CU-CSD-71-118

A FAST METHOD FOR INTERPOLATION USING PRECONDITIONING

by

Ellis Horowitz

Technical Report
No. 71-118

Department of Computer Science
Cornell University
Ithaca, New York 14850

A FAST METHOD FOR INTERPOLATION USING PRECONDITIONING

by

Ellis Horowitz

Department of Computer Science
Cornell University
Ithaca, New York 14850

__Abstract__:

Given n points (x_i, y_i) the best algorithms for finding the unique interpolating polynomial $G(x)$ such that $G(x_i)=y_i$ take $O(n^2)$ arithmetic operations. If the x_i are known in advance then an algorithm for finding $G(x)$ is presented which takes only $O(n(\log n)^3)$ steps. Also, it is shown how to pre-compute certain functions of the x_i, in $O(n^2)$ steps, such that this restricted interpolation algorithm can be easily used. Finally, it is shown that speeding up the general interpolation problem is possible if one can solve a simpler problem, namely to find a polynomial $G(x)$ such that $G(x_i)=0$ for $1 \leq i \leq j$ and $G(x_i)=1$ for $j+1 \leq i \leq n$.

A FAST METHOD FOR INTERPOLATION USING PRECONDITIONING

by

Ellis Horowitz

Given n points (x_i,y_i), $1 \leq i \leq n$ we wish to determine the coefficients $a_{n-1}, \ldots, a_0$ of the unique polynomial $G(x)$ of degree $\leq n-1$ such that $G(x_i)=y_i$ for $1 \leq i \leq n$. This is the general problem of interpolation and the best existing algorithms take $O(n^2)$ arithmetic operations. A restricted form of the interpolation problem can be stated as follows: assuming that n values x_i, $1 \leq i \leq n$ have been previously given, find the unique interpolating polynomial $G(x)$ of degree $\leq n-1$ such that $G(x_i)=y_i$, $1 \leq i \leq n$. Of course this problem does not include interpolating arbitrary sets of data. However, this restricted form often does occur when first evaluation and later interpolation are used jointly. Moreover, there is an immediate analogy between being able to precondition a polynomial's coefficients before evaluation and preconditioning the x_i before interpolation. Whereas preconditioning for evaluation improves the method by only a constant factor, it will be shown that preconditioning for interpolation will produce an order of magnitude decrease in the computing time.

To give a specific example of restricted interpolation, suppose we wish to solve a linear system with polynomial entries. The numerical analysis approach would evaluate the entries at many points producing systems with rational number elements. Each of these systems is solved and then all of these answers are interpolated to produce the final result. Alternatively one may use symbolic manipulation to work with the polynomials themselves, but recent work on symbolic algorithms of this type show that the most efficient methods will first evaluate and then later interpolate. Since the points of evaluation and hence for subsequent interpolation are arbitrary these points can be chosen in advance, thus producing the restricted interpolation problem. For example, Collins in [COL71] and Brown in [BRO71] choose $x_i = i$.

In this paper an algorithm for the restricted interpolation problem will be given which takes $O(n(\log n)^3)$ steps. Also it will be shown how to pre-condition the values x_i in order to use this algorithm. The time necessary to compute the needed functions of the x_i, will be $O(n^2)$. Finally, it will be shown how to reduce the general interpolation problem to a "simpler" problem such that if the simpler problem can be solved in less than $O(n^2)$ operations so may the general interpolation problem.

Let us begin with Lagranges formula for determining $G(x)$ given (x_i, y_i) $1 \le i \le n$. Let $D(x) = \prod_{1 \le i \le n} (x - x_i)$ and $L_i(x) = D(x)/(x - x_i)$. Then,

$$G(x) = \sum_{1 \le i \le n} \frac{L_i(x)}{L_i(x_i)} \cdot y_i \ .$$

It is easy to verify that $G(x_i) = y_i$ since $L_j(x_i) = 0$ for $j \ne i$ and $(L_i(x_i)/L_i(x_i))y_i = y_i$. Also, degree $(L_i) = n-1$ and hence degree (G) $\le n-1$. Now, if we wish to compute $G(x)$ using Lagrange's formula we have the following:

Steps	Times
1) $D(x)$	$O(n^2)$
2) $L_i(x)$ $1 \le i \le n$	$O(n^2)$
3) $G(x)$	$O(n^2)$

A second method commonly used for interpolation, see [COL71], is an iterative scheme. Letting $D_i(x)=(x-x_1)\cdots(x-x_i)$ and assumin we have already computed $G_{i-1}(x)$ such that $G_{i-1}(x_j)=y_j$ $1\le j\le i-1$ we compute $G_i(x)$ as follows:

$$G_i(x) = \frac{y_i-G_{i-1}(x_i)}{D_{i-1}(x_i)} \cdot D_{i-1}(x) + G_{i-1}(x)$$

Note that $G_i(x_i) = y_i-G_{i-1}(x_i)+G_{i-1}(x_i) = y_i$ and $G_i(x_j) = G_{i-1}(x_j)$ for $1\le j\le i-1$ since $D_{i-1}(x_j)=0$. The computing time is $O(i)$ for the iteration since the evaluation of G_{i-1} and D_{i-1} at x_i both take O(steps as well as the multiplication of a constant times $D_{i-1}(x)$. Thus the total time for this algorithm is also $O(n^2)$.

Finally, consider Newton's interpolation formula,

$$G(x) = a_0+a_1(x-x_1)+a_2(x-x_1)(x-x_2)+\cdots+a_{n-1}(x-x_1)\cdots(x-x_{n-1})$$

where the a_j's are determined by computing the divided differences of (x_i,y_i) $1\le i\le n$, see [KNU71] pp. 429. For n=4 we would have

$$\begin{array}{llll}
x_1,y_1 & & & \\
 & \dfrac{y_2-y_1}{x_2-x_1} = y_1' & & \\
x_2,y_2 & & \dfrac{y_2'-y_1'}{x_3-x_1} = y_1'' & \\
 & \dfrac{y_3-y_2}{x_3-x_2} = y_2' & & \dfrac{y_2''-y_1''}{x_4-x_1} = y_1''' \\
x_3,y_3 & & \dfrac{y_3'-y_2'}{x_4-x_2} = y_2'' & \\
 & \dfrac{y_4-y_3}{x_4-x_3} = y_3' & & \\
x_4,y_4 & & &
\end{array}$$

Then $a_0=y_1$, $a_1=y_1'$, $a_2=y_1''$ and $a_3=y'''$. The total number of divided differences to be computed is

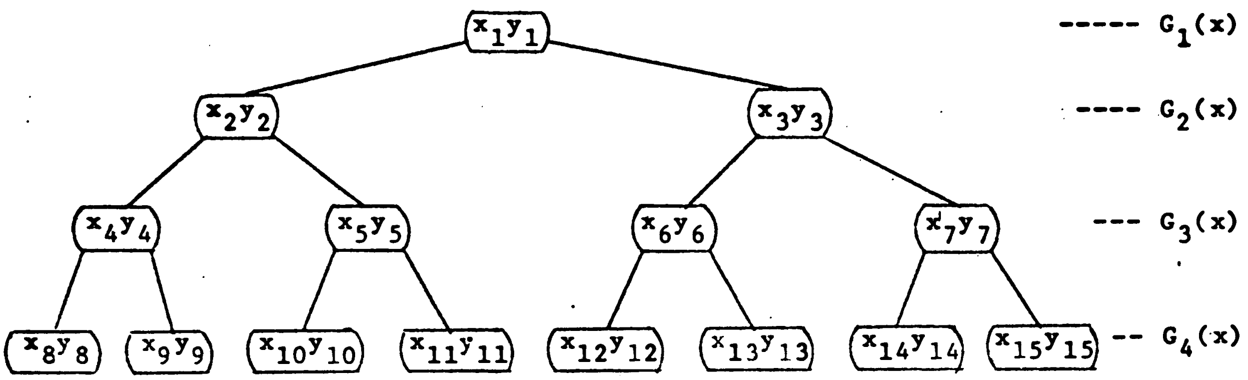
$$(n-1)+(n-2)+\cdots+2+1 = \frac{n(n-1)}{2} = O(n^2)$$

implying $O(n^2)$ arithmetic operations are needed to compute the a_i, $0 \leq i \leq n-1$.

Now, we ask can the computing times for these methods be reduced if we assume that we know the x_i in advance and that we can precompute any values based on these x_i's. Determining the coefficients in Newton's interpolation formula clearly relies on the y_i's and requires $O(n^2)$ steps even if $x_i - x_j$ are precomputed for all i,j $1 \leq i,j \leq n$. In the iterative method we could precompute $D_i(x)$ for $1 \leq i \leq n$ as well as $D_{i-1}(x_i)$ for $1 \leq i \leq n-1$. However, the time needed to compute $G_{i-1}(x_i)$ and the multiplication of the constant times $D_{i-1}(x)$ still takes $O(i)$ steps and hence no improvement can be effected in the overall computing time for this method. Finally, returning to Lagrange's formula we may well precompute $L_i(x)/L_i(x_i)$ for all i, $1 \leq i \leq n$. Nevertheless, the last stage requires n multiplications of y_i by $L_i(x)/L_i(x_i)$ for $1 \leq i \leq n$. Since degree $(L_i(x)) = n-1$ it follows that the time for Lagrange's method is also not reduced.

An algorithm, INTERP, is now presented which computes the interpolating polynomial in less than $O(N^2)$ steps where we assume $N = 2^n$ values x_i, $1 \leq i \leq N$ are given a priori.

By way of explanation of this algorithm, consider the 2^n points (x_i, y_i) displayed in a binary tree, e.g. if $n=4$



Then, for each of the n levels in this tree, INTERP computes $G_i(x)$ which interpolates the points on the i^{th} level, such that $G_i(x_j)=y_j$ for $2^{i-1} \leq j \leq 2^i-1$. Then, to find G(x) INTERP uses n precomputed polynomials $A_i(x)$ such that

$$G(x) = A_1G_1+\ldots+A_nG_n.$$

Hence, the criteria for the $A_i(x)$ is precisely

$$A_i(x_j) = \begin{cases} 0, & 1 \leq j \leq 2^{i-1}-1,\ 2^i \leq j \leq 2^n \\ 1, & 2^{i-1} \leq j \leq 2^i-1 \end{cases}$$

and these interpolating polynomials are independent of the y_i. Thus they may be precomputed and this is precisely what is done by Algorithm PRE-COMPUTE.

<u>Algorithm INTERP</u> $((x_j,y_j),\ 1 \leq j \leq N)$

Input: $N=2^n$ points (x_j,y_j) $1 \leq j \leq N$ plus the precomputed polynomials $A_1(x), \ldots, A_n(x)$ where

$$A_i(x_j) = \begin{cases} 0 & \text{otherwise} \\ 1 & \text{if } 2^{i-1} \leq j \leq 2^i-1 \end{cases}$$

Output: G(x) such that $G(x_i)=y_i$ $1 \leq i \leq N$ and $\deg(G) \leq N-1$.

1) [Initial Case] If N=1, INTERP←y_1; end;

If N=2, INTERP←$\dfrac{(y_3-y_2)(x-x_2)}{x_3-x_2} + y_2$; end;

2) [Recursion] $n \leftarrow \log_2 N$; $G \leftarrow 0$;

For $i \leftarrow 1, \ldots, n$ do

2.1) $G_i(x) \leftarrow$ INTERP$((x_jy_j)\ 2^{i-1} \leq j \leq 2^i)$;

2.2) $G(x) \leftarrow G(x)+A_i(x) \cdot G_i(x)$;

3) [end:] Return(G(x)); end;

Let us now derive the computing time for Algorithm INTERP. Let $I(2^n)$ be the time to find the polynomial, $G(x)$ which interpolates 2^n points. Then, Step (1) takes $O(1)$, Step (2.1) takes $I(2)+I(2^2)+\cdots+I(2^{n-1})$. Now degree $(A_i)=2^n$ since A_i is defined at 2^n points and degree$(G_i)=2^{i-1}$. Thus, the time to compute the polynomial product using Fourier transforms is $O(n2^n)$, see [KNU71], pp. 441, and since the step is executed n times the total time for Step 2.2 is $O(n^2 2^n)$. Then, the total time for Algorithm INTERP is

$$
\begin{aligned}
I(2^n) &= n^2 2^n + I(2^{n-1}) + \cdots + I(2) = n^2 2^n + (n-1)^2 2^{n-1} + 2(I(2^{n-2}) + \cdots + I(\\
&= n^2 2^n + (n-1)^2 2^{n-1} + 2(n-2)^2 2^{n-2} + 3(I(2^{n-3}) + \cdots + (I(2)) \\
&= \sum_{1 \leq i \leq n-1} \left\{ i^2 2^i (n-i) \right\} + n^2 2^n \\
&< n^3 (2^n - 1) + n^2 2^n = O(n^3 2^n).
\end{aligned}
$$

Or, letting $N=2^n$ the total time for Algorithm INTERP(N) is $O(N(\log N)^3)$.

Now an algorithm is given which determines all the necessary precomputed polynomials.

Algorithm PRE-COMPUTE

Input: $N-1=2^n-1$ values $x_1, \ldots, x_{N-1}$.

Output: $A_1(x), \ldots, A_n(x)$ such that $A_i(x_j) = \begin{cases} 0 & \text{otherwise} \\ 1 & 2^{i-1} \leq j \leq 2^i - 1 \end{cases}$.

1) For $i \leftarrow 1, \ldots, n$ do $D_i(x) \leftarrow (x - x_{2^{i-1}}) \cdots (x - x_{2^i - 1})$;

2) $D(x) \leftarrow \prod_{1<i<n} D_i(x)$

3) For $i \leftarrow 1, \ldots, n$ do SUM$\leftarrow 0$;

   3.1) For $j \leftarrow 2^{i-1}, \ldots, 2^i - 1$ do

      3.1.1) $E_j(x) \leftarrow D(x)/(x-x_j)$;

      3.1.2) Form $E_j(x_j)$;

      3.1.3) $D_{i_j} \leftarrow D_i(x)/(x-x_j)$;

      3.1.4) SUM$\leftarrow$SUM$+D_{i_j}/E_j(x_j)$;

   3.2) $A_i(x) \leftarrow (D/D_i) * \mathrm{SUM}$;

4) end;

Then to correspond to each recursive call of INTERP we need to call PRE-COMPUTE n times with the values $x_{2^{i-1}}, \ldots, x_{2^i-1}$ for $i=1, \ldots, n$.

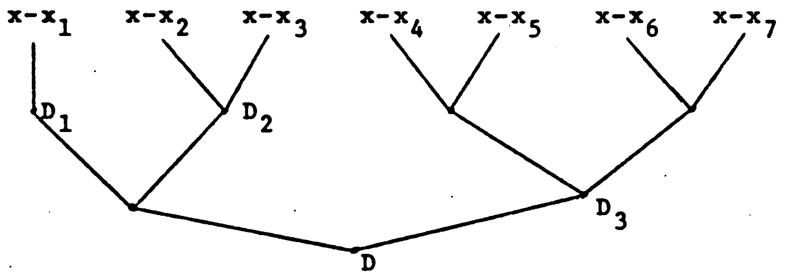
What is the computing time for Algorithm PRE-COMPUTE. Let $P(2^n)$ be its computing time. Then,

Step	Computing Time
(1)	$\sum_{1 \le i \le n-1} \left(2^n/2^i\right)(2^{i-1})(i-1) = O(2^{n-1}n^2)$ Using fast Fourier transforms for polyno multiplication and a binary tree structure the "multiplications".
(2)	$\sum_{1 \le i \le n} 2^n \cdot n = O(n^2 2^n)$. (same reasons as in Step (1)).
* (3.1.1)	$2^n \cdot (1+2+\cdots+2^{n-1}) = 2^{2n}$
* (3.1.2)	$2^n \cdot 2^n = 2^{2n}$
* (3.1.3)	$2^n \cdot 2^n = 2^{2n}$
* (3.1.4)	$2^n(1+\cdots+2^{n-1}) = 2^{2n}$
(3.2)	$n \cdot \sum 2^{n-i}(n-i) < n \sum_{1 \le i \le n} 2^n \cdot n = n^3 2^n$

Thus, the total time for Algorithm PRE-COMPUTE is $O(2^{2n}) = O(N^2)$.

What conclusions can be drawn about the general interpolation problem. If one could compute $A_1(x)$, ..., $A_n(x)$ in less than $O(N^2)$ steps, then Algorithm INTERP could be used to produce a better than $O(N^2)$ interpolation method. The $A_i(x)$ are interpolating polynomials which are independent of the y_i and thus the general interpolation problem has in some sense been simplified.

Therefore, let us examine more closely Algorithm PRECOMPUTE. By way of further explanation of steps (1) and (2) suppose that $N=2^3$ and that we wish to compute $D_1(x)$, $D_2(x)$, $D_3(x)$ and $D(x)$. This would be done as shown below.



Thus we see that by using this binary tree structure to guide the multiplications plus a fast Fourier transform algorithm for polynomial multiplication we can produce, in general, $D_1(x)$, ..., $D_n(x)$ and $D(x)$ in $O(N(\log N)^2)$ operations. Remembering that $D(x)=(x-x_1)\cdots(x-x_N)$, the above gives a fast method for finding the values of the elementary symmetric functions of x_1, ..., x_N: i.e. the functions

$$\sum x_i, \quad \sum x_i x_j, \quad \ldots, \quad \sum x_{i_1} \cdots x_{i_N}$$

since these values are merely the coefficients of $D(x)$.

Then, the steps which really bound the computing time of PRE-COMPUTE are the computation of the 2^n polynomials $E_i(x)=D(x)/(x-x_i)$ and the 2^n values $E_i(x_i)$. These steps can be simplified by observing that the values $E_i(x_i)$ can be obtained by evaluating a single polynomial, the derivative of D(x), at x_i, $1\leqslant i\leqslant N$. This follows by considering the Taylor series expansion of D(x) about the point x_i. Then,

$$D(x) = D(x_i) + \sum_{j\geqslant 1} \frac{D^j(x_i)(x-x_i)^j}{j!}$$

where $D^j(x_i)$ is the j^{th} derivative of D(x) evaluated at x_i. Now $D(x_i)=0$ and dividing by $(x-x_i)$ and evaluating at x_i we get

$$E_i(x_i) = D(x)/(x-x_i)\Big|_{x_i} = D'(x_i)$$

Thus instead of computing 2^n polynomials and then evaluating each one at a single point we can simply evaluate $D'(x)$ at x_i, $1\leqslant i\leqslant 2^n$. Now, in [BOR71], pp. 67 Borodin and Munro show that any iterative scheme which evaluates an N^{th} degree polynomial at N distinct points must take $O(N^2)$ arithmetic operations. Therefore, any scheme for speeding up the calculation of the $A_i(x)$ will likely have to make use of a radically different computational procedure than iteration.

References:

[BOR71] Borodin, A. and I. Munro, Evaluating Polynomials at Many Points, Information Processing Letters 1 (1971) North-Holland Publishing Company, pp. 66-68.

[BRO71] Brown, W.S., On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, J.ACM., Vol. 18, No. 4, October 1971, pp. 478-504.

[COL71] Collins, G.E., The Calculation of Multivariate Polynomial Resultants, J.ACM., October 1971, Vol. 18, No. 4, pp. 515-532.

[KNU69] Knuth, D.E., The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Addison-Wesley, 1969.