Modulo Difference Cover Bounds

Jacob Charboneau, David Gregory, and Dr. Ankur Gupta

Department of Mathematics at Butler University

Definitions

Let $P = \{0, 1, 2, \ldots, p-1\}$.
We call $S \in P$ a **modulo difference cover** when $\forall n \in P, \exists s_i, s_j \in S$ such that $s_i - s_j = n$ mod p.
We call S **minimum** when there exists no other cover T of the same P such that $|T| < |S|$.
We call S **minimal** when $\forall s_i \in S, S - \{s_i\}$ is not a cover of the same P.

Example

Let $P = \{0, 1, 2, 3, 4, 5, 6\}$.
Is $S = \{0, 1, 3\}$ a cover of P?
If the cover is nonempty, the element 0 is covered.
$0 - 1 = 6$ mod 7 and $1 - 0 = 1$ mod 7. This means that we cover $\{0, 1, 6\}$ with 0 and 1.
$0 - 3 = 4$ mod 7, $3 - 0 = 3$ mod 7, $3 - 1 = 2$ mod 7, and $1 - 3 = 5$ mod 7. These are the four remaining integers that needed covered in P.
Thus, $S = \{0, 1, 3\}$ covers our P.

Motivations

You can attempt to find a minimum cover with brute force in exponential time. This is not ideal, as nothing that runs in exponential time tends to be practical. Thus we want to make an algorithm that will make a cover that is approximately minimal in polynomial time. It is not difficult to, through the use of combinatorics and calculus, find a simple method to make a cover for a given P in $O(\sqrt{p})$ time where $|S| = 2\sqrt{p}$. This method is our upper bound in terms of size. Our goal is to make an algorithm that runs in polynomial time that yields a smaller cover.

Greedy Algorithm

```
int [] Greedy(int p)
  P = {0, 1, ..., p-1}
  Pcov = {1, 1, 0, 0, ..., 0, 1}
  S = {0, 1}
  x = 2; best = 0; bestdiff = 0; diff=0;
  while S is not a Cover of P do {
    while x<p do {
      if x ∉ S
       for i from 0 to |S| do {
         if Pcov[x-S[i] mod p]==0
           diff=diff+1;
         if Pcov[S[i]-x mod p]==0
           diff=diff+1;
       if diff>bestdiff
         bestdiff=diff; best=x; diff=0;}
      x=x+1;}
    S = S+{best};
    x=2; best=0; bestdiff = 0;}
  return S;
```

Chinese Remainder Theorem

Let $n_1, \ldots, n_i$ be coprime integers greater than 1.
then there exists uniquely one integer $x \in \mathbb{Z}_N$ such that $x = a_1$ mod $n_1, x = a_2$ mod $n_2, \ldots, x = a_i$ mod n_i, where $N = \prod_1^i n_k$.

How is it used?

if $p = q * r$ where q, r are coprime, then $S = S_q \times S_r$ is a cover of $P = \mathbb{Z}_p$ and $S_q \in \mathbb{Z}_q, S_r \in \mathbb{Z}_r$ are covers for their parent sets.

Proof. Let us assume that S does not cover $\mathbb{Z}_p$. So there is some element $z \in \mathbb{Z}_p$ that is not covered by S.
This means there exists some (x, y) such that $x \in \mathbb{Z}_q, y \in \mathbb{Z}_r$ that is congruent to $z \in \mathbb{Z}_p$ under the Chinese Remainder Theorem.
But because S_q covers $\mathbb{Z}_q$ and S_r covers $\mathbb{Z}_r$, $\exists q_i, q_j, r_k, r_l$ such that $(x, y) = (q_i - q_j, r_k - r_l) = (q_i, r_k) - (q_j, r_l)$, where $(q_i, r_k), (q_j, r_l) \in S$. Thus $(x, y) \in S$ and S is a cover of $\mathbb{Z}_p$ $\square$

Chinese Remainder Algorithm

In the following pseudocode, `getCover(int n)` retrieves the recorded cover for $|P| = n$ from a list, and `findBest(int [] factors)` returns two coprime integers m, n where $p = mn$ and m and n yield the smallest possible $|S_m||S_n|$ where S_m, S_n are retrieved from a list.
$T(n) \in O(1)$ for getCover(int n)
$T(n) \in O(2^k)$ for findBest(int [] factors), where k is the number of primes that p factors into.

```
int [] CRAlg(int p)   int k = 0;   factors[] = sieve(p);
 Best[2] = findBest(factors);
 S1 = getCover(Best[0]);S2 = getCover(Best[1]); S = new
int[Best[0]*Best[1]];
 for(i=0; i<S1.length; i++)
  for(j=0; j<S2.length; j++)
   S[k]=CRT(S1[i], S2[j], Best[0], Best[1]);k++;
 return S;

int CRT(int x, int y, int m, int n)
 while(x!=y)
  while(x<m*n)
   x=x+m;
   if(x==y)
    break;
  if(x!=y)
   x=x%m; y=y+n;
 return x;
```

Algorithm Data

P size	Greedy Algorithm
10000	161
20000	240
30000	300
40000	352
50000	397
60000	441
70000	483
80000	517
90000	553
100000	586

P size	CRA Cover Size	Greedy Cover Size
6	4	3
21	6	6
35	9	8
50	12	9
74	14	12
75	12	11
92	18	12
100	18	13

The Chinese Remainder Algorithm fails to do better than the Greedy Algorithm in terms of size. Even for small cases we find that the Greedy Algorithm outperforms the Chinese Remainder Algorithm. The reason for this is not because of the Chinese Remainder Algorithm's run time. It has a runtime of $O(p\sqrt{p})$, where the Greedy Algorithm has a runtime of $O(p^2)$. The Chinese Remainder Algorithm is tested with yields from minimum covers, and we only have minimum covers generated for $p \leq 123$. If applied with covers that are not minimum, the $|S|$ that results will be larger. Given that Greedy outperforms the Chinese Remainder Algorithm in flexibility and accuracy, it is superior to this version of the Chinese Remainder Algorithm. The data from both algorithms shows that up to their largest p tested, $|S| < 2\sqrt{p}$.

Remarks

The Chinese Remainder Algorithm is quite interesting as there exist some covers that cannot be built as minimal covers. This means that a possible revision to the algorithm could be to add a checker to the algorithm to pull out unnecessary elements and compare the sizes of the built covers to that of the unchanged Chinese Remainder Algorithm and the Greedy Algorithm. This is more than likely the avenue that will be pursued in future research.