Homework 20

Brian Knotten, Brett Schreiber, Brian Falkenstein

October 14, 2018

3

To prove that if there is an $O(n^k)$ algorithm for squaring an degree n polynomial (call this $Square(C)$), then there is an $O(n^k)$ algorithm for multiplying 2 degree n polynomials (call this $Mult(A, B)$), we must transform the input of $Mult(A, B)$ to the input of $Square(C)$ in polynomial time, and then transform the output of $Square(C)$ to the output of $Mult(A, B)$, also in polynomial time.
Given as input to $Mult(A, B)$ two degree n polynomials A and B, we can construct a polynomial C by doing the following:

- Initialize the solution polynomial P to 0, so $P = 0$.
- Let j be the number of coefficient variable pairs in A and k be the number of pairs in B. Note that since A and B are both degree n, j and k can be no greater than n.
- For $1 <= i <= j$, take the $i'th$ coefficient variable pair from A and add it to B, so $C_i = c_{ai}x_{ai} + B$
- Pass C_i into $Square(C)$.
- From the output of $Square(C_i)$, add the $c_2x_2...c_{k+1}x_{k+1}$terms to P (that is, ignore the first term, and all terms after the $k + 1$'th term), so $P+ = c_2x_2...c_{k+1}x_{k+1}$
- Repeat for all $i's$ up to j

Our code will then simply return P as the solution to $Mult(A, B)$. Note that the code for $Mult(A, B)$ will call $Square(C)$ j times, although since j is a constant less than n, and we are assuming $Square(C)$ has an $O(n^k)$ algorithm, the running time for calling $Square(C)$ will still be polynomial (constant times polynomial). The transformation of A into C_i takes constant time, as we just add a term from A to B. The transformation of the output also takes poly time, as we are simply grabbing k terms from the output of $Square(C)$ and adding them to P, and repeating this a constant number of times.
As an example, consider the two polynomials $A = a^2+b$ and $B = c^2+d$. This algorithm will construct $C_1 = a^2+c^2+d$. The output of $Square(C_1)$ will then be $a^4 + a^2c^2 + a^2d$... (we don't care about the rest of the output). We ignore the first term (a^4) and add the terms a^2c^2 and a^2d to P, so now $P = a^2c^2 + a^2d$. Now, construct $C_2 = b + c^2 + d$. Squaring C_2 gives us $b^2 + bc^2 + bd$.... We then add bc^2 and bd to P, giving us $P = a^2c^2 + a^2d + bc^2 + bd$, the solution to $A \times B$.

5

To prove that if an algorithm for one of Undirected Graph Isomorphism, Directed Graph Isomorphism, and Mixed Graph Isomorphism implies that they all do, the following reductions need to be made:

- Undirected Graph Isomorphism $\leq_p$ Mixed Graph Isomorphism
- Mixed Graph Isomorphism $\leq_p$ Directed Graph Isomorphism
- Directed Graph Isomorphism $\leq_p$ Undirected Graph Isomorphism

Using these three reductions, if any of the problems has a poly-time algorithm, then they all do, since, given one poly time algorithm, each reduction implies the existence of a second poly-time algorithm for any of the problems, and the third poly-time algorithm can be derived from the second poly-time algorithm. Here are the reductions:

Undirected Graph Isomorphism $\leq_p$ Mixed Graph Isomorphism

Assume there exists an algorithm for Mixed Graph Isomorphism called $MISO$. Then it is possible to construct a poly-time algorithm for Undirected Graph Isomorphism $UISO$ as follows:

$UISO(G, H)$:
    return $MISO(G, H)$

A purely undirected graph can be thought of as a special case of a mixed graph, so an algorithm for Mixed Graph Isomorphism works just as well on inputs that only have directed edges. Moreover, since $MISO$ is poly-time, and since $UISO$ makes no transformations from input to input or from output to output, then $UISO$ is also poly-time.

Mixed Graph Isomorphism $\leq_p$ Directed Graph Isomorphism

Assume there exists an algorithm for Directed Graph Isomorphism called $DISO$. Then it is possible to construct a poly-time algorithm for Mixed Graph Isomorphism $MISO$ as follows:

$MISO(G, H)$:
    Let $G' = Directed(G)$
    Let $H' = Directed(H)$
    return $DISO(G', H')$

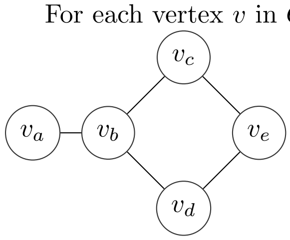
Where $Directed$ is a supplementary function as follows:
$Directed(G)$:
    for each undirected edge e between u and v in G:
        add a directed edge from u to v into G' and
        add a directed edge from v to u in G'
    return G'

An mixed graph can be thought of as a special case of a directed graph where each undirected edge corresponds to two mutual directed edges. Since two directed graphs can only be isomorphic if they both share these pairs of mutual directed edges, the algorithm remains correct after the input has been transformed. Since this transformation is polynomial in the number of edges, and since the output does not need to be transformed, then $UISO$ is also a poly-time algorithm.

Directed Graph Isomorphism $\leq_p$ Undirected Graph Isomorphism

This reduction is the most sophisticated and requires the use of a gadget for transforming a directed graph G into an undirected graph G'.

For each vertex v in G, G' contains the following subgraph:



Any directed edge from u to v in G corresponds to an undirected edge between u_a and v_e in G'. This means that all edges outgoing v in G correspond to a "source" edge of v_a in G' and all edges going into v in G correspond to a "destination" edge of v_e. The subgraph gadget must be assymmetrical in order to distinguish these source and destination vertices.

So if there exists a poly-time algorithm for Undirected Graph Isomorphism $UISO$, then there also exists a poly-time algorithm for Directed Graph Isomorphism $DISO$ as follows:

$DISO(G, H)$:
    Let G' = undirected transformation of G as described above

Let H' = undirected transformation of H as described above
return $UISO(G', H')$

Since a directed edge corresponds to an edge from a source vertex to a destination vertex, then the transformed graphs are isomorphic iff the directed graphs are isomorphice. Moreover, the transformation requires a constant factor (5) vertices for a single vertex in the directed vertex, so the transformation of the inputs takes poly-time, and therefore $DISO$ is poly-time.