)(f_1(x) - g_1(z))$.

We now have all the necessary tools to construct the algorithm.

3 Two Decomposition Algorithms

We will begin by introducing a potential polynomial decomposition algorithm: Suppose a polynomial f in $R[x]$ has degree n and can be decomposed into $g \circ h$ for some g and h in $R[x]$. Suppose g has degree s and h has degree r, then $n = rs$. We can solve for g and h by solving the linear system that arises from equating the coefficients of f and $g \circ h$. Since f has degree $n = rs$, we will get $rs + 1$ equations, and $r + s + 2$ unknown coefficients. In most cases, attempting to solve the resulting system is nontrivial, and perhaps impossible, as it is over-determined.

In the following section, we will analyze two decomposition algorithms that avoid attempting to solve the system directly, and follow the same general structure:

- compute a potential h with degree less than f as a *candidate*
- find g if it exists, so that $f = g \circ h$
- recursively decompose g and h until they are *indecomposable*

We will see later that the algorithms differ in their computation of h, and have trade-offs in terms of runtime and guaranteed convergence. Since both algorithms compute g the same way, we will investigate that first.

3.1 Finding g(x)

Suppose we have found both f and h (for convenience, we will take the case where $h(0) = 0$, so h has no constant term), and now we want g such that $f = g \circ h$. This problem breaks down to essentially computing a p-adic expansion of f at h, similarly to how we've seen in class [1]:

Algorithm 1: p-adic expansion of $f(x)$ about $h(x)$

Input: f(x) and candidate h(x)
Initialization;
find $q_1(x), r_0(x)$ such that $f(x) = q_1(x)h(x) + r_0(x)$, deg($r_0$)<deg($h$)
while $q_i(x) \neq 0$ **do**
    find $q_{i+1}(x), r_i(x)$ such that $q_i(x) = q_{i+1}(x)h(x) + r_i(x)$, increment i
end

[1] Since h is a candidate, we must have deg(h)|deg(f), so the algorithm will run $s = n/r$ times before breaking. Recursively plugging the q_i's into our equation $f(x) = q_1(x)h(x) + r_0(x)$, we get:

$$f = r_0 + r_1 h + ... + r_s h^s.$$

Hence, the obvious choice for $g(t)$ is

$$g(t) = r_0 + r_1 t + ... + r_s t^s.$$

In our formulation, we need that g is a univariate polynomial of degree s, so we must have that $r_0, ..., r_q$ are all constants. Assuming they are constants, and using that we defined h

such that $h(0) = 0$, we get:

$$\begin{aligned} f(0) &= q_1(0)h(0) + r_0(0) = r_0 \\ q_1(0) &= q_2(0)h(0) + r_1(0) = r_1 \\ &\vdots \\ q_s(0) &= 0h(0) + r_s(0) = r_s \end{aligned} \tag{3}$$

So we have $r_i = q_i(0), i = 0, .., s$. Plugging in this condition, we can then rearrange the formulas to yield:

$$\begin{aligned} q_1(x) &= \frac{f(x) - r_0}{h(x)} = \frac{f(x) - f(0)}{h(x)} \\ &\vdots \\ q_s(x) &= \frac{q_{s-1}(x) - r_{s-1}}{h(x)} = \frac{q_{s-1}(x) - q_{s-1}(0)}{h(x)} \end{aligned} \tag{4}$$

and from this, we find $r_i = q_i(0)$. If at any point in **(4)** the division is not exact, there is no g corresponding to our candidate h, so we can stop the algorithm.

Example 3.1. Consider the polynomial:

$$f(x) = 2x^8 + 32x^7 + 195x^6 + 548x^5 + 656x^4 + 192x^3 + 2x^2 + 8x - 4$$

and a candidate polynomial for its decomposition $h(x) = x^2 + 4x$. We find:

$$\begin{aligned} q_1 &= \frac{f(x) - f(0)}{h(x)} = 2x^6 + 24x^5 + 99x^4 + 152x^3 + 48x^2 + 2 \\ q_2 &= \frac{q_1(x) - q_1(0)}{h(x)} = x(2x^3 + 16x^2 + 35x + 12) \\ q_3 &= \frac{q_2(x) - q_2(0)}{h(x)} = 2x^2 + 8x + 3 \\ q_4 &= \frac{q_3(x) - q_3(0)}{h(x)} = 2 \\ q_5 &= 0 \end{aligned}$$

So, according to our algorithm, $g(x) = 2x^4 + 3x^3 + 2x - 4$. As intended, we do have $f(x) = g(h(x))$.

Next we will bound the number of field operations of computing g assuming a naive cost table [1]. We have deg(f)= n, deg(h)= r and deg(g)= s. Notice that deg(q_i)= $n - ir$. From the naive cost table, computing q_{i+1} from q_i and h costs $O(r(n-ir))$ field operations. So our total cost is:

$$Cr\sum_{i=1}^{s}(n - ir) = Cr\left(sn - r\sum_{i=1}^{s} i\right) = Cr\left(sn - r\frac{s(s+1)}{2}\right) \in O(n^2) \quad \text{(since } sr = n\text{)}$$

So we need $O(n^2)$ field operations to compute g from f and h.

3.2 Finding *h(x)*: Attempt 1

Now that we know how to find $g(x)$ given a candidate h, we will derive our first algorithm for computing h. Notice that if we have found g and h such that $f = g \circ h$, according to *Theorem 2.1*, we know that $f(x) - f(y)$ is divisible by $h(x) - h(y)$. Suppose

$$g(x) = g_0 + ... + g_s x^s$$

for some $g_0, ..., g_s \in R$. Then

$$\begin{aligned} f(x) - f(y) &= g(h(x)) - g(h(y)) \\ &= g_1 h(x) + ... + g_s h^s(x) - g_1 h(y) - ... - g_s h^s(y) \\ &= g_1(h(x) - h(y)) + ... + g_s(h^s(x) - h^s(y)). \end{aligned} \tag{5}$$

From this, we can see that $h(x) - h(y)$ divides each term in equation **(5)**. As such, we only need to look for divisors of $f(x) - f(y)$ that do not have terms with mixed powers. Applying the reverse direction of *Theorem 2.1* shows us that every divisor of $f(x) - f(y)$ with no mixed powers is an element of our component $h(x)$. To find the decomposition of f, we factor $f(x) - f(y) = a_1(x, y)...a_r(x, y)$, where our a_i's are irreducible polynomials in $R[x, y]$. After we have our decomposition, we find the any combination of a_i's that have no mixed powers, and this combination is a candidate for $h(x) - h(y)$. If the trivial combination is the only one in which there are no mixed powers, the $h(x)$ we want does not exist. Once we have found our candidate, we can compute the corresponding $g(x)$ using *Algorithm 1*.

We can make some realizations to simplify this otherwise expensive computation. First, we note that we want $h(0) = 0$, so $x - y$ must divide $h(x) - h(y)$. Hence, $x - y$ also divides $f(x) - f(y)$. So when factoring $f(x) - f(y)$, we can instead factor

$$\frac{f(x) - f(y)}{x - y},$$

and since $x - y$ has no mixed terms, we can set it to $a_1(x, y)$, and always include $x - y$ when finding a candidate $h(x) - h(y)$. For convenience, we can also assume $h(x)$ is normalized and primitive. With this information, we can formalize the construction of $h(x)$ according to [1]:

Algorithm 2: Decomposition of a polynomial *f(x)* in a ring *R*

1. *Initialization:* set $h_0(x) = g_0(x) = x$
2. *Factor:* Factor $f(x) - f(y) = a_1(x, y)...a_r(x, y)$ over $R[x]$, $a_1(x, y) = x - y$
3. *Find Candidate:* Find product of a_j's with smallest possible degree that is a multiple of $h_{i-1}(x) - h_{i-1}(y)$ and has no mixed powers. Call this polynomial $h_i(x) - h_i(y)$
4. *p-adic Expansion:* Use *Algorithm 1* to find $g_i(x)$ such that $h_i(x) = g_i(h_{i-1}(x))$
5. *Loop:* If $h_i(x) = f(x)$, STOP, $f = g_i \circ ... \circ g_1$. Else, $i \leftarrow i + 1$ repeat step 3.

Example 3.2. Consider the following polynomial to decompose:

$$f(x) = -2x^4 - 16x^3 - 32x^2 + 3$$

Now we follow the steps of *Algorithm 2* to get:

$$\begin{aligned} f(x) - f(y) &= -2(x^4 - y^4) + 16(x^3 - y^3) - 32(x^2 - y^2) - 8(x - y) \\ &= -2(x - y)(x + 4 + y)(x^2 + y^2 + 4x + 4y) \\ &= -2(x^2 + 4x - y^2 - 4y)(x^2 + y^2 + 4x + 4y) \end{aligned}$$

So we have a candidate $h(x) - h(y) = x^2 + 4x - y^2 - 4y$, or $h(x) = x^2 + 4x$. Applying *Algorithm 1* to this h gives $g(x) = -2x^2 + 3$, which satisfies $f = g \circ h$, as desired.

According to Barton and Zippel [1], a cost analysis of the above algorithm is difficult, but is exponential in deg(f) in the worst case. This corresponds to the case where f is indecomposable, but $f(x) - f(y)$ can be decomposed into linear factors. The most expensive part of this algorithm comes in factoring $f(x) - f(y)$, so we might hope to find an algorithm where this computation is not necessary.

3.3 Finding *h(x)*: Attempt 2

In the previous algorithm, our set of potential candidates for decomposition is fairly small, because they need to not have any mixed powers of x and y (that is, we must be able to write our candidate as $h(x) - h(y)$). In this section, we will consider an algorithm that uses only univariate polynomials, so there is no criteria involving mixed powers. An upside of this algorithm is we do not have to factor a multivariate polynomial, which Barton and Zippel claim consumes most of the computation time of *Algorithm 1*. The downside to omitting the mixed-power criteria is we have tend to have substantially more candidate polynomials to consider.

Once again, we assume that we have f, g and h such that $f(x) = g(h(x))$, with the additional assumption that $h(0) = 0$. For this to hold, we must have $f(0) = g_0$. So in constructing h, we need only look at the divisors of $f(x) - f(0)$. We take

$$g(x) = g_0 + g_1 x + ... + g_s x^s,$$

then:

$$f(x) - f(0) = g_1 h(x) + ... + g_s h^s(x).$$

Since $h(0) = 0$, we know x divides $h(x)$, and must also divide $f(x) - f(0)$, so we include x in all our possible candidates for h. One further condition to limit our candidates comes in that deg(h) must divide deg(f). Finally, We assume that our candidate must also have degree > 1 to make our decomposition nontrivial. This gives rise to a simpler algorithm:

Algorithm 3: Decomposition of a polynomial *f(x)* in a ring *R* using univariate polynomials

1 *Initialization:* set $h_0(x) = g_0(x) = x$
2 *Factor:* Factor $f(x) - f(0) = a_1(x)...a_r(x)$ over $R[x]$, $a_1(x) = x$
3 *Find Candidate:* Find product of a_j's with smallest possible degree that is a multiple of $h_{i-1}(x)$. Call this polynomial $h_i(x)$.
4 *p-adic Expansion:* Use *Algorithm 1* to find $g_i(x)$ such that $h_i(x) = g_i(h_{i-1}(x))$
5 *Loop:* If $h_i(x) = f(x)$, STOP, $f = g_i \circ ... \circ g_1$. Else, $i \leftarrow i + 1$ repeat step 3.

Example 3.3. Consider the same polynomial as in *Example 3.2*:

$$f(x) = -2x^4 - 16x^3 - 32x^2 + 3.$$

We now factor $f(x) - f(0)$ as:

$$\begin{aligned} f(x) - f(0) &= -2x^4 - 16x^3 - 32x^2 + 3 - 3 \\ &= -2x^2(x^2 + 8x + 16) \\ &= -2x^2(x + 4)^2 \end{aligned}$$

Here we have three candidates: $h(x) = x$, $h(x) = x^2 + 4x$ and $h(x) = x^3 + 8x^2 + 16x$. We throw out the first option because it is linear, and the third option because its degree does not divide deg(f). We are left with $h(x) = x^2 + 4x$. Applying it to *Algorithm 1* once again yields $g(x) = -2x^2 + 3$.

Example 3.4. We will use the algorithm on perhaps a more interesting polynomial:

$$f(x) = -x^6 - 3x^5 - 3x^4 - x^3 + 4x^2 + 4x - 5$$

Factoring $f(x) - f(0)$:

$$\begin{aligned} f(x) - f(0) &= -x^6 - 3x^5 - 3x^4 - x^3 + 4x^2 + 4x \\ &= -x(x - 1)(x + 2)(x + 1)(x^2 + x + 2) \end{aligned}$$

So we have candidates:

$$\begin{aligned} h_1(x) = x^2 - x, \quad h_2(x) = x^2 + 2x, \\ h_3(x) = x^2 + x, \quad h_4(x) = x^3 + x^2 + 2x \\ \vdots \end{aligned}$$

There are several more candidates that satisfy deg(h_i)|deg(f), but according to *Algorithm 3*, we can check candidates incrementally using *Algorithm 1*:

1. For $h_1(x)$, finding $q_2(x)$ gives a non-exact division, so there is no corresponding $g(x)$.
2. For $h_2(x)$, finding $q_2(x)$ gives a non-exact division, so there is no corresponding $g(x)$.
3. For $h_3(x)$, all corresponding $q_i(x)$ divide exactly, and so:
$$g(x) = -x^3 + 4x - 5$$

In comparing *Example 3.2* and *Example 3.4*, we see that *Algorithm 2* seems to require more expensive factoring, whereas *Algorithm 3* often leads to more candidates that need to be

tested before a decomposition is found. Barton and Zippel [1] found that *Algorithm 3* typically runs slightly faster than *Algorithm 2*, but in some cases may take much longer than *Algorithm 2*. Consider the following case:

Example 3.5. Let $f(x) = 1 + x(x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)$, so:

$$f(x) - f(0) = x(x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)$$

In this case, we have deg(f)=8, so any combination of factors with degrees 2 or 4 is a candidate. Splitting into linear factors like this means steps *3 - 5* in *Algorithm 3* in will dominate the cost. In the case of *Algorithm 2*, factoring the associated bivariate polynomial immediately gives a single candidate $h(x) = x^2 - 7x$, which results in a consistent decomposition for f. If *Algorithm 3* is used, steps *3* to *5* will need to be repeated 7 times before the appropriate candidate is found.

Now that we have introduced and compared polynomial decomposition algorithms, we can look into applications of the techniques and results.

4 An Improved Algorithm for Polynomial Decomposition

The algorithms discussed in this paper were constructed in 1985 and in the worst case run in exponential time. These algorithms were chosen because of their relative simplicity, and to introduce the first publication of algorithms for polynomial decomposition. Shortly after, some breakthroughs were made to drastically decrease the worst-case runtime. In 1989, Kozen and Landau [2] introduced an algorithm that runs in $O(n^2 r)$ (where $n = \deg(f)$ and $r = \deg(g)$). Perhaps the most interesting aspect of the improved algorithm is at no point does it use polynomial factorization. In this section, we will look at the algorithm and do a cost analysis. For the sake of space, we will not investigate its derivation.

Suppose f, g and h are monic, where:

$$\begin{aligned} f &= x^{rs} + a_{rs-1}x^{rs-1} + ... + a_0 \\ g &= x^{r} + b_{r-1}x^{r-1} + ... + b_0 \\ h &= x^{s} + c_{s-1}x^{s-1} + ... + c_0 \end{aligned}$$

With this we can define our algorithm:

Algorithm 4: Polynomial decomposition over a Commutative Ring R

Solve for coefficients of h:

Input: $f = a_0 + a_1x + ... + x^{rs}$

1 set $q_0(x) = x^s$, find $q_0^i = x^{is}$ for $0 \leq i \leq r$

2 **for** $k = 1$ *to* $s - 1$ **do**

3 $d_k =$ coefficient of x^{rs-k} in q_{k-1}^r

4 $c_{s-k} = \frac{1}{r}(a_{rs-k} - d_k)$ (these are the coefficients of h corresponding to x^{s-k}), where a_{rs-k} are the coefficients of f

5 compute c_{s-k}^i for $0 \leq i \leq r$

6 set $q_k^i = \sum_{j=0}^{i} \binom{i}{j} c_{s-k}^j x^{j(s-k)} q_{k-1}^{i-j}$ for $0 \leq i \leq r$, with c_{s-k} found in step **4**

end

Solve for coefficients of g:

7 construct $A \in R^{n \times n}$ and $a \in R^n$ where:

A_{ij} is the coefficient of x^{is} in h^j for $0 \leq i,j \leq r$

$a = (a_0, ..., a_{rs})^\intercal$ from f

8 solve $Ab = a$ for $b = (b_0, ..., b_r)^\intercal \in R^r$

This algorithm is designed to solve rs equations for $r+s$ unknowns, so is over-determined. Thus, if step **(7)** does not have a solution, a decomposition with $\deg(h) = s, \deg(g) = r$ does not exist. We can now analyse the cost of computing h and g: step **(6)** dominates the cost of computing h, with $O(n^2r)$ operations in R. In computing g, our matrix A is triangular and we can thus solve for b in $O(r^2)$ operations. Our total cost for *Algorithm 4* is $O(n^2r)$. This is a substantial improvement from our previous two algorithms in multiple ways: it is easy to analyse, concise, requires no polynomial factoring, and some practical implementation has shown it is consistently faster than Barton and Zippel's methods.

With all these algorithms in our toolkit, we can now investigate some applications.

5 Applications

Polynomial decomposition has applications ranging from root finding, to fast evaluation at many points. In this section, we will investigate some of the applications through example.

Example 5.1. Consider the polynomial

$$f(x) = x^6 + 3x^5 + 6x^4 + 7x^3 + 6x^2 + 3x + 10.$$

We can evaluate f at a point $x = \alpha$ using *Horner's scheme* with 6 multiplications and 6 additions. Instead, if we decompose f into $g \circ h$ with $g(x) = x^3 + 9$ and $h(x) = x^2 + x + 1$, we can evaluate $f(\alpha)$ using standard polynomial evaluation with only 3 multiplications and 4 additions, saving 3 multiplications and 2 divisions per evaluation. So if we needed to evaluate a polynomial at very many points, it could be substantially more efficient to

first find an appropriate decomposition for the polynomial.

In *Section 1*, we outlined a method for finding roots of a polynomial by first finding a decomposition, and then solving for the roots with respect to the decomposition. Our example only looked at integer polynomials with integer roots. We can easily extend the procedure to $\mathbb{R}$ and $\mathbb{C}$, and greatly simplify root finding procedures:

Example 5.2. Let

$$f(x) = x^4 + 2x^3 + x^2 + 1 \in \mathbb{C}[x].$$

This polynomial has no linear factors in $\mathbb{Z}[x]$, so we would have to use the quartic formula to find its roots. The quartic formula is, however, massive and expensive. We instead perform a polynomial decomposition using *Algorithm 3* on f to find $f = g \circ h$ for $h(x) = x^2 + x$ and $g(x) = x^2 + 1$. Now $g(x)$ has roots at $x = \pm i$, so we find the roots of:

$$h(x) - i = x^2 + x - i$$
$$h(x) + i = x^2 + x + i$$

now $h - i$ has roots $x = \frac{-1 \pm \sqrt{1+4i}}{2}$, and $h - i$ has roots $x = \frac{-1 \pm \sqrt{1-4i}}{2}$, which we know must be the roots of f.

References

[1] D. R. Barton and R. Zippel. Polynomial decomposition algorithms. *Journal of Symbolic Computation*, 1(2):159–168, 1985. doi: 10.1016/s0747-7171(85)80012-2.

[2] D. Kozen and S. Landau. Polynomial decomposition algorithms. *Journal of Symbolic Computation*, 7(5):445–456, 1989. doi: 10.1016/s0747-7171(89)80027-6.