Finding Anagrams Via Lattice Reduction

Alexander D. Healy
ahealy@fas.harvard.edu

Abstract

This paper describes a technique for finding a certain types of anagrams, which we call *subset anagrams*. Specifically, given a list of words or phrases, we look for two (disjoint) subsets of these words or phrases that are anagrams of each other. The approach presented here makes use of powerful *lattice reduction* algorithms, and is due to Noam Elkies [Elk] (see also [CS03]), who used it to produce an award-winning anagram [Ana02]. We also describe some implementation details of the script at http://www.alexhealy.net/anagram.html that uses this technique to find subset anagrams.

Introduction

On Sunday May 5, 2002, the following puzzle was posed as part of the National Public Radio (NPR) show "Weekend Edition": find two countries, such that the letters in their names can be rearranged to spell two other countries; for example: MALI + QATAR = IRAQ + MALTA. It turns out that there are exactly three other solutions:

ALGERIA + SUDAN = ISRAEL + UGANDA
BELARUS + INDIA = LIBERIA + SUDAN
GABON + ITALY = LIBYA + TONGA

In this case, such anagrams can easily be found by a computer. Indeed, there are (at present) 192 countries in the world. So there are $\binom{192}{2} = 18336$ pairs of countries, and from each pair it is easy to compute the vector $(n_A, n_B, \ldots, n_Z)$, where n_A is equal to the number of occurrences of the letter 'A' in the given pair of countries. Finally, we can sort the list of pairs of countries by ordering these vectors and then search for duplicates. Any two pairs of countries that have the same vector $(n_A, n_B, \ldots, n_Z)$ are clearly anagrams of each other. Such sorting and searching operations are routinely performed on millions of records, so performing these task on a list of size 18336 is trivial for a modern computer. In general, if we use fast sorting algorithms that take time $O(m \log m)$ to sort a list of m objects, then we have a list of $m = O(n^2)$ pairs of "countries", which can be sorted in time $O(n^2 \log(n^2)) = O(n^2 \log n)$. However, this approach quickly becomes impractical when we are looking for a large set of countries that anagram to a different (large) set of countries, apart from just recombining smaller anagrams to form a larger one. The next section discusses an approach for finding subset anagrams when this is the case.

Finding Subset Anagrams

Let $f : \{A, B, \ldots, Z\} \to \mathbb{Z}^{26}$ be the map that takes a word or phrase, w, and computes $f(w) = (n_A, \ldots, n_Z)$, where n_A is the number of occurrences of the letter A, etc. For example

$$f(\text{“ANAGRAM”}) = (3, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)$$

From this definition, it is clear that $f(w_1 + w_2) = f(w_1) + f(w_2)$, where $w_1 + w_2$ represents the phrase that is obtained by concatenating w_1 and w_2.

Now suppose that we are given n words or phrases, $w_1, \dots, w_n$. A subset anagram of these words is simply a pair of sets of indices, $(\{i_1, i_2, \dots, i_s\}, \{j_1, j_2, \dots, j_t\})$ where $i_k, j_k \in \{1, \dots, n\}$ and all i_k and j_k are distinct, such that

$$f(w_{i_1} + \cdots + w_{i_s}) = f(w_{j_1} + \cdots + w_{j_t})$$

or equivalently

$$f(w_{i_1}) + \cdots + f(w_{i_s}) = f(w_{j_1}) + \cdots + f(w_{j_t})$$

For convenience of notation, we write $v_i = f(w_i)$, and so this last expression can be rewritten as

$$v_{i_1} + \cdots + v_{i_s} = v_{j_1} + \cdots + v_{j_t}$$

or more suggestively, as

$$v_{i_1} + \cdots + v_{i_s} - v_{j_1} - \cdots - v_{j_t} = \vec{0}$$

Thus, if we let u be the n-dimensional vector defined by

$$u_k = \begin{cases} 1 & k \in \{i_1, \dots, i_s\} \\ -1 & k \in \{j_1, \dots, j_t\} \\ 0 & \text{otherwise} \end{cases}$$

then we have that u is in the kernel of the $26 \times n$ matrix

$$V = \left(\begin{bmatrix} | \\ v_1 \\ | \end{bmatrix} \quad \begin{bmatrix} | \\ v_2 \\ | \end{bmatrix} \quad \cdots \quad \begin{bmatrix} | \\ v_n \\ | \end{bmatrix} \right)$$

whose columns are the vectors v_i, i.e. $Vu = \vec{0}$. Conversely, any vector $u' \in \{1, 0, -1\}^n$ that is in $\ker(V)$ defines a subset anagram $(\{i_1, i_2, \dots, i_s\}, \{j_1, j_2, \dots, j_t\})$ where i_k are the indices of 1's in u', and j_k are the indices of -1's in u'. Thus, the problem of finding subset anagrams is equivalent to finding $\{1, 0, -1\}$-vectors in $\ker(V)$.

Let us now consider the structure of the $ker(V)$. Since we are only interested in integer vectors in the kernel (particularly those with entries that are 1, 0 or -1), we may restrict our attention to the so-called integer kernel of V, i.e. $\ker(V) \cap \mathbb{Z}^n$. It is not hard to see that $\ker(V) \cap \mathbb{Z}^n$ is a discrete subgroup of $\mathbb{Z}^n$ i.e. an *integer lattice*, and efficient algorithms exist [Coh93] for computing a (lattice) basis for the integer kernel of an integer matrix such as V. That is, there is an efficient algorithm that, given V, computes a set of linearly independent vectors $B = \{b_1, \dots, b_m\} \in \mathbb{Z}^n$ such that $\ker(V) \cap \mathbb{Z}^n = \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_m$.

Now we are left with the task of finding $\{1, 0, -1\}$-vectors in the lattice generated by B, denoted $\mathcal{L}(B)$ (i.e. $\mathcal{L}(B) = \ker(V) \cap \mathbb{Z}^n$). Intuitively, a $\{1, 0, -1\}$-vector u in this lattice, is a relatively short vector. In fact, the $\{1, 0, -1\}$-vectors in $\mathcal{L}(B)$ are exactly those vectors, u, whose ℓ_∞ norm, $\|u\|_\infty = \max_i\{|u_i|\}$, is at most 1. Also, all such vectors have ℓ_2-norm at most $\sqrt{n}$, although the converse is not true in general: there may be other lattice vectors of ℓ_2-norm at most $\sqrt{n}$ that are not $\{1, 0, -1\}$-vectors. Nonetheless, we will search for lattices vectors that are short with respect to the ℓ_2-norm, in the hopes that they will also be short for the ℓ_∞-norm, simply because the best known algorithms for finding short lattice vectors are designed for the ℓ_2-norm (by virtue of the natural inner-product that accompanies it). The problem of lattice basis reduction is that of computing a basis for a given lattice that (in a precise sense) consists of short, nearly-orthogonal, vectors. There are a variety of definitions of reduced and approximately-reduced lattice bases and many of them are $\mathcal{NP}$-hard to compute for arbitrary lattices [MG02]. Nonetheless, there exist efficient (polynomial-time) approximation algorithms ([LLL82], [Coh93], [MG02]) that have theoretical guarantees on the quality of the reduced basis that they return, and that often work extremely well (i.e., much better than the theoretical guarantees) in practice. Thus, it seems appropriate to use such algorithms to find short vectors in the lattice $\mathcal{L}(B)$, and this is the approach taken here.

The subset anagram finder at `http://www.alexhealy.net/anagram.html` consists of a CGI script written in Python that calls a C++ program that applies the above algorithm to the given words or phrases. Victor Shoup's NTL library [Sho] is used for the lattice routines. In particular, the function `image()` is used to compute a basis for the kernel of the matrix V (the function is called "`image()`" because it computes a basis for the image and a basis for the kernel simultaneously), and the function `BKZ_FP()` is used to apply "Block Korkin-Zolotarev" reduction to the basis for $\ker(V) \cap \mathbb{Z}^n$. The block size is originally set to be 5, and then is incremented by 5 after each subsequent call to `BKZ_FP()`, until the block size is 100 or until the script has run for more than 2 seconds. (A larger block size results in a more reduced basis, but consequently causes the lattice reduction routine to run more slowly.) This is to ensure that the CGI script returns promptly and does not use too much CPU time on the server. Finally, any basis vectors that belong to $\{1, 0, -1\}^n \subset \mathbb{Z}^n$ are returned. Therefore, the subset anagrams that are returned form a basis for a sublattice of $\ker(V) \cap \mathbb{Z}^n$, and hence are "linearly independent".

Generalizations and Variations

The above algorithm can clearly be extended to larger alphabets to include numbers and symbols for example, or to be case-sensitive. (The current implementation is not case-sensitive.) It is also interesting to note that if there are numbers that are naturally associated with each word or phrase (such as atomic weights for elements, or dates of birth for authors, etc.), then these values can be placed in a special additional coordinate of the transformed vectors v_i, and the resulting anagrams (if any are found) will not only be subset anagrams, but they will have the property that the sum of the numbers on one side of the anagram is equal to the sum of the numbers on the other side. These are just a few of many possible variants and generalizations of this technique.

References

[Ana02] Anagrammy. *Anagrammy - Winners 2002.* `http://www.anagrammy.com/winners/history2002.html`, 2002.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory.* Springer, 1993.

[CS03] Barry Cipra and Charles Seife. Meeting: Joint mathematics meetings. *Science*, 299:650–651, 2003.

[Elk] Noam D. Elkies. Personal communication, May 2002.

[LLL82] Arjen K. Lenstra, H. W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective.* Kluwer, 2002.

[Sho] Victor Shoup. *NTL: A Library for doing Number Theory.* `http://shoup.net/ntl/`.