Algorithms for Constructing Generalized Skolem-type Sequences

Honours Thesis
David Churchill
Dept. of Computer Science
April 29, 2005

Contents

1 Abstract

We define a Skolem sequence of order n as a sequence $S = (s_1, s_2, ..., s_{2n})$ *of* $2n$ *elements taken from* $\{1, 2, ..., n\}$ *such that every* $s_i \in \{1, ..., n\}$ *appears exactly twice in* S *and for* $j > i$*, when* $s_i = s_j = k \in \{1, ..., n\}$ *then* $j - i = k$*. These sequences play a vital role in the construction of Steiner Triple systems, which are used for creating interference-resistant missile guidance systems. Due to the exponential number of such sequences, and the exponential time complexity required to make them, efficient algorithms are needed if they are to be produced. We devise two such algorithms for the construction of these sequences, and analyze them in terms of speed and feasability. First, we present an exhaustive algorithm to enumerate all possible generalized Skolem-type Sequences of given parameters, as well as a genetic algorithm for the construction of larger more difficult sequences. We also give some results in the form of higher order sequences we have constructed.*

2 Introduction

Thoralf Skolem in 1957 [5] studied various types of combinatorial designs, the most notable of his results came in the form of what are now known as Skolem sequences. The motivation for these sequences came from the area of balanced incomplete block designs, in particular, Steiner Triple Systems. These systems are used for, (amongst other things), interference resistant message code for missile guidance systems [2]. Let us define these terms as follows:

Definition 2.1 *A balanced incomplete block design BIBD(v, k, λ) is a collection of k-subsets (or blocks) from a set V, where $|V| = v$, $k < v$, where each pair of elements from V occur in exactly λ of the blocks.*

Theorem 2.1 *For a BIBD(v,k,λ) with b blocks the following two conditions hold:*

$$bk = vr \tag{1}$$
$$r(k-1) = \lambda(v-1) \tag{2}$$

Definition 2.2 *A Steiner Triple System of order v denoted $STS(v)$ is a pair (B, V) where V is a set of v elements, and B is a family of 3-subsets from V such that every pair in V is in exactly one of these subsets. This corresponds to a BIBD(v,3,1).*

Example 2.1 STS(7)

$$V = \{1, 2, 3, 4, 5, 6, 7\}$$
$$B = \{\{1, 2, 4\}, \{2, 3, 5\}, \{3, 4, 6\}, \{4, 5, 7\}, \{5, 6, 1\}, \{6, 7, 2\}, \{7, 1, 3\}\}$$

Theorem 2.2 *Steiner triple systems of order v exist for $v = 1, 3$(mod 6).*

These two cases of $v = 1, 3(\text{mod } 6)$ are split into two further sub-cases with respect to values (mod 4). We will show for each of these four cases the sequences that are used to construct them, we will prove their existence and give their respective constructions to Steiner triple systems.

2.1 Skolem Sequences

Definition 2.3 *A sequence $S = (s_1, s_2, ..., s_{2n})$ of 2n elements taken from $\{1, 2, ..., n\}$ is said to be a Skolem sequence of order n if:*

1. *every $k \in \{1, ..., n\}$ appears exactly twice in S, and*
2. *if $s_i = s_j = k$ for $j > i$, then $j - i = k$.*

Example 2.2 A Skolem sequence of order 8

$$S = (4, 1, 1, 5, 4, 7, 8, 3, 5, 6, 3, 2, 7, 2, 8, 6)$$

Theorem 2.3 *A Skolem sequence of order n only exists if $n = 0, 1(mod 4)$.*

Proof Consider a Skolem sequence S. Consider pairs of the form (a_i, b_i) where $i \in \{1, 2, ..., n\}$, and a_i, b_i represent the indices of S where i occurs. Realize that (a_i, b_i) must range over every $j \in \{1, ..., 2n\}$, since they represent the $2n$ disjoint indices of a sequence of length $2n$. Then we know

$$\sum_{i=1}^{n} a_i + b_i = 1 + 2 + ... + 2n \tag{3}$$

$$= \frac{2n(2n+1)}{2}. \tag{4}$$

Now consider the sum of all unique $i \in S$. We know that this can be represented as $(b_i - a_i)$, where (a_i, b_i) are the indices of their occurences in S. By the definition of a Skolem sequence, we know this to be the sum of all $i \in \{1, ..., n\}$, so this implies:

$$\sum_{i=1}^{n} b_i - a_i = 1 + 2 + ... + n \tag{5}$$

$$= \frac{n(n+1)}{2}. \tag{6}$$

Adding (4) and (6) we obtain:

$$\sum_{i=1}^{n} 2b_i = \frac{5n^2 + 3n}{2}, \tag{7}$$

so,

$$\sum_{i=1}^{n} b_i = \frac{5n^2 + 3n}{4}. \tag{8}$$

Since each b_i is an index of the sequence, this sum must always be an integer. The only values for n which satisfy this constraint occur when $n = 0, 1 (\text{mod } 4)$, so we are done.

Skolem used these sequences of order n to generate Steiner Triple Systems of order $6n+1$ using the following construction:

1. For each $s_i = s_j \in S$, form pairs form (i, j).
2. Transform each pair (i, j) into triples $(l, i + n, j + n)$ where $l = s_i = s_j$.
3. Transform each triple $(l, i + n, j + n)$ into sets $\{0, l, j + n\}$.
4. Add 1 to each triple mod $(6n + 1)$ to form the STS$(6n + 1)$.

Example 2.3 We construct an STS(25). Construct a Skolem sequence of order 4, for this example we use $S = (1, 1, 4, 2, 3, 2, 4, 3)$. From step 1 we obtain the pairs (1,2),(4,6),(5,8),(3,7). We then convert these to triples as described in step 2 to obtain (1,4,5),(2,8,10),(3,9,12),(4,7,11). Using the transformation in step 3 we construct the sets {0,1,5},{0,2,10},{0,3,12},{0,4,11}. We then add mod(25) to each of these blocks to obtain a Steiner triple system of size $4 * 25 = 100$.

Since Skolem Sequences exist only for $n = 0, 1 (\text{mod } 4)$ then, as we said before, we must define a variation of these sequences in order to construct an STS$(6n + 1)$ for the cases of $n = 2, 3 (\text{mod } 4)$. These are obtained using what is known as a Hooked Skolem Sequence. We can define a hook as 0 in the sequence. In some areas of the literature a hook is also denotred by a *.

2.2 Hooked Skolem Sequences

Definition 2.4 *A sequence $S = (s_1, s_2, ..., s_{2n-1}, 0, s_{2n+1})$ of $2n + 1$ elements taken from $\{1, 2, ..., n\}$ is said to be a Hooked Skolem sequence of order n if:*

1. *every $k \in \{1, ..., n\}$ appears exactly twice in S,*
2. *if $s_i = s_j = k$ for $j > i$ then $j - i = k$, and*
3. *a hook is placed in position $2n$.*

Example 2.4 Hooked Skolem sequence of order 6

$$S = (5, 6, 1, 1, 4, 5, 3, 6, 4, 3, 2, 0, 2)$$

We follow the same construction used for Skolem sequences for the case of hooked Skolem sequences in order to form Steiner triple systems of order $6n + 1$ for $n = 2, 3 (\text{mod } 4)$. To cover the cases of STS$(6n + 3)$ we must define what is known as a Rosa Sequence.

2.3 Rosa Sequences

Definition 2.5 *A sequence $S = (s_1, ..., s_n, 0, s_{n+2}, ..., s_{2n+1})$ of $2n + 1$ elements taken from $\{1, 2, ..., n\}$ is said to be a Rosa sequence of order n if:*

1. *every $s_i \in \{1, ..., n\}$ appears exactly twice in S,*
2. *if $s_i = s_j = k$ for $j > i$ then $j - i = k$, and*
3. *a hook is placed in position $n + 1$.*

Example 2.5 Rosa sequence of order 6

$$S = (3, 1, 1, 3, 6, 7, 5, 8, 0, 4, 6, 5, 7, 4, 2, 8, 2)$$

This type of sequence produces a solution for the Heffter problem of the second kind, generating a Steiner Triple System of order $6n + 3$. The first four steps of the construction are similar to the others, and are as follows:

1. For each $s_i = s_j \in S$, form pairs form (i, j).
2. Transform each pair (i, j) into triples $(l, i + n, j + n)$ where $l = s_i = s_j$.
3. Transform each triple $(l, i + n, j + n)$ into sets $\{0, l, j + n\}$.

4. Add 1 to each triple (mod $6n+3$).

Example 2.6 We use the example of $n = 3$ to show that there are two final steps. We use the sequence $S = (1, 1, 3, 4, *, 3, 2, 4, 2)$ to generate the triples (0,1,5), (0,2,19), and (0,3,9) by the method shown above. If we then add to all of these triples (mod $6n+3$) we see that we form 63 triples. We know however that since this is a BIBD, by (1) and (2): $b = \frac{v(v-1)}{k(k-1)}$, which for this case means $b = 70$. To form the missing triples, we must perform one final step. We see from our triples that the number 7 is missing. We need to add the triple (0,7,14) and peform the addition (mod 21) to get the remaining 7 triples. So the final step in the construction is:

5. Add the triple of the form $\{0, 2n+1, 4n+2\}$.

6. Add 1 to to this triple (mod $6n+3$) to form the the $STS(6n+3)$.

2.4 Hooked Rosa Sequences

Definition 2.6 *A sequence $S = (s_1, ..., s_n, 0, s_{n+2}, ..., s_{2n}, 0, s_{2n+2})$ of $2n+2$ elements taken from $\{1, 2, ..., n\}$ is said to be a Rosa sequence of order n if:*

1. *every $s_i \in \{1, ..., n\}$ appears exactly twice in S,*

2. *if $s_i = s_j = k$ for $j > i$ then $j - i = k$, and*

3. *hooks are placed in positions $n+1$ and $2n+1$.*

Example 2.7 Hooked Rosa sequence of order 6

$$S = (5, 6, 1, 1, 4, 5, 0, 6, 4, 2, 3, 2, 0, 3)$$

To construct an STS($6n+3$) from a hooked Rosa sequence of order n, simply follow the same construction as a normal Rosa sequence.

3 Generalized Skolem-type Sequences

In our study of Skolem sequences, many questions are posed as to what other types of sequences can be constructed using the basic Skolem constraints. We have seen two variants of these types of sequences, namely, hooked Skolem and Rosa variants. Let us now see which other parameters must be considered in order to fully generalized the notion of a Skolem-type sequence.

3.1 Hooks

The introduction of hooks was needed in order to construct hooked, and Rosa sequences for certain orders. In these cases one or two hooks in fixed positions were set, then a Skolem-type sequence was built around them. We can extend this idea to cover Skolem-type sequences in which there are an arbitray number of hooks in any positions in the sequence.

Example 3.1 Skolem-type sequence of order 10 with 4 hooks

$$S = (10, 8, 9, 3, 4, 7, 3, 0, 4, 8, 10, 9, 7, 0, 5, 6, 0, 1, 1, 5, 2, 6, 2, 0)$$

Theorem 3.1 *A Skolem-type sequence of order n and at least one hook exists for all n.*

Proof We show a construction for a sequence of length $2n+1$ with one hook for any given n. Put the largest odd number in the first index of the sequence, followed by each remaining odd number in decreasing order. Put the largest even number in the next available position followed by each even number in decreasing order. Place the hook between the 2's and we are done. It is then trivial to place hooks at the end of the sequence to cover the case of multiple hooks.

Example 3.2 Skolem-type sequence of order 9 with 1 hook formed using the above construction

$$S = (9, 7, 5, 3, 1, 1, 3, 5, 7, 9, 8, 6, 4, 2, 0, 2, 4, 6, 8)$$

3.2 Appearance Values and the Notion of Combs

So far we have been concerned with Skolem-type sequences of order n in which each number $k \in \{1, ..., n\}$ has appeared exactly twice. We can generalize this as well, by introducing a value λ, where each $k \in \{1, ..., n\}$ appears in S exactly λ times.

Example 3.3 Skolem-type sequence with $n = 9$, $\lambda = 3$

$$S = (7, 5, 3, 6, 9, 3, 5, 7, 3, 6, 8, 5, 4, 9, 7, 6, 4, 2, 8, 2, 4, 2, 9, 1, 1, 1, 8)$$

Theorem 3.2 *A necessary condition for the existence of a Generalized Skolem sequence of order n and $\lambda = 3$ is:*

$$n = \{0, 1, .., \lambda - 1\}(mod \lambda^2)$$

for any λ which is a prime power.

Now that we have defined λ, we will use it to introduce the notion of a k, λ comb.

Definition 3.1 *A subsequence of the form $s_i, s_{i+k}, s_{i+2k}, ..., s_{i+(\lambda-1)k}$ of S is called a k, λ comb if $s_{i+xk} = k$ for all $x \in \{0, 1, ..., \lambda - 1\}$.*

Example 3.4 A sequence S containing a 2,4 comb

$$S = (2, 0, 2, 0, 2, 0, 2)$$

So now we can redefine the notion of a hook in terms of a k, λ comb:

Definition 3.2 *A hook in a sequence is an occurence of a $0, \lambda$ comb.*

3.3 Definition of a Generalized Skolem-type Sequence

The last parameter we need to consider in finalizing a defnition for a generalized Skolem-type sequence is the case where we would like various k, λ combs to occur with different multiplicities. We can summarize all of our parameters then in the following definition:

Definition 3.3 *S is called a generalized Skolem-type sequence GSTS($\{q_0, q_1, ..., q_n\}, \lambda$) if S is a union of combs involving precisely q_k disjoint k, λ combs for each $k \in \{0, ..., n\}$. The set $\{q_0, q_1, ..., q_n\}$ can be denoted as Q.*

Example 3.5 A GSTS$(Q, 2)$ with $Q = \{2, 1, 1, 0, 1, 1, 1, 2, 1\}$

$$S = (1, 1, 8, 6, 4, 7, 5, 7, 4, 6, 8, 5, 7, 0, 7, 2, 0, 2)$$

Let us now, for illustrative purposes, redefine a Skolem sequence in terms of a GSTS.

Definition 3.4 *A Skolem sequence of order n is a GSTS($\{q_0, ..., q_n\}, 2$) with $q_0 = 0$ and $q_i = 1$ for all $i \in \{1, ..., n\}$.*

4 Generating Algorithms

Constructing Skolem-type sequences by hand is not an easy task. When we approach values of $n \geq 8$ it becomes very difficult to construct more than just a few sequences. In certain situations we also would like to enumerate all possible sequences of a given order with some given parameters. It has been shown by Abrham and Gotzig [3] that there is a lower bound of $2^{\lfloor \frac{n}{3} \rfloor}$ sequences of the orderinary Skolem type. This is, infact, a very inaccurate lower bound, as the number of sequences for a given order is often far larger than this value. For example, for $n = 13$ there are actually 3040560 Skolem sequences, while the lower bound would indicate 'at least' 16. Nevertheless, the exponential lower bound given by Abrham and Gotzig indicates that it would be impossible to compute them all by hand. For these reasons, we must devise algorithms for the construction and enumeration of such sequences.

There are two types of algorithms which lend themselves quite well to the construction of these sequences. The first being a deterministic algorithm for the enumeration of sequences, the second being a heuristic for attempting to construct more difficult sequences of higher order.

4.1 Exhaustive Algorithm

For the problem of the enumeration of sequences, we devise an exhaustive algorithm to try all possible placement positions for given parameters, stepping through to eventually construct all sequences. We first present this in its base form of a normal Skolem sequence, and then generalize it to accomodate the construction of a GSTS(Q, λ) for arbitrary values of Q and λ.

The exhaustive algorithm gets its name from the method it follows to constructs sequences. Initially, we construct a blank array of length $2n$. We then recursively attempt to

fill in the array one number at a time. The function is defined based on an integer value we call 'depth', which is the current number we are attempting to insert into the sequence array. We set initial depth equal to the order of the sequence n, and begin the recursive insertion into the blank array.

The following steps are carried out within the exhaustive algorithm:

1. An array S of length $2n$ is constructed and initially set to zeros
2. A recursive function is defined based on an integer 'depth'. This depth will initially be set to n, the order of the sequence
3. The recursive algorithm begins. The structure of the algorithm is as follows:

```
function recursive_insert(int depth)

1     if ( depth == 0 )
2          we are done, output the sequence
3     else
4          for i=1 to (2n - depth)
5               if ( S[i] == 0 and S[i+depth] == 0 )
6                    set S[i] = depth
7                    set S[i+k_n] = depth
8                    recursive_insert(depth-1)
9               else
10                    number will not fit
```

This exhaustive algorithm will step through and enumerate all possible Skolem sequences of length n. As mentioned, we are interested in generalizing this to meet the parameters mentioned in Section 2.

4.2 Generalizing the Exhaustive Algorithm

The process of generalizing the exhaustive algorithm to meet our parameters involves taking the hard-coded assumptions from the exhaustive algorithm shown previously, and generalizing them.

For the case of constructing a sequence with a number of hooks h, we do the following:

1. Make a blank array of size $2n + h$, set each index to 0.
2. Perform the exhaustive algorithm described previously.
3. The numbers of the sequence fill in around the zeros.
4. When depth=0, we count the left over zeros as hooks, and we are done.

This construction succeeds, since it steps through all possible insertions around the zeros, which is analagous to inserting zeros into all possible places around the numbers. Similarly, for the case of sequences such as Hooked Skolem, or Rosa sequences we need the hooks to be in a specified location. One way to do this would be to look through all of the sequences and check for hooks in the correct positions, but a more efficient way would be the following.

1. Make a blank array of size $2n + h$, set each index to 0.
2. For each hook, set the index of the array in which you want a hook to -1.
3. Perform the exhaustive algorithm on this array.
4. When the boolean check for S[i]=0 fails, the index will remain unmodified.
5. When depth=0, change the -1's to zeros, and we are done.

The next parameter we are interested in generalizing is the appearance number λ. We see in the exhaustive algorithm we had previously defined, that there is a line which checks to see if the index at S[i] and S[i + depth] are both equal to 0. This step made an assumption that we were inserting various $k, 2$ combs. To generalize this, we must make a loop which steps through and checks each possible position that our k, λ comb will fill.

```
function recursive_insert(int depth, int lambda)

1     if ( depth == 0 )
2          we are done, output the sequence
3     else
4          for i=1 to (2n - lambda*depth)
5               boolean fits = true
6               for x=0 to (lambda-1)
7                    if (S[i + x*depth] != 0)
8                         fits = false
9               if (fits)
10                   for y=0 to (lambda-1)
11                        S[i + y*depth] = 0
12                        recursive_insert(depth-1)
13              else
14                   all values will not fit
```

Once we know how the basic recursion works, along with how to construct a sequence with a number of hooks, we can now generalize the algorithm fully using our definition of a GSTS(Q, λ). We will take as input Q and λ, and from them construct our sequence. The steps to produce the GSTS are as follows:

1. Make an array S initially set to all zeros. This array will be of size $q_0 + \sum_{i=1}^{n} q_i \lambda$.
2. Set any index of the array we wish to be a fixed hook equal to -1. The number of such placed hooks must not exceed q_0.

3. Perform the following recursive function on S with initial k equal to n. Note that our new value of k corresponds to depth in the previous two algorithms, it has changed to reflect our definition of a GSTS(Q, λ).

```
function recursive_insert(int k, int lambda, int[] Q)

1     if (k == 0)
2          we are done, output sequence
3     else if (Q[k] == 0)
4          recursive_insert(k-1, lambda, Q)
5     else
6          for i=RMO to (2n-lambda*k)
7               boolean fits = true
8               for x=0 to (lambda-1)
9                    if (S[i + x*k] != 0)
10                        fits = false
11              if (fits)
12                   for y=0 to (lambda-1)
13                        S[i + y*k] = 0
14                        Q[k] -= 1
15                        recursive_insert(k, lambda, Q)
16              else
17                   will not fit
```

The only line which requires explanation for the algorithm above is line 6. The value RMO denotes the right-most occurence of all first indices of all other k, λ combs in the sequence for the current k (in other words, where the right-most comb started). The reason this is done to prevent duplicate sequences from occuring in our output. By restricting this value to be to the right of any previously placed values of k, then this guarentees uniqueness placements for all values of k. We set up a simple loop to check where this value is and then insert it in place of RMO.

4.3 Initial Patterns

While the purely exhaustive algorithm is the most basic Skolem Sequence construction, from it we can imagine a situation where a partial sequence is initially filled out, and then the remaining positions are filled in exhaustively. This method seems quite intuitive in a way, because, if we were to attempt to construct a sequence by hand, we would not start generating random numbers and placing them exhaustively. Most times, simple patterns are found and an initial portion of the sequence is easily filled. A very basic example of this involves the placing of either, even or odd values, for s_i. Since the values for increasing odd or even numbers increases by 2, then we see that each successive number will fit outside the previously inserted number.

Example 4.1 Take the following initial steps for order 8:

$$S = (0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$

$$S = (0, 0, 4, 2, 0, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$
$$S = (0, 6, 4, 2, 0, 2, 4, 6, 0, 0, 0, 0, 0, 0, 0, 0)$$
$$S = (8, 6, 4, 2, 0, 2, 4, 6, 8, 0, 0, 0, 0, 0, 0, 0)$$

We place the even numbers on the left, and then attempt to fill in the rest. The advantage of this method is that it reduces the number of recursions we have to carry out, which speeds up the process, but at the same time, producing far less sequences then the purely exhaustive method. The danger we run into, when performing partial pattern insertions, is that we could possibly insert a pattern which will not produce any sequences. An example of this pitfall can be extended from above. After inserting all even numbers, start placing odd numbers in the same way on the right side of the sequence:

Example 4.2

$$S = (8, 6, 4, 2, 0, 2, 4, 6, 8, 0, 0, 0, 1, 1, 0, 0)$$
$$S = (8, 6, 4, 2, 0, 2, 4, 6, 8, 0, 0, 3, 1, 1, 3, 0)$$
$$S = (8, 6, 4, 2, 0, 2, 4, 6, 8, 0, 5, 3, 1, 1, 3, 5)$$

We see that now we are stuck. The only remaining number to place is 7, but this is not possible, so great care must be taken in choosing the pattern to initially insert. Pattern insertions are quite useful for testing mathematical constructions of Skolem sequences, as they can automate in milliseconds what might take minutes or even hours by hand.

4.4 Genetic Algorithm

Genetic algorithms are a specific type of heuristic algorithm which use probability and a notion of fitness, to mimic the natural process of evolution. A given population or species (of in our case, Skolem sequences) will go through the process of evolution, eventually arriving at either an exact, or near solution. New generations are formed through the process of natural selection and reproduction, where fitter parents are selected to reproduce, while less fit sequences will die off. Mutations may also happen to the sequences to mimic natural environmental impact, or problems in genetic recombination. This process repeats until a break condition is reached, at which point we will output a given number of the most fit sequences we have produced.

To form the generic shell of any genetic algorithm, the following elements are needed:

1. Each member of the population has two associated values: a phenotype, and a genotype. These correspond to (in the example of human evolution) the human being itself, and it's encoded representation as DNA.

2. A popluation of individuals must be established, and evolved. The initial generation of a set of individuals is usually generated randomly, to mimic a fresh set of beings inserted into an environment.

3. A fitness function must be defined on the phenotype. This fitness is a function of the phenotype, and its environment, and mimics natural selection by selecting only those more fit sequences to reproduce. A higher fitness means a better ability to survive in the environment. For example, a faster predator would be able to catch more prey, so its fitness would be higher than that of a slower member of the same species.

4. A reproduction method is established. When phenotypes have been selected based on their fitness, their associated genotypes are recombined to form one or more child genotypes for the next generation.

5. A mutation method is established. A mutation involves setting up a statistical percentage within the algorithm, when this percentage is satisfied a genotype becomes mutated, often creating small changes to the genotype itself. This process can either hurts or help the individual's fitness.

We take each of these elements and make them specific to the case of generalized Skolem sequence construction. We define the genotype-phenotype relationship, the method of reproduction, and the functions for both fitness and mutation. We then combine these elements into our algorithm, and give the psuedocode for running it.

4.4.1 Skolem Specific Genetic Algorithm Attributes

When setting out to construct a genetic algorithm, the first thing which must be established is the all important genotype to phenotype relationship. In our case, the phenotype will be a representation of a Skolem-type sequence itself, while the genotype will be its encoding. The only difference between our phenotype, and a normal Skolem-type sequence is that our phenotype P is a sequence of sets, where each value of P_i may contain 0 or more elements, based on its construction from its genotype. Currently, our genetic algorithm does not support a completely generalized GSTS(Q, λ), but it does accept a base order n, and a λ value for construction, which correspond to GSTS($\{0, 1, 1, .., 1\}, \lambda$),

Example 4.3 A sample Skolem phenotype of order 4, with $\lambda = 2$

$$P = (\{4, 2\}, \{1\}, \{1, 2\}, \{3\}, \{4\}, \{\}, \{3\}, \{\})$$

We see that the same Skolem-type distance constraints still apply, although now the numbers are in sets, and their distance pertains to the distance between the sets in which they are contained. We see that although p_1 above contains both 4 and 2, the relative distances between it and the sets which contain the other 4 and 2 are still within the Skolem constraint. We will now define our fitness function based on the phenotype description above.

Definition 4.1 *We define our fitness function fit(P) to be the total length of P minus the number of empty sets it contains.*

By this definition, the fitness for the eample above would be $8 - 2 = 6$. So we can think of the maximum, or perfect fitness for a given phenotype P to be $|P|$, the length of P itself.

The genotype on the other hand is an encoding of the phenotype which follows the following definition:

Definition 4.2 *The Skolem genotype used in our algorithm is a sequence G of length n in which g_k denotes the first occurence of k in the corresponding phenotype of length $n\lambda$.*

Example 4.4 Genotype and corresponding phenotype for a Skolem sequence of length 5

$$G = (9, 2, 5, 3, 1)$$

$$P = (\{5\}, \{2\}, \{4\}, \{2\}, \{3\}, \{5\}, \{4\}, \{3\}, \{1\}, \{1\})$$

Now that we have defined a genotype, phenotype, and fitness function we can start thinking about reproduction and how it will be done in our algorithm. Traditionally two genotypes are chosen based on the fitness of their respective phenotypes and reproduce to produce any number of children. Our algorithm takes the entire population, and selects two parents with a linear probability on their fitnesses. One can think of this as a virtual dart board, where each genotype is given a slice of the board based on its probability, then two darts are thrown randomly at the board, we take the genotypes hit by the two darts and these are our parents.

Reproduction consists of taking two of these parents $G1$ and $G2$ and selecting a random number r such that $0 < r < n$. We then concatenate the first r values of $G1$ with the last $n - r$ values from $G2$ to form a child.

The last function which needs to be defined is the mutation function. In the main shell of the algorithm, we define a mutation rate percentage. Whenever we generate a new child, before we insert it into the population we generate a random percentage number. If this number is less than our mutation rate percentage, then we mutate the genotype before inserting it. The mutation involves a swapping of one or more elements of the array of the genotype. The number of swaps depends on the current state of the evolution. At the beginning of the evolution we perform many mutations and swaps to effictively 'sitr up' the gene pool and get a diverse array of values. After a few generations we turn down the mutation and swap rate to let evolution take its course. Now that we have defined all the necessary attributes of the algorithm, let us now see the pseudocode itself.

4.4.2 The Algorithm

The algorithm for the shell of the genetic algorithm goes as follows:

1. Define a mutation rate percentage m
2. Randomly Generate an initial population set T of genotypes
3. Calculate the fitnesses for all phenotyes associated with the genotypes in T
4. If any of the phenotypes have a perfect fitness, stop and output
5. Generate a new population as follows:
   (a) Select two parents $G1$ and $G2$ from T based on their fitness

(b) Carry out reproduction on $G1$ and $G2$ to create a number of children

(c) If a randomly selected percentage falls within m, mutate the child

(d) Insert the child into the new generation

6. If we have reached our maximum generation number, exit and output the best sequence that we have found so far

7. Return to step 3

5 Results and Analysis

We now see the results from having run both the exhaustive and genetic algorithms with various input parameters. We show some sample enumerations of Skolem-type sequences using the exhaustive algorithm, then show plots of fitness vs. time for the case of some runs of the genetic algorithm for more complex sequences. We finish our results by displaying some of the most complicated sequences which we have been able to create.

5.1 Exhaustive Results

We show here some sample enumerations from our exhaustive algorithm. These programs were written in C++, running under Gentoo linux on an AMD Athlon(tm) XP 1700+ 32-bit processor. The first table shows various enumerations of some regular Skolem sequences with given order n, with $\lambda = 2$. The 'Sequences' column shows how many total sequences were constructed, and the 'time' column shows how many milliseconds the program took to run to completion.

Exhaustive Algorithm Enumerations of Skolem Sequences		
Order n	Sequences	Time (ms)
4	6	0
5	10	0
8	504	10
9	2656	70
12	455936	31170
13	3040560	272240

Next we show enumerations of all possible extended Skolem sequences for a given order n. An extended Skolem sequence is a Skolem sequence with a hook in any arbitrary position. We can see from our table that simply dding one hook to the sequences increases the number of sequences dramatically.

Exhaustive Algorithm Enumerations of Extended Skolem Sequences (1 hook)		
Order n	Sequences	Time (ms)
6	160	0
7	636	0
8	3556	30
9	19488	240
10	95872	1170
11	594320	14070
12	4459888	120100
13	32131648	1085390

We can also use the exhaustive algorithm to prove or disprove the existence of certain GSTS. For example, the following table shows the minimum number of hooks required to succesfully construct a Skolem-type sequence of order n with $\lambda = 4$, the third column indicates the number of such sequences.

Exhaustive Algorithm Minimum Number of Hooks Required for $\lambda = 4$		
Order n	Minimum Hooks	Sequences
7	5	2
8	3	4
9	3	6
10	4	12
11	3	16
12	5	38
13	5	104
14	3	2

5.2 Genetic Algorithm Results

We will now show some results from runs of our genetic algorithm for some particularly difficult sequences. Each result will have a graph plotting fitness vs. time, and a brief explanation of the phenomenom observed in the graph. The first example is of the case of a Skolem-type sequence with $n = 25$ and $\lambda = 5$, which is a particularly difficult example, because nobody in the literature has yet been able to construct such a sequence.

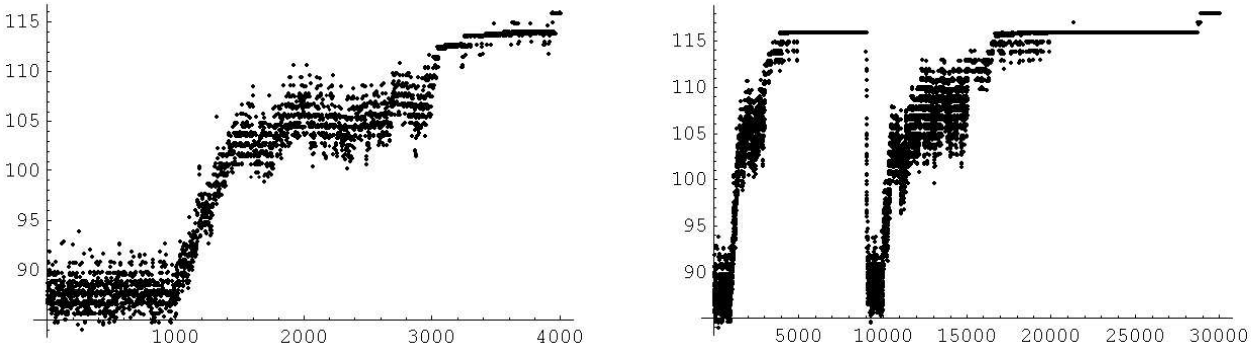

Figure 1: Plot of fitness vs. time for a genetic algorithm run for a Skolem sequence with $n = 25$, $\lambda = 5$

In Figure 1, we see that the initially randomly generated population has a fitness of around 85-90. This initial population is 'stirred up' using a high rate of mutation for 1000 generations, at which point the mutation rate is turned down and we see evolution really start to take place. At generation 2000 we see that the fitness has reached approximately 110 out of a maximum 125. Since our $\lambda = 5$ this correspond to approximately 3 numbers being out of place in the sequence. When the algorithm continues to run until generation 4000 we see tht the maximum fitness has reached 116. The graph on the right shows the same algorithm run up until generation 30000. However there is a noticable decline in fitness at generation 10000. This occured because we saw that from generation 4000 until 10000 there was no noticable increase in fitness, our population had reached its maximum. Because of this fact, we turn up the mutation rate for a few generations in order to attempt to help our algorithm reach a new maximum fitness. At first glance the mutation killed off our fit genotypes, but when the algorithm is left to run until generation 30000 we see that we reach a maximum of 118 of a possible 125. This means that the process of mutation actually does work, although being initially bad for our fitness, led to a better result in the end.

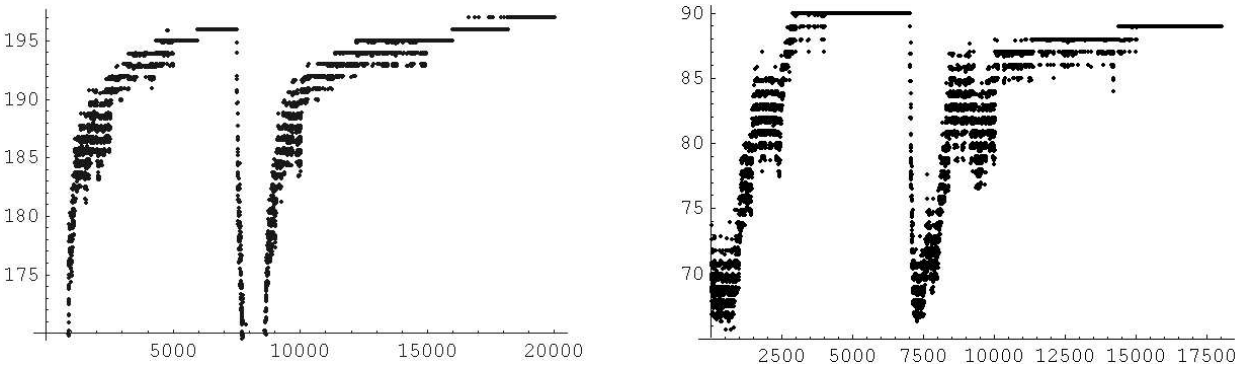

Figure 2: Two more genetic algorithm runs. The graph on the left shows a run for $n = 100, \lambda = 2$ while the graph on the right shows a a run for $n = 24, \lambda = 4$

As with Figure 1, these graphs show a very typical trend of genetic algorithms. Initially, the gene pool is made diverse in order to provide many different genotypes of various fitnesses to work in our environment. We see a sharp rise in fitness, followed by a long plateau of

little increase. We attempt to break this plateau and achieve better results by increasing the rate of mutation within the algorithm. We can see that this oftentimes works as in Figure 1, and also in Figure 2 (left), but as the case of Figure 2 (right) our mutated population never again achieved the same maximum fitness. This illustrates the fact the never ending struggle to obtain the perfect settings for the parameters we use in our genetic algorithm.

5.3 Sample Sequence Results

We now show some examples of more complex sequences which we have been able to construct using both the genetic and exhaustive algorithms.

Example 5.1 Skolem sequence of order 80

$$S = \{84, 82, 83, 79, 77, 81, 74, 80, 71, 69, 78, 66, 64, 76, 61, 75, 58, 56, 73, 53, 72, 50, 48, 70, 45, 43, 68, 40, 67, 37, 34, 65, 31, 28, 63, 24, 62, 21, 12, 60, 16, 59, 9, 5, 57, 10, 7, 55, 5, 54, 12, 9, 52, 7, 51, 10, 16, 49, 21, 24, 47, 28, 46, 31, 34, 44, 37, 40, 43, 45, 48, 50, 53, 56, 58, 61, 64, 66, 69, 71, 74, 77, 79, 82, 84, 83, 81, 80, 78, 76, 75, 73, 72, 70, 68, 67, 65, 63, 62, 60, 59, 57, 55, 54, 52, 51, 49, 47, 46, 44, 42, 39, 41, 38, 35, 33, 30, 27, 36, 26, 17, 11, 25, 32, 14, 8, 13, 29, 15, 6, 1, 1, 11, 8, 23, 6, 22, 17, 14, 13, 20, 18, 19, 15, 27, 26, 30, 25, 33, 35, 39, 38, 42, 41, 36, 32, 29, 23, 22, 18, 20, 19, 4, 2, 3, 2, 4, 3\}$$

Example 5.2 Skolem-type sequence of order 27, $\lambda = 3$

$$S = \{27, 25, 23, 26, 24, 8, 5, 13, 11, 22, 12, 5, 3, 8, 21, 3, 5, 20, 3, 11, 13, 8, 12, 9, 19, 23, 25, 27, 24, 26, 11, 22, 9, 13, 12, 21, 14, 20, 4, 10, 18, 9, 4, 19, 17, 15, 4, 16, 23, 10, 14, 25, 24, 22, 27, 26, 21, 20, 18, 10, 15, 17, 19, 16, 14, 6, 7, 1, 1, 1, 2, 6, 2, 7, 2, 15, 18, 6, 17, 16, 7\}$$

Example 5.3 Skolem-type sequence of order 13, $\lambda = 4$, with 5 hooks. This example is analagous to lining up a deck of 52 playing cards with the minimum number of hooks inserted to make the sequence work.

$$S = \{13, 3, 9, 6, 3, 0, 0, 3, 10, 6, 3, 9, 5, 13, 11, 6, 8, 5, 10, 12, 9, 6, 5, 0, 8, 11, 13, 5, 10, 9, 0, 12, 8, 4, 0, 7, 11, 4, 10, 13, 8, 4, 7, 12, 2, 4, 2, 11, 2, 7, 2, 1, 1, 1, 1, 12, 7\}$$

Example 5.4 Skolem-type sequence of order 24 with $\lambda = 4$. This is the most complex of the sequences we have constructed, with approximate computation time of approximately one month on the system described in Section 4.1

$$S = \{14, 23, 24, 2, 4, 2, 7, 2, 4, 2, 12, 6, 4, 7, 14, 10, 4, 6, 22, 20, 7, 16, 12, 6, 23, 10, 24, 7, 14, 6, 21, 18, 11, 15, 12, 10, 8, 16, 19, 20, 22, 17, 14, 11, 8, 10, 12, 23, 15, 18, 24, 21, 8, 16, 11, 13, 5, 19, 17, 20, 8, 5, 22, 15, 9, 11, 5, 18, 13, 16, 23, 5, 21, 9, 24, 17, 19, 3, 15, 20, 3, 13, 9, 3, 22, 18, 3, 1, 1, 1, 1, 9, 17, 21, 13, 19\}$$

6 Conclusion and Future Work

A result we have seen from executions of our genetic algorithm is that it often comes to approximate solutions very quickly, while sometimes never yielding exact solutions for a given set of parameters. This is caused (in our case) by the discrete nature of Skolem-type sequences. A sequence S may be of fitness $|S| - 1$ but may actually be quite far from constructing an actual sequence. This trait of genetic algorithms lends to a possible hybrid algorithm for future consideration. This hybrid algorithm would combine both the genetic and exhaustive algorithms in order to produce a much faster way of constructing one or more sequences of complex order. To do this, we would run the genetic algorithm to produce an approximate solution very quickly, then at a given fitness we would hand this sequence over to the exhaustive algorithm in order to perform insertion and swap operations to effectively 'finish off' the rest of the sequence.

Another consideration with respect to our genetic algorithm, is that there can always be improvements made to both the fitness and recombination functions. We chose our particular functions due to their intuitive nature and feel about what a 'fit' sequence should be. More testing needs to be done with various fitness and recombination functions, as it is very difficult to produce provable results without the process of simulation being carried out. We wish to obtain an optimal combination of gene diversity while maintaining a steady rate of increase in maximum fitness.

Statistical analysis is also considered for future work with these algorithms. Statistical feedback between the enumerative and genetic algorithms could be very beneficial for speed increases. For example, if we enumerated a number of sequences for given parameters and performed some statistical information with respect to number placement or sequence design in general, we could use this analysis in either the construction, reproduction, or fitness functions in the genetic algorithm. Conversly, statistical information from results of the genetic algorithm can be used to aid the enumerative algorithm in calculating new partial sequences which could be filled out, as mentioned in Section 4.3.

A final goal for these types of algorithms would be to combine all of the mentioned genetic, exhaustive and statistical based algorithms to try and form an algorithm for the discovery of patterns or constructions of Skolem-type sequences of given parameters. Normally constructions and patterns are found by hand, often taking weeks or even years of testing to validate. If we could devise an algorithm to calculate such constructions/patterns, it would our ultimate goal, reducing the time complexity of finding certain sequences from exponential, to linear.

References

[1] N. Shalaby, Skolem Sequences, *CRC Handbook of combinatorial designs* (1996), 457-461

[2] A.R. Eckler, Construction of missile guidance codes resistant to random interference, *Bell Syst. Tech. J.* (1960), 937-994

[3] J. Abrham, Exponential lower bounds for the number of Skolem and external Langford sequences, *Ars Combin.* 22 (1986), 187-198

[4] C. J. Jacob, *Illustrating Evolutionary Computation with Mathematica, MKP Morgan Kaufmann*, (2001)

[5] Th. Skolem, On crtain distributions of integers in pairs with given differences, *Math Scand.*, 5:57-68 1957