Finding Longest Arithmetic Progressions

Jeff Erickson*

Abstract

We describe efficient output-sensitive algorithms to find the longest arithmetic progression in a given set of numbers.

1 Introduction

This paper describes efficient algorithms for finding the longest arithmetic progression in a set of n integers. That is, given an array $A[1..n]$ of integers, we wish to find the largest sequence of indices $\langle i_0, i_1, \ldots, i_{k-1}\rangle$ such that $A[i_j] - A[i_{j-1}] = A[i_1] - A[i_0]$ for all j. Note that the indices themselves need not form an arithmetic progression. There is an $\Omega(n \log n)$ lower bound on the complexity of this problem in the algebraic decision tree model of computation [1], so without loss of generality, we assume that the input array A is sorted and free of duplicate elements.

2 Dynamic Programming

An interesting subproblem, which we call AVERAGE, is to determine whether the input contains a three-term arithmetic progression, or equivalently, if any array element is the average of two others. AVERAGE can be solved by the following simple $O(n^2)$-time algorithm. This is the fastest algorithm known. There is a matching $\Omega(n^2)$ lower bound in the *3-linear decision tree* model, in which every decision depends on the sign of an affine combination of three or fewer input elements [4], so at least in that model, this algorithm is optimal.

```
AVERAGE(A[1..n]):
for j ← 2 to n − 1
    i ← j − 1
    k ← j + 1
    while (i ≥ 1 and k ≤ n)
        if A[i] + A[k] < 2A[j]
            k ← k + 1
        else if A[i] + A[k] > 2A[j]
            i ← i − 1
        else
            return TRUE
return FALSE
```

AVERAGE is closely related to the class of *3SUM-hard* problems defined by Gajentaan and Overmars [5]. A problem is 3SUM-hard if there is a sub-quadratic reduction from the problem 3SUM: Given a set A of n integers, are there elements $a, b, c \in A$ such that $a + b + c = 0$? It is not known whether AVERAGE is 3SUM-hard. However, there is a simple linear-time reduction from AVERAGE to 3SUM, whose description we omit. (Thus, 3SUM-hard problems might better be called "AVERAGE-hard".)

The longest arithmetic progression can be found in $O(n^2)$ time using a dynamic programming algorithm similar to our algorithm for AVERAGE. The algorithm shown below computes only the length of the longest arithmetic progression; computing the actual progression requires only a few extra lines. When our algorithm terminates, $L[i, j]$ stores the maximum length of an arithmetic progression whose first two terms are respectively $A[i]$ and $A[j]$. Our algorithm can be described by a family of 3-linear decision trees, and the $\Omega(n^2)$ lower bound for AVERAGE [4] implies that it is optimal in that model of computation.

```
LONGESTARITHPROG(A[1..n]):
L* ← 2
for j ← n − 1 downto 1
    i ← j − 1;  k ← j + 1
    while (i ≥ 1 and k ≤ n)
        if A[i] + A[k] < 2A[j]
            k ← k + 1
        else if A[i] + A[k] > 2A[j]
            L[i, j] ← 2
            i ← i − 1
        else
            L[i, j] ← L[j, k] + 1
            L* ← max {L*, L[i, j]}
            i ← i − 1;  k ← k + 1
    while i ≥ 1
        L[i, j] ← 2;  i ← i − 1
return L*
```

Theorem 1. *The longest arithmetic progression in an n-element set can be found in time $O(n^2)$, which is optimal in the 3-linear decision tree model of computation.*

3 Output-Sensitive Divide and Conquer

Another problem related to finding largest arithmetic sequences is finding an element of a multiset with largest multiplicity. This problem can be solved in $O(n \log n)$ time by sorting and scanning the multiset. Since determining whether the maximum multiplicity is at least two (the *element uniqueness* problem) requires $\Omega(n \log n)$ time in the algebraic compu-

*Computer Science Dept., University of Illinois, Urbana-Champaign; jeffe@uiuc.edu; http://www.uiuc.edu/~jeffe

tation tree model [1], this algorithm is worst-case optimal. However, we can "beat" the lower bound if the maximum multiplicity m is large. A simple divide-and-conquer algorithm computes m in $O(n \log(n/m))$ time [7]; a matching lower bound was proved by Björner and Lovász [2].

Another similar problem is the *exact fitting problem* considered by Guibas, Overmars, and Robert [6]: Given a set of points in $\mathbb{R}^d$, find the largest subset that lies on a common hyperplane. This problem can be solved in $O(n^d)$ time by constructing the dual hyperplane arrangement, and known lower bounds suggest that this approach is optimal in the worst case. However, Guibas *et al.* describe a complex divide-and-conquer algorithm that runs in time $O((n^d/k^{d-1}) \log(n/k))$, which is significantly faster when the output size k is large.

We can make a similar improvement to our worst-case-optimal algorithm LongestArithProg by exploiting the following simple lemma.

Lemma 2. *Any set of n numbers contains $O(n^2/k^2)$ maximal arithmetic progressions of length k or more.*

Proof: Say that two elements are *close* if their ranks (positions in sorted order) differ by less than $n/2(k-1)$. There are less than $n^2/2(k-1)$ close pairs. Any progression of length k or more must include at least $(k-1)/2$ close consecutive pairs. Any two elements are consecutive in exactly one maximal progression. □

Tight bounds on the maximum number of maximal k-term arithmetic progressions are not known, even in very simple cases. Lemma 2 appears to be the best upper bound known. Any improvement would have to exploit maximality in some essential way, since the sequence $\langle 1, 2, \ldots, n \rangle$ contains roughly $n^2/2k^2$ (non-maximal!) k-term progressions. On the other hand, the best published lower bound is only $\Omega(n^{\log_k(k+2)})$, which Erdős and Simmons prove by considering a generic projection of a regular $k \times k \times \cdots \times k$ lattice [3]. In the simplest nontrivial case $k = 3$, Simmons and Abbott improve the lower bound to $\Omega(n^{\log_{11} 49}) \approx \Omega(n^{1.623})$ [8]. These lower bounds are almost certainly *not* tight—better bounds would follow immediately from better small examples by an easy product construction.

The following divide-and-conquer algorithm finds *all* maximal progressions of length at least k. To simplify both the presentation and analysis, we assume without loss of generality that both n and k are powers of two.

```
AllLongProgs(k, A[1 .. n]):
if k ≤ lg n lg lg n
    use dynamic programming
else
    P♭ ← AllLongProgs(k/2, A[1 .. n/2])
    P♯ ← AllLongProgs(k/2, A[n/2 + 1 .. n])
    return Extend(P♯, k, A) ∪ Extend(P♭, k, A)
```

The subroutine Extend takes a set P of progressions, a target length k, and an array A, and attempts to extend each progression across the array as far as possible. Each progression is stored as a triple (x, Δ, ℓ), where x is its smallest term, Δ is its step size, and ℓ is its number of terms. If a progression cannot be extended to the target length k, it is discarded; the successfully extended progressions are returned.

```
Extend(P, k, A[1 .. n]):
for each progression (x, Δ, ℓ) ∈ P
    search in A for all terms of (x − kΔ, Δ, 2k)
    ℓ′ ← maximum number of consecutive terms found
    if ℓ′ ≥ k
        x′ ← first of ℓ′ consecutive terms found
        add (x′, π, ℓ′) to the output
```

The correctness of these algorithm is fairly obvious, since any k-term progression in A must contain a $k/2$-term progression in one of the two halves of A. Searching for the terms of $(x - k\Delta, \Delta, 2k)$ in A takes $O(k \log(n/k))$ time, so the total running time for Extend is $O(pk \log(n/k))$, where p is the number of progressions in P. By Lemma 2, $p = O(n^2/k^2)$ whenever we call Extend, so the running time is $O((n^2/k) \log(n/k))$.

Finally, the running time for AllLongProgs satisfies the recurrence

$$T(n, k) \leq 2T(n/2, k/2) + O((n^2/k) \log(n/k)),$$

whose solution is $T(n, k) = O((n^2/k) \log(n/k) \log k)$. This is faster than the quadratic dynamic programming algorithm whenever $k > \lg n \lg \lg n$.

Theorem 3. *We can determine whether an n-element set contains a arithmetic progression with k or more terms in $O((n^2/k) \log(n/k) \log k)$ time.*

Finally, we can find the longest arithmetic progression using a standard doubling trick, also used in [6]. We call AllLongProgs several times, halting as soon as it returns at least one progression. In the ith iteration, we look for progressions with at least $n/2^i$ terms. Omitting further details, we conclude:

Theorem 4. *The longest arithmetic progression in an n-element set can be found in time $O(\min\{n^2, (n^2/k) \log(n/k) \log k\})$, where k is the output size.*

References

[1] M. Ben-Or. Lower bounds for algebraic computation trees. *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pp. 80–86. 1983.

[2] A. Björner and L. Lovász. Linear decision trees, subspace arrangements, and möbius functions. *J. Amer. Math. Soc.* 7(3):677–706, 1994.

[3] P. Erdős. Problems and results on combinatorial nuymber theory. *A Survey of Combinatorial Theory*, pp. 117–138. North-Holland, 1973.

[4] J. Erickson. Lower bounds for linear satisfiability problems. *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pp. 388–395. 1995. ⟨http://www.uiuc.edu/~jeffe/pubs/linsat.html⟩. To appear in *Chicago J. Theoret. Comut. Sci.*

[5] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.* 5:165–185, 1995.

[6] L. J. Guibas, M. H. Overmars, and J.-M. Robert. The exact fitting problem in higher dimensions. *Comput. Geom. Theory Appl.* 6:215–230, 1996.

[7] J. Misra and D. Gries. Finding repeated elements. *Sci. Comput. Prog.* 2(2):143–152, 1982.

[8] G. J. Simmons and H. L. Abbott. How many 3-term arithmetic progressions can there be if there are no longer ones? *Amer. Math. Monthly* 84(8):633–635, 1977.