

Subset-Sum in $O(n^{11} \log(n))$

Rion Tolchin
riont@mit.edu

September 14th, 2022

Abstract

This paper describes the "Dragonfly Algorithm" by which the subset-sum problem can be solved in $O(n^{11} \log(n))$ time complexity; it will first detail the generalized "product-derivative" method (and the more efficient version of this method which will be used in the algorithm) by which a pair of monic polynomials can be used to generate a system of unique monic polynomials for which each polynomial in the system will share with every other a set of roots equivalent to the intersection of the roots of the original pair; this method will then be applied on a pair of polynomials one of which, $\phi(x)$, exhibiting known roots based on the instance of the subset-sum problem and the other of which, $t(x)$, containing unknown placeholder coefficients and representing an unknown subset of the linear factors of $\phi(x)$.

Foreword

This paper is a continued revision of a paper I released to faculty—first of MIT, then to multiple universities—starting on March 7th and 8th and continuing through March 2022, the most recent revision having been posted in April 2022.

In a previous version of this paper, I included a brief summary in Section 5.4 about how it may be even more efficient to solve the same-sum partition problem by substituting in the known sum for the final coefficient a_{d-1} instead of iterating over multiple possible a_{d-1} of known order.¹ This edition, like the last edition, of the paper is a revision which further improves the description and readability of the detailing of the Dragonfly Algorithm by applying it to the subset-sum problem^[1] and contains improvements on the efficiency of certain processes and on the calculated complexity bound.²

1 Introduction

The Dragonfly Algorithm, primarily detailed in Section 4, is a polynomial time-complexity process to solve an instance of the subset-sum problem of known NP-Complete complexity. Hence the following result:

Theorem 1.1. *$P=NP$, or rather the complexity classes of polynomial-time solvable problems and non-deterministically polynomial-time solvable problems are equivalent.*

Proof of Theorem 1.1. The result follows constructively from the processes described in Section 4 and their necessary determination of a solution to an instance of the subset-sum problem, Lemma 4.2, with the fact that the processes described require no more than $O(n^{11} \log(n))$ time-complexity, Lemma 4.1. \square

¹Since the same-sum partition problem is a special instance of the subset-sum problem, the Dragonfly Algorithm works on it with one efficiency improvement being that in place of iterating over possible sizes of subsets, the only degree of the partition polynomial $t(x)$ needing to be checked is $\frac{1}{2}|S|$ resulting in $O(n^{10} \log(n))$ time complexity on that version of subset-sum.

²Special thanks go to Professor Ryan Williams for, in response to one of the previous editions of the paper, pointing out that the early sections (used to encode Monotonic Exactly-1-in-3-SAT as an integer partition problem modulo a commutative ring) were similar to existing reductions from 3-SAT to subset-sum.

Abstractly, the result is possible by the embedding of an instance of the subset-sum problem into a relationship between the roots of monic polynomials with linear factors corresponding to the relevant set and subset(s) with the correctly related set of unique monic polynomials being generable by the use of multiplicity ≥ 1 roots and derivation.

2 Product Derivatives

2.1 General Product Derivative Method

Definition 2.1. For two polynomials, $A(x) \in \mathbb{R}[x]$ and $B(x) \in \mathbb{R}[x]$, define their ϵ th product-derivative to be $R_{\epsilon,AB}(x) = \frac{d^\epsilon}{dx^\epsilon} (A^{\delta_1}(x)B^{\delta_2}(x))$ where $\delta_1 + \delta_2 = \epsilon + 1$.

Theorem 2.1. For any two polynomials $A(x) \in \mathbb{R}[x]$ and $B(x) \in \mathbb{R}[x]$, their product-derivatives $R_{\epsilon,AB}$ will necessarily have a set of real roots including the intersection of the sets of real roots of $A(x)$ and $B(x)$ respectively.

Proof of Theorem 2.1. For any real root shared between any such $A(x) \in \mathbb{R}[x]$ and $B(x) \in \mathbb{R}[x]$ with multiplicities m_A and m_B respectively, the product $A(x)B(x)$ will contain that root of multiplicity $m_A + m_B$. This is a direct consequence of the fact that the product may be rewritten as

$$(x-r)^{m_A}(x-r)^{m_B} \frac{A(x)}{(x-r)^{m_A}} \frac{B(x)}{(x-r)^{m_B}} = (x-r)^{m_A+m_B} \frac{A(x)B(x)}{(x-r)^{m_A+m_B}}$$

using standard algebra. Then, by Lemma 2.2, the ϵ th derivative of $A(x)B(x)$ must contain the factor $(x-r)$ so long as $m_A + m_B > \epsilon$.

By the definition of a product derivative, it must be true that $R_{\epsilon,AB}$ necessarily contains every real root shared by both $A(x)$ and $B(x)$ with multiplicity $m > \epsilon$ because $A^{\delta_1}(x)$ contains—with multiplicity $\delta_1 + m$ —any real root of $A(x)$ with original multiplicity m , and vice versa for $B^{\delta_2}(x)$. \square

Lemma 2.2. For any polynomial $f(x) \in \mathbb{R}[x]$, the set of roots of $\frac{d^\epsilon}{dx^\epsilon} f(x)$ will necessarily contain every real root of $f(x)$ that is of multiplicity $m > \epsilon + 1$ in $f(x)$.

Proof of Lemma 2.2. Any $f(x)$ as described may be written as

$$f(x) = \prod_{i=1}^d c_i(x - r_i)$$

where r_i is the i th real root of $f(x)$, d is the degree of $f(x)$, and c_i is a leading constant of that linear factor. For any real root of multiplicity in $f(x)$, $m_i > 1$, it will be the case that there must be m distinct indices of i for which the roots r_i are all identical. Another way of saying this is that each integer root will correspond to m values of i where m is that root's multiplicity.

Then, by the extended product rule, we get that

$$\frac{d}{dx} f(x) = \sum_{j=1}^d \left(\prod_{i=1, i \neq j}^d c_i(x - r_i) \right)$$

which is the sum of $\binom{d}{d-1}$ of the linear factors of $f(x)$. In each term being summed, exactly one linear factor has had its derivative taken thus taking that factor from $c_{i_1}(x - r_{i_1})$ to c_{i_1} . Therefore, for any real root, r_{i_1} , of $f(x)$ that is of multiplicity $m_{i_1} > 1$, there must be at least one other index, i_2 , for which the linear factor $c_{i_2}(x - r_{i_2})$ is still present in that summed term where $r_{i_1} = r_{i_2}$. In fact, there will be exactly $m_{i_1} - 1$ other such duplicate factors of that term.

Using the fact that there will be exactly $m_{i_1} - 1$ other such linear factors in each summed term corresponding to the "removed" linear factor, induction over the single derivative case into the ϵ th derivative case shows that for each summed term of the ϵ th derivative there must still be a linear factor corresponding to every root of multiplicity $\epsilon + 1$. \square

2.2 Alternative Product Derivative Processes

Following the same rationale, a system of monic polynomials sharing the same set of roots may be constructed not by using higher powers, δ_1 and δ_2 , of monic polynomials $A(x)$ and $B(x)$, and consequently higher order derivatives, but by instead generating each equation as $R_{i,AB} = \frac{d}{dx} [(x + c_i)A(x)B(x)]$ for a set of distinct constants, $\{c_i\}$, such that no $(x + c_i)$ is a linear factor of either $A(x)$ or $B(x)$.

Lemma 2.3. *For distinct constants, $c_i \neq c_j$, the modified product-derivatives $\frac{d}{dx} [(x + c_i)A(x)B(x)]$ and $\frac{d}{dx} [(x + c_j)A(x)B(x)]$ will not share an additional root other than the intersection of the roots of $A(x)$ and $B(x)$, even with the $(x + c_{i,j})$ terms.*

Proof of Lemma 2.3. Applying the standard product rule, the two described derivatives are as follows:

$$\frac{d}{dx} [(x + c_{i,j})A(x)B(x)] = A(x)B(x) + (x + c_{i,j}) \frac{d}{dx} [A(x)B(x)]$$

which, for $c_i \neq c_j$, will not share between each other an additional root other than the shared roots of $A(x)B(x)$ and $\frac{d}{dx} [A(x)B(x)]$ which, by the described rationale of each root of $A(x)B(x)$ decreasing in multiplicity by one when taking $\frac{d}{dx} [A(x)B(x)]$ hence making the shared roots between $A(x)B(x)$ and $\frac{d}{dx} [A(x)B(x)]$ the roots of $A(x)B(x)$ of multiplicity two or greater, will be the intersection between the roots of $A(x)$ and $B(x)$. \square

It is apparent that this method will result in a system of equations (to be used in the Dragonfly Algorithm) of lower degree and coefficient through the avoidance of repeated multiplications of $\phi(x)$ and derivations; hence, it presents the straightforward computational efficiency motivation. This entails the following, slightly distinct, notation.

Definition 2.2. Let $R_{i,AB} = \frac{d}{dx} [(x + c_i)A(x)B(x)]$ where $(x + c_i)$ is not a possible shared linear factor of monic polynomials $A(x)$ and $B(x)$.

Then the necessary system of equations for the algorithm will be a set of $R_{i,\phi t}(x) = 0$ for distinct i (this notation is further elaborated in Sections 3 and 4).

Naturally, the original product derivative method is also sufficient to maintain the efficient polynomial time complexity of the algorithm.

3 Non-cancelling of Placeholder Coefficients in Algebraic Substitution

This section is recommended to be read after (or in close conjunction with) Section 4. In Section 4's description of the algorithm's steps, a system of equations, $R_{i,\phi t}(x) = 0$ for varying i , will be generated in terms of x and the placeholder coefficients a_k for $k = 0$ to $k = d - 2$ where d is the degree of the expected valid partition polynomial, $t(x)$. Then, algebraic substitution will be used to solve for the placeholder coefficients strictly in terms of x .³ To prevent cases in which—during this algebraic substitution of variables—the coefficient of a given variable becomes 0,⁴ the following constraint applies.

³This follows from the conventional process of using n equations to solve for n unknown variables.

⁴As in this arbitrary example scenario, substituting $2y = -3z - x$ into $3z + 2y + x = 0$ which just gives an identity $0 = 0$ providing little information.

3.1 Recursion Constraint

For the set of $R_{i,\phi t}(x)$ having been solved in terms of the placeholder coefficients, a_k , by setting x equal to a trial integer, x_0 , which is expected to be in a desired subset of S , S_T , of target sum T , the set of $R_{i,\phi t} = 0$ equations will take the form of $R_{i,\phi t} = {}_i q_d + T {}_i q_{d-1} + {}_i q_{d-2} a_{d-2} + \dots + {}_i q_1 a_1 + {}_i q_0 a_0 = 0$ for unique constant terms, ${}_i q_k$ corresponding to the value of the x terms at $x = x_0$.

The substitution rewriting process for the constant coefficients, ${}_i q_k$, of a given placeholder coefficient, a_k , will follow the form of rewriting ${}_1 q_k a_k \rightarrow ({}_0 q_0 {}_1 q_k - {}_0 q_k) a_k$, then ${}_2 q_k a_k \rightarrow ({}_1 q_1 {}_2 q_k - ({}_0 q_0 {}_1 q_k - {}_0 q_k)) a_k$, and so forth where each of these new coefficients of a_k must be nonzero for all k values greater than or equal to the iteration instance; thus, the recursion process can be generalized in the following form:

$${}_i Q_k = {}_i q_i ({}_{i+1}) q_k - ({}_{i-1}) Q_k$$

where ${}_{(-1)} q_k = 1$ and will be subject to the constraint

$$\forall k | (k > i \geq 0) : {}_i Q_k \neq 0$$

for the purpose of avoiding the cancellation of terms.⁵

3.2 Algorithmic Implementation

In implementation of the algorithm, this constraint may be accomplished in the following manner: firstly, using a set of c_i for which every c_i is greater than the maximum integer element of set S to generate the $R_{i,\phi t}(x_0)$, upon performing the algebraic substitution of the placeholder a_k constants, if at any step the solved constant multiple of an a_k is 0 in an equation, recreate that and every higher indexed (higher indexed implying $R_{i,\phi t}$ generated with a c_i greater than the c_i for that equation) equation by incrementing all of their c_i values up one and then regenerating their equations and re-performing the previous substitution steps on them. This technique would, in a hypothetical edge-case, at most require regenerating each equation once for every a_k and then regenerating every higher indexed equation at each of those instances, resulting in a hypothetical upper-bound of $(d-1)^2 + (d-1)(d-2) + \dots + (d-1)(2) + (d-1) = (d-1) \sum_{v=0}^{d-1} v$, where d is the degree of the expected partition polynomial $t(x)$, necessary instances of regenerating an individual $R_{i,\phi t}$ with a new c_i value.

For a provisional bound on the maximum c_i , even in the hypothetical upper-bound scenario, the maximum value of c_i used will be less than $(d_{max} - 1)^3$ multiplied by the maximum integer element in S . With a maximum degree being the number of elements in S , the highest c_i will be less than $(|S|^3)_{s_{max}}$.

In practice, slightly more efficient implementation will be obtainable by, rather than using a minimum c_0 that is still greater than the maximum element of set S , using a set of c_i for which no c_i is an element of S and, at each step of incrementing an equation's and all higher indexed equations' unique c_i , shift each equation's c_i not simply upwards by an increment of 1, but shift each affected equation's c_i value to the next lowest integer that is greater than its pre-shift value, greater than every lower indexed equation's post shift c_i values, and is not an integer that is an element of S .

For the purposes and scope of this paper, the previous upper-bound that $c_{max} < (|S|^3)_{s_{max}}$ is sufficient. With the simplifying bound of the integers in S being of a number of bits on the order of $|S|$, labeled n , this results in a c_{max} on the order of $\log(n)$ bits; the additional rewriting steps will, in the upper-bound case, require an extra $O(n^3)$ steps of polynomial generation.

⁵An initial ${}_i q_k$ of an a_k won't be 0 since derivation was applied to $(x + c_i)\phi(x)a_k$ which doesn't contain double roots.

4 Subset Sum

4.1 Algorithmic Steps

Let S be a set of positive integers (without duplicates) and T be a target sum where the hypothetical subset of S being searched for, S_T , contains elements that exactly sum to T . For complexity calculations, $n = |S|$ and the positive integer elements of S are assumed to contain $O(n)$ bits.

1. Construct polynomial $\phi(x)$ by the following definition:

Definition 4.1.

$$\phi(x) = \prod_{i=1}^{|S|} (x + r_i)$$

where every r_i is an integer corresponding to an element of the set S . In other words, $\phi(x)$ is a monic polynomial with known integer coefficients which is of degree $d = |S|$ and which has roots exactly equal to the sign inversion of the set of integers, S .

2. Now, loop over all sizes of the possible subset S_T . This is simply a loop from 1 to $|S|$.⁶
 - (a) Now, create a polynomial, $t(x)$, of degree, d , equal to the currently looped over size from step 2; this $t(x)$ will be written in terms of placeholder constant coefficients, a_0 through a_k , and will be constrained (by the later algebraic processes) to have roots corresponding to the sign inversion of a subset, S_T , of S iff such a subset exists containing elements that exactly sum to the target sum, T . This will correspond to setting

$$t(x) = x^d + Tx^{d-1} + a_{d-2}x^{d-2} + \dots + a_1x + a_0$$

for each $d = 1$ through $d = |S| - 1$. This takes $O(n)$ steps where $n = |S|$. The a_{d-1} term is constrained to equal T due to the a_{d-1} term of a monic polynomial being equivalent to the sum of the sign inversion of the polynomial's roots.

- (b) Now generate a system of $d - 1$ unique equations, $R_{i,\phi t} = \frac{d}{dx} [(x + c_i)\phi(x)t(x)]$, via the process described in Section 2.2.
- (c) Loop over the elements, s_j , of set S as expected elements of set S_T to be searched for. Rewrite the system of equations strictly in terms of integer multiples of the placeholder constants, a_k , and integer constants by setting x to equal the sign inversion of the current expected element (current s_j in the loop) of S_T (labeled $x_0 = -s_j$ in Section 3).
 - i. Perform algebraic substitution (along with the additional described steps in Section 3 if the relevant situation occurs) on the new system of equations strictly in terms of integer multiples of placeholder a_k to solve each a_k as a constant. Due to the equations being $R_{i,\phi t}(x_0) = 0$ and $t(x) = 0$, denominator terms will be circumvented just by multiplying them out.
 - ii. Substitute the solved a_k placeholder constants into the original $t(x) = 0$ partition equation (multiplying out all denominator terms) to get a rewritten $t_c(x)$ as a monic polynomial with known integer coefficients.
 - iii. Iterate over all elements, s_j , of set S and, setting $x_0 = -s_j$, solve $t_c(x_0)$, keeping track of all elements for which $t_c(x_0) = 0$ and hence correspond to roots of $t_c(x)$. By Lemma 4.2, iff the set of elements of S that are sign inverted roots of $t_c(x)$ is of magnitude d , then this set is a valid S_T and the instance of the subset-sum problem is solved.

⁶In practice, such edge cases as $|S_T| = 1$ or $|S_T| = |S|$ may be much more efficiently checked by just checking each element of S for equivalence with T or checking if the sum of all elements in S equals T .

3. If all such previous steps and loops of the algorithm have been completed without there having been found a valid S_T of the target sum T , then (by Lemma 4.2) no such S_T exists and the determination of this information constitutes the instance of the subset-sum problem being solved.

4.1.1 Complexity Bounding

Lemma 4.1. *The complexity of the described algorithm is $\leq O(n^{11} \log(n))$.*

Proof of Lemma 4.1. With $n = |S|$ and $s_{max} \sim 2^n$, the generation and storing of the information describing a given polynomial equation with integer and placeholder a_k coefficients (including $\phi(x)$, $t(x)$, and the $R_{i,\phi t}(x)$ equations from steps 1, 2a, and 2b respectively) requires an upper-bound of $O(n^5 \log(n))$ steps from there being $O(n)$ placeholder coefficients being multiplied by $O(n)$ terms of powers of x each with integer coefficient having been the product of $O(n)$ steps of integer multiplication each of which having been performed in $O(n^2 \log(n))$ time (from the integers using $O(n)$ bits^[2]).

There being $O(n)$ such equations constructed per the $O(n)$ possible values of $x_0 = -s_j$ looped over in step 2c brings the complexity to $O(n^7 \log(n))$. Rewriting a given equation for $x = x_0$ entails a maximum of d (hence $O(n)$) integer multiplications per x term (by raising x_0 to the term's power) bringing an upper-bound of the complexity to $O(n^8 \log(n))$. The additional steps detailed in Section 3 during the process of this rewriting of the system of equations for $x = x_0$ contributes the factor of $O(n^3)$ steps bringing the complexity upper-bound to $O(n^{11} \log(n))$. Those last two complexity contributions are of leading order in the sub-processes of step 2c hence implying an upper-bound on the algorithm's complexity of $O(n^{11} \log(n))$. \square

4.2 Bijection Between S_T and Generated $t_c(x)$

It follows directly from the fact that the a_{d-1} coefficient of a monic polynomial is equivalent to the sum of the sign inverted roots of the polynomial that, for any such valid S_T with the sought target sum, T , there will be a polynomial simply equal to the product of $(x + r_i)$ for all elements, r_i , of S_T which will have $a_{d-1} = T$ and have roots equal to the sign inversion of the elements of S_T .

To show that iff there is a valid S_T then the $t_c(x)$ (of degree $|S_T|$) constructed by the algorithmic processes in Section 4.1 will be such a representative polynomial of an S_T of $|S_T| = d$, it is relevant to follow the algebraic processes in the construction and substitution of variables in the system of equations of integer multiples of a_k .

Lemma 4.2. *The determination of a $t_c(x)$ with all of its roots being sign inverted elements of S will occur iff there is a valid subset S_T .*

Proof of Lemma 4.2. If there exists a valid $t_c(x)$ of degree d , then the placeholder constants, a_k , in $t(x)$ which were used to generate the system of equations $R_{i,\phi t}(x) = 0$ may have their integer values from $t_c(x)$ substituted for each a_k placeholder in each equation and at any step in the algebraic rewriting of the a_k without there being any mathematical contradiction with the equivalences dictated by the equations. By Definitions 2.2 and 4.1 and Lemma 2.3, the set of equations $R_{i,\phi t_c}(x) = 0$ will necessarily be valid for each value of x that is within the intersection of the roots of $\phi(x)$ and $t_c(x)$. Thus, by the described non-contradiction of substituting any placeholder a_k for its integer value in $t_c(x)$ in any of the $R_{i,\phi t}(x) = 0$ equations, the equations $R_{i,\phi t}(x) = 0$ will also necessarily be valid for each value of x that is within the intersection of the roots of ϕ and $t_c(x)$.

Therefore, if there is a valid $t_c(x)$, which is a monic polynomial with integer coefficients for which the roots of $t_c(x)$ are a subset of the sign inversions of the elements of S and for which the a_{d-1} coefficient is equal to target sum T , the processes used to construct the $R_{i,\phi t}(x_0) = 0$ equations (for $x_0 = -s_j$ where $s_j \in S_T$) will result in a set of algebraic steps that validly (non-contradictorily) solve for each a_k and hence produce $t_c(x)$.

If no such S_T exists (for given size $|S_T| = d$), then the solved integer values of a_k when substituted into $t(x) = 0$ will result in a generated $t(x)$ polynomial with a set roots that does not equal the sign inversions of a subset of set S (due to every possible multiplication of d linear factors $(x + r_i)$ for $r_i \in S$ resulting in a value of a_{d-1} that is not equal to T and hence would be a contradiction with the initial $t(x)$ constructed with $a_{d-1} = T$). \square

Notably, if there exist multiple distinct subsets of S with elements summing to target value T , the specific subset corresponding to the generated $t_c(x)$ polynomial will depend on the equations used in the system of equations and will hence depend on the specific choice of constants c_i in the additional linear factors $(x + c_i)$.

Conclusion

This edition of the paper hopefully improves the tractability of the Dragonfly Algorithm's description and better emphasizes the essential components its processes. For questions or inquiries about the algorithm, I can be contacted by my university email, riont@mit.edu.

References

- [1] Horowitz, Ellis, and Sartaj Sahni. "Computing partitions with Applications to the Knapsack Problem". *Journal of the ACM*, vol. 21, no. 2, 1974, pp. 277-292., <https://doi.org/10.1145/321812.321823>.
- [2] Harvey, David, and Joris Van der Hoeven. "Integer Multiplication in Time $O(n \log n)$." *Annals of Mathematics*, vol. 193, no. 2, 2021, <https://doi.org/10.4007/annals.2021.193.2.4>.

Acknowledgements

I would like to thank everyone who gave me any kind of feedback on the original editions of the paper: Ryan Williams, Henry Cohn, Bjorn Poonen, Thomas Cormen, and Boaz Barak. I would like to specifically thank Ryan Williams for his response to one of the early editions of the paper referenced in footnote 1 (as well as for his comment on one of the previous edition's references to a theorem which would have altered the scope of the result) as well as Henry Cohn for his response on a counter-example for one of the intermediary results in one of the earlier editions of the paper which resulted in me simplifying the paper by circumventing the result.

.....

Dedicated to my love, support, friend, inspiration and muse, Rosaline; dedicated to my mother, Jo, to my father, Alex, to my stepmother, Darlene, to my friends Ian, Tyler, Camille, Jasmine, Angela, to my friends from MIT, Alassia, Nina, Thomas, Christian, Winston, Janice, Caleb, Andrew, Julian, Greyson, Carlos, Ben, Alex, Alexa, to Amber, Marci, Julian, and to all of my family, to my friends and family who have supported me in recent time, Christina, Josh, Max, Jack, Sam, Tara, Kathryn, Cliff, Isaac, my advisor Marin, my departmental advisor Catherine, to all of the professors and teachers and role models for the ways they've inspired me over the years such as Mr. Mycroft who first introduced me to complexity theory, my advisor Patrick, my philosophy teacher Carl, to all of my loved ones (encompassing those thus far and even encompassing myself), to my allies, to humanity and the world and all of the life of the world, as well as to the Father of Christ, from whom much of my inspiration was received.