Algorithms in Discrete Minimization Applications
Dylan M. Copeland
Department of Mathematics
Southeastern Louisiana University
Hammond, LA 70401

Advisor:
Prof. Dennis I. Merino

Abstract

We develop computer algorithms to solve minimization problems where the variables take only integral values, that is the variables are of the discrete type. In particular, we study the following type of problems. Given a set of positive integers less than 100, say \{a_1, ..., a_n\}, we wish to minimize \(\sum_{i=1}^{99} f(i)\), where \(f(i)\) is the minimum among \(\sum_{k=1}^{n} g(b_k)\), and \(i = \sum_{k=1}^{n} b_k a_k\). A special application is a study of the efficiency of the current US coins in making change of up to 99 cents.

1 Introduction

Among the most essential problems solved by mathematics are those of minimization, which can usually be solved by calculus when working with variables of the continuous type. But many applications of minimization involve discrete variables whose range is restricted to the integers. Ordinary calculus fails in such cases, which means that purely mathematical methods to solution can be very difficult if not impossible. In this paper, we demonstrate the use of computer algorithms to simplify discrete minimization problems that would otherwise be baffling.

Take for instance the problem of minimizing the number of coins required on average in making change. The currency of the United States has five types of coins with the values 1 (penny), 5 (nickel), 10 (dime), 25 (quarter), and 50 (half-dollar). The values of our coins were selected as numbers that are easy to use when adding up change, but it turns out that they are not very efficient in terms of the average number of coins that must be used to make a sum. How then can we select five coins that would minimize the average number of coins required for an amount less than a dollar? Finding that set of coins would reduce the number of coins the U.S. government has to manufacture, and dealing with coins would be less cumbersome.

A purely mathematical solution to this problem is difficult to find due to the necessity of making a quantitative conclusion regarding the effect of
changes in the five variables. Instead, we develop a computer algorithm which exhausts all possible sets of coins and finds the most efficient one. The ability of computers to rapidly execute many instructions allows one to make much needed simplifications to the problem of minimization at the expense of computing time. As we now show, this daunting problem and similar others can be solved by rather short and simple algorithms.

2 The Coin Efficiency Problem

We are interested in finding the set of five coins for which the average number of coins needed to form sums from 1 to 99 is minimized. To do this, it is necessary to find the minimum number of coins needed for a particular sum and a given set of coins. Since the coins are being added, each sum can be written as a linear combination of the coins in which each coefficient is nonnegative. Therefore, we will refer to this minimum sum as the MSNC (minimum sum of nonnegative coefficients). Since the MSNC yields one and only one integer value for a given integer "amount", we define it as a function from the set of natural numbers to itself. Note that one of the coins, or elements of the set \( D \), must be equal to 1 so that the sum 1 can be formed, ensuring that the function is well-defined.

**Definition 1** Let \( x \) be a positive integer and let \( D = \{d_1, d_2, \ldots, d_n\} \) be a set of \( n \) positive integers, where \( d_1 = 1 \) and \( d_1 < d_2 < \cdots < d_n \). The function \( MSNC : \mathbb{N} \rightarrow \mathbb{N} \) is defined by

\[
MSNC(x) = \min \left( \sum_{i=1}^{n} c_i \right)
\]

such that \( \sum_{i=1}^{n} c_i d_i = x \), where \( c_i \) is a nonnegative integer.

To illustrate the behavior of the MSNC function, we will consider two examples. First, take \( D_1 = \{1, 5, 10, 25, 50\} \). Then \( MSNC(32) = 4 \), since the linear combination \( 2 \times 1 + 1 \times 5 + 1 \times 25 = 32 \) minimizes \( \sum_{i=1}^{5} c_i \) to 4. Now let \( D_2 = \{1, 4, 16, 27, 41\} \). In this case, the linear combination \( 2 \times 16 = 32 \) yields \( MSNC(32) = 2 \), and we observe that it is not true for all sets that the best linear combination maximizes the use of the largest coins. That is, the procedure for finding the best linear combination is not simply to use as many large coins as possible, since in this case the use of the largest possible coin, 27, would have given \( 1 \times 1 + 1 \times 4 + 1 \times 27 = 32 \) and \( \sum_{i=1}^{5} c_i = 3 \), which is not the minimum.

**Theorem 1** For \( n = 5 \), the five-element set \( D = \{1, 5, 16, 23, 33\} \) uniquely minimizes \( \sum_{x=1}^{99} MSNC(x) \) to 329.
The proof of this theorem is established by the following exhaustive algorithm, of which a C++ implementation yielded the stated results. For a given five-element set $D$, one can evaluate $MSNC(x) = \min(\sum_{i=1}^{5} c_i)$ for each $x \leq 99$ by exhausting all possible values of $c_i$ and finding those that yield $\min(\sum_{i=1}^{5} c_i)$. This must be done for all permissible sets $D$ in order to find those for which $\sum_{x=1}^{99} MSNC(x)$ is minimum.

1. Select a set $D$ of five integers such that $1 \leq d_i \leq 99$. For simplification and speed, we make the restrictions that $d_i \neq d_k$ for $i \neq k$ and that $d_i < d_k$ for $i < k$. This ordering of the elements of $D$ prevents the repeated use of sets that are the same but ordered differently. Under these restrictions, the program begins with the set $D$ for which all $d_i$’s are minimum; i.e., $D = \{1, 2, 3, 4, 5\}$. Then a looping structure is used which selects all possible sets $D$ such that $5 \leq d_5 \leq 99$, $4 \leq d_4 < d_5$, $3 \leq d_3 < d_4$, $2 \leq d_2 < d_3$, and $d_1 = 1$ (see definition 1), which must always be 1 so that the condition $\sum_{i=1}^{n} c_i d_i = x$ will hold for $x = 1$.

2. For the chosen set $D$, evaluate $MSNC(x)$ for all $x$ from 1 to 99. To ensure that all possible linear combinations of the elements of $D$ are checked, the program finds $\sum_{i=1}^{5} c_i$ using every possible value for $c_i$. This is done by means of a looping structure which selects all values of $c_i$ such that

$$
0 \leq c_5 \leq 99 \text{ div } d_5 \\
0 \leq c_4 \leq (99 - c_5 d_5) \text{ div } d_4 \\
0 \leq c_3 \leq (99 - c_5 d_5 - c_4 d_4) \text{ div } d_3 \\
0 \leq c_2 \leq (99 - c_5 d_5 - c_4 d_4 - c_3 d_3) \text{ div } d_2 \\
c_1 = x - c_5 d_5 - c_4 d_4 - c_3 d_3 - c_2 d_2
$$

Of all values of $c_i$, those for which $\sum_{i=1}^{5} c_i$ is minimum are summed to find $MSNC(x)$.

3. After evaluation of $MSNC(x)$ for all $x$ from 1 to 99, find $\sum_{x=1}^{99} MSNC(x)$. Of all the sets $D$, those for which this sum is minimum are recorded as solutions.

The solution set, $D = \{1, 5, 16, 23, 33\}$, yields $\sum_{x=1}^{99} MSNC(x) = 329$. It follows that the average number of coins required for a sum between 1 and 99 inclusive is $329 / 99 = 3.323232...$, which is a minimum. The solution set is much more efficient than the standard set of coins $\{1, 5, 10, 25, 50\}$, for which $\sum_{x=1}^{99} MSNC(x) = 420$. For this set, an average of $420 / 99 = 4.242424...$ coins are required for sums between 1 and 99, which is almost one coin more than the solution set requires. This improvement is due in part to the fact that the elements of the solution set are relatively prime to each other, which eliminates cases where a positive integer $x$ less than or equal to 99 is equal to multiples.
of more than one element of the set, thus increasing the number of integers between 1 and 99 that are multiples of elements of the set. Since in most cases where \( x = kd_i \) for a small integer \( k \) we observe that \( MSNC(x) = k \), the increase of integers such as \( x \) lowers the average \( MSNC \).

Another cause of the inefficiency of the standard set \{1, 5, 10, 25, 50\} is the element 50, which can be used less frequently than the solution set’s largest element, 33. Since the largest of a set should generally be used frequently, the use of an excessively large coin is relatively limited and has an adverse effect on the average \( MSNC \).

### 3 The Weights Efficiency Problem

In the previous problem, sums had to be formed by adding the values of coins. That is, each sum could be written as a linear combination of the elements of \( D \), where the coefficients could only be nonnegative. Now consider this same problem with the variation that the elements of \( D \) can be either added or subtracted to form each sum. A good visualization of such a problem is a pan balance with two arms, where weights can be placed either in the left pan (which subtracts from the net weight) or in the right pan (which adds to the net weight). We are interested in finding the set of five weights for which the average number of weights needed to weigh an amount is minimized. Again we allow the amounts to be weighed to range from 1 to 99 so that we can compare the results with those of the previous problem.

Since each amount can now be written as a linear combination of the weights in which the coefficients can be negative, the total number of weights used is equal to the sum of the absolute values of the coefficients. Therefore, we are to find the \( MSAC \) (minimum sum of the absolute value of coefficients) for each amount from 1 to 99.

**Definition 2** Let \( x \) be a positive integer and let \( D = \{d_1, d_2, \ldots, d_n\} \) be a set of \( n \) positive integers, where \( d_1 < d_2 < \cdots < d_n \). The function \( MSAC : \mathbb{N} \to \mathbb{N} \) is defined by

\[
MSAC(x) = \min \left( \sum_{i=1}^{n} |c_i| \right)
\]

such that \( \sum_{i=1}^{n} c_i d_i = x \), where \( c_i \) is any integer.

Note that this function is similar to the \( MSNC \) defined in the first problem, with the allowance that \( c_i \) may be negative. Also, the restriction that \( d_1 = 1 \) is no longer needed, since elements of \( D \) can be subtracted to form the sum 1. For example, if \( D = \{2, 11, 22, 23, 37\} \), then \(-1 \cdot 11 + 1 \cdot 37 = 26\) and \( MSAC(26) = 2 \). In this case, any positive sum can be formed since \( 23 - 22 = 1 \) can play the role of 1 as an element (although the coefficient is doubled since two coins must be used).
Theorem 2  For $n = 5$, the five-element set $D = \{13, 24, 30, 32, 33\}$ uniquely minimizes $\sum_{i=1}^{99} MSAC(x)$ to 270.

The proof of these results is supplied by the following algorithm, for which some development is necessary. Consider the linear combination $\sum_{i=1}^{5} c_i d_i = x$ for which $\sum_{i=1}^{5} |c_i| = \min(\sum_{i=1}^{5} |c_i|) = MSAC(x)$. Since $c_i$ is allowed to be any integer, it follows either that all of the terms are positive or that some are positive and some are negative. If all terms are positive, then $MSAC(x) = MSAC(x)$, which is found by the algorithm of the previous problem.

If instead it happens that some of the terms in $\sum_{i=1}^{5} c_i d_i$ are negative, then we consider $\sum_{i=1}^{5} c_i d_i = x$ to be the sum of positive and negative terms. Thus we may group the positive terms together and the negative terms together, so that $x = \sum_+ c_i d_i + \sum_- c_i d_i$, where $\sum_+$ denotes the sum of the positive terms and $\sum_-$ denotes the sum of the negative terms. Since each term in the sum $\sum_- c_i d_i$ is negative, we can say that $\sum_- c_i d_i = -s$, where $s$ is a positive integer. If we let $t = \sum_+ c_i d_i$, then $x = t - s$, which is the difference of two positive integers. In this form, it is obvious that $MSAC(x) = MSAC(t) + MSNC(s)$ where $t > x$. Based on this reasoning, the solution to this problem is found by the following algorithm.

1. Select a set $D$ of five integers such that $1 \leq d_i \leq 99$, $d_i \neq d_k$ for $i \neq k$, and $d_i < d_i$ for $i < k$. Under these restrictions, the program begins with the set $D$ for which all $d_i$’s are minimum; i.e., $D = \{1, 2, 3, 4, 5\}$. Then a looping structure is used to select all possible sets $D$ such that $5 \leq d_5 \leq 99$, $4 \leq d_4 < d_5$, $3 \leq d_3 < d_4$, $2 \leq d_2 < d_3$, and $1 \leq d_1 < d_2$. Note that here the simplification $d_1 = 1$ made in the first problem is not acceptable, since for $x = 1$ there could be a linear combination involving subtraction that equals 1.

2. Evaluate $MSNC(x)$ for all $x$ such that $1 \leq x \leq 99$ using the algorithm of the first problem, for which one slight modification is necessary due to the new possibility that $d_1 \neq 1$. Let

$$c_1 = \frac{x - c_5 d_5 - c_4 d_4 - c_3 d_3 - c_2 d_2}{d_1}$$

to ensure that $\sum_{i=1}^{5} c_i d_i = x$. If $c_1$ is an integer then it is used as a possible solution, but if $c_1$ is not an integer then the current set of coefficients is abandoned.

3. For each $x$, if for all $t > x$ it is true that $MSNC(x) < MSNC(t) + MSNC(t - x)$ then let $MSAC(x) = MSNC(x)$. If there exists $t > x$ such that $MSNC(t) + MSNC(t - x) < MSNC(x)$ then let

$$MSAC(x) = \min(MSNC(t) + MSNC(t - x)).$$
Unlike the first problem, where the MSNC had to be calculated only for values of $x$ less than or equal to 99, it is now necessary to find the MSNC for all integers $t > x$ since it is possible to form a sum greater than 99 and then subtract the difference between that sum and $x$. For example, if $D = \{1, 10, 23, 34, 41\}$, then the linear combination yielding $MSAC(89)$ would be $89 = (0 \cdot 1) + (0 \cdot 10) + (0 \cdot 23) + (-1 \cdot 34) + (3 \cdot 41)$. In this case, the algorithm would have found this linear combination only if for $3\cdot 41 = 123$ it had calculated $MSNC(123) = 3$ and then determined that $MSNC(89) < MSNC(123) + MSNC(34) = 4$.

Thus it is essential to the validity of the algorithm that the MSNC be calculated for integers greater than 99. But our finite algorithm must stop somewhere, and to extend the range of integers over which the MSNC must be calculated tremendously increases the duration of the algorithm’s execution. To determine a reasonably small integer at which the algorithm can stop and not miss the linear combination that yields the MSAC, we note that $d_5$, the largest element of the solution set $D$, should be less than 50. Therefore, it would not be reasonable to form a sum $t > 99 + 50 = 149$ and then subtract $t - x > 50$ to get $x$, so we conclude that some integer $t < 150$ yields the desired sum $MSNC(t) + MSNC(t-x) = \text{min}(MSNC(t) + MSNC(t-x)) = MSAC(x)$.

4. After evaluation of $MSAC(x)$ for all $x$ from 1 to 99, we find the sum $\sum_{x=1}^{99} MSAC(x)$. Of all the sets $D$, those for which this sum is minimum are recorded as solutions.

An assembly language implementation of the above algorithm found that the set $D = \{13, 24, 30, 32, 33\}$ minimizes $\sum_{x=1}^{99} MSAC(x)$ to 270, yielding $270 / 99 = 2.727272\ldots$ as the minimum average number of weights required for sums from 1 to 99 inclusive. Since only this one set was found to give this minimum sum, the solution is unique. Remarkably, there are no single digit elements of the solution set, and the largest three elements are very close to each other. This is plausible when one considers that small ”weights” are not necessary to form small amounts since the larger ones in this solution set can be used in subtraction to form small amounts. The greater efficiency in this problem (a total sum of 270 as opposed to 329 in the first problem) is attributed to the much improved efficiency in forming large sums, which more than compensates for any loss seen in forming some of the small sums.

4 The Minimum Weights Problem

In the previous problem we found the set of five weights for which the average number of weights needed for amounts from 1 to 99 is minimized. Now we impose the restriction that each weight in the set $D$ may be used at most once,
regardless of whether it is used to subtract or to add to the net weight in the pan balance. We wish to know the minimum size of a set of weights such that every amount from 1 to some positive integer \( m \) can be formed under this restriction.

The algorithms presented so far have begun with the selection of a set, which is then evaluated as a possible solution for a particular integer sum. In this problem, it is simpler to select the size of a set and then find all linear combinations of all sets of that size to determine if one of the sets is a solution. Given a positive integer \( m \), we find the smallest possible solution set by starting with singleton sets and then incrementing the size until a solution is found. For a positive integer \( m \) there may be many sets of the minimum size, each of which can be considered as a solution, but the following algorithm stops as soon as the first solution set is found since we are mostly interested in the minimum size of the solution sets and not the sets themselves.

1. Initialize \( n \), the number of elements of the set \( D \), to 1.
2. Select an \( n \)-element set \( D = \{ d_1, d_2, ..., d_n \} \) such that \( 1 \leq d_i \leq m \), \( d_i \neq d_k \) for \( i \neq k \), and \( d_i < d_j \) for \( i < k \). A looping structure ensures that all such sets are tested.
3. Select an \( n \)-element set of coefficients \( C = \{ c_1, c_2, ..., c_n \} \) such that \( c_i \in \{-1, 0, 1\} \), since each element of \( D \) can be added or subtracted once or not used at all. Again, a looping structure ensures that all possible coefficients are tested.
4. Find the linear combination suggested by the sets \( D \) and \( C \), namely \( \sum_{i=1}^{n} c_i d_i \). If the integer given by this sum is between 1 and \( m \) inclusive, then record that that integer can be expressed as a linear combination of the elements of \( D \).
5. If all integers between 1 and \( m \) inclusive have been recorded as linear combinations of the elements of \( D \), then \( D \) is a solution (all sets \( D \) of size \( n \) have been checked according to step 2). If not, then increment \( n \) and repeat step 2.

A C++ implementation of the above algorithm yielded the below table of solutions for \( m \) from 1 to 40. Every integer from 1 to \( m \) can be expressed as a linear combination of the elements of \( D \), where each coefficient is -1, 0, or 1. The table indicates that the smallest size of a solution set for \( m = 2 \) is 2, for \( m = 5 \) is 3, and so on. Observe that as \( n \) increases, the range of values of \( m \) for which \( D \) is a solution increases dramatically. Also, the solutions are not always unique, considering for instance \( n = 5 \), for which both sets \( \{1, 1, 3\} \) and \( \{1, 3, 9\} \) are solutions.

<table>
<thead>
<tr>
<th>( m )</th>
<th>( D )</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1}</td>
<td>1</td>
</tr>
<tr>
<td>2, 3, 4</td>
<td>{1, 3}</td>
<td>2</td>
</tr>
<tr>
<td>5, 6, ..., 13</td>
<td>{1, 3, 9}</td>
<td>3</td>
</tr>
<tr>
<td>14, 15, ..., 40</td>
<td>{1, 3, 9, 27}</td>
<td>4</td>
</tr>
</tbody>
</table>
5 The Sum of Squares Problem

This problem differs from the first two in that we are to minimize the sum of the squares of the coefficients. Just as we demonstrated the advantage of allowing coefficients to be negative in the minimum sum of coefficients problems, we now present algorithms which minimize the sum of the squares of the coefficients in two cases, the first of which restricts coefficients to be nonnegative, and the second of which has no such restriction. As we will see, the latter case again yields much greater efficiency.

5.1 Case 1 - Nonnegative Coefficients

This case requires that we find the minimum sum of squares of nonnegative coefficients, which we will refer to as $MSSNC$ and define as a function as follows.

Definition 3 Let $x$ be a positive integer and let $D = \{d_1, d_2, ..., d_n\}$ be a set of $n$ positive integers, where $d_1 = 1$ and $d_1 < d_2 < \cdots < d_n$. The function $MSSNC : \mathbb{N} \rightarrow \mathbb{N}$ is defined by

$$MSSNC(x) = \min(\sum_{i=1}^{n} c_i^2)$$

such that $\sum_{i=1}^{n} c_i d_i = x$, where $c_i$ is a nonnegative integer.

Since $c_i^2$ tremendously increases as $c_i$ increases, the most significant characteristic of a linear combination that yields $MSSNC(x)$ is that each $c_i$ is very small. It often happens that such a linear combination does not minimize $\sum_{i=1}^{n} c_i$ because more coefficients must be nonzero in order to prevent one coefficient from being too large. For example, if $D = \{2, 7, 13, 17, 25\}$, then the linear combination $1 \cdot 2 + 1 \cdot 7 + 1 \cdot 13 + 1 \cdot 17 = 39$ yields $MSSNC(x) = 4$ (and $\sum_{i=1}^{5} c_i = 4$), whereas the linear combination $3 \cdot 13 = 39$ minimizes $\sum_{i=1}^{5} c_i$ to 3.

Theorem 3 For $n = 5$, the five-element set $D = \{1, 3, 8, 21, 44\}$ uniquely minimizes $\sum_{x=1}^{99} MSSNC(x)$ to 498.

The algorithm to prove these results is exactly the same as that which was given as the proof of Theorem 1, with the exception that in step 2 the values of $c_i$ are found such that $\sum_{i=1}^{5} c_i^2$ is minimum.

5.2 Case 2 - Integer Coefficients

In this case, each coefficient can be any integer. We refer to the minimum sum of squares of integer coefficients as $MSSIC$ and define it as a function.
Definition 4 Let $x$ be a positive integer and let $D = \{ d_1, d_2, \ldots, d_n \}$ be a set of $n$ positive integers, where $d_1 < d_2 < \cdots < d_n$. The function $\text{MSSIC} : \mathbb{N} \to \mathbb{N}$ is defined by

$$\text{MSSIC}(x) = \min \left( \sum_{i=1}^{n} c_i^2 \right)$$

such that $\sum_{i=1}^{n} c_i d_i = x$, where $c_i$ is any integer.

Unlike case 1, the case of integer coefficients cannot be solved by modifying any of the previous algorithms. In particular, the algorithm of the weights efficiency problem will not help because it does not directly find the coefficients, for the sake of simplicity. Instead, it calculates the MSAC in terms of the MSNC function, which is a simplification that cannot be made here since we must know each coefficient in order to find the sum of squares of coefficients.

Although the coefficients can be any integer, we can develop algorithms in which the coefficients are limited to a certain size beyond which the sum of the squares of coefficients would be too large to possibly yield the $\text{MSSIC}$. This makes feasible the method of exhaustion that we have used in previous algorithms. However, exhaustive algorithms for the problem of minimizing $\sum_{x=1}^{40} \text{MSSIC}(x)$ have shown to take a matter of months to execute even on the latest computers using assembly language implementations. Therefore, we will simplify our problem by minimizing $\sum_{x=1}^{40} \text{MSSIC}(x)$.

Theorem 4 For $n = 5$, the five-element set $D = \{ 6, 11, 18, 19, 21 \}$ uniquely minimizes $\sum_{x=1}^{40} \text{MSSIC}(x)$ to 90.

The algorithm again involves exhausting all possible sets $D$ and all of the possible corresponding sets of coefficients. However, in this problem there are infinitely many sets of coefficients since if we let one of the coefficients be a variable dependent on the other four, then there are four independent integer variables, each of which can range over the entire set of integers. Thus we must impose limits on the coefficients without excluding possible solutions.

This can be accomplished by finding a set for which $\sum_{x=1}^{40} \text{MSSIC}(x)$ is relatively low. It has been found empirically that the set $\{ 6, 11, 18, 19, 21 \}$ yields $\sum_{x=1}^{40} \text{MSSIC}(x) = 90$. We will use this to restrict the values of coefficients that are considered by the algorithm. Note that $\text{MSSIC}(x) \geq 1$ for any positive integer $x$, since by definition the codomain of $\text{MSSIC}$ is the set of natural numbers. Hence, for any 39 positive integers $y_i$, we have that $\sum_{i=1}^{39} \text{MSSIC}(y_i) \geq 39$. This implies that, under the restriction that $\min \left( \sum_{x=1}^{40} \text{MSSIC}(x) \right) \leq 90$, for any integer $x$ such that $1 \leq x \leq 40$ the solution set must yield $\text{MSSIC}(x) \leq 90 - 39 = 51$. This in turn implies, by definition of $\text{MSSIC}$, that for the solution set $\text{MSSIC}(x) = \min(\sum_{i=1}^{5} c_i^2) \leq 51$, which gives us the restriction on the coefficients that is required for a finite algorithm.

1. Select a set $D$ of five integers such that $1 \leq d_i \leq 40$, $d_i \neq d_k$ for $i \neq k$, and $d_i < d_j$ for $i < k$. Under these restrictions, the program begins with
the set \( D \) for which all \( d_i \)'s are minimum; i.e., \( D = \{1, 2, 3, 4, 5\} \). Then a looping structure is used to select all possible sets \( D \) such that \( 5 \leq d_5 \leq 40 \), \( 4 \leq d_4 < d_5 \), \( 3 \leq d_3 < d_4 \), \( 2 \leq d_2 < d_3 \), and \( 1 \leq d_1 < d_2 \).

2. For each \( x \), check all permissible sets of coefficients and find the one that minimizes \( \sum_{i=1}^{40} MSSIC(x) \). This is done by means of a looping structure which selects all values of \( c_2 \), \( c_3 \), \( c_4 \), and \( c_5 \) such that

\[
\begin{align*}
-7 & \leq c_5 \leq 7 \\
-\sqrt{49 - c_5^2} & \leq c_4 \leq \sqrt{49 - c_5^2} \\
-\sqrt{49 - c_5^2 - c_4^2} & \leq c_3 \leq \sqrt{49 - c_5^2 - c_4^2} \\
-\sqrt{49 - c_5^2 - c_4^2 - c_3^2} & \leq c_2 \leq \sqrt{49 - c_5^2 - c_4^2 - c_3^2}
\end{align*}
\]

To meet the condition \( \sum_{i=1}^{5} c_i d_i = x \), let

\[
c_1 = \frac{x - c_5 d_5 - c_4 d_4 - c_3 d_3 - c_2 d_2}{d_1}
\]

If \( c_1 \) is an integer then it is used as a possible solution, but if \( c_1 \) is not an integer then the current set of coefficients is abandoned. The limits imposed on the coefficients by this structure ensure that all sets of coefficients are selected such that \( \sum_{i=1}^{5} c_i^2 \leq 51 \). Therefore, no set of coefficients other than those selected by the algorithm could possibly yield \( \sum_{x=1}^{40} MSSIC(x) \leq 90 \).

3. Of all possible sets \( D \), those for which \( \sum_{x=1}^{40} MSSIC(x) \) is minimum are recorded as solutions.