Difference Covers

Adam Savard

asavard@uvm.edu

Advisor: Robert Snapp

March 6, 2015

Abstract

I explore the concept of difference covers, as well as the related concepts of difference packing and sets, with the primary goal of finding smaller difference covers than those already known for modulo groups of order greater than 128. Using Golomb rulers and tree searches, I find several difference covers and compile a comprehensive list of the smallest covers I found for various groups. This method, although initially promising, did not provide the results I was hoping to find, which led me to further explore search methods and the patterns that arise in difference covers. A known method of finding solutions to NP-Hard problems, simulated annealing, ultimately led me to discover many difference covers I was unable to find with Golomb rulers. I applied different cooling schedules to the problem and had best results using simulated annealing with a constant thermodynamic velocity.

1 Introduction

1.1 Difference Covers

A difference cover D is a subset of a group G such that for all $x \in G$, $x = i - j$ for some $i, j \in D$. That is, every element in G is expressible as a difference of two elements in D. For example, for the modulo group under addition of order 3, $\mathbf{Z}_3$, the set {0, 1} would be

a minimum difference cover. Since $0 = 0 - 0 \; mod \; 3$, $1 = 1 - 0 \; mod \; 3$ and $2 = 0 - 1 \; mod \; 3$, the set $\mathbf{Z}_3$ is covered by the differences of {0, 1}. The entire set $\mathbf{Z}_3$ is also a difference cover as well as {0, 2}.

Proposition 1.1 For the group modulo v, the smallest order a difference cover for that group can possibly have is the smallest integer n such that $n(n-1) \geq v$.

A subset of n elements can have as many as $n(n-1)$ unique nonzero differences since each pair of elements represents two differences. In order for a subset to cover modulo v, there must be at least $v-1$ distinct nonzero differences since every difference from 1 to $v-1$ needs to be covered. If $n(n-1) < v-1$, it is not possible for v differences nonzero to be covered by the subset, because the maximum number of differences is less than the number of difference required to cover the group.

Minimum difference covers, or optimal difference covers, of G are difference covers whose length is the smallest possible length of a difference cover for G [2]. As of the time I started my research, difference covers of minimum size for modulo groups up to 128 have already been found by brute force searches [5]. The problem of finding minimum difference covers has been shown to be NP-Hard and thus more robust search methods are required to continue finding them for larger groups [15].

Note that because modulo groups under addition are cyclic groups, {0, 1} and {0, 2} are effectively the same difference cover of $\mathbf{Z}_3$. Adding a constant to each element in a subset does not change the set of differences that occur in that subset. Because of this, and since the difference 1 always appears of covers of modulo groups, I will ignore all subsets of these groups that do not include {0, 1} since they are additively equivalent to those that do include {0, 1}. That is, an integer can be added to each element of a cover that does not contain {0, 1} to create a cover that does. Likewise, I will ignore covers that include 2 when searching the entire search space since they will be additively equivalent to a cover that does not contain 2. This is because as long as 2 is contained, each element can have 1 subtracted from it as many times as it takes for 2 to no longer be contained. This works as long as there is an element in the set not contained by the cover, which will always be the

case. It is also possible to multiply each element of a cover by -1 to create a cover that is multiplicatively equivalent. The cover {0, 1, 3} in modulo 6 for instance can be multiplied by -1 which gives {0, 3, 5} which can have 1 added to each element to produce {0, 1, 4}. The differences that appear in {0, 1, 4} must be the same as the differences that appear in {0, 1, 3}. While I did take additive equivalence into account in my searches by assuming 0 and 1 are included and 2 is not, I did not implement a way to remove sections of a search tree to avoid the redundancy of finding two covers that can be shown to be equivalent through multiplication.

Table 1.1

An additively inequivalent list of all minimum difference covers for modulo groups 1 through 13.

Modulo							
1	{0}						
2	{0, 1}						
3	{0, 1}						
4	{0, 1, 3}						
5	{0, 1, 3}	{0, 1, 4}					
6	{0, 1, 3}	{0, 1, 4}					
7	{0, 1, 3}	{0, 1, 5}					
8	{0, 1, 3, 4}	{0, 1, 3, 5}	{0, 1, 3, 7}	{0, 1, 4, 6}	{0, 1, 4, 7}	{0, 1, 5, 6}	{0, 1, 5, 7}
9	{0, 1, 3, 4}	{0, 1, 3, 5}	{0, 1, 3, 6}	{0, 1, 3, 7}	{0, 1, 3, 8}	{0, 1, 4, 6}	
	{0, 1, 4, 7}	{0, 1, 4, 8}	{0, 1, 5, 7}	{0, 1, 5, 8}	{0, 1, 6, 7}	{0, 1, 6, 8}	
10	{0, 1, 3, 5}	{0, 1, 3, 6}	{0, 1, 4, 6}	{0, 1, 4, 9}	{0, 1, 5, 7}	{0, 1, 5, 8}	
	{0, 1, 6, 8}	{0, 1, 6, 9}					
11	{0, 1, 3, 7}	{0, 1, 3, 5}	{0, 1, 3, 8}	{0, 1, 4, 6}	{0, 1, 4, 9}	{0, 1, 4, 10}	
	{0, 1, 7, 10}	{0, 1, 5, 9}	{0, 1, 6, 8}	{0, 1, 7, 9}			
12	{0, 1, 3, 7}	{0, 1, 4, 6}	{0, 1, 6, 10}	{0, 1, 7, 9}			
13	{0, 1, 5, 11}	{0, 1, 4, 6}	{0, 1, 8, 10}	{0, 1, 3, 9}			

Colbourn and Ling (2000) used difference covers as a tool to produce systems of quorums and in doing so created a difference cover construction which was later used to find covers for large modulo that were better than those previously known. A quorum is a set of sites in a system of sites that communicate over a network that grant permission to another site to access a resource. To ensure mutual exclusion, each quorum must have at least one site in common with each other quorum. While the paper created a method of

constructing more optimal systems of quorums, it was mostly cited for its use of difference covers. This construction builds a sequence of $6r + 3$ elements from a specific sequence of integers that acts as a difference cover for any modulo $v \leq 12r^2 + 18r + 6$ [1]. This construction has been useful for finding covers for larger modulo, finding length 20 covers for modulo 256 and length 29 covers for modulo 512 [5]. These covers however are unlikely to be minimum and methods of finding better covers for such modulo still need to be found.

Kärkkäinen et al. (2006) later utilized difference covers to create a linear time construction algorithm for suffix arrays. In doing so, they utilized the concept of a difference cover sample. They define a v-periodic sample C of [0, n] with the period D, where D is a difference cover modulo v, by $C = \{i \in [0, n] | i \ mod \ v \in D\}$. The members of C will become the indices of the suffixes used as a sample to be sorted before the rest of the array is sorted. The properties of difference covers happen to be useful in this case to more efficiently sort the remaining suffixes. This is due to the periodic nature of difference covers and the fact that for any indices $i, j \in [0, n - v + 1]$ there is an l such that $i + l$ and $j + l$ are both members of the sample [5]. While the existing applications of difference covers are narrow, it is likely that there will be other uses for difference covers found gradually in the future.

1.2 Difference Packings

A difference packing P is a subset of a group G such that for all $x \in G$, $x = i - j$ for no more than one $i, j \in C$. This is similar to a difference cover except we actually want to search for the largest possible packings, since we want to get as close as possible to having every difference covered once without repeating a difference. By their nature, difference packings tend to be smaller than difference covers, and are smaller on all but a few specific modulo. The problem of finding maximum difference packings is NP Hard as well since it scales in the same way finding minimum difference covers does as the order of the group increases. For any given group however, finding maximum packings will be faster than finding minimum covers using backtracking or a tree search because the same difference can never appear twice. Finding packings using other methods that can be applied to covers like simulated annealing may actually be more difficult.

Table 1.2

A non-redundant concise list of maximum difference packings for modulo groups 1 through 13.

Modulo						
1	{0}					
2	{0}					
3	{0, 1}					
4	{0, 1}					
5	{0, 1}					
6	{0, 1}					
7	{0, 1, 3}	{0, 1, 5}				
8	{0, 1, 3}	{0, 1, 6}				
9	{0, 1, 3}	{0, 1, 4}	{0, 1, 6}	{0, 1, 7}		
10	{0, 1, 3}	{0, 1, 4}	{0, 1, 7}	{0, 1, 8}		
11	{0, 1, 3}	{0, 1, 4}	{0, 1, 5}	{0, 1, 7}	{0, 1, 8}	{0, 1, 9}
12	{0, 1, 3}	{0, 1, 4}	{0, 1, 5}	{0, 1, 8}	{0, 1, 9}	{0, 1, 10}
13	{0, 1, 5, 11}	{0, 1, 4, 6}	{0, 1, 8, 10}	{0, 1, 3, 9}		

1.3 Difference Sets

A difference set S is a subset of group G such that for all $x \in G$, $x = i - j$ for exactly λ pairs $i,\ j \in G$ where $\lambda \in \boldsymbol{N}$. Difference sets are defined by the parameters v, k and λ where v is the order of the group, k is the order of the subset and λ is the number of times each difference appears [14]. Difference sets are a much more heavily studied concept than covers and packings, occurring naturally in many combinatorial problems [9]. Since they are a similar concept to difference covers, I considered them valuable to learn about in order to better understand difference covers. Difference sets are much more specific in nature than difference covers. When searching for a difference set with a specific set of parameters, the order of the set is already known as well as how many times each difference appears. This makes tree searches markedly simpler than in the case of difference covers.

Difference sets, difference covers and difference packings overlap when every non-identity element of G can be represented by the difference of two elements in a subset in exactly one way. Since a subset of length n has $(n-1)n$ nonzero differences, only modulo groups expressible as $(n-1)n+1$ for $n \in Z$ can potentially have subsets that fit this specific constraint. Not all such groups will contain such a subset, although it has been

proven that these sets will exist for certain modulo that fit this description as will be explained later.

The search methods I find for difference covers should be applicable to difference sets as well. Perhaps a tree search for a difference set would ignore nodes with too many repetitions of any difference and value nodes based on proximity to the correct set of differences. Simulated annealing methods, which I later cover, would be similarly applicable.

1.4 Golomb Ruler

A Golomb ruler is a set of nonnegative integers such that no two pairs are the same distance apart. Golomb rulers by convention include 0 since the smallest value can otherwise be subtracted from each element. The order of a Golomb ruler is the number of elements in the ruler and the length is the value of the largest element. The applications of Golomb rulers vary widely and include error correcting codes, radio frequency selection, radio antenna placement and current transformers [10, 11, 12].

Golomb rulers are conveniently similar in nature to difference packings, since they consist of integers that lack repeated differences. In fact, any Golomb ruler of length L is by definition a difference packing for modulo groups of order 2L and greater. While finding shortest length Golomb rulers is an NP-hard problem, all such rulers have already been found up to order 27. These rulers are very similar in nature to difference packings since no difference repeats and in theory, minimum difference covers may contain nontrivial Golomb rulers, thus leading to the idea that rulers may be helpful in finding minimum difference covers. I explore the potential of these Golomb rulers to be used as starting points in searches for minimum length difference covers of modulo groups large enough such that the rulers will function as difference packings, which usually means the modulo is at least twice the length of the ruler.

Suboptimal Golomb rulers could theoretically also be useful but the higher length to order ratios means they are only applicable to larger modulo groups at the same order, which means having a smaller base to start with for any given modulo group. There are fast construction methods to produce Golomb rulers but they produce rulers so large they will not be useful for this purpose. Because of this, I have stuck to using the optimal

Golomb rulers.

Table 1.3

A comprehensive list of optimal Golomb rulers up to order 15 as listed on the IBM website [14].

Order	Length	Marks
1	0	{0}
2	1	{0, 1}
3	3	{0, 1, 3}
4	6	{0, 1, 4, 6}
5	11	{0, 1, 4, 9, 11}
5	11	{0,2, 7, 8, 11}
6	17	{0, 1, 4,10, 12, 17}
6	17	{0, 1, 4, 10, 15, 17}
6	17	{0, 1, 8, 11, 13, 17}
6	17	{0, 1, 8, 12, 14, 17}
7	25	{0, 1, 4, 10, 18, 23, 25}
7	25	{0, 1, 7, 11, 20, 23, 25}
7	25	{0, 1, 11, 16, 19, 23, 25}
7	25	{0, 2, 3, 10, 16, 21, 25}
7	25	{0, 2, 7, 13, 21, 22, 25}
8	34	{0, 1, 4, 9, 15, 22, 32, 34}
9	44	{0, 1, 5, 12, 25, 27, 35, 41, 44}
10	55	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}
11	72	{0, 1, 4, 13, 28, 33, 47, 54, 64, 70, 72}
11	72	{0, 1, 9, 19, 24, 31, 52, 56, 58, 69, 72}
12	85	{0, 2, 6, 24, 29, 40, 43, 55, 68, 75, 76, 85}
13	106	{0, 2, 5, 25, 37, 43, 59, 70, 85, 89, 98, 99, 106}
14	127	{0, 4, 6, 20, 35, 52, 59, 77, 78, 86, 89, 99, 122, 127}
15	151	{0, 4, 20, 30, 57, 59, 76, 100, 111, 123, 136, 144, 145, 151}

2 Search Techniques

2.1 Tree Searches

Tree searches, such as depth first and Monte Carlo, are a simple and effective way to search for minimum difference covers. Each node in the tree search represents a subset of the given modulo group with the root being {0, 1} and each node having a child for each element that can be added. These nodes can be assigned a value based on how many differences are repeated, where the nodes with less repeated differences are more promising. If the size of the cover being searched for is known, then we know how many repeated differences the cover will have, thus we can remove all nodes from our search

that reach a point of having too many differences repeated to possibly have the cover we want among its descendants. The tree is constructed during the search, starting with the node and repeatedly choosing a node that has not already been chosen to add their children to the search. Nodes corresponding to order $s - 1$ subsets can be quickly checked to see if a cover of length s can be constructed from them and then they are removed from the tree. Nodes whose children have been removed can also be removed, so that eventually there will be no remaining tree to search.

In depth first tree searches, only the highest length nodes are considered when deciding which node's children to look at next. In a Monte Carlo tree search however, every node in the tree has to be considered. Because of this, nodes are valued based on both how few differences are repeated and how many elements their subset contains. If the value of increased length is high enough, a depth first search is effectively being conducted and if it is low enough, the search will be nearly equivalent to a breadth first search. Because the breadth of these trees is too great to search, the increase in value with each additional element must be sufficient to quickly reach higher depth nodes.

The tree can also be simplified to eliminate duplicate nodes. To do this, a node's children will contain each element in that node as well as an element that is larger than any element in the parent node. This way, you are only adding elements to the cover in ascending order, so that, for instance, the subsets {0, 1, 3, 5} and {0, 1, 5, 3} do not both appear in the tree. This creates a lopsided tree since {0, 1, 3} has a large number of descendants while {0, 1, n – 1} has no children at all. If the tree is going to be broken up into sub trees for multiple threads to search, this imbalance should be taken into account.

A more sophisticated version of this which I have found much more useful is to first sort the children of a node, then when traversing into any of their subtrees to eliminate elements from their children that were added by their previous sibling. For instance, we generate the children for {0, 1, 5} then sort those children in descending order of how promising they seem to be. If for instance {0, 1, 26}, {0, 1, 24} and {0, 1, 23} are the first three children, respectively, we would avoid children of {0, 1, 24} containing 26 and children of {0, 1, 23} containing 26 or 24. This creates the same effect of a skewed tree but one that is skewed in a way specifically designed to produce faster results.

Golomb rulers can be easily applied to tree searches. Instead of the root node being {0, 1}, it would be a specific Golomb ruler. In this way, only a tiny subset of the entire tree is actually searched. We generally only need to find at least one of the difference covers each modulo contains so this will work as long as there is a single difference cover in this subset of the tree. The number of nodes in a tree for modulo v is 2^{v-3} since we are counting the subsets of {3, 4, ..., v – 1}. The number of nodes in a tree that starts with a Golomb ruler length G is $(v - G)!$ so for $G > 3$, this reduces the search space by a factor of nearly $v(G - 3)$.

2.2 Backtracking

Backtracking is an algorithm used to find solutions to certain NP-Hard problems. The idea, in the case of difference covers or packings, would be to start with a small set, perhaps {0, 1} and recursively add elements. The algorithm backtracks when it is headed down a path that looks like it will not produce results and does not search that space again. It is similar in nature to tree searches in that it searches through every possible solution. It is in essence a more generalized version of a depth first tree search. The key difference being that you backtrack faster when it becomes apparent that a solution is unlikely to be found.

Ruskey and Sawada (1999) applied a backtracking algorithm that had been designed to find k-aray necklaces to generate difference covers. K-aray necklaces of length n are equivalence classes of strings of length n over an alphabet of k characters in which rotations are considered equivalent. They generate fixed-density necklaces of length n over an alphabet of size k where the position of each element of the necklace corresponds to a number in a difference cover of modulo k. The runtime complexity of their algorithm is $O(\binom{n-1}{k-1})$ for k order covers in modulo n. This algorithm found them minimal covers as large as modulo 131 [9]. This algorithm appears to be a marginally more effective and much more in-depth version of more naïve algorithms I have applied.

2.3 Direct Products

The direct product of two Groups, G and H $G \times H$ is defined such that every element of $G \times H$ is an ordered pair (g, h) such that $g \in G$ and $h \in H$. The operation of the group is defined as $(g_1, h_1) \cdot (g_2, h_2) = (g_1 \cdot g_2, h_1 \cdot h_2)$ for $g_1, g_2 \in G$ and $h_1, h_2 \in H$. The

direct product of two modulo groups with relative prime orders n and m is isomorphic to modulo $m * n$. The elements of these group map such that 0 maps to {0, 0}, 1 maps to {1, 1}, etc. If the two modulo are relatively prime, repeatedly adding {1, 1} will cycle through every possible combination of elements in the two modulo groups. If the two modulo are not relatively prime however, this does not work since repeatedly adding {1, 1} does not cycle through each possible combination of elements. I explored whether this could be used to find any patterns in difference covers that were not otherwise apparent and whether I could use known difference covers from the modulo being multiplied together to, in any way, get to a minimal cover in the larger group faster.

3 Computing

3.1 System Used

My implementations for this thesis were done entirely using Java. Every difference cover and packing I found was found using my personal computer. I am using a 64-bit Windows 8 machine with 8 GB of RAM and an Intel Core i5-4210U Processor.

3.2 Simulated Annealing

Simulated annealing was another search method I explored because it provides a promising way to explore the search space stochastically. This method is an analogy to statistical mechanics in which total entropy production is minimized while reaching the lowest energy state. Simulated annealing is generally applied to NP-Hard combinatorial problems that want minimum or maximum values of a function with many variables, i.e. a high dimensional space that can be searched [7].

The idea is to begin with a specific modulo v and a cover length s. This represents the state of our analogical system. An order s subset of modulo v that is relatively close to a difference cover to be easily generated by gradually adding elements to {0, 1} while avoiding too many repetitions in differences. An initial temperate T_0 is then set. The subset repeatedly has one of its elements, other than 0 or 1, changed, effectively moving to a neighbor in an (s-2)-dimensional search space. The temperature determines the tendency of the current subset to move towards a higher energy state, with a larger temperature

corresponding to a higher willingness to move to a higher energy state. The energy of a state can simply be determined for this problem by the number of differences that are not covered. The temperature gradually decreasing over time, making the movement of the state increasingly stable and ultimately making it unable to move out of a local minimum. Once the temperature reaches a temperature so low that the state can no longer escape from a local minimum that is not a difference cover, i.e. energy greater than zero, the temperature needs to be reset so that the process can be repeated [7].

The cooling schedule is crucial to the probability of ending up in a global minimum and the amount of time required to get there. The two simplest and commonly used cooling strategies are linear and exponential cooling, defined by $T(t) = T_0\alpha^t$ and $T(t) = T_0 - nt$ respectively. These are simply schedules with the idea of gradually subtracting a small constant from the current temperature or repeatedly multiplying the temperature by some number close to and less than 1. Another theoretically important cooling schedule is $T(t) = \frac{c}{\log(t+d)}$ where d is usually 1 and c is greater than or equal to the maximum value of $\frac{dE}{dt}$. This logarithmic cooling strategy has been proven to inevitably lead to the global maximum but the cooling is slow enough to render the method impractical. Linear and exponential were the first two cooling methods I tried because of their simplicity. I did not try logarithmic cooling because it is known to be generally useless in practice [7].

I also tried using a cooling method with a constant thermodynamic speed where the cooling rate is defined by $\frac{dT}{dt} = \frac{-vT}{\varepsilon\sqrt{C}}$ where v is the constant thermodynamic speed, ε is the relaxation time and C is the heat capacity. ε and C are both functions that depend on the way the energy of the system changes over time, thus the temperature at any given time cannot be computed ahead of time like it can be with the previous cooling methods I used. This provides a dynamic cooling method that adapts to the way the state changes over time [7].

Salamon et. al. (1988) demonstrated experimentally that constant thermodynamic speed works better than naive cooling schedules. To do this, they used to problem of placing 3,000 circuit elements on each of two chips while minimizing the number of wires connecting the chips. This was chosen because it is a difficult problem that demands a

robust cooling schedule and also has real applications. The effectiveness of cooling schedules was measured by the mean energy levels at large numbers of run steps. This cooling schedule was shown to perform better than $\frac{dT}{dt} = \frac{r}{\varepsilon}$, $\frac{dT}{dt} = rT^2$, $\frac{dT}{dt} = rT$, $T = \frac{2}{\log(1+k)}$, $T = 0$ and $\frac{dT}{dt} = r$. While the cooling schedule defined by $\frac{dT}{dt} = \frac{-vT}{\varepsilon\sqrt{C}}$ did perform as well as any other schedule, it performed no better than $\frac{dT}{dt} = \frac{-vT}{\varepsilon C}$ and $\frac{dT}{dt} = \frac{-vT}{\varepsilon C^{0.3}}$, indicating that the exponent of C has minimal impact on performance. Because of this, I used $\frac{dT}{dt} = \frac{-vT}{\varepsilon C}$ for the sake of speeding up the computation [4].

The heat capacity of the system, C, can be obtained from $C = \frac{dE}{dT} = \frac{\sigma^2(e)}{T^2}$ where $\sigma(e)$ is the variance in energy [4]. This variance in energy can be calculated exactly by determining the energy for every possible order s subset of modulo v and solving $\sigma^2(E) = \sum \frac{s!(v-s)!(E_i-\overline{E})^2}{v!}$ which would take $O(\frac{v!}{s!(v-s)!})$ time. Since this is not an option, I can roughly estimate the value instead. The energy of any given subset of modulo v with s elements will vary from a minimum of 0, where the subset is a difference cover, to a maximum of $\frac{v}{2} - s + 1$ where differences repeat as much as possible. Whenever a single element is changed, the energy can change by as much as $s - 1$, since the element being changed can be responsible for anywhere from 0 to $s - 1$ differences that are not otherwise covered. The rate of energy change does not depend on the current energy since a single change in one element causes as much change in a cover as it does in the highest energy state. I expect the average value of $\frac{dE}{dt}$ to be the same regardless of whether E was equal to 0 or $\frac{v}{2} - s + 1$. Thus, $\frac{dE}{dt}$ is roughly proportional to s. $\frac{dE}{dt}$ can be estimated by $\frac{-(s-1)}{2}$ when a subset is being mutated randomly. $\sigma(E)$ measures the typical differences between energy values, thus should be roughly proportional to $\frac{dE}{dt}$. Since this is my closest estimate of $\sigma(E)$, a value which can not be easily calculated, it is the one I will use. This gives us a heat capacity of $C = \frac{(s-1)^2}{4T^2}$.

To find the relaxation time, Salamon et. al. (1988) used the model $\frac{dE}{dt} = \frac{-1}{\varepsilon}(E - E^{eq})$ where E^{eq} is the equilibrium energy, which in this case is zero since the equilibrium is a difference cover. This gives us $\varepsilon = \frac{-E}{dE/dt} = \frac{2E}{s-1}$. Substituting into our cooling model, $\frac{dT}{dt} =$

$\frac{-vT}{\varepsilon C} = \frac{-vT*4T^2*(s-1)}{(s-1)^2*2E} = \frac{-2vT^3}{E(s-1)}$. This model gives us a rate of cooling that is proportional to the cube of the current temperature and inversely proportional to the current energy. This implies the temperature is going to decrease much more rapidly than it was with other cooling schedules at high temperature values. It also means that when we are closer to equilibrium, temperature drops faster, keeping the system from moving far from the global minimum state. This model is merely a rough estimate of the thermodynamic cooling equation since I could not calculate the values of the relaxation time and heat capacity and used trial and error to determine the best constant to use for thermodynamic velocity. I expect that there is a better model for this problem out there, but this one should be close enough to get results. Luckily, there is a great deal of leeway afforded to me since modifications to the equation such as changing the exponent of the heat capacity have negligible impact on performance.

Simulated annealing can also be implemented in a way such that, when the order of the subset we are looking for is not known, we can increase and decrease the number of elements with each mutation. Having additional elements in the subset would correspond to an increase in energy. A minimum order for the subset can be defined by the mathematically smallest possible order or the smallest order we believe a solution to be able to have. The advantage of this is that when the algorithm fails to find a cover at a specified minimum length, it can find one larger than that instead. While this method can be useful, I have found it more advantageous to constrain the search space to a specific order and increment the order manually after the algorithm runs for hours without finding a cover. This is largely because I want to be confident that I am not going to find a cover of a specific order before I accept a cover of a larger order as my result. Another reason to avoid this approach is the tremendous increase in search space as each order s in modulo v has a search space of $\binom{v}{s-2}$ elements.

In the code sample below, I have an acceptance probability function that gives us a probability of 1 to transition to a lower energy state and a probability of $e^{-\Delta E/T}$ to move to a higher energy state. To determine the energy of a state, I determine the number of uncovered differences and finish as soon as a state of energy zero is found. The way

temperature is adjusted is dependent on which cooling method is currently being used.

```
public void anneal(){
	temp = MAX_TEMP;
	List<Integer> solution = generateRandomCover();
	List<Integer> solutionP = new ArrayList<Integer>();

	while(true){
		while(temp > MIN_TEMP){
			solutionP = mutate(solution);
			if (acceptanceProb(energy(solution), energy(solutionP), temp) > rand.nextFloat())
				solution = solutionP;
			tempAdjust();
		}
		temp = MAX_TEMP;
	}
}

public double acceptanceProb(int eOld, int eNew, double temp){
	if(eNew <= eOld) return 1.0;
	return Math.pow(Math.E, (-1 * (eNew - eOld)) / temp);
}

public int energy(List<Integer> subset){
	int e = HALF_MOD - countDifsCovered(subset);
	if (e == 0){ // It's a difference cover!
		System.out.println(subset);
		System.exit(0);
	}
	return e;
}
```

4 Results

4.1 Golomb Ruler to Difference Cover

The use of Golomb rulers as starting points in searches for minimum difference covers worked to a lesser extent than Ling predicted but still produced a large amount of results. I was able to find a minimum cover for modulo 113 by brute force searches alone but requiring about a day of searching to do so, thus I began using difference covers at modulo 114 instead. The length 10 optimal ruler, {0, 1, 6, 10, 23, 26, 34, 41, 53, 55} proved to be exceptionally effective, finding minimum difference covers for most modulo groups of order 114 to 140. For certain modulo however, this ruler failed to find a difference cover and I had to gradually try using smaller covers until finding ones that worked. The table below details my results up to modulo 142.

Table 4.1

Difference covers obtained from optimal Golomb rulers.

Modulo	Golomb Ruler	Order	Minimum Cover
114	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 76, 91, 92}
115	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 48, 53, 55, 79, 99}
116	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 80, 91, 113}
117	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 89, 101, 114}
118	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 61, 82, 105}
119	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 81, 83, 69}
120	{0, 1, 4, 9, 15, 22, 32, 34}	8	{0, 1, 4, 9, 15, 22, 32, 34, 45, 58, 74, 82, 109}
121	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 83, 85, 97}
122	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 84, 86, 57}
123	{0, 1, 4, 9, 15, 22, 32, 34}	8	{0, 1, 4, 9, 15, 22, 32, 34, 87, 63, 89, 43, 16}
124	{0, 1, 11, 16, 19, 23, 25}	7	{0, 1, 11, 16, 19, 23, 25, 76, 55, 97, 56, 114, 63}
125	{0, 1, 11, 16, 19, 23, 25}	7	{0, 1, 11, 16, 19, 23, 25, 77, 57, 97, 36, 66, 99}
126	{0, 1, 11, 16, 19, 23, 25}	7	{0, 1, 11, 16, 19, 23, 25, 79, 53, 96, 92, 52, 117}
127	{0, 1, 8, 12, 14, 17}	6	{0, 1, 8, 12, 14, 17, 69, 42, 106, 88, 62, 32, 92}
128	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 90, 91, 92}
129	{0, 1, 5, 12, 20, 30, 44, 57, 66}	9	{0, 1, 5, 12, 20, 30, 44, 57, 66, 60, 108, 91, 106}
130	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 92, 94, 73, 66}
131	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 92, 93, 97}
132	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 99, 72, 118}
133	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 32, 42, 44, 48, 51, 59, 72, 77, 97, 111}
134	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 92, 97, 98, 73}
135	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 97, 99, 85, 67}
136	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 92, 114, 117}
137	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 99, 101, 121, 104}
138	{0, 1, 4, 10, 18, 23, 25}	7	{0, 1, 4, 10, 18, 23, 25, 79, 106, 53, 94, 7, 38, 65}
139	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55}	10	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 74, 82, 101, 103}
140	{0, 1, 4, 9, 15, 22, 32, 34}	8	{0, 1, 4, 9, 15, 22, 32, 34, 42, 58, 71, 94, 96, 109}

141	{0, 1, 4, 10, 18, 23, 25}	7	{0, 1, 4, 10, 18, 23, 25, 89, 61, 101, 34, 69, 49, 47}
142	{0, 1, 4, 9, 15, 22, 32, 34}	8	{0, 1, 4, 9, 15, 22, 32, 34, 86, 113, 77, 93, 46, 116}

The order 10 optimal ruler found 273 distinct covers of length 15 in modulo 143 and continued to find such covers until finding a single length 15 cover in modulo 155. After that, the effectiveness of the order 10 ruler is again lost, and the runtime became too high for me to proceed with brute force searches with smaller covers at that point. Starting with modulo 130, where I started finding order 14 covers, these covers may very well not be minimum but they are the smallest known covers for their modulo. For Modulo 143, I have found covers of order 15 but not 14. I have no means of proving or disproving the existence of smaller covers. At the very least however, these are unlikely to be more than 1 element above the true minimum covers.

Table 4.2

Order 15 difference covers obtained from the order 10 optimal Golomb ruler.

Modulo	Cover
143	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 93, 92, 101, 117, 73}
144	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 106, 108, 94, 81, 67}
145	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 109, 107, 88, 95, 14}
146	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 110, 108, 96, 81, 13}
147	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 97, 111, 109, 67, 75}
148	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 97, 89, 72, 16, 84}
149	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 111, 92, 113, 84, 11}
150	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 112, 76, 104, 83, 141}
151	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 101, 97, 92, 12}
152	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 92, 99, 111, 127}
153	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 115, 92, 117, 91, 79}
154	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 92, 118, 97, 116, 104}
155	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 97, 117, 91, 101, 92}

There were a few cases in which I gave up on certain rulers before moving on to the next smaller ruler. While I did complete brute force searches starting with the order 10 ruler for each of these modulo, modulo 141 happened to find a minimum difference cover quickly the length 7 ruler while it took too long with the length 8 ruler for me to continue the search. It appears based on this evidence that certain covers work well at certain modulo and will work over a scattered list of modulo as opposed to a sequential list. The order 10 ruler for example fails to find a cover for what appears to be a random subset of

modulo groups on the interval where it otherwise works. This could be explained by the way the nature of subtraction changes at each modulo, causing the same ruler to cover a different set of differences for different modulo groups. This way, any given ruler may only be useful for a subset of modulo that appears to be random, although there is likely a discernable pattern dictating where rulers will be part of minimum covers. I have not been able to determine any such pattern.

While compiling a database of difference covers for each modulo group, I observed that for the most part the size of minimum covers increased gradually as the order of the group increased. There were however a few specific exceptions. For every integer n greater than 1, the modulo group of order $n(n-1)+1$ may have covers with n elements that are also difference packings and sets, i.e. no differences will overlap. In each of these cases, the length of these covers is less than the lengths of the covers of the modulo groups adjacent to them. These modulo groups are exceptions to the general trend of minimum difference covers increasing in order as the order of the group increases.

Table 4.3

The overlap of difference covers, sets and packings.

Order	Modulo	Maximum Packing
2	3	{0, 1}
3	7	{0, 1, 3}
4	13	{0, 1, 4, 6}
5	21	{0, 1, 4, 14, 16}
6	31	{0, 1, 4, 10, 12, 17}
7	43	None
8	57	{0, 1, 6, 21, 28, 44, 46, 54}
9	73	{0, 1, 12, 20, 26, 30, 33, 35, 57}
10	91	{0, 1, 5, 7, 27, 35, 44, 67, 77, 80}
11	111	None
12	133	{0, 1, 8, 10, 32, 36, 52, 55, 66, 95, 116, 128}

Looking into difference sets, I found that these particular constructions were already well known. Singer's construction of Singer difference sets prove that for any prime power q, at least one $(q^2+q+1, q+1, 1)$-difference set exists. These are difference sets for modulo q^2+q+1 with $q+1$ elements and each difference repeating once. This accounts for each of these packings and confirms that they will continue to be found at

higher modulo. These can be found much faster than difference covers at other modulo since the search space includes only subsets of the modulo group that have no overlapping differences. There are some orders, like 43 and 111, where even though they are $n(n-1)+1$ for a certain n, they do not have any Singer difference sets. Even though the math suggests they could, these modulo do not have difference covers that are also packings [6].

The only other discrepancy I found in minimum difference cover sizes was at modulo 39, which has length 7 minimum difference covers despite modulo 38 having length 8 minimum covers. This is not a modulo group in which difference covers can also be packings, so the Singer construction does not explain this phenomenon. This anomaly emphasizes the notion that the true smallest minimum difference cover cannot be known with complete certainty until the entire search space is searched. This means for larger modulo where complete searches cannot be carried out, covers that are suspected to be minimum, or at least the smallest that can be found, will have to suffice. More of these anomalies are likely to exist but will require extensive brute force searches to discover. They may also be an explanation for this that I am unaware of, but these covers are not difference packings or sets, so constructions for those do not explain this. If there are other modulo groups with covers like this, I suspect that a construction can be built to find others like it.

Alan Ling believed that for larger modulo, the search for minimum difference covers would be greatly speed up using these rulers. However, I was able to use backtracking searches to show that for the optimal Rulers of order 7 and up, minimum covers for modulo 256 and 512 are not found. I was able to find covers of order 19 for modulo 256, which are 1 element smaller than the cover previously found. I was not able to find covers for 512 that were smaller than the previously found 28 order covers. I believe that there likely exists order 17 or 18 difference covers for modulo 256 and order 25 or 26 difference covers for 512. Golomb rulers were unfortunately not an effective enough strategy to achieve results that effective at modulo this large. For modulo 256, the order 19 covers are still an improvement, however small, upon existing covers. See Appendix B for a list of some of the order 19 covers found for this group.

4.2 Difference Packings with Backtracking

Difference packings are similar enough in nature to difference covers that the same methods can easily be applied. A simple backtracking algorithm that avoids repeated differences can find difference packings much more efficiently that it can covers. Choosing the order of elements to search stochastically appears to work well. I am not sure whether this exists a pattern that would allow certain elements to be valued more highly than others as I can do in difference cover searched. Certain patterns do appear to crop up in difference packings like the sequence {0, 1, 3, 7} which is an order four Golomb ruler. It is not an optimal ruler however as {0, 1, 4, 6} is more efficient. The packings do generally resemble Golomb rulers, as would be expected, but I have not been able to find known optimal Golomb rulers of order greater than four actually appearing within the packings I have found. Based on the packings I have found, I believe that it is not feasible to speed up the search for packings by using Golomb rulers as a starting point. This is because there is a sufficient enough variation in the way differences are distributed in larger packings that the optimal Golomb rulers do not appear within them.

Table 4.4

Difference packings obtained with backtracking.

Modulo	Packing
128	{0, 1, 3, 7, 12, 20, 30, 44, 78, 93}
129	{0, 1, 3, 7, 12, 20, 30, 45, 69, 95}
130	{0, 1, 3, 7, 12, 20, 30, 46, 78, 93}
131	{0, 1, 3, 7, 12, 20, 30, 44, 70, 86}
132	{0, 1, 3, 7, 12, 20, 30, 44, 65, 93}
133	{0, 1, 3, 7, 12, 20, 30, 44, 69, 100}
134	{0, 1, 3, 7, 12, 20, 30, 44, 65, 80}
135	{0, 1, 3, 7, 23, 35, 49, 73, 78, 117, 125}
136	{0, 1, 3, 7, 26, 35, 43, 55, 65, 76, 92}
137	{0, 1, 3, 7, 12, 43, 60, 73, 93, 112, 122}
138	{0, 1, 3, 7, 19, 65, 86, 91, 106, 114, 128}
139	{0, 1, 3, 7, 12, 29, 39, 62, 86, 105, 126}
140	{0, 1, 3, 7, 12, 27, 44, 58, 80, 93, 122}

4.3 Direct Products

Another question I wanted to explore was whether I could learn anything from covers of smaller groups to help me find covers of larger ones. Because the nature of subtraction changes with each order in modulo groups, intuition dictates that this should not be the case. This did not stop me from looking for patterns that might emerge in a large list of covers. This led me to looking into the direct product of modular groups. Since there is an isomorphism between the product of two relatively prime modulo groups and the group whose order is the product of their orders, this means certain modulo groups can be expressed in different ways. For example, difference covers in modulo 66 can be viewed as corresponding difference covers in modulo 6 x modulo 11. By looking through the direct product representation of known covers I hoped to find patterns emerge and find a way to narrow down my search.

Looking through known minimum difference covers for modulo that can be converted to a direct product, patterns did appear to emerge. Every element in the larger modulo would usually appear only once and in the smaller modulo the elements would likewise be very spread out. This pattern gives me a more powerful metric to use in tree searches for specific modulo. For example, for modulo 138 (6 x 23), I can assign a value to nodes in a tree search not just on the number of repeated differences, but on how well spread out the values are among both modulo. This means a subset with three elements whose modulo 6 coordinates are 1, 2 and 3 would be much more interesting than a subset with elements whose modulo 6 coordinates are 5, 5 and 5, since the latter seems unlikely to produce good results. This is relevant since both subsets in question could have the same number of repeated differences in modulo 138 thus being valued equally by the usual metric.

To demonstrate the existence of this pattern, I analyzed the complete list of minimum covers in modulo 66. I had obtained this list from a brute force search that lasted roughly an hour. .This modulo was chosen not only for convenience, since it is 6 x 11, but since this modulo group has 153,578 minimum covers, giving me an enormous sample. I am not aware of another modulo group with a similar magnitude of minimum covers though I suspect larger modulo where the order of minimum covers ticks up will have

similarly long lists of covers. I converted the elements of each cover to points in modulo 6 x modulo 11 to verify that the coordinates of the elements in each cover were in fact relatively spread out among their respective modulo.

Out of the minimal covers in modulo 6 x modulo 11, each element of modulo 6 appears at least once in 39,026 of them. 5 elements of modulo 6 appear in another 102,153 of them and the remaining 12,399 covers contain 4 elements in modulo 6. Not a single cover contains only 3 or fewer elements of modulo 6. For modulo 11, the numbers were more spread out but were similarly skewed towards a large representation of the set with 92,442 elements containing 7 elements of the modulo and none containing 3 or fewer elements. These covers only have ten elements so the entirety of modulo 11 could not be represented. This implies that, when searching modulo 66, we could theoretically round off paths that lead to too many modulo 6 coordinates being repeated since three repeats would eliminate the chance of a cover and even two repeats make it relatively unlikely.

If we extend this concept to a larger modulo group, say modulo 6 x modulo 19, we could value nodes in a tree based on the number of repeated differences in modulo 114 as well as whether the modulo 6 and modulo 11 coordinates of the new element are already represented. This would result in a less naïve tree search that could be more effective for finding a single cover for a given order and modulo.

This observation resulted in a theoretically increase in the speed of tree searches for certain modulo, however this observation is only useful for specific modulo groups. Furthermore, it is only useful for larger modulo groups beyond 128 where it is normally difficult to find covers. This method of finding covers however is not nearly as efficient as simulated annealing, as detailed in the next section. Because of this, the use of direct products unfortunately serves no practical purpose unless a better way to utilize them is found. Because this is essentially just a way of rewriting the same problem, this is unlikely. I was hoping that exploring this would lead me somewhere useful and perhaps it did give me some insight into the nature of difference covers, but it regrettably lead me to no direct results.

4.4 Simulated Annealing

My results with simulated annealing were much more promising than I expected and made me more hopeful than I am about Golomb rulers, moving forward into high modulo groups. With linear and exponential cooling I was able to find order 13 minimum difference covers from modulo 114 to 129 fairly quickly, which demonstrates potential with this method since this significantly improves upon greedy tree searches and compares well to the use of Golomb rulers.

Table 4.5

Minimum covers found while testing simulated annealing with linear and exponential cooling schedules.

Cooling Schedule	Modulo	Difference Cover
Exponential	114	{0, 1, 59, 43, 32, 70, 107, 19, 112, 55, 84, 9, 90}
Exponential	115	{0, 1, 95, 33, 39, 67, 56, 25, 83, 76, 93, 85, 80}
Exponential	116	{0, 1, 69, 103, 59, 32, 26, 87, 47, 50, 67, 110, 5}
Exponential	117	{0, 1, 37, 26, 66, 23, 16, 10, 42, 8, 87, 105, 70}
Exponential	118	{0, 1, 109, 80, 21, 64, 61, 69, 97, 95, 18, 86, 14}
Exponential	119	{0, 1, 43, 36, 57, 61, 89, 81, 17, 70, 84, 55, 33}
Exponential	120	{0, 1, 22, 57, 115, 27, 77, 10, 73, 113, 61, 102, 24}
Exponential	121	{0, 1, 12, 51, 45, 103, 90, 49, 73, 78, 87, 66, 113}
Exponential	122	{0, 1, 42, 46, 97, 59, 73, 112, 61, 80, 115, 103, 75}
Exponential	123	{0, 1, 43, 73, 58, 77, 13, 75, 82, 118, 102, 113, 110}
Exponential	124	{0, 1, 80, 62, 32, 12, 103, 36, 26, 70, 65, 28, 19}
Exponential	125	{0, 1, 28, 20, 38, 59, 35, 68, 113, 117, 6, 61, 17}
Exponential	126	{0, 1, 63, 94, 35, 77, 53, 83, 37, 28, 98, 40, 75}
Exponential	127	{0, 1, 101, 6, 125, 17, 79, 21, 40, 77, 70, 115, 42}
Exponential	128	{0, 1, 96, 47, 126, 115, 109, 101, 85, 87, 37, 32, 103}
Exponential	129	{0, 1, 39, 35, 73, 63, 30, 12, 14, 123, 83, 31, 9}
Linear	114	{0, 1, 109, 105, 21, 73, 40, 23, 37, 80, 22, 61, 48}
Linear	115	{0, 1, 73, 16, 83, 33, 32, 54, 24, 15, 35, 28, 79}
Linear	116	{0, 1, 89, 107, 9, 85, 47, 114, 64, 14, 20, 35, 59}
Linear	117	{0, 1, 48, 29, 25, 108, 100, 44, 55, 5, 42, 32, 65}
Linear	118	{0, 1, 97, 56, 9, 50, 100, 40, 93, 52, 29, 14, 73}
Linear	119	{0, 1, 102, 29, 17, 59, 8, 66, 14, 64, 92, 44, 40}
Linear	120	{0, 1, 49, 116, 65, 30, 107, 84, 80, 110, 82, 60, 8}
Linear	121	{0, 1, 6, 85, 99, 33, 51, 21, 41, 109, 60, 4, 116}
Linear	122	{0, 1, 13, 22, 6, 103, 36, 51, 24, 100, 118, 69, 61}
Linear	123	{0, 1, 32, 45, 90, 15, 39, 98, 36, 43, 88, 61, 20}
Linear	124	{0, 1, 46, 49, 79, 50, 71, 32, 86, 55, 114, 112, 52}
Linear	125	{0, 1, 117, 94, 75, 70, 16, 37, 123, 12, 104, 42, 77}
Linear	126	{0, 1, 12, 63, 69, 119, 72, 115, 112, 34, 10, 99, 94}
Linear	127	{0, 1, 92, 55, 6, 63, 25, 106, 13, 53, 124, 107, 24}

Linear	128	{0, 1, 123, 121, 61, 85, 96, 19, 64, 113, 73, 15, 32}
Linear	129	{0, 1, 96, 47, 126, 115, 109, 101, 85, 87, 37, 32, 103}

With both strategies, the amounts of time required to find a cover vary significantly, due to the random nature of the process. Both of these cooling methods are fast, generally taking less than 15 minutes, up until around modulo 124. By the time modulo 129 is reached however, it already takes around an hour or two. The times for both of these cooling schedules are comparable to each other and to Golomb rulers. With both cooling methods, I began with a temperature of 30 and reset the temperature every time it reached 0.001. At larger modulo, I increased the starting temperature to around 50 which appeared to help. I suspect based on my observations that the initial temperature wants to increase with an increase in modulo, if only to allow the system more room to move around a larger search space. With linear cooling, n = 0.01 worked well and with exponential cooling, α = 0.99992 to 0.99994 worked better than other values I tried. Values closer to 0.99 are clearly too fast and result in very high average energy. I believe the value of α should become closer to 1 as the order of the covers increases since the cooling needs to be done slower. Likewise, for linear cooling, n wants to slowly approach 0. Exponential worked much better than linear at higher modulo however, and found me a few possibly minimum covers much farther out when I looked to continue where difference covers had left off.

Table 4.6

Covers found with an exponential cooling schedule at higher order modulo groups.

Modulo	Difference Cover
153	{0, 1, 17, 25, 95, 15, 101, 41, 61, 104, 83, 106, 74, 102, 139}
154	{0, 1, 133, 33, 4, 22, 153, 105, 62, 68, 74, 17, 98, 18, 145}
155	{0, 1, 38, 7, 16, 74, 50, 130, 99, 103, 61, 69, 116, 71, 51}
156	{0, 1, 48, 119, 8, 33, 2, 56, 73, 61, 67, 26, 155, 77, 13}
157	{0, 1, 123, 133, 137, 77, 5, 100, 27, 79, 9, 8, 154, 117, 64}
158	{0, 1, 71, 4, 99, 69, 112, 38, 51, 144, 80, 104, 87, 77, 48}
159	{0, 1, 87, 153, 39, 100, 43, 97, 151, 76, 128, 17, 147, 9, 133}
160	{0, 1, 28, 108, 19, 139, 113, 80, 131, 39, 73, 3, 97, 13, 117}
161	{0, 1, 140, 115, 50, 36, 108, 131, 142, 7, 123, 120, 149, 64, 40}
162	{0, 1, 106, 14, 52, 85, 147, 128, 32, 157, 5, 98, 2, 25, 119, 161}
163	{0, 1, 3, 98, 149, 63, 89, 159, 117, 37, 104, 119, 11, 25, 116, 143}
164	{0, 1, 77, 27, 58, 82, 54, 150, 66, 95, 140, 65, 83, 98, 36, 34}

165	{0, 1, 20, 32, 60, 15, 83, 110, 163, 93, 7, 44, 33, 131, 67, 2}
166	{0, 1, 124, 48, 139, 4, 149, 136, 26, 72, 69, 67, 107, 62, 155, 116}
167	{0, 1, 124, 63, 150, 3, 103, 117, 82, 69, 140, 95, 91, 165, 14, 39}
168	{0, 1, 117, 74, 78, 73, 44, 100, 80, 33, 151, 103, 54, 86, 109, 16}
169	{0, 1, 157, 105, 23, 29, 72, 21, 5, 130, 71, 94, 113, 160, 120, 74}
170	{0, 1, 134, 56, 165, 79, 18, 68, 81, 95, 143, 35, 52, 42, 3, 153}
171	{0, 1, 37, 116, 48, 25, 57, 98, 81, 168, 12, 86, 106, 132, 79, 100}
172	{0, 1, 157, 49, 164, 78, 55, 88, 30, 28, 89, 120, 73, 4, 140, 14}
173	{0, 1, 30, 107, 12, 79, 120, 137, 104, 110, 131, 88, 135, 74, 114, 165}

Even though these covers cannot be proven to be minimum, based on every minimum cover that is known and the rate at which minimum covers increase, these are not only almost certainly minimum, I am even surprised that it was possible to get as high as modulo 161 with 15 elements. I proceeded to find order 16 covers for modulo 162 to 173. The runtime increased significantly each time the length of the cover increased. This can be explained by the size of the search space being $\binom{v-3}{s-2}$. Increasing the value of v and s, with $s-2<\frac{v-3}{2}$, and v increasing more quickly than s, will scale the search space upwards dramatically. Finding order 15 covers took a few hours each but order 16 covers took around twenty four hours with the same machine with exponential cooling. Continuing to use simulated annealing for larger modulo groups will require a much more efficient algorithm or significantly more processing power than I have had.

Constant thermodynamic speed took a while for me to perfect since there were constants that I was merely guessing the appropriate values of. Eventually I got the algorithm performing well with a maximum temperature of 40 and a minimum temperature of 0.05. These numbers were arrived at partially from trial and error, as well as estimating the minimum temperature at which the probability of moving to another state is sufficiently low and a temperature at which the system effectively moves in an entirely stochastic way. I found that starting temperatures $T>100$ caused problems where the temperature would accelerate downwards too quickly. Additionally, a minimum temperature $T<0.01$ can result in too much time being spent trapped in a single state while the temperature asymptotically approaches its minimum value. The cooling schedule I used ended up being defined by $\frac{dT}{dt}=\frac{-0.002T^3}{E(s-1)}$ where the constant

velocity was determined with trial and error, keeping track of average energy values with each constant I tried. As soon as I started finding results, I had no reason to change this constant and found every cover in the table below with the same constant. This cooling schedule found many different covers for the same modulo and cover lengths as had been found by previously used cooling schedules as well as covers I had not been able to find using any other method.

Table 4.7

Difference covers found with constant thermodynamic speed. In bold, covers that I had not been able to find previously. Some of these covers are for modulo in which I had found covers using previous methods but ones that were an order larger.

Modulo	Difference Cover
129	{0, 1, 26, 45, 42, 13, 101, 112, 64, 40, 95, 122, 49}
130	{0, 1, 35, 7, 12, 82, 48, 20, 72, 3, 87, 108, 34, 18}
131	{0, 1, 78, 112, 50, 52, 116, 5, 34, 123, 109, 10, 40, 95}
132	{0, 1, 16, 113, 43, 78, 40, 100, 108, 37, 26, 49, 44, 47}
134	{0, 1, 122, 49, 116, 29, 65, 32, 95, 25, 42, 121, 100, 40}
135	{0, 1, 52, 108, 48, 13, 128, 117, 14, 72, 46, 23, 91, 133}
136	{0, 1, 81, 79, 112, 118, 77, 68, 51, 43, 130, 48, 91, 16}
137	{0, 1, 80, 119, 109, 95, 36, 112, 27, 103, 124, 47, 134, 38}
138	{0, 1, 78, 57, 91, 23, 99, 85, 19, 42, 94, 74, 109, 111}
139	{0, 1, 23, 7, 83, 95, 136, 92, 19, 43, 78, 131, 108, 129}
140	{0, 1, 87, 115, 130, 57, 128, 25, 60, 17, 82, 134, 39, 121}
141	{0, 1, 32, 95, 75, 108, 22, 113, 26, 24, 59, 15, 101, 56}
142	{0, 1, 107, 79, 18, 104, 32, 28, 42, 113, 97, 20, 53, 5}
143	**{0, 1, 93, 31, 67, 85, 22, 141, 3, 44, 10, 15, 42, 38}**
144	{0, 1, 84, 32, 111, 7, 121, 57, 56, 86, 10, 100, 20, 82, 15}
145	{0, 1, 102, 82, 128, 120, 45, 126, 66, 34, 31, 40, 30, 59, 18}
146	{0, 1, 61, 131, 2, 60, 54, 6, 28, 42, 73, 137, 84, 81, 35}
147	{0, 1, 131, 24, 30, 96, 18, 28, 97, 70, 147, 95, 136, 75, 88}
148	{0, 1, 110, 93, 142, 63, 48, 51, 129, 3, 124, 77, 114, 40, 44}
149	{0, 1, 125, 72, 43, 132, 95, 101, 55, 40, 123, 35, 88, 76, 139}
150	{0, 1, 131, 24, 30, 96, 18, 28, 97, 70, 147, 95, 136, 75, 88}
151	{0, 1, 57, 37, 83, 110, 67, 116, 71, 48, 22, 35, 17, 148, 144}
152	{0, 1, 138, 75, 49, 21, 13, 3, 97, 115, 148, 30, 91, 32, 113}
153	{0, 1, 88, 115, 94, 101, 82, 34, 9, 132, 32, 72, 37, 112, 86}
154	{0, 1, 23, 47, 136, 148, 85, 112, 81, 44, 92, 77, 138, 33, 72}
155	{0, 1, 151, 75, 70, 143, 91, 21, 17, 3, 137, 125, 128, 59, 115}
156	{0, 1, 3, 91, 73, 155, 122, 34, 121, 44, 60, 66, 20, 148, 15}
157	{0, 1, 13, 109, 78, 40, 23, 42, 123, 46, 98, 86, 152, 16, 69}

158	{0, 1, 24, 135, 18, 56, 61, 72, 123, 22, 149, 138, 136, 86, 130}
159	{0, 1, 80, 42, 17, 84, 110, 144, 3, 123, 10, 115, 33, 132, 104}
160	{0, 1, 158, 58, 41, 88, 134, 81, 121, 146, 152, 140, 156, 30, 104}
161	{0, 1, 44, 89, 159, 155, 132, 84, 52, 114, 20, 148, 99, 61, 120}
162	**{0, 1, 151, 53, 108, 48, 16, 127, 66, 56, 129, 133, 94, 33, 120}**
163	**{0, 1, 143, 41, 156, 158, 114, 28, 96, 16, 153, 100, 134, 65, 82}**
164	{0, 1, 14, 11, 114, 124, 97, 117, 71, 140, 4, 136, 91, 5, 35, 99}
165	{0, 1, 161, 44, 61, 17, 131, 106, 120, 133, 91, 125, 54, 23, 35, 41}
166	{0, 1, 97, 52, 139, 144, 69, 83, 19, 49, 148, 8, 3, 59, 13, 112}
167	{0, 1, 152, 20, 22, 140, 88, 41, 155, 127, 110, 163, 34, 96, 158, 85}
168	{0, 1, 71, 115, 155, 73, 81, 64, 133, 26, 15, 28, 101, 109, 104, 85}
169	{0, 1, 159, 60, 118, 45, 72, 155, 50, 61, 71, 37, 54, 90, 93, 92}
170	{0, 1, 152, 52, 82, 146, 109, 139, 96, 23, 105, 154, 12, 61, 2, 151}
171	{0, 1, 145, 16, 50, 134, 124, 166, 13, 141, 120, 41, 19, 39, 112, 77}
172	{0, 1, 10, 96, 82, 94, 35, 140, 11, 30, 8, 71, 26, 65, 21, 145}
173	{0, 1, 165, 95, 53, 160, 72, 158, 83, 100, 18, 93, 24, 49, 141, 138}
174	**{0, 1, 114, 153, 129, 23, 101, 30, 32, 110, 12, 15, 148, 67, 116, 164}**
175	**{0, 1, 156, 40, 11, 129, 152, 55, 42, 168, 143, 128, 37, 77, 105, 122}**
176	**{0, 1, 149, 13, 115, 14, 170, 67, 6, 69, 135, 105, 17, 45, 164, 106}**
177	**{0, 1, 116, 27, 91, 123, 163, 128, 17, 129, 152, 8, 19, 31, 70, 28}**
178	**{0, 1, 13, 42, 174, 131, 96, 141, 163, 173, 110, 24, 129, 21, 51, 89}**
179	**{0, 1, 48, 145, 46, 72, 153, 176, 16, 155, 64, 7, 101, 167, 59, 84}**
180	**{0, 1, 91, 120, 6, 117, 105, 92, 140, 162, 179, 37, 170, 33, 104, 161}**
181	**{0, 1, 58, 68, 126, 120, 112, 131, 110, 86, 9, 83, 55, 170, 143, 90}**
182	**{0, 1, 86, 13, 181, 44, 128, 124, 69, 9, 74, 40, 20, 92, 102, 77}**

To my amazement, many of the covers above were found in less than ten seconds, a speed that I did not believe possible at modulo this large. This algorithm failed to find a minimum cover for modulo 133, one of the few groups where the minimum cover is also a packing, but this is a modulo whose covers can be much more easily found with a tree search anyway. To put these results into perspective, each of the above covers were found on the same day after I had spent months finding a similar set of covers. This cooling method even gave me results further than the covers I had previously found for modulo 173. It also managed to find order 15 covers for modulo groups 162 and 163. 162 was previously the first modulo at which I increased the order of covers I was searching for to 16 since I could not find an order 15 cover for the group. Likewise, an order 14 cover for modulo 143 was found. Around modulo 180, covers were still being found in a matter of minutes. At modulo 183 however, no results were found even after several hours

suggesting that this might be the next point where the order of the minimum difference cover increases or perhaps the runtime is simply taking off at this point.

Simulated annealing does not appear to be well suited for finding difference packings. Modulo 133 was the only instance on this range where annealing failed. This can be explained by the decreased density of solutions since packings are more specific in nature. The more differences overlap in a cover for a given modulo v and order s, the larger the number of covers there is expected to be since those repeated differences represent room for error. Without that, packings likely occupy a smaller portion of their search space. I suggest backtracking be used to find these packings instead.

5 Observations and Future Work

It appears that at larger order modulo groups, the magnitude of the search space results in tools like Golomb rulers becoming less effective since there is increasingly more room for minimum covers that would not resemble a Golomb ruler. Because of this, I have little faith in the ability of Golomb rulers to continue to be effective into larger modulo groups. For modulo 256, Golomb rulers were able to find an order 19 length cover but the optimal cover for this modulo should be about 15. For modulo 512, the optimal cover should be about order 24, whereas Golomb rulers only found order 28 covers. I am not convinced that Golomb rulers have the potentially to find covers closer to minimum for groups this large.

Simulated annealing seems promising for going into higher modulo with a robust enough cooling schedule and access to as many processors as possible. Simulated annealing with a constant thermodynamic speed is the best cooling schedule for tackling this problem that I am aware of. A more refined version of the algorithm than mine needs to be implemented. I believe this method will be capable of finding significantly more covers than it found for me. I would not be surprised if near minimum covers up to modulo 256 were mapped out using this idea.

References

[1] Colbourn, C. J., & Ling, A. C. (2000). Quorums from difference covers. *Information Processing Letters*, *75*(1), 9-12.

[2] Haanpää, H. (2004). Minimum sum and difference covers of abelian groups. *Journal of Integer Sequences*, *7*(2), 3.

[3] Arasu, K. T., & Sehgal, S. (2005). Cyclic difference covers. *Austral. J. Combin*, 32, 213-223.

[4] Salamon, P., Nulton, J. D., Harland, J. R., Pedersen, J., Ruppeiner, G., & Liao, L. (1988). Simulated annealing with constant thermodynamic speed. Computer Physics Communications, 49(3), 423-428.

[5] Kärkkäinen, J., Sanders, P., & Burkhardt, S. (2006). Linear work suffix array construction. *Journal of the ACM* (JACM), 53(6), 918-936.

[6] Singer, J. (1938). A theorem in finite projective geometry and some applications to number theory. *Transactions of the American Mathematical Society*, 43(3), 377-385.

[7] Nourani, Y., & Andresen, B. (1998). A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, *31*(41), 8373.

[8] Whiteman, A. L. (1962). A family of difference sets. *Illinois Journal of Mathematics*, *6*(1), 107-121.

[9] Ruskey, F., & Sawada, J. (1999). An efficient algorithm for generating necklaces with fixed density. *SIAM Journal on Computing*, 29(2), 671-684.

[10] Robinson J, Bernstein A (1967). A class of binary recurrent codes with limited error propagation. *IEEE Transactions on Information Theory* **13** (1): 106–113.

[11] Thompson, A. Richard; Moran, James M.; Swenson, George W. (2004). *Interferometry and Synthesis in Radio Astronomy* (Second ed.). Wiley-VCH. p. 142.

[12] Arsac, J. (1955). Transmissions des frequences spatiales dans les systemes recepteurs d'ondes courtes [Transmissions of spatial frequencies in shortwave receiver systems]. *Optica Acta* (in French) **2** (112).

[13] (n.d.) *Optimal Golomb Ruler Table*. Retrieved from http://www.research.ibm.com/people/s/shearer/gropt.html.

[14] Moore, Emily H. & Pollatsek, Harriet S. (2010). *Difference Sets*. Providence, Rhode Island: American Mathematical Society.

[15] Palano, Beatrice & Mereghetti, Carlo. (2005). The complexity of minimum difference cover. *Journal of Discrete Algorithms 4*, 239-254.

6 Appendix A

Below is a list of known list of a known minimum difference cover for modulo groups up to 128 for future reference. These covers were all found by me.

Modulo	Example of Minimum Difference Cover
1	{0}
2	{0, 1}
3	{0, 1}
4	{0, 1, 3}
5	{0, 1, 3}
6	{0, 1, 3}
7	{0, 1, 3}
8	{0, 1, 3, 4}
9	{0, 1, 3, 4}
10	{0, 1, 3, 5}
11	{0, 1, 3, 7}
12	{0, 1, 3, 7}
13	{0, 1, 5, 11}
14	{0, 1, 3, 10, 11}
15	{0, 1, 7, 12, 3}
16	{0, 1, 8, 11, 13}
17	{0, 1, 8, 12, 3}
18	{0, 1, 9, 13, 16}
19	{0, 1, 9, 15, 3}
20	{0, 1, 10, 14, 6, 3}
21	{0, 1, 8, 18, 6}
22	{0, 1, 11, 16, 14, 4}
23	{0, 1, 11, 16, 7, 3}
24	{0, 1, 12, 17, 15, 11}
25	{0, 1, 12, 18, 21, 3}
26	{0, 1, 13, 19, 22, 11}
27	{0, 1, 13, 19, 24, 17}
28	{0, 1, 14, 9, 25, 7}
29	{0, 1, 14, 21, 24, 18, 3}
30	{0, 1, 15, 22, 5, 19, 7}
31	{0, 1, 15, 22, 28, 20}
32	{0, 1, 3, 7, 12, 17, 20}
33	{0, 1, 3, 7, 12, 20, 30}
34	{0, 1, 3, 7, 12, 20, 2}
35	{0, 1, 3, 7, 12, 20, 17}
36	{0, 1, 3, 7, 21, 26, 9}
37	{0, 1, 3, 7, 12, 17, 25}
38	{0, 1, 13, 20, 24, 29, 32, 34}

39	{0, 1, 19, 30, 25, 23, 27}
40	{0, 1, 20, 29, 14, 24, 22, 3}
41	{0, 1, 20, 30, 14, 37, 33, 16}
42	{0, 1, 8, 18, 21, 16, 22, 31}
43	{0, 1, 10, 12, 15, 37, 14, 18}
44	{0, 1, 3, 7, 12, 20, 30, 23}
45	{0, 1, 5, 7, 27, 10, 31, 39}
46	{0, 1, 3, 7, 12, 22, 30, 33}
47	{0, 1, 3, 7, 12, 20, 22, 36}
48	{0, 1, 3, 7, 19, 24, 9, 35}
49	{0, 1, 3, 7, 12, 20, 34, 24}
50	{0, 1, 3, 8, 17, 28, 32, 38}
51	{0, 1, 3, 13, 17, 22, 28, 21}
52	{0, 1, 3, 7, 12, 20, 30, 38, 36}
53	{0, 1, 27, 14, 38, 45, 33, 10, 3}
54	{0, 1, 3, 7, 12, 20, 30, 33, 16}
55	{0, 1, 27, 40, 18, 8, 51, 21, 10}
56	{0, 1, 3, 7, 12, 20, 30, 34, 16}
57	{0, 1, 28, 44, 21, 6, 54, 46}
58	{0, 1, 3, 7, 12, 20, 30, 44, 36}
59	{0, 1, 3, 7, 12, 22, 30, 46, 29}
60	{0, 1, 3, 7, 12, 28, 29, 42, 52}
61	{0, 1, 3, 7, 12, 20, 30, 36, 22}
62	{0, 1, 3, 8, 20, 30, 34, 9, 47}
63	{0, 1, 3, 7, 17, 43, 58, 34, 45}
64	{0, 1, 3, 8, 18, 34, 40, 53, 44}
65	{0, 1, 3, 11, 15, 20, 36, 42, 29}
66	{0, 1, 33, 49, 20, 11, 26, 28, 14, 62}
67	{0, 1, 3, 7, 12, 20, 30, 46, 32, 15}
68	{0, 1, 3, 7, 12, 20, 30, 54, 43, 53}
69	{0, 1, 3, 7, 12, 20, 30, 48, 55, 32}
70	{0, 1, 35, 52, 21, 12, 45, 29, 48, 50}
71	{0, 1, 36, 19, 51, 60, 44, 38, 15, 41}
72	{0, 1, 36, 54, 21, 45, 5, 43, 60, 46}
73	{0, 1, 35, 53, 11, 65, 48, 7, 51}
74	{0, 1, 37, 55, 22, 13, 30, 64, 61, 59}
75	{0, 1, 38, 20, 54, 28, 31, 25, 40, 71}
76	{0, 1, 38, 57, 17, 26, 30, 24, 35, 45}
77	{0, 1, 39, 20, 55, 64, 8, 5, 37, 11}
78	{0, 1, 39, 58, 15, 67, 71, 42, 48, 76}
79	{0, 1, 40, 17, 65, 7, 35, 43, 67, 76}
80	{0, 1, 40, 60, 23, 12, 50, 16, 15, 21, 34}
81	{0, 1, 41, 21, 59, 70, 31, 46, 37, 48, 56}
82	{0, 1, 41, 61, 24, 13, 51, 76, 8, 15, 12}

83	{0, 1, 42, 22, 60, 71, 32, 75, 78, 57, 59}
84	{0, 1, 42, 63, 24, 13, 33, 70, 68, 60, 66}
85	{0, 1, 43, 22, 62, 73, 8, 47, 57, 40, 49}
86	{0, 1, 43, 64, 25, 13, 53, 79, 70, 15, 82}
87	{0, 1, 44, 23, 63, 12, 53, 60, 68, 82, 86}
88	{0, 1, 44, 66, 25, 12, 58, 51, 60, 61, 40}
89	{0, 1, 45, 23, 65, 76, 35, 19, 28, 68, 74}
90	{0, 1, 45, 67, 26, 12, 84, 37, 28, 7, 87}
91	{0, 1, 44, 67, 27, 80, 35, 7, 77, 5}
92	{0, 1, 46, 69, 26, 58, 10, 18, 39, 41, 4}
93	{0, 1, 47, 26, 74, 59, 9, 63, 87, 45, 4}
94	{0, 1, 47, 70, 27, 14, 59, 77, 55, 61, 56, 37}
95	{0, 1, 48, 25, 69, 82, 36, 16, 30, 38, 29, 79}
96	{0, 1, 48, 72, 27, 14, 60, 80, 57, 79, 68, 62}
97	{0, 1, 49, 25, 71, 84, 37, 56, 89, 45, 68, 39}
98	{0, 1, 49, 73, 28, 15, 61, 90, 31, 51, 7, 94}
99	{0, 1, 50, 26, 72, 85, 38, 9, 92, 81, 33, 3}
100	{0, 1, 50, 75, 28, 15, 39, 84, 80, 57, 47, 45}
101	{0, 1, 50, 75, 29, 16, 64, 94, 70, 84, 72, 33}
102	{0, 1, 51, 76, 29, 16, 37, 69, 57, 60, 59, 5}
103	{0, 1, 51, 76, 30, 16, 64, 95, 58, 54, 87, 56}
104	{0, 1, 52, 77, 30, 16, 42, 97, 86, 80, 47, 84}
105	{0, 1, 52, 78, 30, 16, 97, 72, 13, 11, 48, 23}
106	{0, 1, 52, 78, 29, 16, 66, 74, 63, 87, 6, 105}
107	{0, 1, 53, 79, 31, 18, 68, 45, 21, 12, 43, 5}
108	{0, 1, 54, 80, 31, 15, 91, 99, 87, 29, 96, 35}
109	{0, 1, 54, 81, 31, 14, 63, 11, 48, 36, 52, 55}
110	{0, 1, 55, 82, 24, 35, 14, 85, 18, 23, 16, 88}
111	{0, 1, 55, 82, 24, 96, 33, 61, 101, 99, 8, 59}
112	{0, 1, 56, 82, 33, 20, 68, 72, 54, 71, 47, 76}
113	{0, 1, 56, 86, 20, 69, 51, 54, 40, 44, 92, 77}
114	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 76, 92, 91}
115	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 79, 99, 48}
116	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 80, 91, 113}
117	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 101, 114, 89}
118	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 82, 61, 105}
119	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 83, 81, 69}
120	{0, 1, 4, 9, 15, 22, 32, 34, 74, 58, 82, 45, 109}
121	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 85, 83, 97}
122	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 84, 86, 57}
123	{0, 1, 4, 9, 15, 22, 32, 34, 87, 63, 89, 43, 16}
124	{0, 1, 11, 16, 19, 23, 25, 76, 55, 97, 56, 114, 63}
125	{0, 1, 11, 16, 19, 23, 25, 77, 57, 97, 36, 66, 99}
126	{0, 1, 11, 16, 19, 23, 25, 79, 53, 96, 92, 52, 117}

127	{0, 1, 8, 12, 14, 17, 69, 42, 106, 88, 62, 32, 92}
128	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 90, 91, 92}

These are examples of the smallest difference covers I found for modulo beyond 128.

These were all found using either a Golomb ruler or simulated annealing.

Modulo	Example of Smallest Known Cover
129	{0, 1, 5, 12, 20, 30, 44, 57, 66, 60, 108, 91, 106}
130	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 92, 94, 73, 66}
131	{0, 1, 78, 112, 50, 52, 116, 5, 34, 123, 109, 10, 40, 95}
132	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 99, 72, 118}
133	{0, 1, 32, 42, 44, 48, 51, 59, 72, 77, 97, 111}
134	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 92, 97, 98, 73}
135	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 97, 99, 85, 67}
136	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 92, 114, 117}
137	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 99, 101, 121, 104}
138	{0, 1, 4, 10, 18, 23, 25, 79, 106, 53, 94, 7, 38, 65}
139	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 74, 82, 101, 103}
140	{0, 1, 4, 9, 15, 22, 32, 34, 42, 58, 71, 94, 96, 109}
141	{0, 1, 4, 10, 18, 23, 25, 89, 61, 101, 34, 69, 49, 47}
142	{0, 1, 4, 9, 15, 22, 32, 34, 86, 113, 77, 93, 46, 116}
143	{0, 1, 93, 31, 67, 85, 22, 141, 3, 44, 10, 15, 42, 38}
144	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 106, 108, 94, 81, 67}
145	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 109, 107, 88, 95, 14}
146	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 110, 108, 96, 81, 13}
147	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 97, 111, 109, 67, 75}
148	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 97, 89, 72, 16, 84}
149	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 111, 92, 113, 84, 11}
150	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 112, 76, 104, 83, 141}
151	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 101, 97, 92, 12}
152	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 92, 99, 111, 127}
153	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 115, 92, 117, 91, 79}
154	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 92, 118, 97, 116, 104}
155	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 97, 117, 91, 101, 92}
156	{0, 1, 5, 12, 25, 27, 35, 41, 44, 105, 126, 89, 110, 138, 50}
157	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 91, 121, 94, 135, 84}
158	{0, 1, 6, 10, 23, 26, 34, 41, 53, 55, 120, 122, 108, 101, 30}
159	{0, 1, 4, 9, 15, 22, 32, 34, 99, 72, 119, 46, 135, 81, 52}
160	{0, 1, 28, 108, 19, 139, 113, 80, 131, 39, 73, 3, 97, 13, 117}
161	{0, 1, 140, 115, 50, 36, 108, 131, 142, 7, 123, 120, 149, 64, 40}
162	{0, 1, 151, 53, 108, 48, 16, 127, 66, 56, 129, 133, 94, 33, 120}
163	{0, 1, 143, 41, 156, 158, 114, 28, 96, 16, 153, 100, 134, 65, 82}
164	{0, 1, 77, 27, 58, 82, 54, 150, 66, 95, 140, 65, 83, 98, 36, 34}
165	{0, 1, 20, 32, 60, 15, 83, 110, 163, 93, 7, 44, 33, 131, 67, 2}

166	{0, 1, 124, 48, 139, 4, 149, 136, 26, 72, 69, 67, 107, 62, 155, 116}
167	{0, 1, 124, 63, 150, 3, 103, 117, 82, 69, 140, 95, 91, 165, 14, 39}
168	{0, 1, 117, 74, 78, 73, 44, 100, 80, 33, 151, 103, 54, 86, 109, 16}
169	{0, 1, 157, 105, 23, 29, 72, 21, 5, 130, 71, 94, 113, 160, 120, 74}
170	{0, 1, 134, 56, 165, 79, 18, 68, 81, 95, 143, 35, 52, 42, 3, 153}
171	{0, 1, 37, 116, 48, 25, 57, 98, 81, 168, 12, 86, 106, 132, 79, 100}
172	{0, 1, 157, 49, 164, 78, 55, 88, 30, 28, 89, 120, 73, 4, 140, 14}
173	{0, 1, 30, 107, 12, 79, 120, 137, 104, 110, 131, 88, 135, 74, 114, 165}
174	{0, 1, 114, 153, 129, 23, 101, 30, 32, 110, 12, 15, 148, 67, 116, 164}
175	{0, 1, 156, 40, 11, 129, 152, 55, 42, 168, 143, 128, 37, 77, 105, 122}
176	{0, 1, 149, 13, 115, 14, 170, 67, 6, 69, 135, 105, 17, 45, 164, 106}
177	{0, 1, 116, 27, 91, 123, 163, 128, 17, 129, 152, 8, 19, 31, 70, 28}
178	{0, 1, 13, 42, 174, 131, 96, 141, 163, 173, 110, 24, 129, 21, 51, 89}
179	{0, 1, 48, 145, 46, 72, 153, 176, 16, 155, 64, 7, 101, 167, 59, 84}
180	{0, 1, 91, 120, 6, 117, 105, 92, 140, 162, 179, 37, 170, 33, 104, 161}
181	{0, 1, 58, 68, 126, 120, 112, 131, 110, 86, 9, 83, 55, 170, 143, 90}
182	{0, 1, 86, 13, 181, 44, 128, 124, 69, 9, 74, 40, 20, 92, 102, 77}

7 Appendix B

Some of the smallest known covers for modulo 256 and 512.

Modulo	Ruler Used	Cover
256	{0, 4, 6, 20, 35, 52, 59, 77, 78, 86, 89, 99, 122, 127}	{0, 4, 6, 20, 35, 52, 59, 77, 78, 86, 89, 99, 122, 127, 195, 200, 183, 189, 187}
512	{0, 1, 6, 25, 32, 72, 100, 108, 120, 130, 153, 169, 187, 190, 204, 231, 233, 242, 246}	{0, 1, 6, 25, 32, 72, 100, 108, 120, 130, 153, 169, 187, 190, 204, 231, 233, 242, 246, 296, 427, 364, 382, 373, 387, 413, 381, 467}