The Polynomial Method and All-Pairs Shortest Paths

May 8, 2016

Abstract

We examine a recent development in circuit complexity used to speed up algorithms with circuit representations, and how it can be applied to the All-Pairs Shortest Path problem specifically. Additionally, we look at a few other problems on which improvements can be made in this manner. We also examine other consequences of possible extensions of this work.

1 Introduction

All-Pairs Shortest Paths (APSP) is a very well-known problem in theoretical computer science, and a staple of introductory algorithms classes. Given a graph with n nodes, the goal is to construct a data structure which can in $\tilde{O}(1)$ time produce the length of a given shortest path or in $\tilde{O}(l)$ time produce a shortest path of length l.

The canonical solution to APSP is the Floyd-Warshall algorithm, published in 1962 by Robert Floyd and Stephen Warshall independently; it solves the problem relatively simply by repeatedly improving estimates of optimal paths and runs in time $O(n^3)$ where n is the number of vertices in the graph. Many solutions to this problem are based on min-plus matrix multiplication - it was shown in the 1970s that these two problems are deeply related and an improved result on one leads to an equal result on the other.

2 The Polynomial Method in Circuit Complexity

Recall that a boolean circuit is a mathematical object represented by a directed acyclic graph, each of whose nodes is a boolean function. In particular we are interested in the class ACC of boolean circuits, where the functions are restricted to be NOT, OR, AND, and $MODn$ for some value of n, where $MOD3(a_1, \ldots, a_k) = 1$ iff $\sum_i a_i \equiv 0 \mod n$. ACC is reasonably well-studied and useful; it is known not to contain $NEXPTIME$.

2.1 The Polynomial Method

The polynomial method in circuit complexity refers to show that circuits are computable by some simple (low-degree, low-number of terms, etc.) polynomial,

and then use properties of polynomials to prove things about the circuits. For example, the polynomial method has been used to show lower bounds on the complexity of circuits to compute problems such as $MAJORITY$ in the 1980s. However, it was much more recently that the technique has been applied to solving algorithmic problems with polynomial methods.

2.2 Polynomial Representations

There are several different kinds of polynomial representations that can be made of a circuit. The most intuitive and simplest to define are exact representations, defined as follows:

Definition A polynomial $p(x_1, x_2, \ldots, x_n)$ over a ring R is an *exact representation* of a boolean function f if for all inputs $a_1, \ldots, a_n \in \{0,1\}^n$, $f(a_1, \ldots, a_n) = p(a_1, \ldots, a_n)$.

For example, we might represent the AND function as $f(x_1, x_2) = x_1 x_2$, so we say f is an exact representation of AND.

Exact representations are the most obvious kind of representation but less useful for our purposes than probabilistic representations and symmetric representations due to their being larger and harder to shrink. For dealing with problems like $PARITY$, however, variants of them are quite useful.

2.2.1 Probabilistic polynomials

We now define the polynomial analog of a randomized algorithm.

Definition [1] A distribution D of polynomials is a *probabilistic polynomial representing f with error δ* over a ring R if, for all $(a_1 \ldots a_n) \in \{0,1\}^n$, $Pr_{p \sim D}[p(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)] > 1 - \delta$.

Probabilistic polynomials were used in some important results in circuit complexity by Razborov and Smolensky in 1987 to show that various functions cannot be computed with ACC circuits satisfying certain properties. Here is a theorem used in order to show these things to be impossible to compute with ACC circuits.

Theorem 2.1 *[2] For every ACC circuit C of depth d, size s, modulus 2, and n inputs, for every $\epsilon > 0$, there is a probabilistic polynomial D_C over $\mathbb{F}_2$ with error ϵ, and degree at most $(4 \log s)^{d-1} \cdot (\log 1/\epsilon)$, such that to sample a polynomial from D_C takes $n^{O(\log s)^{d-1} \cdot (\log 1/\epsilon)}$ time.*

Note that this is a useful theorem because, if we show that having a polynomial satisfying these properties compute a certain problem (e.g. $MAJORITY$) is hard, then it is also hard to have an ACC circuit with $m = 2$ that solves that same problem. A more general result is found below which allows any value of the modulus for a not-too-substantial increase in size. We can also use this result to generate a distribution which can be sampled from in a randomized algorithm for solving a problem expressible as a circuit.

The proof of Theorem 2.1 is a construction involving replacing each gate with a low-degree function.

- NOT gates are exactly represented with a simple polynomial $g(h) = 1+h$.
- $MOD2$ is easy in $\mathbb{F}_2$; gates are exactly represented with a simple polynomial $g(h_1, \ldots, h_n) = 1 + \sum_i h_i$.
- OR is the interesting and nontrivial polynomial here. To sample an OR polynomial with error ϵ, we repeatedly sample random subsets of the inputs and sum them. A sum will only be 1 if the OR is true, and the probability that will happen is $\frac{1}{2}$. Thus, if we sample several times and multiply the outputs together (rather, multiply the inverses together and take the inverse), we can, with degree logarithmic in $\frac{1}{\epsilon}$, simulate an OR gate.
- Finally, AND can be represented as an OR:
  $AND(a_1, \ldots a_n) = NOT(OR(NOT(a_1), \ldots, NOT(a_n)))$

By combining the gates described above, we only use logarithmically many gates at each level in the size of the circuit, so the entire polynomial is reasonably low-degree. For another $(logs)^{d-1}$ factor, we can use any field $\mathbb{F}_p$ in OR gates (Aspnes, Beigel, Furst, Rudich 1994), allowing the use of this theorem more generally.

2.3 Symmetric Representations

A third kind of representation, and one that is useful for making deterministic algorithms for problems like APSP is called a *symmetric representation*.

Definition A polynomial $h(x_1, \ldots, x_n)$ over a ring R and an arbitrary function g over the image of h that maps to $\{0, 1\}$ are a *symmetric representation* of a boolean function f if for all $(a_1, \ldots, a_n) \in \{0, 1\}^n$,

$$f(a_1, \ldots, a_n) = g(h(a_1, \ldots, a_n))$$

holds.

Symmetric representations are so-named because the output function g does not depend on any one input but can be used for a more general symmetric purpose like checking whether there are at least a certain number of the inputs are true. Recall that $MAJORITY$ is hard for polynomials to compute, so we gain a reasonable amount of power by adding this extra output function.

3 All-Pairs Shortest Paths

Papers before Williams 2014[3] came up with only bounds of $n^3/\log^k n$ for $k < 3$ on APSP runtime for dense graphs. APSP has more solutions that work much better on sparse graphs, but little progress has been made on a truly subcubic algorithm for dense APSP.

3.1 OR-AND-COMP

Define the boolean function $OR - AND - COMP$ on $2d^2 \log n$ inputs as the or over all rows of the and over all columns of a table of comparisons between the

2 tables of input. That is, is there any i for which every entry in row i in the left half of the input is larger than the corresponding element in the right half? More formally,

$$OR-AND-COMP(a_{1,1}, a_{1,2}, \ldots, a_{d,d}, b_{1,1}, b_{1,2}, \ldots, b_{d,d}) = \bigvee_{i=1}^{d} \bigwedge_{j=1}^{d} [a_{i,j} \leq b_{i,j}]$$

Theorem 3.1 *[3] Given two sets $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ of n vectors of length d^2, if the function $OR - AND - COMP$ can be computed in $\tilde{O}(n^2)$ time for all n^2 pairs (a_i, b_j) simultaneously, then APSP is solvable in $\tilde{O}(n^3/d)$ time.*

This works basically because we are doing a part of the min-plus matrix multiplication, and by combining a lot of these, we cover a whole matrix. Or it can be thought of as APSP on a smaller tripartite graph whose partitions have size n, d, and n, many of which compose together the whole general graph.

Now we show that if Theorem 3.1 is true, we can solve APSP quickly. We need to choose a small value of d in order for this to take $\tilde{O}(n^2)$ time instead of $O(n^2 d^2 \log n)$. Choose $d = 2^{c\sqrt{\log n}}$. Now we need to compute $OR - AND - COMP$ in a short amount of time with this value of d. We are trying to simulate d^2 comparisons n^2 times in $\tilde{O}(n^2)$ time, so we approximate the comparisons with low-depth circuits.

Before we begin making a small polynomial for an $AND - OR - COMP$ circuit, we write the following claim.

3.2 Circuit shrinking

Consider the entries of the matrices A and B to be bit strings of length $1+\log n$. Then consider the following construction of a comparison circuit:

$$LEQ(a,b) = \left(\bigwedge_{j=1}^{t} (1 + a_i + b_i) \right) \oplus \bigvee_{i=1}^{t} \left((1 + a_i) \wedge b_i \wedge \bigwedge_{j=1}^{i-1} (a + a_j + b_j) \right) \quad (1)$$

$$= \left(\bigwedge_{j=1}^{t} (1 + a_i + b_i) \right) \oplus \bigoplus_{i=1}^{t} \left((1 + a_i) \wedge b_i \wedge \bigwedge_{j=1}^{i-1} (a + a_j + b_j) \right) \quad (2)$$

The first part of the circuit activates if $a = b$, the second if $b > a$, by checking each bit whether it's the bit at which they differ. The OR can be replaced by an XOR since only one clause can be true at most. Crucially, this form can be further reduced to get something that looks like

$$\bigoplus_{t+1} \left[\bigwedge_{e'} [2 \bigoplus \text{gates}] \right] \quad (3)$$

for some parameter e' that we will still need to tune, as follows: we started out with an expression which was an XOR of $t+1$ ANDs of fan-in $\leq t$ of XORs of fan-in at most 3. We then apply the transformation we would apply to convert an AND gate into polynomial form, but we skip the final step for now. This gives us (3) but with an extra XOR of fan-in $\leq t$ in the middle. We

can rearrange this XOR into an XOR of random bits from each of a and b and a constant. We can precompute the $t(e'+1)$ XORs for each of the nd^2 entries in each of A and B yields the necessary comparisons in $\tilde{O}(nd^2 \cdot (t+2)e'$)time. Now we have the circuit in the form of (3). Now set $e' = 3+2\log d+\log t$ so that the error probability over all the calculations this circuit does is, int total, constant and $< 1/4$. We now apply the technique described above to convert the rest of the circuit into a polynomial, and we end up with a size whose dominant term is $(\log d)^2$. Thus we can set d to $2^{c\sqrt{\log n}}$ for sufficiently small c and fit our polynomial size within $n \cdot 1$.

Now, we apply Coppersmith's rectangular matrix multiplication to evaluate the polynomial on alll n^2 pairs in $n^2 poly(\log n)$ time. Each entry in the resulting table is correct with constant probability, so we can apply a "majority amplification" trick for high probability success, doing all of this logarithmically many times in d^2 and taking the majority for each entry.

Now that we can compute $OR-AND-COMP$ efficiently, we can apply theorem Theorem 3.1 to solve APSP in $O(n^3/2^{c\sqrt{\log n}})$ time.

3.3 Strong Exponential Time Hypothesis

APSP is not the only problem we can apply the math behind Theorem 3.1 to. One plausible avenue of further exploration of these techniques is trying to resolve the open problem of whether APSP is possible on a dense graph in *truly subcubic* time ($O(n^{3-\epsilon})$, for $\epsilon > 0$). If we can satisfy the conditions for theorem 3.1 with $d = n^\epsilon$ for some positive ϵ, then we could apply theorem 3.1 and be done. However, there are other consequences of such a result. In particular, we can show the Strong Exponential Time Hypothesis to be false in such a situation (that is, we could solve SAT in less than 2^n time.

Consider a simpler version of $OR-AND-COMP$, $OR-AND-OR2$. It's the same except the last item is an OR instead of a comparison. Note that any fast solution to $OR-AND-COMP$ is an equally fast solution to $OR-AND-OR2$. We can use fast evaluation of this function to solve things other than $APSP$.

We examine a problem called Orthogonal Vectors (OV). In OV, we have a set of vectors in $\{0,1\}^d$ and we want to tell if there are any pairs which are orthogonal to one another. There is an obvious algorithm which runs in time $O(n^2 d)$ and another which runs in time $O(2^d n)$. However, this begs the question, can we do any better when these algorithms are about as good as one another, namely when $d > \log n$. First, note that this problem is equivalent to the case where the vectors come from different sets A and B.

Theorem 3.2 *Let A and B be sets of vectors with n vectors, where each vector has $d_1 d_2$ bits. If we can compute $OR-AND-OR2$ on all pairs of vectors $(a \in A, b \in B)$ in time $\tilde{O}(n^2)$ then OV can be solved in time $\tilde{O}(n^2/d_1)$.*

Proof.[4] Partition both A and B into subsets of size $\sqrt{d_1}$. We will check d_1 pairs of vectors at a time for overlap by running $OR-AND-OR2$ on subsets we generated - if $OR-AND-OR2(a, 1^n - b) = 1$, then a and b are orthogonal. By our assumption this is doable in $(n/\sqrt{d_1})^2 = n^2/d_1$ time.

Now we show that such a situation would disprove SETH. Given a SAT formula on n variables and m clauses, divide the variables into two sets and enumerate the possible assignments to each half independently in $2^{n/2}$ time.

For each assignment, generate a vector of length m where the jth entry is 1 if and only if the partial assignment does not satisfy the CNF. Put all these vectors into our OV solver. The OV solver runs in $O(m^{2-\epsilon})$ time, or (if we choose m and n appropriately) $o(2^n)$ time, which violates SETH.

4 Open Problems

It is not known whether any method can produce a truly subcubic result for APSP - before Williams 2014 [3] it was conjectured by some that no better than $O(n^3/\log^c n)$. The question of whether these methods can do that is a slightly less interesting one because it requires $SETH$ to be false.

References

[1] Richard Biegel. The polynomial method in circuit complexity. *Structure in Complexity Theory Conference*, 1995.

[2] Ryan Williams. The polynomial method in circuit complexity applied to algorithm design. *Leibniz International Proceedings in Informatics*, pages 1–14, 2014.

[3] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *Symposium on the Theory of Computing*, 2014.

[4] Ryan Williams Amir Abboud and Huacheng Yu. More applications of the polynomial method to algorithm design. *SODA*, 2015.

18.405J / 6.841J Advanced Complexity Theory

Spring 2016