Blankits and Covers
Finding Cover with an Arbitrarily Large Lowest Modulus

Amina Dozier, Charles Fahringer, Martin Harrison, Jeremy Lyle

Dr. Neil Calkin, Dr. Kevin James, Dr. Dave Penniston

July 13, 2006

Abstract

Blankits and Covers

Finding Covers with a Large Lowest Modulus

Amina Dozier, Charles Fahringer, Martin Harrison, Jeremy Lyle

Dr. Neil Calkin, Dr. Kevin James, Dr. Dave Penniston

Computaional Number Theory and Combinatorics 2006 REU

Clemson University, Clemson, SC 29634

In this paper we will look at different ways to find covers with a large lowest modulus. We found some conditions neccessary for a collection of congruence classes to cover the integers. We determined that certain types of collections of congruence classes cannot cover the integers. To compute a cover we created a program that uses a greedy algorthm to create potential covers by covering the most remaining numbers with each congruence class choosen. In the end, we have a cover with lowest modulus 14.

Introduction

A covering, defined formally below, is a set of congruence classes such that every integer is in at least one of the congruence classes. A distinct cover (or simply a cover, for convenience) is a cover all of whose moduli are distinct. Quite a bit of work has been done in studying covers, and two major problems (posed by Erdős) have not yet been solved. The first, which we will discuss, states that the smallest modulus of a distinct cover can be arbitrarily high. The second, which we did not research as extensively, asks whether a cover exists with all odd moduli. Our task, in this paper, is to construct a cover with a large smallest modulus using computation. Now, some definitions:

Definition 1. *A blankit,* (a, m)*, is a collection of congruence classes*
$a_i \pmod{m_i}$,$1 \leq i \leq k$

We are interested in a particular type of blankit, those with distinct moduli and which include every integer. These blankits are called *covers* or *coverings*. If a blankit is also a cover, we say that it *covers*.

Definition 2. *Arithmetic progressions*

$$a_1 \pmod{n_1}, a_2 \pmod{n_2}, \ldots, a_k \pmod{n_k},$$

where

$$1 < n_1 < n_2 < \ldots < n_k,$$

form a ***distinct covering system*** *(or* ***cover****) if*

$$\forall m \in \boldsymbol{Z}\ \exists i$$

such that

$$m \equiv a_i \pmod{n_i}.$$

0.1 An Example of a Cover

The blankit $\{0 \pmod 2, 0 \pmod 3, 1 \pmod 4, 1 \pmod 6, 11 \pmod{12}\}$ is a cover. Proof: Note that every number that is 0 mod 12 is also 0 mod 2. In fact, every number that is 2,4,6,8, or 10 mod 12 is also 0 mod 2. Furthermore, this applies to all residues mod divisors of 12: if a is congruent to b mod c (c divides 12), then all numbers in a mod 12 are covered by b mod c. Thus, to cover the integers (all of which are in one of the congruence classes mod 12), we only need to cover the integers mod 12 using divisors of 12 as moduli. Checking our example, we find that 0 mod 2 covers (0,2,4,6,8,10), 0 mod 3 covers (0,3,6,9), 1 mod 4 covers (1,5,9), 1 mod 6 covers (1,7), and 11 mod 12 covers (11). Thus, the example covers the integers. We use this general idea to create covers. If we can cover the integers mod d using only moduli that divide d, then we have covered the integers.

We do not attempt to prove the smallest modulus conjecture, but we establish some elementary conditions necessary for a blankit to be a cover.

Theorem 1. *If a blankit covers, then the sum of the reciprocals of its moduli must be at least* 1.

Proof. Let (a, m) be a cover with moduli $m_1, m_2, \ldots, m_k$ with corresponding residues $a_1, a_2, \ldots, a_k$, and let M be the least common multiple of the moduli. In general, m_i's congruence class will contain $\frac{M}{m_i}$ numbers mod M.

Since, by assumption, every number $1, \ldots, M$ is covered, we have

$$\left| \bigcup_{1 \leq i \leq k} S_i \right| = M$$

where $S_i = \{n | 1 \leq n \leq M, n \equiv a_i \pmod{m_i}\}$.

This tells us $\sum_{i=1}^{k} \frac{M}{m_i} \geq M$. Dividing by M yields the desired result.

□

We will use this result to establish some results which show us where not to look when trying to increase the size of our moduli.

Theorem 2. *The "smallest modulus" problem is equivlent to the "minimum height" problem. That is, the following are equivalent:*
(i) Given any integer N, there exists a covering whose smallest modulus is larger than N.
(ii) Given any integer N, there exists a blankit, b, with the property that every integer belongs to at least N of b's congruence classes.

Proof. Suppose we can cover the integers with an arbitrarily high smallest modulus. Then to cover the integers j times, we make a cover of the integers with smallest modulus n. This has a highest modulus, m. We make a new cover of the integers using m+1 as the smallest modulus. This has a highest modulus, repeat the process j times.

Suppose we can cover the integers arbitrarily many times. Such a multiple covering has a lowest modulus. Since we cover everything the lowest congruence class covers more than once, we can eliminate this congruence class. Repeat this process until we have the smallest modulus as large as we like. □

Material

In this research we will find covers from various mathematical and computional calculations. This is done using Maple and C++. Maple is a program designed to help do computations in a math friendly enviorment. C++ is an object-oriented programming language that is mostly used for lage applications. In this problem we use Maple and C++ to create covers with a given smallest modulus.

Methods

For a blankit to be a cover we conclude that sum of the reciprocals must be at least one. More generally, the sum of the reciprocals of a set of moduli must be greater than 1+(the sum of the reciprocals of the numbers in the set below n) in order to make a cover with lowest modulus n, using members of the set as moduli. Thus, the divisors of $n!$ must be greater than or equal to $1 + log(n) + \gamma$, to have a cover with lowest modulus n. Note that γ (gamma) is the Euler-Mascheroni constant, the error term between $log(n)$ and the harmonic series.

Proposition: The function σ_{-1}, where $\sigma_{-1}(m) = \sum_{d|m} \frac{1}{d}$, is multiplicative. That is, if naturals m and n are relatively prime, then $\sigma_{-1}(mn) = \sigma_{-1}(m)\sigma_{-1}(n)$.

Proof. Note that for any integer m, we have that

$$\sigma_{-1}(m) = \frac{\sigma(m)}{m}$$

where $\sigma(m)$ gives the sum of the divisors of m. We will show that σ is multiplicative.

Let m and n be positive integers. Let $\sigma(m) = a_0 + a_1 + \ldots + a_k$ and $\sigma(n) = b_0 + b_1 + \ldots + b_l$.

Then

$$\sigma(m)\sigma(n) = \sum_{0 \le i \le k, 0 \le j \le l} a_i b_j$$

Note that if $a_i | m$ and $b_j | n$, then $a_i b_j | mn$, so each of the summands is a divisor of mn.
Now suppose that some number d divides mn. Let $c = gcd(m, d)$.

Then

$$\frac{d}{c} | \left(\frac{m}{c} n\right)$$

and $gcd(\frac{m}{c}, \frac{d}{c}) = 1$, so we must have that$\frac{d}{c}$ divides n. That is, $\frac{d}{c} = b_j$ for some j, and so d is of the form $a_i b_j$.

Let's check that we have not added any divisor twice:
Suppose that $a_i b_j = a_p b_q$. Then $a_i | a_p b_q$, and since $gcd(a_i, b_q) = 1$, we must have $a_i | a_p$. A similar argument shows $a_p | a_i$, and therefore $a_i = a_p$. It follows that $b_q = b_j$.

Note that

$$\sigma_{-1}(mn) = \frac{\sigma(mn)}{mn} = \frac{\sigma(m)\sigma(n)}{mn} = \frac{\sigma(m)}{m}\frac{\sigma(n)}{n} = \sigma_{-1}(m)\sigma_{-1}(n)$$

□

Theorem 3. $\sigma_{-1}(n!) \ge 1 + log(n) + \gamma$

Proof. To get the powers of the prime divisors in $n!$, add 1 for every multiple of p up to n, add an additional 1 for every multiple of p^2 up to n, continuing this pattern, finally add$\lfloor log_p(n) \rfloor$ for the only multiple of $p^{\lfloor log_p(n) \rfloor}$ up to n. The sum of the reciprocals of the divisors of each of these prime powers is simply a geometric series

$$\sigma_{-1}(p^k) = 1 - (1/p)^{k+1}/(1 - 1/p).$$

Since we have the prime powers in $n!$, we will replace k with each of these.

$$k_p = \sum_{i=1}^{log_p(n)} \left\lfloor n/p^i \right\rfloor .$$

We approximate this by

$$k_p = \sum_{i=1}^{log_p(n)} n/p^i - 1 = n \sum_{i=1}^{log_p(n)} 1/p^i - log_p(n).$$

The summation is a geometric series, so we have

$$n \sum_{i=1}^{log_p(n)} 1/p^i - log_p(n) = n(1/p)(1-(1/p)^{log_p(n)})/(1-(1/p)) - log_p(n) =$$

$$n(1/p)(1-(1/n))/(1-(1/p)) - log_p(n) = (n-1)/(p-1) - log_p(n).$$

Since σ_{-1} is multiplicative, we can take σ_{-1} of the prime powers of $n!$ and multiply these to get

$$\sigma_{-1}(n!).$$

$$\sigma_{-1}(p^k), \text{ where } k = N_p(n!),$$

is the geometric series

$$\sum_{i=0}^{k} 1/p^i = 1 - (1/p)^{k+1}/(1-(1/p)).$$

We replace k by

$$(n-1)/(p-1) - log_p(n)$$

to get

$$(1-(1/p)^{(n-1)/(p-1)-log_p(n)+1})/(1-(1/p)).$$

So

$$\sigma_{-1}(n!) \geq \prod_{p=2}^{n} (1 - n(p^{(1-n)/(p-1)-1}))/(1-(1/p)).$$

Now, Merten's theorem tells us that

$$\prod_{p \leq n} (1-(1/p)^{-1})$$

is asymptotic to

$$e^{\gamma} log(n).$$

Since the product of the numerator goes to 1, we have that

$$\sigma_{-1}(n!)$$

is asymptotic to

$$e^{\gamma} log(n),$$

which is greater than

$$1 + log(n) + \gamma.$$

□

0.2 Coverings we Can't Make

Since we know what types of blankits can't be covers, we know where not to check with our computations. The common idea of the next two theorems tells us that if we are trying to make a cover of the integers using certain moduli, we can check whether they can't be sufficient without checking every set of congruence classes we could use.

For instance, we can't make a cover using powers of 2 because the sums of the reciprocals of powers of 2 is always less than 1. We also know that we can't make a cover using 2,3, and 4 as moduli.

Proof. : We can't choose better congruence classes than 0 (mod 2), 0 (mod 3), and 1 (mod 4). We have to choose so that the congruence classes (mod 2) and (mod 3) intersect every 6 integers, and so that the congruence classes of (mod 3) and (mod 4) intersect every 12 integers. By the Chinese Remainder Theorem, we can't avoid that. But we can prevent the congruence class (mod 4) from ever intersecting the congruence class (mod 2), and we do that.

So this is the most efficient potential cover we can make using 2,3, and 4 as moduli. Checking to see whether this covers the integers mod the lcm of these numbers, we find that it doesn't. Specificially, it does not cover 7 or 11 (mod 12). □

Intuitively, the next proposition says that if we wish to find coverings of larger and larger smallest modulus, we must also increase the number of prime divisors of the product of our moduli. First, we need a new definition:

Definition 3. *An integer is $b-$ smooth if each of its prime divisors is at most b.*

Theorem 4. *For any integer b, there is an integer n such that no blankit,(a, m), satisfying:*

$$(i) m_i > n \text{ for all } i$$

$$\text{and}$$

$$(ii) m_i \text{ is } b\text{-smooth for all } i$$

can be a covering.

Proof. We will show that given b, there is a number N with the property that the sum of the reciprocals of b-smooth numbers greater than N is strictly less than 1.

Note that for each prime, p, the geometric series $\sum_{k=0}^{\infty} \left(\frac{1}{p}\right)^k$ converges absolutely to $\frac{p}{p-1}$.

Note also that the product $\prod_{p \leq b} \frac{p}{p-1}$ is finite and equals the sum of the reciprocals of the b-smooth numbers (plus 1).

Since all of these geometric series converge absolutely, so does their Cauchy product, as well as any rearrangement of it. So, given b, consider the series $\sum_{1 \leq i} \frac{1}{a_i}$, where $\{a_i\}$ is the sequence of b-smooth numbers and is increasing.

By the convergence of this series, we can find n so large that $\sum_{i=1}^{i=n} \frac{1}{a_i}$ is within 1 of the limit ($\left(\prod_{p \leq b} \frac{p}{p-1}\right) - 1$). It follows from Proposition 1 that no blankit with smallest modulus a_{n+1} or greater can cover. □

Theorem 5. *There does not exist a cover all of whose moduli are pairwise co-prime:*

Proof. By the Chinese Remainder Theorem, we get just as many intersections up to the lcm of the moduli among 2, 3, or more congruence classes with pairwise co-prime moduli, no matter what congruence classes we choose.

Therefore, we cover just as many things no matter what congruence classes we choose. Therefore, we can choose the congruence classes arbitrarily, and if we show that we can't cover using our arbitrary choice of congruence classes, then we can't cover at all. We choose 0 (mod p) for all of the congruence classes. This obviously does not cover 1. Since we couldn't have choosen so as to get fewer intersections (and therefore cover more numbers), we always get at least this many uncovered numbers.

Therefore, we always have at least 1 uncovered number, for any set of congruence classes with pairwise co-prime moduli. □

Theorem 6. *There does not exist a cover whose moduli are all prime powers:*

Proof. We can prevent two congruence classes in such a potential cover from intersecting if

they have not co-prime moduli. We do this by choosing 1 mod p, p mod p^2, etc. For every set of congruence classes with not co-prime moduli (the powers of a particular prime).

This prevents any intersections among those congruence classes because p^{n-1} is congruent to 0 mod all the previous powers of the prime, but we choose p^{j-1} mod p^j for all of those. Therefore, to minimize (to 0) the amount of intersections among moduli that aren't co-prime, we can take p^{n-1} mod p^n for every prime power we use.

Since we always get just as many intersections among congruence classes with relatively prime moduli, we've minimized the number of intersections we can get. Therefore, if something remains uncovered, we cannot make a covering in this way. 0 remains uncovered (since a prime to an integer power is never 0). Therefore, we will always have at least 1 uncovered number for a potential cover using congruence classes with prime power moduli. Therefore, a cover with congruence classes with prime power moduli does not exist. □

0.3 Covers we Can Make

On the other hand, some types of covers can be made that are convenient for computation. For instance, covers mod n can be made using the divisors of n for certain types of numbers (with many divisors). The integers mod $n!$ can be covered using the divisors of n!. Considering that we can cover the integers mod 12 using the divisors of 12, it should be apparent that we can cover the integers mod 24 using its divisors, and any other multiple of 24 including any higher factorial.

In fact, this method is so efficient that we found (up to a certain point, at least) that we can cover the integers mod n! without using any of the divisors up to n as moduli. Unfortunately, n! grows so quickly that it stopped being feasible to check for coverings of the integers mod n!.

It should not surprise us that we can make a covering of the integers using the divisors of n!. After all, the divisors of n! eventually will include every number.

However, there are some types of covers that can be made that we were not expecting to find. For instance,making a covering of the integers using only square free numbers (numbers without prime power factors other than p^1). This first happens when we cover the integers (mod 210) with its divisors. It also seems possible that we can cover the integers using congruence classes that don't intersect unless their moduli are relatively prime. We are looking into this right now.

0.4 Writing the Code

Our aim is to produce a program whose input is a list of moduli (or a number whose divisors will serve as moduli) and whose output is a potential cover. We also want this program to tell us whether its output is a cover, or merely a blankit. So what should this program do? First, it should use the greedy algorithm to choose an appropriate residue class for each modulus (this will be discussed in greater detail later). Next, it should check whether or not each integer less than or equal to the lcm of the moduli belongs to at least one of the chosen congruence classes. Remember, a blankit with this property covers all the integers. We are also interested in the number of distinct congruence classes to which each such integer belongs, and in particular, the height of the blankit. This tells us how wasteful we are being with our choices of congruence classes (greater height means we've wasted more potential "'eliminations"' up to the LCM).

Now that we know the desired procedure, lets discuss how well implement it. In determining residue classes, we used a greedy algorithm. A greedy algorithm is an algorithm that will choose, at each stage, the local optimum. In the context of our problem, this means choosing the residue class that contains the largest number of integers not already belonging to some previously chosen congruence class. We used an array to keep track of the integers blanked out by our collection of congruence classes. Due to the growth rate of the array used, the original code had to be modified so that it would run within the memory restrictions.

The first program (Program 1) uses an array that allows us to tally the number of congruence

classes to which each integer belongs. It uses the Least Common Multiple (LCM) algorithm given by Euclid to minimize the number of integers checked: we need only check the integers less than or equal to the lcm of the moduli up to the one whose optimal residue were seeking. After that, the behavior of the set of congruence classes repeats. This program will also compute the sum of the reciprocals of the moduli, and the number of moduli used (it stops once it has found a cover). This code contains a function, boolgreedy, which is not shown as part of the results. In order to confirm that we do indeed have a cover, we run this function. It simply goes through the array and prints the height at each number. The height at the number n is the number of distinct congruence classes (of a given blankit) to which n belongs. The height of the blankit is the maximum of these local heights.

The second program (program 2) took the array and converted it to a bit vector to use one eighth as much memory. Now we will receive the same results as the first program but more quickly, and we can run the program for higher numbers. In addition we added an algorithm that uses random variables to decrease the time it takes to compute the code. Instead of checking, at each modulus we could use, the number of things covered in the entire array by each potential congruence class, we pick 100*the modulus random numbers lower than the size of the big array (these should be approximately equi-distrubited among the congruence classes), and we choose "'semi-greedily"' whichever congruence class has the most uncovered numbers in it among the ones randomly choosen. This allows us to remove the array completely and just run through the loop inputting the random numbers, and recording whether or not numbers are covered, instead of a list of all covered or uncovered numbers.

Before running the code for these programs we chose to make the size of our arrray (N) to be $(n+2)!$. Thus, we our covering the numbers mod (n+2)! with congruence classes mod its divisors. As noted above, this seems like a reasonable place to look for covers. In fact, it allows us to eliminate low moduli by increasing the n. Furthermore, after running the code for larger smallest modulus, we noted that we could begin using (n+1)!, and we'd still get a cover. In fact, at lowest modulus 12 we find a cover using divisors of 12!. We suspect that

using the divisors of n! continues to create covers with lowest modulus n, but the size of the array (13! at the next step) required more memory than we had.

To fix this problem we are looking at what happens to the sum of the reciprocals of the divisors of numbers as we multiply by primes. We begin with 2, then multiply by 3 because this increases $\sigma_{-1}(n)$ more than multiplying by any other prime. Then, we use another 2, and so on. Later, instead of choosing greedily like this, we try to limit the size of our array by multiplying by small primes instead of large ones, even though multiplying by the larger ones might increase $\sigma_{-1}(n)$ more. Of course, by theorem 5, we must eventually multiply by new primes in order to get a larger smallest modulus. So far, we have no algorithmic way of doing this, and we've been checking various ways of forming numbers in this way that we think will probably work for getting a cover with lowest modulus n, for arbitrarily chosen n. Most recently, we have used $N = 128*27*25*49*11*13*17$, which we suspect will get us a cover with smallest modulus 20. In order to get a cover with modulus greater than 20, however, we probably have to multiply N by 19, which would take too long on one machine, given our random algorithm. However, if we parallelize the code to run on 3 or more machines, we might get it to work. Furthermore, multiplying N by 19 gets us so far that we can reduce the size of some of our lower prime's powers. Instead of using $N = 128*27*25*49*11*13*17*19$ we might get away with $N = 32 * 27 * 25 * 49 * 11 * 13 * 17 * 19$.

Procedure

Now lets look at the two codes used for solving this problem.

0.5 Program 1

0.6 Program 2

Results

0.7 Program 1

By using the Program 1 we were able to find a cover of least modulo 14.

The results (14div4.txt) shows a list of the blankit in its entirety. It also shows the total cover height ,sum of the recprical, total number of moduli used, and the total time(seconds) it took to run.

This results should be read in the following form:

(residue) mod *(divisors)* covers: *(#)* holes up to *(the lcm)*

0.8 Program 2

We ran program 2 for a long time, and got no results. After choosing the first several congruence classes, the program finds 100*the modulus (or less, for larger divisors) random numbers up to the lcm of the numbers checked so far, and checks whether these are already covered, and if they aren't, which congruence classes contain them. However, if none of the randomly choosen numbers are uncovered, we choose again. We suspect that when the computer is close to making a cover, we have so few uncovered numbers left that we are not likely to get any uncovered numbers from our random choices, and we continue to choose randomly with very little chance of getting any further in the program. If we fix this problem, program 2 should run in a decent amount of time.

Discussion

In the end we are now able to prove that there does exist a cover with lowest modolus 14. Although this is not quite our goal of 25, we have a method that we think could work, if we had computers with infinite memory and infinite processing speed. Given that we don't have these things, we will continue to work on improving our code until we reach our goal. Also, we have begun writing code to search for a cover using odd moduli.

Acknowledgements

We acknowledge Clemson University's Department of Mathematical Science for Hosting the 2006 Computational Number Theory and Combinatorics Summer REU. We thank the National Science Foundation(NSF), who provided the grant to allow the REU to take place. A special thanks to Philip Lee and Kenny Stauffer who helped modify the code. We would like to express much gratitude to the graduate students and professors who are working with us throughout this summer and in the future.

References

[1] Mark Herkomme Number Theory: A programmer's Guide
Copywright 1999 by McGraw-Hill Company Inc.

[2] Ivan, Nieven, Herbert S Zuckerman,Hugh L. Montgomery
An Introduction to the Theory of Number (5th edition)
Copyright 1960, 1966, 1975, 1980, 1991 by John Wiley & Sons Inc.

[3] Choi, S.L.G. "Covering the Set of Integers by Congruence Classes of Distinct Moduli." Mathematics of Computation 25:116 *(October 1971): 885-895.*

[4] Filaseta, Micheal, et al. "Sieving by Large Integers and Covering Systems of Congruences." *2006 The ArXiv. 9 May 2006* <http://arxiv.org/abs/math.NT/0507374>