Non-Standard Heuristic Search Applied to the Syndrome Decoding Problem

Andrew P. Woods

3rd Year Project 2004
Department of Computer Science
University of York

$\approx 15,500$ words counted by WC

Abstract

The concept of zero-knowledge schemes was introduced by Goldwasser, Micali and Rackoff in 1985, with practical schemes being conceived in later years. These schemes all have in common, as with most public-key systems, a reliance upon arithmetic operations upon large numbers. An identification scheme based on syndrome decoding was introduced by Stern in 1994 which does not have this reliance. Quite recently successful attacks, using a non-standard optimisation based technique which takes into account the actual dynamics of the optimisation process, have been carried out by Knudsen and Meier upon a problem with certain similarities; the Permuted Perceptron Problem. The technique has considerable power and so was implemented in a cryptographic too and used to attack the syndrome decoding problem. Unfortunately the technique did not prove to be as successful in attacking this problem as it was against the PPP.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Security is a concern for many people. Physical security, financial security, and now the increasing use of the Internet has made communications security of critical importance. Every day millions of on-line transactions take place which depend upon secure communication; e-commerce, e-banking, e-mail to name just a few.

Many people in the world will try to gain advantage in situations where security is lax, and the relative anonymity the internet offers can make these opportunities more tempting. The sheer volume of people involved in these transactions presents new challenges. New protocols need to be developed to allow people to communicate securely without requiring a detailed understanding of the underlying systems. The cryptographic community is continually at work to develop such protocols, in addition to the software systems required, and make them as secure as is possible.

Obviously it is necessary for these systems to be difficult to break. In history ciphers started life extremely simply, as far back as the time of the Romans, with primitive substitution ciphers (sometimes known as Caesar ciphers). Cryptography did not make much progress until 1467, when the first (published) polyalphabetic cipher was invented by Leon Alberti. Probably the most famous polyalphabetic substitution cipher was invented in the 20th century by Arthur Scherbius for use by Germany in the 2nd World War, known as the Enigma code. A team of British cryptanalysts, Alan Turing at the forefront, changed the course of the Second World War by successfully attacking the cipher and in the process created the foundation for the modern computer.

One branch of cryptographic systems relies on the computational intractability of solving certain hard Math problems. Traditionally known is the Knapsack public key cipher, which has now been repeatedly broken. Less well known schemes rely upon problems such as constrained linear equation solving. In particular there are protocols such as zero knowledge identification schemes which are based on the Permuted Perceptron Problem, Permuted Kernel Problem and finally, and more recently, the problem of Syndrome Decoding; taken from the field of error correcting codes.

Given the previous history of the Knapsack based schemes, for example the polynomial-time algorithm created by Shamir and the more recent successful optimisation based attacks upon the Permuted Perceptron Problem, simply assuming the intractability of these problems seems questionable. This project examines the application of these more modern techniques upon the Syndrome Decoding problem, and in doing so will verify the apparent difficulty of solving it.

The project report is structured as follows:

- Chapter 1 - Introduction
- Chapter 2 - Optimisation and Cryptanalysis
- Chapter 3 - Designing A Cryptanalytic Framework
- Chapter 4 - Experimenting With The System
- Chapter 5 - Evaluation
- Chapter 6 - Conclusions

It has been attempted, where possible, to format the paper in a similar fashion to and use similar notation to the more influential papers referenced in the bibliography. This has been done to facilitate the ease of reading the other papers in the context of this report.

Chapter 2

Optimisation And Cryptanalysis

2.1 Optimisation

Heuristic optimisation techniques such as genetic algorithms and simulated annealing have shown their worth over a great number of engineering disciplines. It is applicable to any problem which can be expressed as a scalar field, i.e. a scalar function over a vector domain. The level of success in solving a problem cannot, of course, be guaranteed but practical application has proved the technique to be useful.

2.1.1 Hill-Climbing Algorithms

Hill-climbing optimisation techniques are derived from classical mathematical methods developed in recent centuries. Essentially, these methods move toward an optimum by following an uphill path within the current points neighbourhood, basically following a positive local gradient. This has the advantage of reaching an optimum relatively fast, but there are also several drawbacks. Firstly, it is assumed that the search space is reasonably 'continuous' or smooth. Unfortunately this is not the case in many real world situations, especially in the field of cryptology where the space is intentionally noisy and discontinuous.

The second disadvantage of hill-climbing techniques is that they are only capable of finding the optima in the neighbourhood of the start point; they in no way consider the global picture (see Figure 2.1). Starting a number of hill-climbing runs at multiple points in the search space offers a way forward, but this still suffers from having no guarantee of finding the global optimum, especially in high dimensional problems or where there are large number of local optima.

There are both deterministic and stochastic variations on hill climbing. In the deterministic variant the entire neighbourhood of the current state is searched for the state with the greatest improvement. A stochastic technique would choose a random improving state from the current state's neighbourhood. This has the advantage that it can very quickly ascend close to an optimum, but cannot easily tell if it has reached it, whereas the deterministic variant immediately knows when an optimum has been found.

2.1.2 Simulated Annealing

Simulated annealing (**SA**) is an optimisation process based loosely upon the process of the physical annealing of cooling metals. For any state in the search space there is an associated energy, mapped by some function usually denoted E. This corresponds the physical state of the particles having an energy level determined by its configuration.

In the process of finding the state with the minimum energy, only small changes can be made to go from one state, say S, to the next state S' (Algorithm 1, part 2(a)i). So the state S' must be in what is referred to as the *neighbourhood* of S. More precisely $S' \in Neighbourhood(S)$, where $Neighbourhood(S)$ is the set of all states 'close' to S.

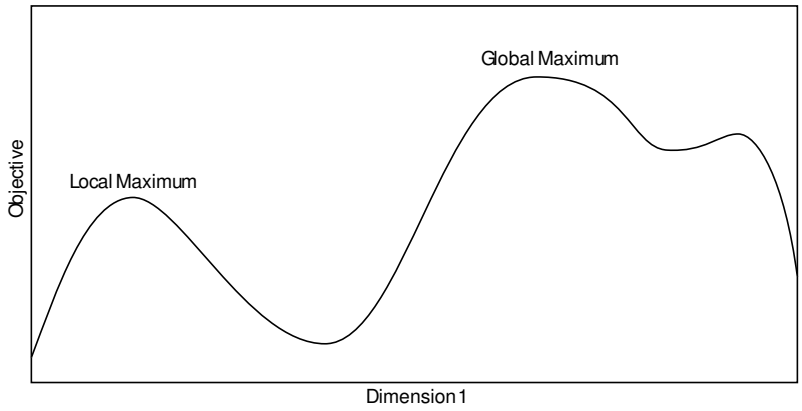

Figure 2.1: Global vs Local Maxima

Although similar, **SA** differs from hill-climbing techniques because it has the capacity to accept worsening moves. To progress from the current state S another state S' is randomly chosen from $Neighbourhood(S)$. If the new state S' has a lower energy, i.e. $E(S') < E(S)$, then it is accepted and becomes the current state S. This forms the hill-climbing part of the algorithm. If $E(S') \geq E(S)$ then there is a chance that the state will be accepted, depending upon how much higher $E(S')$ is than $E(S)$ and also upon another parameter T, the *temperature* (See below). The greater the difference between $E(S')$ and $E(S)$, the less likely the move is to be accepted.

As with the physical annealing process, **SA** has a temperature T which controls the probability of accepting worsening moves. Initially the temperature is set high so that almost any move will be accepted (Algorithm 1, part 1(a)), practically it is usually increased until at least a certain percent (something like 95%) of moves are acceptable. Then it is gradually reduced in a geometric fashion with some parameter $\beta \in [0, 1]$ (Algorithm 1, part 1(b)ii) so that it becomes harder and harder to accept worsening moves. This is done until the problem 'freezes'; when there is effectively zero probability of a worsening move being accepted.

SA will be capable of finding an optimal state in the same situations as a hill-climbing algorithm due to its operation at low temperatures. It has the additional advantage of being able to search globally when the temperature is high which means that when there are multiple optima it is more likely to find the global optimum than the basic hill-climbing approach. The downside of this additional versatility comes at the cost of running time; **SA** will spend time searching globally at the start whereas basic hill-climbing algorithms will not.

2.1.3 Genetic Algorithms

Genetic algorithms (**GAs**), as exemplified in 2, are inspired by the natural biological process of evolution. More specifically by the similarity between a real world chromosome and a state from a search space. A 'population' of states is maintained by the **GA**. The states within this population are analogous to the chromosomes of real world creatures in the same way the elements of data within the states are analogous to alleles. An initial population of states is generated randomly and the states ranked by a fitness function (similar to the energy function from **SA**). Some of the states are then selected for *reproduction* based on their fitness; the states with more desirable fitness being given greater priority. When the states are interbred, the natural process of *crossover* is emulated; the newly created child-state taking information randomly from parts of both parent states. To prevent stagnation within the population, there is a low possibility that the child state may undergo *mutat*ion. So a new population is formed by the children of the fittest states from the old population. This process is then iterated, and a continually evolving population is run through a number of generations.

Algorithm 1 Simulated Annealing

1. proc sa(S_0)
   (a) find initial T by doubling it until InnerLoop(S_0, T) accepts 95% and let $S \leftarrow S_0$
   (b) while (not frozen) do
      i. InnerLoop(S, T)
      ii. set $T \leftarrow T \times \beta$
      iii. if ($E(S)$ low enough) return S
   (c) return S
2. proc InnerLoop(S, T)
   (a) for (do n times) do
      i. set $S' \in Neighbourhood(S)$, $\Delta \leftarrow E(S') - E(S)$
      ii. if ($\Delta < 0$) then set $S \leftarrow S'$
      iii. otherwise if $\left(e^{\frac{-\Delta}{T}} > rand\right)$ then set $S \leftarrow S'$

Reproduction

Reproduction is carried out by the *reproduction operator* which selects individual states to be used for possible inclusion in the next generation. The chance a state will be chosen is based upon its fitness, determined by the fitness function. For each generation, the reproduction operator chooses strings that are placed into a mating pool, which is used as the basis for creating the next generation. An example population is as below in Table 2.1.

State	Fitness	Percentage of total fitness
810459	2	$4.5 \approx 100 \times (2 \div 44)$
140801	21	$47.7 \approx 100 \times (21 \div 44)$
210168	12	$27.3 \approx 100 \times (12 \div 44)$
532569	9	$6.3 \approx 100 \times (9 \div 44)$

Table 2.1: Example **GA** population of size 4

There are vast number of different types of reproduction operators, but most common sense method is the Roulette Wheel method, one other is Tournament Selection. In the Roulette Wheel method the states are allocated a space on a 'roulette wheel' of size proportional to their fitness. In population from the table for example, the state '140801' is the fittest and should be allocated a 47.7% 'slice' of the wheel. '810459' is the weakest and should only be allocated a 6.3% slice. When selecting the n states for the mating pool, the wheel is spun n times, with the result of each spin indicating the state to be added to the pool.

Crossover

The *crossover operator* now mimics biological mating in terms of the mixing of chromosomes from parents to produce new chromosomes for the child. The **GA** selects two states from the mating pool at random to be parents. It is then decided if crossover should take place or not randomly using a parameter called the *crossover probability*; simply a scalar value $p \in [0, 1]$, used by comparing it to a random number also between 0 and 1.

If the **GA** decides not to perform crossover, the two selected states are simply copied into the new population. If crossover does take place, then there are a variety of methods for combining the two states. A popular method for this is as follows. The two parents's states, S_1 and S_2, are

Algorithm 2 Genetic Algorithm

1. generate initial population P
2. for (n generations) do
   (a) set $P' \leftarrow \{\text{top fittest members from } P\}$
   (b) for all $(a, b \in P')$ do
      i. set $c \leftarrow breed\,(a, b)$
      ii. set $c \leftarrow mutate\,(c)$
      iii. add c to P'
   (c) set $P \leftarrow P'$

split at a randomly selected splicing point to give $S_1 = S_{1,1}S_{1,2}$ and $S_2 = S_{2,1}S_{2,2}$. The rightmost halves of each are then exchanged to give two children, $C_1 = S_{1,1}S_{2,2}$ and $C_2 = S_{1,2}S_{2,1}$. These children are then added to the new population.

Mutation

During the **GA** search, it is possible that the population starts to contain a large number of copies of the same state. If this happens the population can quickly be overwhelmed and end up entirely consisting of many copies of this one state. This can be overcome by introducing a *mutation operator*. This is another probability, m, which dictates the frequency at which mutation occurs and is usually performed during crossover.

For each element of each state in the mating pool, the **GA** randomly decides if a mutation should be performed. If it should then the element is assigned a new value randomly. In the case of decimal strings (as in the example), the digit is changed to any of the other 9 possible values in the range $0 \ldots 9$. For example, if the **GA** decided to mutate element 5 from the state '123456' the resulting state could be '1234596'. The probability should be set very low ($\approx 0.07\%$) as a high mutation rate would devastate every state and the **GA** would be tuned into a random search.

2.1.4 Applications of Optimisation To Cryptology

Optimisation is used in the field of cryptology to automate the the cryptanalysis process. Cryptographic systems are roughly speaking black boxes which produce an output for a given input together with a key. Cryptanalysis of these systems involves attempting to find this key. Given an appropriate heuristic/energy function which derives a scalar value representing the 'closeness' of the guessed key to the actual key from the algorithm's output we have a scalar field defined. This is all that is required for an optimisation framework to begin attacking a particular problem. Obviously such a trivialisation of the system cannot successfully attack a real problem, but the general idea is conveyed.

2.2 Cryptanalysis of Classical Ciphers

2.2.1 Classical Substition Ciphers

A *monoalphabetic substitution cipher* is a cipher in which every plaintext character in the plaintext is mapped to a corresponding element of the ciphertext alphabet by a one to one substitution function. It is very common for the plaintext and ciphertext alphabets to be the same, as differing them has no effect on the security of the system. An common example of this kind of function is the 'ROT13' function which is defined in fig. 2.2.

The plaintext "abcde" would be transformed to the ciphertext "nopqr" for this function. To decrypt the ciphertext an inverse function is required; this is very easy to derive from the original function, and in the case of ROT13 $f\,(x) = f^{-1}\,(x)$.

$$f(x) = \begin{cases} n,\ x = a \\ o,\ x = b \\ p,\ x = c \\ q,\ x = d \\ r,\ x = e \\ \vdots \\ m,\ x = z \end{cases}$$

Figure 2.2: Example substitution function

An efficient way to attack a cipher of this kind is described in [Jak95]. In the paper a hill-climbing algorithm was applied using a cost function $F(t) = \sum_{ij} |D_{ij}(t) - E_{ij}|$, where $D(t)$ is defined as the matrix of digram frequencies of the text t. Additionally E_{ij} is the matrix of digram frequencies of an 'average' piece of English text; in this case the text from the novel "Moby Dick" was used. In other words the cost is the sum of the differences between all the elements in the digram frequency matrices. This method will work well in solving the cipher, but in order to name itself 'fast' a nice trick was used so that the digram frequency matrix D need only be generated once. This works because of the simplicity of the cipher, the effect on D of changing the key by a small amount is easily deducible.

A *polyalphabetic substitution cipher* is a set of *monoalphabetic substitution ciphers* used looped through sequentially for each character. The cipher was designed to improve upon the original ciphers by eliminating the patterns in digram frequencies throughout the ciphertext. It is possible though to find the 'period' (i.e. the number of monoalphabetic ciphers used) of the cipher and then make attempts on the key by looking at the characters congruent to this period. The cost function needs to be changed little to achieve success with this kind of cipher.

2.2.2 Classical Transposition Ciphers

Classical *transposition ciphers* operate on the plaintext by permuting the characters in some way to create the ciphertext. Usually a simple transposition cipher will split the plaintext into fixed size blocks and permute each of these in the same way according to some permutation function P. Unfortunately employing this technique leads to the unigram frequencies of the ciphertext being exactly the same as those of the plaintext. Additionally the size of the blocks the cipher uses is known as its *period*. So for example if the period was 7 and the permutation function P was defined as in fig 2.3 then the plaintext would be split up into blocks of 7 characters and each of them operated upon by P. So the character in the 1^{st} position would be moved to the 5^{th} position and so on. Obviously if the permutation function P is known then it is very simple to construct the inverse function for decryption P^{-1}.

$$P = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 3 & 4 & 1 & 7 & 2 & 6 \end{pmatrix}$$

Figure 2.3: Example permutation function with a period of 7

In order to attack transposition ciphers Dimovski and Gligoroski [DG03] applied three different heuristic search techniques; **SA, GA** and Tabu Search (**TS**). As with the attacks upon substitution ciphers a heuristic comparing the digram statistics of attempted decryptions against the expected English digram statistics was used (Fig. 2.4). Where again $D(t)$ is the matrix of digram frequencies of the text t. The attacks were successful in that in the majority of cases almost all of the key could be retrieved (13 out of 15 characters set correctly) and from this point an enumerative search of the remaining possibilities should be trivial.

$$F(t) = \sum_{ij} \left| (D_{ij}(t))^R - (E_{ij})^R \right|$$

Figure 2.4: Heuristic used in attacking classical transposition ciphers

2.3 Cryptanalysis of Modern Minimalistic Schemes

Almost all viable public-key schemes are based on hard problems from number theory. This has remained the case still with the zero-knowledge identification protocols proposed in 1985 by Goldwasser, Micali and Rackoff [GMR85]. A practical example of one of these schemes was soon suggested in 1986 by Fiat and Shamir [FS87]; a scheme based on zero-knowledge proofs of quadratic residuosity to establish user identity and digitally sign documents. As is the case in the majority of public-key cryptography, the scheme involves manipulating large numbers. The same applies for its decedents [GQ88, OO89, OS91].

More recently, it has become desirable to create a system which relies upon a problem using only simple arithmetic operations of small numbers. Problems of this kind are considered useful as they have the potential to be deployed upon a wide variety of platforms including highly resource constrained environments such as smart cards owing to the minimal nature of their memory and arithmetic requirements. The first examples of this were in 1989 [Ste90, Sha90] when two attempts were made to make identification protocols that used only this simpler arithmetic. Of these two only the second was really viable; it was based upon the Permuted Kernel Problem (**PKP**). Since then there has been Pointcheval's proposed new identification scheme [Poi95] based on the so called Permuted Perceptrons Problem (**PPP**) in 1995. Following on from this, Jaques Stern suggested two schemes [Ste94, Ste97] in 1993, one based on Syndrome Decoding (**SD**) taken from the field of error correcting codes, and the other Constrained Linear Equations (**CLEs**).

In order to attack one of these systems it is only necessary to cryptanalyse the underlying problem it is based on, so attacks that have been carried out on the actual $\mathcal{NP}$-complete problems are all that are of concern. Hopefully some inspiration can be gained from a combination of the techniques which have been applied previously.

2.3.1 The Permuted Kernel Problem

In 1989 Shamir introduced an identification scheme [Sha90] based on the intractability of the Permuted Kernel Problem (**PKP**). The **PKP** is defined as follows.

Permuted Kernel Problem

In this problem all arithmetic operations are carried out modulo p, where p is a small prime.

Definition 1. *Permutations:*

Vector : *If v is an n-vector and π is a permutation over $\{1,\ldots,n\}$ then the vector permutation v_π is defined as the vector w such that $w_j \equiv v_{\pi(j)}$ for $1 \leq j \leq n$.*

Matrix : *If A is an $m \times n$ matrix and π is a permutation over $\{1,\ldots,n\}$ then the matrix permutation A_π is defined as the matrix B such that $B_{i,j} \equiv A_{i,\pi(j)}$ for $1 \leq i \leq m$, $1 \leq j \leq n$.*

So for any matrix A and vector v $A_\pi v_\pi = \sum_{j=1}^{n} A_{i,\pi(j)} v_{\pi(j)} = \sum_{j=1}^{n} A_{i,j} v_j = Av$.

Definition 3. *Composition of permutations:*

If v is an n-vector and π, σ are permutations over $\{1,\ldots,n\}$ then the composed permutation $v_{\pi\sigma}$ is defined as the vector w such that $w_j \equiv v_{\pi(\sigma(j))}$ for $1 \leq j \leq n$.

Definition 4. *Kernels:*

If A is an $m \times n$ matrix then the kernel of A, $K(A)$, is the set of n-vectors W such that $AW = 0$, where 0 is the m-vector of zeros.

It is easy to see that $K(A)$ is a linear subspace of $\mathbb{Z}_p^n$ and that $K(A_\sigma) = (K(A))_\sigma$.

Definition 5. *Permuted Kernel Problem:*

Input : *A $m \times n$ matrix A, a n-vector v, and a prime p.*
Problem : *Find a permutation π such that $v_\pi \in K(A)$.*

The **PKP** has been proved to be strongly $\mathcal{NP}$-complete [MRG79], which allows small numbers to be used in the implementation of the identification system. This simplifies and speeds up the system considerably without reducing its intractability to an insecure level.

Analysis of The Problem

Shamir reasoned that good values for the parameters are n between 32 and 64, prime $p = 251$ (for 8-bit machines), m based on $p^m \approx n!$. This was because at that time (1989) the best known attacks for those sizes of problems ranged in complexity from 2^{76} (for $n = 32$) to 2^{184} (for $n = 64$). But in 1992, two new algorithms to solve the **PKP** were suggested; one by T. Baritaud, M. Campana, P. Chauvard and H. Gilbert [TBG92] and another independently found by J. Georgiades [Geo92]. The former used a memory trade-off to improve time complexity, storing the solution to many tuples in memory for later access. Unfortunately this trade off was very large, for example the solution of a problem of size 32 requiring (say $k = 6$ and $l = 10$ were chosen) around $\frac{32!}{22!} \approx 2^{47.7}$ 10-uples to be stored. The running time of this algorithm is low enough to show that the smaller sizes originally suggested as secure ($n = 32$) are infact insecure.

More recently in 1998 Patarin and Chauvard have improved further upon these results [PC94] by combining both the ideas from [TBG92, Geo92] and some of their own. This resulted in an algorithm which has a significantly lower running time than both of the other algorithms and additionally has far lower memory requirements than the one in [TBG92]. The initial combination of ideas creates in an algorithm the same as [PC94], but with one variable less by using the same Gaussian elimination but with one equation extra. This is then further improved by grouping the variables into sets which can only take part in one of the left or right sides of the equation solution. At this point their algorithms split into a number of different versions, each good at a particular size of the **PKP**. Their algorithms successfully attacked problems of size $n = 32$, but for the larger sizes ($n = 64$) the time complexity was only reduced to 2^{116}, which is still not enough to be practical. Though the possibility for large parallelisation could increase the feasibility.

2.3.2 The Perception Problem and Permuted Perceptron Problem

In 1995 Pointcheval introduced an identification scheme [Poi95] based on a variant of the Perceptrons Problem (**PP**) known as the Permuted Perceptrons Problem (**PPP**). Even though it is possible to create a zero-knowledge protocol using any $\mathcal{NP}$-complete problem, this problem is much harder to solve than the original **PP** so it is quicker and easier to construct a protocol using it. The problems are defined as follows.

Perceptron Problem

As mentioned in Pointcheval's paper, the **PP** appears in physics and in artificial intelligence with certain kinds of learning machines.

Definition 1. *An ϵ-vector (or matrix) is a vector (or matrix) of which all the entries are ± 1.*

Definition 2. *The Perceptrons Problem:*

Input : *An ϵ-matrix A of size $m \times n$.*
Problem : *Find an ϵ-vector S of size n such that $(AS)_i \geq 0 \ \forall \ i = 1, \ldots, m$.*

This problem is hard to solve as Pointcheval showed in [Poi94].

Permuted Perceptron Problem

Pointcheval chose this variant of the **PP** because it was quicker and easier to make a secure identification protocol based on it than on the original **PP**.

Definition 3. *The Permuted Perceptrons Problem:*

Input : *An ϵ-matrix A of size $m \times n$.*
A multi-set I of nonnegative integers, of size m.

Problem : *Find an ϵ-vector S of size n such that* $\left\{\left\{(AS)_j : j = \{1, \ldots, m\}\right\}\right\} = I$

Obviously, all solutions of the **PPP** are also solutions of the **PP,** and there are further restrictions on the solutions, thus the **PPP** must be harder to solve than the **PP**.

Analysis of The Problem

Pointcheval proved theoretically that an S of size $n \approx m + 16$ was optimal for maximising problem difficulty. In order to find a secure key size, Pointcheval tried a few different attacks, the most successful of which being Simulated Annealing (**SA**) with energy function of the form 2.5, against the **PPP**. The conclusion was reached that a value of $m \geq 101$ to be of sufficient complexity to make the problem secure.

$$E(V') = \sum_{i=1}^{m} max\left\{-\left(AV'\right)_i, 0\right\}$$

Figure 2.5: Pointcheval's energy function for attacks on the PP

In 1999 Knudsen and Meier [KM99] attacked the problem using a more unusual approach. They discovered there was a large amount of significant information to be gained from the properties of sets of the results from multiple runs of the **SA** algorithm. It was noticed that if over the set of say t results, if the i^{th} entry is of the same sign all t times then it is very likely that the entry has its sign set correctly. This information proved extremely useful, with some powerful techniques based upon it being developed. Running another batch of **SA** runs with the recently discovered entries either fixed or initially set to their likely value. Repeating this process until a sufficient number of entries have the correct sign to allow an enumerative search of the remaining possibilities proved to be an effective method, as Knudsen and Meier demonstrated.

$$E(V') = g\sum_{i=1}^{m}\left(max\left\{K - \left(AV'\right)_i, 0\right\}\right)^R + \sum_{i=1}^{n}\left(\left|H(i) - H'(i)\right|\right)^R$$

Figure 2.6: Clark and Jacob's new 'warped' energy function for attacks on the PPP

More recently Clark and Jacob [CJ02] have advanced from what Knudsen and Meier originally discovered by employing two further very non-standard approaches; actually taking into account the dynamics of the **SA** optimisation process. One is based around the idea of security fault injection [DBL97]; it was named Problem Warping in this case. Essentially a different energy function is used in the optimisation process. This energy function, which is the first half of 2.6, looks unlikely to produce an actual solution for the problem, but as it is closely related to the original energy function the results can be more highly correlated with the actual solution than those obtained with the original one (2.5). Some very successful results for the attack on the **PP** are produced when the parameters g, K and R are properly tuned; most results for even large problems being only 1–3 bits away from the actual solution. In addition, multiple runs of the optimisation process with differently warped versions of the problem can reveal more information

about the secret. To attack the **PPP** further refinement is required, so Clark and Jacob made another change to the energy function (2.6), this time taking into account the hamming difference between the current histogram and the expected histogram of secret vector entries. This new function allows attacks against the **PPP** to be possible. The second non-standard approach made is a timing analysis channel [Koc96]. In this the status of each entry of the current solution vector is monitored throughout the optimisation process and the last time each changes is recorded. This information proved to be extremely useful, for example some entries can get 'stuck', i.e. remain fixed at a particular value and never change from that point on, early on in the optimisation process. Analysing when this happens can reveal over half of the secret vector's entries in only a single **SA** run. When a large set of runs is observed, if the early stuck secret vector entries agree, then this further concretes the likelyhood they are correct. Unfortunately, sometimes even if over large sets of runs the a particular entry agrees when stuck early, it is possible that it has stuck incorrectly each time, and an enumerative search must be made for these bits.

2.3.3 Syndrome Decoding

Stern proposed a new zero knowledge identification scheme [Ste94] in 1994 based upon the syndrome decoding problem for error-correcting codes.

Syndrome Decoding Problem

In this problem all arithmetic takes place in the two-element field.

Definition 1. *Matrix xor operator:*

If A is an $m \times n$ matrix and S is an n-vector then $A \oplus S$ is defined as follows:

$$A \oplus S = \begin{pmatrix} [(A_{1,1}) \wedge (S_1)] \oplus [(A_{1,2}) \wedge (S_2)] \oplus \ldots \oplus [(A_{1,n}) \wedge (S_n)] \\ [(A_{2,1}) \wedge (S_1)] \oplus [(A_{2,2}) \wedge (S_2)] \oplus \ldots \oplus [(A_{2,n}) \wedge (S_n)] \\ \vdots \\ [(A_{m,1}) \wedge (S_1)] \oplus [(A_{m,2}) \wedge (S_2)] \oplus \ldots \oplus [(A_{m,n}) \wedge (S_n)] \end{pmatrix}$$

Definition 2. *The Syndrome Decoding Problem:*

Input : *A matrix A of size $m \times n$.*
An m-vector I.
A positive integer $p = \frac{n}{8}$.

Problem : *Find a vector S of size n such that $A \oplus S = I$ and S has p ones.*

Analysis

In order to indicate the difficulty of solving this problem Stern noted that in addition it being strongly $\mathcal{NP}$-complete, under the right choice of problem parameters its solution is analogous to the problem of decoding unstructured codes, which is currently believed to be unsolvable. He also noted that at the time (1994) the best known algorithms for its solution have a computing complexity which grows exponentially. Furthermore a correct asymptotic evaluation of the problem was performed in [Cha93] which has also been confirmed with empirical tests. Therefore, as Stern did, it is possible to state an intractability assumption for the **SD** problem.

Searching for non-standard heuristic based attacks has returned no results for this particular problem. Because of this the **SD** problem has been chosen for analysis in the project. It is hoped that some of the recently developed techniques may allow insights into the problem's structure.

Chapter 3

Designing a Cryptanalytic Framework

The Syndrome Decoding (**SD**) problem has been chosen for analysis. In order to best attack **SD** it is intended to put together a cryptanalytic framework capable of detailed analysis of the problem. Although its primary use will be the cryptanalysis of **SD**, the framework will be designed to be as general as possible so that future work may be carried out requiring as few changes as possible to its structure.

3.1 Overview

In the cryptanalysis of any problem there are a number of vital abilities a cryptanalytic framework must have:

- A method of generating instances of the problem with *known* solutions.
- An actual optimisation rig, i.e. an implementation of one or more optimisation algorithms.
- A specification of the problem for the optimisation rig to work with. This is information such as how to perturb a state a small amount (for finding a next state in the optimisation process) and a heuristic to approximate the 'goodness' of any particular state.
- A post-processing setup in order to produce useful statistics about the many optimisation runs.
- A method of visualising this data in some way.

There is a natural order for these processes so logically a system with a structure of that of fig. 3.1 is produced.

3.2 How to Generate a Problem Instance

Since it is not possible (if it were then there would be no point to the analysis this project undertakes) to find the solution to any given instance of **SD**, a method of generating instances of the problem with known solutions is necessary. From the definition in 2.3.3 a problem instance requires the following:

- A matrix A of size $m \times n$ (usually $n = 2m$).
- An m-vector I.
- A positive integer $p = \frac{n}{8}$.

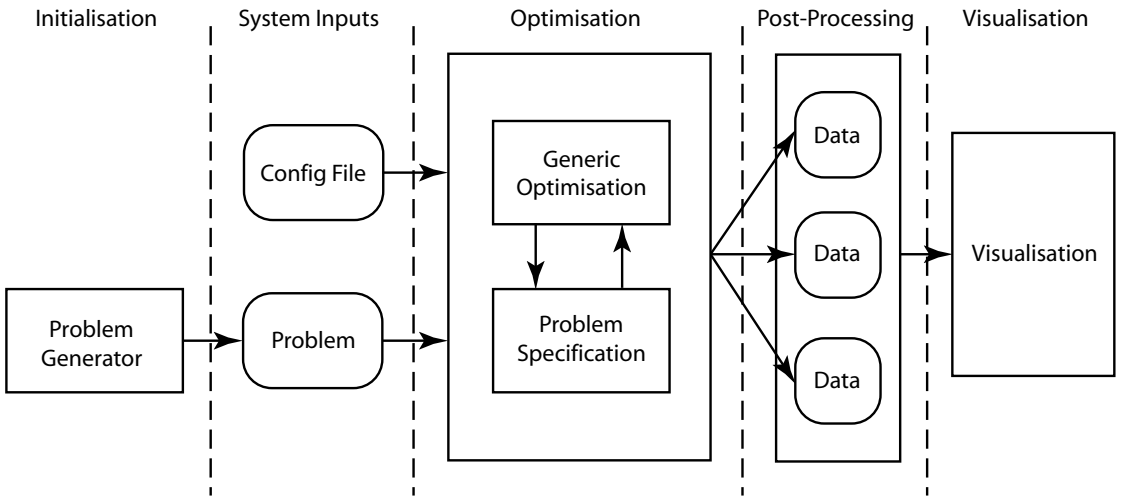

Figure 3.1: System flow diagram

Algorithm 3 Randomisation of vector over the two element field

1. proc randomise(S)
   (a) Set first p bits of S to 1
   (b) Set remaining bits of S to 0
   (c) for $i = 1 \ldots n$
      i. switch S_i with random bit of opposite sign

The matrix A is generated to consist of half 1's and half 0's simply by using a random number generator compared against 0.5 for each element. This approach was taken as guaranteeing precisely half 1's would make no significant difference to the 'randomness' of the matrix. The m-vector I cannot be generated directly because as previously mentioned this would require the solution of a problem instance to be carried out so instead an n-vector S, the secret vector, is generated with p 1's as the problem definition requires. This is done by starting with a vector of appropriate size of the form.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

i.e. p 1's followed by $(n-p)$ 0's. Then each bit is in turn randomly swapped with with another opposite bit; so if it is a 1 then it is swapped with a randomly chosen 0 and vice-versa. See algorithm 3 for pseudocode . Now to generate I the matrix xor operator is used to xor A with S, so $I = A \oplus S$. So now a random problem instance has been created with a known solution S.

Additionally to generating a problem each time the system must have the ability to save and load problem instances. This is a trivial to implement feature but obviously of importance.

3.3 An Optimisation Rig

3.3.1 A Simulated Annealing Implementation

The most important part of this component is the Simulated Annealing (**SA**) implementation, which will be of a very typical construction in the **SD** cryptanalysis. The implementation is designed to be as generic as possible while not sacrificing its power, so that another problem specification can be simply 'plugged in' to allow analysis of a different problem. The specific refinements made to the data structures related to the **SD** problem will be covered later.

Algorithm 4 Simulated Annealing With Timing Channel

1. proc sa(S_0)
   (a) find initial $T \leftarrow \alpha$ by doubling it until InnerLoop(S_0, T) accepts 95% and let $S \leftarrow S_0$
   (b) set timer $Time \leftarrow 0$
   (c) while (not frozen) do
      i. InnerLoop($S, T, Time$)
      ii. increment timer $Time \leftarrow Time + 1$
      iii. set $T \leftarrow T \times \beta$
      iv. if ($E(S)$ low enough) return S
   (d) return S
2. proc InnerLoop($S, T, Time$)
   (a) for (do n times) do
      i. set $S' \in Neighbourhood(S)$ and let i be the bit which changed, $\Delta \leftarrow E(S') - E(S)$
      ii. mark on S' the time bit i changed with $Time$
      iii. if ($\Delta < 0$) then set $S \leftarrow S'$
      iv. otherwise if $\left(e^{\frac{-\Delta}{T}} > ran\right)$ then set $S \leftarrow S'$

To achieve optimal performance from the **SA** the optimisation parameters (β, n) must be tuned. It is generally accepted that for the temperature reduction factor a value of $0.9 < \beta < 0.99$ is preferable. The higher values are more desirable if the evaluation function for the **SA** is rapid enough to allow the increased number of iterations to be possible in a reasonable amount of time. Similarly a value of $n \approx 400$ for the internal iteration count is normally used, but again higher values will present better results if it is possible for the **SA** still to run in a reasonable amount of time. With the **SA** as in algorithm. 4 step 1(a) the initial temperature parameter α is irrelevant as the appropriate temperature for boiling point is found by increasing the temperature until almost any potential move (95% of a selection of 400 random moves) is accepted.

For the non-standard analysis of the problem a *Timing Channel* is required; essentially this is a counter which increments, algorithm. 4 step 1(c)ii, at each temperature level. Using this counter it is then possible to mark, algorithm. 4 step 2(a)ii, on the current key S the time at which any of its components change. This information can be used, as it will be later, to analyse the actual dynamics of the **SA** optimisation process.

3.3.2 Configuration

The optimisation rig has the ability to both generate and attack **SD** problems. Given a directory it will load the following files to establish the appropriate problem instance. It will also write out these files using the procedures created for the problem instance generation after it has been instructed to generate a problem instance. The user specifies the number of desired **SA** optimisation attack attempts to make upon the problem instance.

a.dat *Specifies the ϵ-matrix A:*

ϵ-matrix A : The actual contents of the ϵ-matrix A.

Integer m : The height of the ϵ-matrix A.

Integer n : The width of the ϵ-matrix A.

This file consists entirely of a line by line write-out of the boolean values of the ϵ-matrix A. The values of m and n are established from the size of the ϵ-matrix created by reading the file.

secret.dat *Specifies the ϵ-vector S:*

ϵ-vector S : The contents of the ϵ-vector S (of size n).
Integer p : The number of 1's in S.

This file consists entirely of a single line of the boolean values of the ϵ-vector S. The value of n implied by the file is verified against the initial value taken from 'a.dat' and the value of p is established by counting the number of 1's in the file.

seed.txt *Contains the seed to use for the **SA** runs.*

An example of the contents of the preceding files can be found in table 3.1. In this particular instance m has value 8, n has value 16 and p has value 2. The matrices are as specified in the files.

a.dat	**secret.dat**	**seed.txt**
(0101011101011010)		
(1010111010110100)		
(1010100100101010)		
(0101001010100010)	(0100000001000000)	616054010648406406106840
(1010000101010111)		
(1101100110100011)		
(1001000100101001)		
(1001001010111010)		

Table 3.1: 'a.dat', 'secret.dat', 'seed.txt' example contents

3.3.3 Structure of Problem Instance

The framework uses a class to entirely contain a problem instance. An instance of this class will contain the values of A, S, m, n, p, the seed for the **SA's** stochastic elements and the total number of **SA** runs required. The problem solution can then be executed as a separate thread in its own right; allowing many problem instances to be attacked at once, or perhaps two sets of attacks on the same problem. This would allow a potentially great speedup on multi-processor machines, although none were available at the time of development so this avenue was not further pursued.

In addition to the aforementioned information the class specifies the 3 optimisation parameters for the **SA** runs; the initial temperature α, the geometric temperature schedule coefficient β, the number of internal iterations n for each temperature within the schedule.

As the **SA** runs are executed the problem object maintains a log of all the finalised states from them including all the relevent information required for the post-processing analysis stage. Once all **SA** runs are completed the problem object is stored in a file for later analysis.

3.3.4 Batching System for Bit Fixing

A useful ability for a cryptanalytic framework to have in the attacking of the **SD** problem is to be able to fix a number of components of the attempted solution vector S' throughout the **SA** run. The mechanism for achieving this efficiently is described in section 3.4. In order to use this ability to its fullest a collection of runs with bits fixed differently in each case must be analysed. This requires a method of 'batching' so that a number of these different fixings can be specified in a 'batch' file. The operation of this system is as follows (algorithm 5).

3.4 Problem Specification

The actual structure of the **SD** problem is defined by a number of classes which define both the structure of ϵ-vectors, ϵ-matrices and the operations which may be carried out upon them. This architecture allows any other, albeit necessarily similar in structure, problem to be simply introduced to the framework with a maximal amount of code re-use.

Algorithm 5 Fixed Bit Batching Operation

1. Reads "batch.dat" from given directory. This file contains a list of files to load in order for each **SA** run. Each file specifies an individual configuration of fixed bits. If multiple files are placed on any single line then they are loaded at the same time.

batch.dat
bits_1.dat
bits_1.dat bits_0.dat
foo_0.dat

2. Runs batch files in turn:
   Each of these files determines which bits of the attempted secret ϵ-vector S' should be fixed. If the filename ends in "1.dat" then it determines that the bits should be fixed to '1' dually if it ends "0.dat" then the bits are fixed to '0'. Each file contains a list of which bits should be fixed encapsulated by parenthesis and delimited by spaces in much the same manner as the secret ϵ-vector is stored.

   (a) Runs an **SA** with bits 0, 1, 5, 14 fixed to '1'

bits_1.dat
(0 1 5 14)

   (b) Runs an **SA** with bits 0, 1, 5, 14 fixed to '1' and bits 6, 10, 15, 20 fixed to '0'

bits_1.dat	bits_0.dat
(0 1 5 14)	(6 10 15 20)

   (c) Runs an **SA** with bit 31 fixed to '0'

foo_0.dat
(31)

3.4.1 Bit-Vectors

ϵ-vectors, called BitVectors in implementation, are possibly the most fundamental data structure to the system. Both the secret vector S and the image vector $I = A \oplus S$ are BitVectors in the framework specification. BitVectors are implemented as arrays of integer primitives; with each bit of the integer directly corresponding to a component of the vector. This allows native 'xor', 'and' and shift operations to be carried out which will execute considerably more rapidly than for any other possible representation.

The **SA** algorithm requires a perturbation operator which can slightly modify any given BitVector S to give another BitVector S' in the 'neighbourhood' of S, $(S' \leftarrow N(S))$. The initial thought was to choose any two random components S_i, S_j of S and exchange their values, however this proved to be unsuccessful because in the majority of cases no change would be made to $S' = N(S)$ from S. This is because S is a sparse vector, i.e. the majority of its components are 0, which meant most randomly selected components would have the same value, namely 0. It would also be inefficient to repeatedly try selecting components at random as this would waste valuable CPU time. The solution was to maintain a two arrays of component indices; one with the indices of components of value 1 and another listing the components of value 0 (see fig. 3.2). To then guarantee an actual change to S is made requires simply a random element of 'ones' and a random element of 'zeros' to be selected and the values of those indices (and the corresponding components of S) to be exchanged.

A useful ability for the BitVector specification is the ability to fix certain components of the attempted solution vector S' to particular values throughout the **SA** process. With the previously described structure in place, this is simply a matter of not including all of the components indices of S' in 'ones' and 'zeros'. Any components which are required to be fixed are simply 'left out'. Then the perturbation operation cannot select them for random exchange, so they remain fixed throughout the optimisation process.

$$ones = \begin{pmatrix} 2 \\ 5 \end{pmatrix} \qquad zeros = \begin{pmatrix} 1 \\ 3 \\ 4 \\ 6 \\ 7 \\ 8 \end{pmatrix} \qquad S = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Figure 3.2: Arrays of Components Indices For ϵ-vector S of Size 8

3.4.2 Bit-Matrices

ϵ-matrices, called BitMatrices in implementation, are implemented as arrays of BitVectors. The only operation which will be carried out upon a BitMatrix is ϵ-matrix xor defined in section 2.3.3 on page 11. This process of calculating $I = A \oplus S$ is time consuming, as the time complexity of the process is proportional to the number of elements in the matrix, so some optimisation is necessary to achieve the desired execution speed. It is actually possible to dramatically reduce the amount of calculations necessary to find I. If we have a current value of the image vector $I = A \oplus S$ and wish to find $I' = A \oplus S'$ where $S' \in N(S)$ (i.e. S' = S with components i, j exchanged), then $I' = I \oplus (a_i) \oplus (a_j)$. (a_i) is defined as the ith column of A. This requires only two pointwise vector xor operations - a vast improvement, as now the time complexity is proportional to only $2\times$ the size of the image vector. As the BitMatrix consists of arrays of BitVectors the appropriate pair of these can be extracted and operated upon using the regular BitVector operations which, as noted before, will execute very repidly.

3.4.3 Heuristic

Finally the **SA** needs a way of calculating the 'energy' of a given image vector I'. The initial function used will be the Hamming Distance between the expected image I and the current image I'. The hamming distance $H_I(I')$ is defined as:

$$H_I(I') = \sum_{i=0}^{n} |I'_i - I_i|$$

Figure 3.3: Hamming Distance

i.e. the total number of components differing in I and I'. For example if $I = (1010101110101100)$ and $I' = (1100010100011101)$, then then $H_I(I') = 4$. After further analysis attempts will be made to refine this function to more accurately represent the closeness of S' to S.

3.5 Visualisation and Post-Processing

When a set of **SA** runs on a problem instance is completed a corresponding set of data is produced. This data is a set of tuples of the following configuration:

$$(S', E(S'), T_{end}, T(S'))$$

S' is the final solution for S which the **SA** came up with; $E(S')$ is the energy level for the secret vector S'; T_{end} is the final temperature index; $T(S')$ is an n-vector of the time indexes of when each component of S' last changed. Together with the values of A, S and I this information represents an extremely useful set of information to analyse.

To produce meaningful human-readable information from this large set of data post-processing must take place in order to construct a simpler set of data. This can range from taking the mean of a particular statistic to constructing 3-dimensional scalar fields of values. This visualisation is helpful because it allows 'intuitive' analysis of the data by humans which can often lead to insights into structure in the problem. In implementation the post-processing system will be a separate program from the optimisation rig, but use many of the data classes so that all of the problem specification characteristics may be carried over.

3.6 Proposed Analysis

The following graphs are chosen because they represent the areas where it is anticipated structure in the data could potentially exist:

1. (correctness, stick time), (#bits, stick time):
   As the **SD** problem is similar to the **PPP** in many was it is expected that the correlation between bit sticking time and observed in the analysis of the **PPP** will be prevalent in the **SD** analysis.

2. (correctness, bit weight):
   Structure between the weight of a bit (i.e. the hamming weight of the i^{th} column) and the correctness is a not unreasonable possibly.

3. (av. stick time, bit weight):
   If either of the previous two graphs show correlation then correlation between bit weight and average stick time should hopefully indicate some correlation too.

4. (correctness, bit#):
   It will be necessary to determine how often each bit is set correctly in a batch of **SA** runs as the simple majority vector is hoped to correlate reasonably with the real secret vector.

5. (average stick time, bit#):
   This graph will indicate the accuracy of the majority vector.

Chapter 4

Experimenting With The System

4.1 Judging The Limit of Direct Optimisation

To discover at what sizes of problem instances direct optimisation has the potential to 5 different problem sizes were examined. Each problem was attacked individually by 10,000 separate **SA** runs. For the problem sizes 8×16, 12×24, 16×32 and 24×48 a set of twenty problem instances were created, but for the size 32×64 only ten. This smaller number of 32×64 problem instances was due entirely to time constraints, with each set of **SA** runs taking a large amount of time.

The prediction was that the vast majority of the smaller sizes of problem would be solved by direct **SA** optimisation, with the degree of success decreasing in a fairly continuous manner as the problem sizes increased. Eventually the **SA** breaks down completely and at that point the focus turns to searching for small correlations in the statistics.

All bits of the attempted secret vector were allowed to be free and the remaining parameters for the **SA** were as follows:

- Temperature reduction factor; $\beta = 0.95$
- Number of iterations before temperature reduction; $n = 400$

4.1.1 Results of Attacks On Test Sets

In this range of problem sizes at very least some **SA** attacks were able to find the solution in all problem instances. A summary of the success of the attacks for each problem size is presented in figure 4.1.

Test Set 1 (Twenty 8×16, $p = 2$ Problems)

The total number of **SA** runs which found S and totals of successfully minimised energy function (both out of 10,000) are listed in table 4.1. As is clearly visible the **SA** runs found S in all 10,000 cases for all problems except numbers 10 and 11. Apart from problems 10 and 11 these results were as anticipated, the problems being small enough so that the **SA** can find found S by random movements alone. The reason for these 'failures' is that the particular **SD** instances have multiple solutions. As is visible in the table the heuristic criteria have been met (i.e. the energy function returns zero) so the **SA** is finding the alternate solutions to the problem.

Problem	Found S	Zero Energy	Problem	Found S	Zero Energy
1	10000	10000	11	5030	10000
2	10000	10000	12	10000	10000
3	10000	10000	13	10000	10000
4	10000	10000	14	10000	10000
5	10000	10000	15	10000	10000
6	10000	10000	16	10000	10000
7	10000	10000	17	10000	10000
8	10000	10000	18	10000	10000
9	10000	10000	19	10000	10000
10	4969	10000	20	10000	10000

Table 4.1: Test Set 1 Successes

Test Set 2 (Twenty 12×24, $p = 3$ Problems)

The same as for the first set of problems, the counts of runs which found S and totals of successfully minimised energy function are listed in table 4.2. The mean number of runs that found S ($\mu_{found\,S}$=6830.4) has lowered from that of the smaller set of experiments ($\mu_{found\,S}$=9499.95) as expected, although all **SA** runs still converge to a solution with zero energy ($\mu_{zero\,energy}$=10000). This makes sense, as problem this very small, and with a high cooling factor like 0.95 **SA** should be extremely likely to minimise the energy function.

Problem	Found S	Zero Energy	Problem	Found S	Zero Energy
1	4992	10000	11	5001	10000
2	3280	10000	12	4989	10000
3	10000	10000	13	4971	10000
4	10000	10000	14	4997	10000
5	4989	10000	15	10000	10000
6	5049	10000	16	10000	10000
7	3361	10000	17	10000	10000
8	10000	10000	18	5122	10000
9	10000	10000	19	4871	10000
10	4986	10000	20	10000	10000

Table 4.2: Test Set 2 Successes

Test Set 3 (Twenty 16×32, $p = 4$ Problems)

Again the counts of runs that found S and totals of successfully minimised energy function are listed in table 4.3. There is a less significant drop in the number of runs finding S in this category ($\mu_{found\,S}$=4888.95) than with the previous one. Also this time there is the drop in the number of successfully minimised problems; no longer are all runs ending in a state with zero energy. The problem space is becoming too large for the **SA** to succeed in it every time ($\mu_{zero\,energy}$=6632.45).

It seems rational to predict from this data that given there are multiple solutions for a given problem and that **SA** is capable of solving that problem directly, then the counts of solutions found are evenly distributed between all the possible solutions. For example in problem 2 the total number of minimised runs is ≈8000 while the number of times S was found is ≈4000, suggesting there are 2 possible solutions. Again with problem 9 the number of minimised runs is ≈3200 while the number of times S was found is ≈9600, suggesting there are 3 possible solutions. To further back this conjecture all runs which find S more than 50% of the time have an identical number of minimised states, so that in all of these cases there was only solution, namely S, which successful minimisation always led to.

Problem	Found S	Zero Energy	Problem	Found S	Zero Energy
1	5768	5768	11	5555	5555
2	4260	8131	12	5594	5594
3	5598	5598	13	5205	5205
4	3918	7846	14	5486	5486
5	5110	5110	15	3761	7950
6	5591	5591	16	3791	8054
7	6205	6205	17	6026	6026
8	5096	5096	18	5608	5608
9	3308	9577	19	3048	9089
10	5787	5787	20	3064	9373

Table 4.3: Test Set 3 Successes

Test Set 4 (Twenty 24×48, $p = 6$ Problems)

Counts of runs finding S and totals of successfully minimised energy function are listed in table 4.4. This problem size is very close to the edge of what direct optimisation can solve; only 0.003% of **SA** runs succeed in finding the secret ϵ-vector S. As with the smaller problems, the numbers of successfully minimised runs are roughly multiples of the number of runs finding S indicating the likely number of solutions in each case.

Problem	Found S	Zero Energy	Problem	Found S	Zero Energy
1	29	52	11	30	52
2	29	53	12	32	32
3	25	25	13	46	102
4	18	42	14	24	24
5	29	105	15	20	77
6	22	22	16	19	19
7	29	29	17	27	91
8	25	25	18	32	32
9	24	24	19	41	41
10	19	49	20	27	50

Table 4.4: Test Set 4 Successes

Test Set 5 (Ten 32×64, $p = 8$ Problems)

From table 4.5 it is clear that direct optimisation is in no way capable of solving **SD** instances with secret ϵ-vectors of length 64 or above. In all 100,000 runs S was never found and obviously the energy function was never successfully minimised either. It is at this point that more detailed analysis must be carried out to attempt to discern if any correlations are present in the statistics.

Problem	Found S	Zero Energy	Problem	Found S	Zero Energy
1	0	0	6	0	0
2	0	0	7	0	0
3	0	0	8	0	0
4	0	0	9	0	0
5	0	0	10	0	0

Table 4.5: Test Set 5 Successes

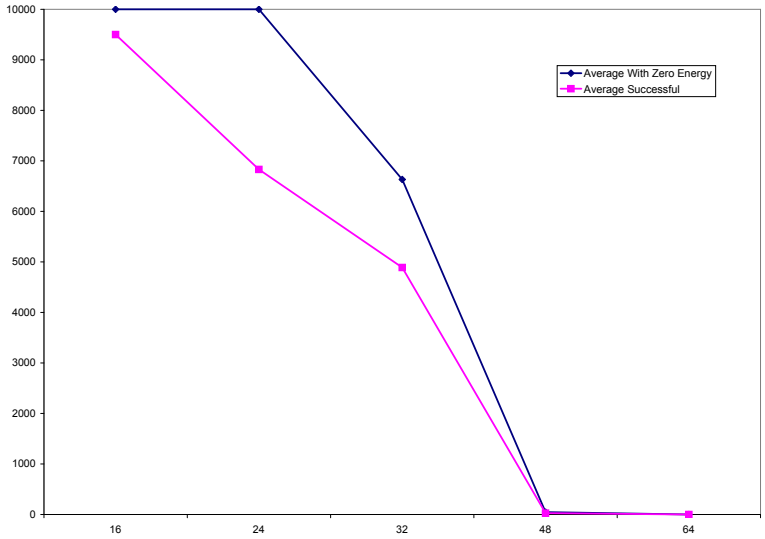

Figure 4.1: Successfulness vs Problem Size

4.2 Further Analysis of Problem 5

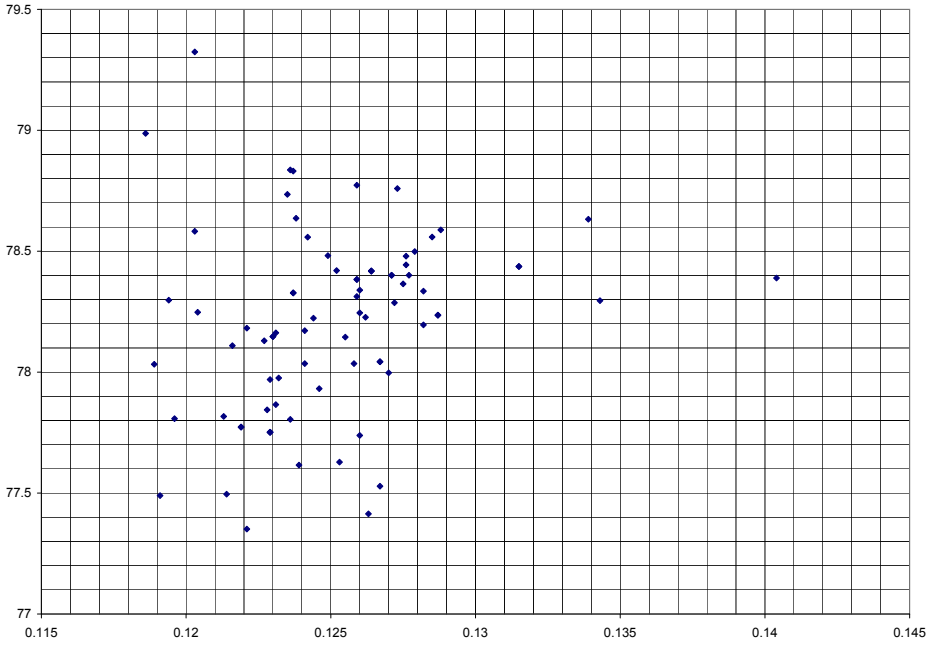

Figure 4.2: Plot of (average stick time, correctness) for each one

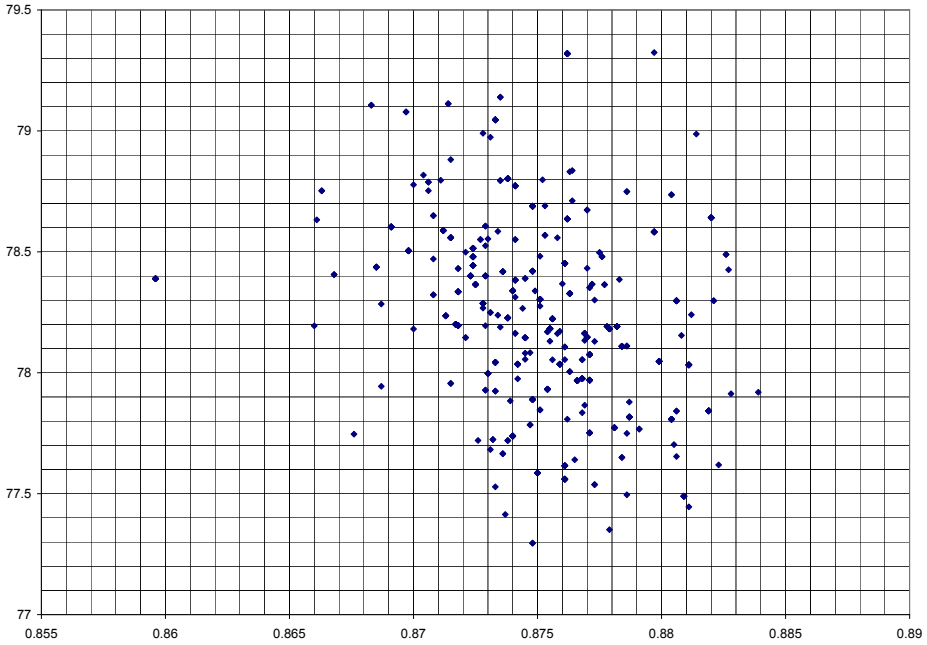

Figure 4.3: Plot of (average stick time, correctness) for each zero

4.2.1 Analysing The Distributions of Correctness

Examining figures 4.2 and 4.3 it appears that there is a reasonable possibility that the mean correctness of each bit is randomly distributed. There seems to be no discernable trend in correctness in either graph.

To determine if the distribution of majority vectors is truly random the data will be analysed using the χ^2 test. It need only be determined if the ones are random, given that the locations of zeros are determined precisely by where ones do not lie.

We assume that the end result for each 'one' is independent of all the other 'ones'. The probability on random assignment of any 'one' being correct is $p = \frac{1}{8}$ and the number of events is $n = 10000$. Given this number of correct 'ones' in a single problem (over all 10,000 **SA** runs) follows a binomial distribution $B(p, n)$. This distribution can be approximated to the following normal distribution:

$$B(n, p) \sim N(np, np(1-p))$$

The results above in figures 4.2 and 4.3 represent the mean number of successes for the size of the problem, i.e. $\mu = \frac{\sum x_i}{n}$. Now the mean μ follows the distribution:

$$\mu \sim N\left(p, \frac{p(1-p)}{n}\right)$$

Substituting in the values of p and n:

$$\mu \sim N\left(\frac{1}{8}, \frac{7}{640000}\right)$$

Define $X = \mu$, now the null hypothesis H_0 can be stated:

- X is approximately normal with mean 0.125, and standard deviation 0.003307

By the central limit theorem; provided the sample size is sufficiently large (in this case 80) the distribution of the sample mean is approximately normal regardless of the shape of the population. To carry out the χ^2 test the observed frequency distribution over around 7-8 classes. The region was divided into 7 of these equally sized classes and the observed frequencies(=O) tallied. Table 4.6 contains the observed frequency distribution.

LCB, A	≥0.115	≥0.11875	≥0.1225	≥0.12625	≥0.13	≥0.13375	≥0.1375
UCB, B	<0.11875	<0.1225	<0.12625	<0.13	<0.13375	<0.13375	<0.14125
O	1	14	37	23	2	2	1

LCB=Lower Class Boundary, UCB=Upper Class Boundary

Table 4.6: Observed Frequencies(=O) For Regular Classes

The regular classes must then be transformed into standardised normal distribution classes and the standard normal probabilities calculated for each new class. From these probabilities the expected frequencies(=E) can easily be calculated by simply multiplying the expected frequency by the number of trials(=80). This expected frequency distribution is presented in table 4.7.

$\overline{LCB}, A'$	$\geq -\infty$	≥-1.890	≥-0.7559	≥0.3780	≥1.512	≥2.646	≥3.780
$\overline{UCB}, B'$	<-1.890	<-0.7559	<0.3780	<1.512	<2.646	<3.780	$< \infty$
$p(X \in [A', B'))$	0.02939	0.1955	0.4224	0.2874	0.06121	0.003997	7.8552E-05
E	2.351	15.64	33.79	23.00	4.898	0.3198	0.006284

$\overline{LCB}$=Standardised Lower Class Boundary, $\overline{UCB}$=Standardised Upper Class Boundary

Table 4.7: Expected Frequencies(=E) For Standardised Classes

If any of the classes' expected frequencies are less than 5 then those classes must be combined until all classes have expected frequency of at least 5. The same must be done for the observed frequencies. The table produced is in table 4.8.

Class	1	2	3	4
O = Observed Freq.	15	37	23	5
E = Expected Freq.	17.98766616	33.79405012	22.99548106	5.222766825
(O-E)^2 / E	0.496237198	0.304139769	8.88035E-07	0.00950168

Table 4.8: Observed And Expected Frequencies χ^2 Values

From table 4.8 the calculated χ^2 sum is $\chi^2_{calc} \approx 0.81$. This must be compared against the reference χ^2 value for the appropriate number of degrees of freedom and significance level. The number of 'degrees of freedom', ν, is the number of restrictions subtracted from the number of classes. In this case $\nu = 4 - 1 = 3$ as there is one restriction; that the total χ^2sum must equal the sum of the expected frequencies=80. If we wish to test at a 5% significance level then the reference χ^2 value will be:

$$\chi^2_{0.95}(\nu) = \chi^2_{0.95}(3) \approx 7.81$$

This value is significantly greater than $\chi^2_{calc} \approx 0.81$ as exemplified in figure 4.4.

$$\chi^2_{calc} \approx 0.81 \ll 7.81 \approx \chi^2_{0.95}(3)$$

$\Longrightarrow$ the distribution of X is very likely to be normal.

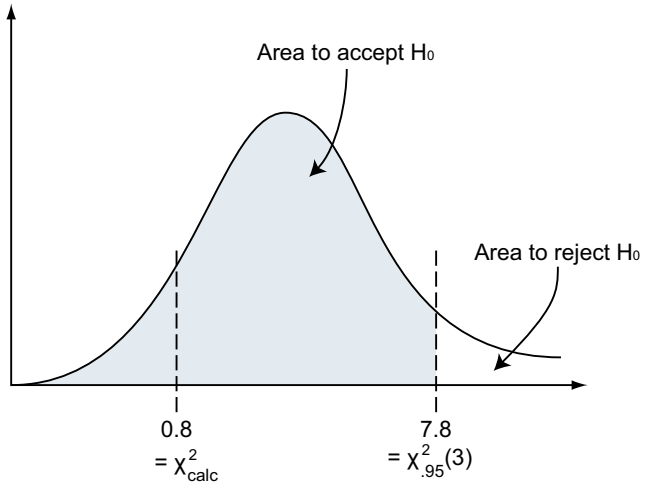

Figure 4.4: Pictorial View of χ^2 Test

4.2.2 Examining Association Between Sticking Time And Correctness

Even though the distribution of the correctnesses is likely to be normal (see section 4.2.1) it may still potentially have some association with average sticking time. For example if figure 4.2 is examined it appears that in the region $stick\,time < 78.2$ that the distribution of bits is distinctly biased. A similar observation can be made about figure 4.3. To test whether stick time and correctness are independent the data can be examined by categorising it into a contingency table and calculating the Pearson χ^2 statistic. The samples were classified into 4 categories as follows:

Observed Frequencies	$0.115 \leq x < 0.125$	$0.125 \leq x < 0.135$	Total
$77 \leq y < 78.2$	26	10	36
$78.2 \leq y < 79.4$	14	30	44
Total	40	40	80

To test if there is any association between the average sticking time of a bit and its correctness define the null hypothesis $H_0$to be:

- There is no association between the average sticking time of a bit and its correctness.

Assuming independence, the expected frequencies are calculated using the following formula:

$$\frac{(row\,total) \times (column\,total)}{grand\,total}$$

Expected Frequencies	$0.115 \leq x < 0.125$	$0.125 \leq x < 0.135$	Total
$77 \leq y < 78.2$	18	18	36
$78.2 \leq y < 79.4$	22	22	44
Total	40	40	80

Since the table is 2×2, the Yates' continuity correction is used as follows:

$$\chi^2 = \sum \frac{\left(|O - E| - \frac{1}{2}\right)^2}{E}$$

$$\chi^2 = \frac{\left(|26 - 18| - \frac{1}{2}\right)^2}{18} + \frac{\left(|10 - 18| - \frac{1}{2}\right)^2}{18} + \frac{\left(|14 - 22| - \frac{1}{2}\right)^2}{22} + \frac{\left(|30 - 22| - \frac{1}{2}\right)^2}{22}$$

$$\chi^2_{calc} \approx 11.36$$

There is 1 degree of freedom $\nu = (rows - 1) \times (columns - 1) = 1$. Using a 5% significance level:

$$\chi^2_{0.95}(1) \approx 6.63$$

Since $\chi^2_{0.95}(3) \approx 6.63 \ll \chi^2_{calc} \approx 11.36$ reject the null hypothesis of independence and conclude there is an association between the average sticking time of a bit and its correctness.

4.2.3 Comparison of The Distributions of Sticking Times

The mean sticking time of 'one' bits from problem 5 is $\overline{x_1} \approx 78.21$ and of 'zero' bits is $\overline{x_2} \approx 78.25$. It may be that these sample distributions are drawn from populations that have the same mean. Let S_1 be the set of 'one' samples and S_2 be the set of 'zero' samples. Now the recorded statistics for each set are as below. As the population variances are unknown they are estimated from the sample variances.

S_1	S_2
$n_1 = 80$	$n_2 = 560$
$\overline{x_1} \approx 78.21$	$\overline{x_2} \approx 78.25$
$s_1^2 \approx 0.142081$	$s_2^2 \approx 0.147144$
$\hat{\sigma}_1^2 = \frac{n_1}{n_1 - 1} s_1^2 \approx 0.143879$	$\hat{\sigma}_2^2 = \frac{n_2}{n_2 - 1} s_2^2 \approx 0.147407$

Now the null hypothesis H_0 is stated:

- The population means are equal, i.e. $\mu_1 = \mu_2$ or $\mu_1 - \mu_2 = 0$.

The alternative hypothesis H_1 is stated to be:

- The population means are not equal, i.e. $\mu_1 \neq \mu_2$ or $\mu_1 - \mu_2 \neq 0$

The test statistic is:

$$z = \frac{(\bar{\mathbf{x}}_\mathbf{1} - \bar{\mathbf{x}}_\mathbf{2}) - (\mu_\mathbf{1} - \mu_\mathbf{2})}{\sqrt{\frac{\hat{\sigma}_1^2}{n_1} + \frac{\hat{\sigma}_2^2}{n_2}}}$$

$$z_{calc} = \frac{(78.21 - 78.25) - 0}{\sqrt{\frac{0.143879}{80} + \frac{0.147407}{560}}} \approx -0.88$$

Now using a two-tail test with a 5% significance level the critical values of z are ± 1.96. Since the test statistic z_{calc} is in the interval $[-1.96, 1.96]$ then the null hypothesis should be accepted.

4.3 Examples of Larger Attacks On Problems

Although largest size of the smaller problems failed to produce useful information, as its results at size 32×64 are apparently random, a selection of larger problems were investigated to determine if there might be other emergent trends in bigger problems. To this end three problems of sizes 32×64, 64×128 and 128×256 were generated and each attacked by the optimisation rig in turn. Each attack on a problem consited of 10,000 **SA** runs one after another, with the results of all runs recorded for analysis. All bits of the attempted secret vector were allowed to be free and the remaining parameters for the **SA** were as follows:

- Temperature reduction factor; $\beta = 0.95$
- Number of iterations before temperature reduction; $n = 400$

For each problem the graphs mentioned in 3.6 on page 18 were formed. These results are included as an example of the performance of the system against larger sized problems.

4.3.1 Problem Instances

The secret ϵ-vectors for these problems are defined in figures A.1, A.10 and A.19 with the one-components listed in A.2, A.11 and A.20. The problems are of size:

1. 32×64, $p = 8$
2. 64×128, $p = 16$
3. 128×256, $p = 32$

4.3.2 Results of Attacks

The results of the (average correctness, sticking time) graphs (figs. A.3, A.12 and A.21) would appear to indicate that bits which are stuck earlier are likely to be stuck correctly, with ranges of 75% to 90% of bits being stuck correctly. Additionally the vast majority of bits stick in this region (between 65 and 90 temperature cycles), so the correlation is over a great sum of bits (figs. A.4, A.13 and A.22). This means that the correlation is very unlikely to have been produced by chance.

Unfortunately these high numbers do not indicate that the majority vector in these circumstances will produce successful results, the reasoning for which is as follows. Let the solution ϵ-vector be S, which contains p ones (i.e. $|S| = p$). Then choose a large set of random ϵ-vectors $R = \{S' : |S'| = p\}$. Then obviously over all ϵ-vectors $S' \in R$ the mean correctness for each component which is 1 will be $\frac{p}{n}$. Similarly the mean correctness for each component which is 0 will be $1 - \frac{p}{n}$. So for the majority vector to provide useful information the correctness for each component much differ from $1 - \frac{p}{n}$ by a discernible margin.

In the case of these problems $p = \frac{n}{8}$, so the correctnesses must exceed $1 - \frac{n}{8n} = 1 - \frac{1}{8} = \frac{7}{8} = 0.875$. This is not the case for any of the bits in the three problems attacked, as figures A.7, A.16 and A.25 clearly show.

A strange artifact present in each of the (average correctness, sticking time) graphs is the small 'bump' (illustrated best in figure A.21, between times of 65 and 75) which emerges. The reason behind this formation is not evident.

Chapter 5

Evaluation

5.1 Objectives of The Project

The primary aim of this project was to investigate the application of heuristic optimisation to the syndrome decoding problem. To achieve this goal would involve the completion of a number of subtasks. It was hoped that through these attacks that information regarding the structure of the syndrome decoding problem would be leaked. At very minimum a new insight into the difficulties of its solution will be gained.

Before any other work could be carried out literature research had to be performed. A thorough review of literature relevant to the field of optimisation and also cryptography. Obviously literature pertinent to optimisation based cryptography will likely be of most importance in the review. This process should also provide a source of ideas which will be extremely useful.

Once the relevent information is acquired a cryptanalytic framework needed to be designed and constructed in order to determine the applicability of heuristic search to the syndrome decoding problem. To maximise the usefulness of this cryptographic tool the following criteria were set out. The tool should satisfy them all then to be considered completely successful.

Efficiency:

1. The framework needs to be very efficient with memory. Over a large number of simulated annealing runs even a small memory leak could have drastic consequences. It is also desirable that the framework's memory useage in general is low. This will make it far easier to distribute a collection of problem attacks over a number of computer systems with potentially varying specifications.

2. The framework, more specifically the optimisation component, should be very time efficient. This is extremely important for cryptanalysis because attempting to solve apparently intractable problems is obviously going to consume a massive amount of processing time.

Extensibility:

1. It will be very useful if the framework can be simply and quickly be modified to to allow the cryptanalysis of other, but necessarily similar, problems. Although obviously all problems with cryptographic potential cannot be succinctly surmised into such a formal description as in a programing language.

2. If the chosen form of optimisation technique does not prove to be useful then further work may involve the use of other techniques. Thus the ability to easily incorporate another of these techniques is a desirable feature.

3. The first heuristic used in the optimisation process for attacking the syndrome decoding problem is likely not to be ideal. This suggests that multiple possibilities may need to be experimented with, although it is not within the scope of this project to attempt to find an optimal heuristic function.

Reliability:

1. An obvious requirement, but nevertheless important. The system is going to be run for extended periods of time and it is not acceptable for it to fail without good reason. Tolerance against possible faults in its input files is desirable.

Post-processing Ability:

1. The optimisation component of the framework should be able to store the results of any set of experiments carried out with it for later analysis by a separate post-processing component.

2. The post-processing component should be able to produce tabular information in formats readable by graphing/statistical packages such as Excel®[1] and GNUPlot.

3. The post-processing component should be able to produce clear visual information to aid human understanding of the results of experiments. It would be logical to use a package such as GNUPlot to achieve this.

Once the framework has been designed and constructed the first objective should be to find the limit of problem size which direct optimisation can successfully find an answer to. Beyond this point at which the optimisation attacks can no longer succeed the focus of analysis should move to finding indications of small correlations in the data. Through this searching it is hoped that the abilities of the framework may be expanded to encompass larger problems.

5.2 Success In Achieving Early Objectives

Literature: A variety of pertinent literature was reviewed revealing a significant amount of work had been performed in applying the technique of optimisation both directly to ciphers and to underlying intractable problems with varying degrees of success. Direct optimisation itself was found to be completely unsuccessful in attacking the more modern problems used in cryptosystems, but more recent reports detailed successful attacks upon the permuted perceptron problem (**PPP**) using a non-standard optimisation based technique. It was conjectured that this technique had the potential to progress further than had previous attacks upon the syndrome decoding problem.

Simulated Annealing: For the optimisation technique simulated annealing was chosen as from the research it appeared to be one of the more powerful tools available. Simulated annealing is also simple to implement, not requiring a high degree of 'tuning' to allow it to perform well, as is the case with techniques such as genetic algorithms. Another reason for the choice was its considerable success in attacking the permuted perceptron problem. The **SA** algorithm implementation, as would be expected for use in a cryptographic tool, was made with the ability to completely customise all modes of its operation.

5.3 Evaluating The Tool

Efficiency: The tool is very time efficient thanks to certain optimisations used in its design. The most significant made is the technique presented in 3.4.2 to vastly improve the speed of calculation of the image $I = A \oplus S$; its time complexity reduced from $O(mn)$ to $O(n)$ by reusing information stored about the previous state. Additionally the **SA** process is made more efficient by guaranteeing that the neighbourhood function returns a different secret vector than the current one, without needing to repeatedly try until a success is made. Furthermore, 'BitVector's are implemented as arrays of integers, so that the 'xor' and 'and' operations performed upon vectors can be done so with a native CPU instruction, which is obviously more efficient that otherwise.

The tool is reasonably memory efficient, as only a single problem instance need be in memory at any one time and there are not many overheads needed in the running of the simulated annealing algorithm.

[1] Microsoft® Excel® is a registered trademark of Microsoft® Corporation.

Expansability: Being written in an OO language allowed the design of the tool to include the facility to easily incorporate problems other than syndrome decoding. Specifying a new problem simply requires the writing of a new class which must implement the correct set of attributes. This can then be plugged into the framework. The same applies to new optimisation techniques.

Currently there is no mechanism in place to allow the user to select other optimisation techniques. The underlying code to incorporate this would be trivial to add but would require a careful redesign of the GUI to allow the functionality to be exploited. As the use of other optimisation techniques was not within the scope of this project this expenditure of development time was considered unnecessary.

Reliability: The framework is completely reliable in the respect of stability. Errors in input files cause loading to halt and a meaningful message to be printed to the output terminal. The main concern in terms of reliability, though, was the issue of memory leaks. Even a small leak over a large number of program iterations could pose a significant problem. This was one of the main reasons for which Java was the language of choice. In a language such as C++, for example, the programmer must handle the process of freeing memory after usage themselves. If this is not done then it cannot be reclaimed until the program is finished. In a large project oversights of this kind are endemic. Due to the nature of Java's operation, memory used for data which is no longer in use is automatically garbage collected, so memory leaks are not a problem.

Statistical Outputs: In combination with GNUPlot and Excel® the post-processing component of the framework is capable of automatically generating a good variety of graphical outputs which help give a clear indication of the performance of the optimisation stage. This information can also be exported in a tabular form to allow analysis to be performed upon it. Additionally the entire set of statistical data from a complete set of **SA** attacks can be exported to cover the possibility that more information about the problem can be derived from this. Unfortunately these outputs did not produce as much useful information as was hoped for a number of reasons which will be discussed later.

5.4 Experiments Carried Out

5.4.1 Finding The Limit of Direct Optimisation

Firstly some informal experimentation was carried out to confirm the framework was behaving correctly and reliably. After this, as planned, the first set of experiments carried out were designed to determine the limit of the ability of direct optimisation to solve the syndrome decoding problem. Five sets of problem instances were generated, each set containing 20 problems of the same size (with the exception of the final one, which had 10 instances) with a different problem size for each set. The problem sizes used are listed in section 4.1.

The success rates of the **SA** runs for each problem set were tabulated and the results presented. From the results it was quite clear that with the current optimisation technique and heuristic the it was impossible to solve problems of size greater than 24×48. The trend of success against problem size (see figure 4.1) was steeply downwards up to the 24×48 where only roughly 30 out of 10000 runs were succeeding. It is conjectured that the exact point at which 0% of problems can be solved is roughly 25×50.

5.4.2 At The Limit of Direct Optimisation

As 32×64 was the smallest problem size tested in a significant quantity to yield no successes the set of statistics was subjected to further analysis.

Analysing The Distributions of Correctness

From the data it appeared that the the mean correctness values for each bit in each problem over 10,000 runs could be distributed randomly. The appropriate null hypothesis was formulated, i.e. that the data was normally distributed with a given mean, and tested using the χ^2 test. The

testing verified the null hypothesis H_0='X is approximately normal with mean 0.125, and standard deviation 0.003307' to be 95% certain, which indicates that the data most likely was, albeit roughly, normally distributed.

Examining Association Between Sticking Time And Correctness

As mentioned in 4.2.2, figures 4.2 and 4.3 indicate that there may be associations between the sticking time of bits and their corresponding mean correctness. Closer inspection using the Pearson χ^2 statistic rejected the null hypothesis of independence and in doing so showed that there is an association between the sticking time and correctness of a bit.

Comparison of The Distributions of Sticking Times

If the distributions of sticking times for 'ones' and 'zeros' differ, then it may be possible to categorise them after a set of optimisation attacks. So therefore the distributions of the mean sticking times of 'ones' and 'zeros' were compared using the Normal Test for equality of means. The results of this test showed with 95% certainty that null hypothesis, H_0='the two sets of results come from distributions with the same mean', was true. This shows that it would be very difficult to pick out any differences in the two distributions, if indeed there are any at all.

5.4.3 Larger Problems

For the purpose of example, the results of attacking single larger problems was presented. As would be anticipated none of these attacks produced any successful results. Additionally from informal analysis it also appeared that the various statistical outputs which it was hoped would produce evidence of associations between subsets of the data completely failed to do so.

5.5 Evaluation of Experimentation

5.5.1 Finding The Limit of Direct Optimisation

This was a simple objective to achieve with the only difficulty in performing the necessary tasks being the vast amount of time required to carry out such a large number of optimisation attacks on the problem set. It would have been desirable to perform less **SA** runs for each problem instance, although this was not possible; it only 30 out of 10,000 runs successfully solved problems with 48 bit key length. Any fewer runs may not have highlighted this slim chance of success.

5.5.2 At The Limit of Direct Optimisation

Although nothing especially useful was proved in the previous set of tests, a positive by-product was that it provided a set of 10 32×64 problems each with 10,000 **SA** runs with full statistical data to analyse.

Analysing The Distributions of Correctness

This analysis indicates what was previously anticipated, that the output from the **SA** runs is close to random. This is useful information in itself as long as it is taken for no more than its face value. It is definitely not that case that the χ^2 test conclusively shows that the distribution is completely normal, but this is not what it was intended to show. Most $\mathcal{NP}$-complete problems will produce almost normal distributions of results when attacked in this manner. It was merely hoped that the distribution was not quite random and this would be evident in the analysis.

Examining Association Between Sticking Time And Correctness

Although this analysis does not present a concrete proof that there is a way of separating out 'ones' from 'zeros', it at least indicates that this approach is not a 'dead end' for improvement over direct optimisation attacks. It also shows that the performance of the direct optimisation will need to be improved to allow the biases in this data to become more prominent.

Comparison of The Distributions of Sticking Times

This analysis concluded that the sample distributions of 'ones' and 'zeros' sticking times come from distributions with the same mean. The Normal Test is not conclusive enough to eliminate all possibility of difference in the distributions, especially since this set of data is not as large as is desirable. It does, however, successfully conclude that differences in the distributions will be difficult to discern.

5.5.3 Larger Problems

The collection of statistical outputs produced for these larger problems was the originally planned set of tests intended to be carried out. Unfortunately the syndrome decoding problem was found to be more difficult than anticipated, which led to the associations between distributions in the data being almost nonexistant even in small problems. For problems of this size the analysis can conclude nothing other than the inadequacy of the **SA** attacks.

5.6 Further Work

Was SA The Right Choice?

The experiments carried out here have all taken place using a very standard **SA** implementation. More sophisticated techniques such as GAs have often had more successful results in attacking difficult problems. The performance difference between these two, though, is usually relatively small. It is conjectured that another form of optimisation could potentially offer improved results. In combination with other improvements this could be a step towards increasing the power of optimisation based attacks.

Fault Injection

The main objective in attacking the syndrome decoding problem is determining from the statistical information **SA** runs provide, if there are particular bits which are more likely to be of a particular value than in a randomly selected case. Fault injection provides a way to examine particular bits individually from the rest. If for a particular problem instance the results of running a large set of **SA** attacks with:

1. No constraints
2. One bit fixed to one
3. One bit fixed to zero

are recorded. Then the results can be compared against each other. In this example of fault injection's use it is predicted that if the results from set 1 are 'similar' to those from set 2 and those from set 3 are 'different', then it is likely that the bit fixed is one. Dually if set 1 is 'similar' to set 3 it is predicted the bit is likely to be zero.

This use of fault injection serves only as an example of its application. Most likely more sophisticated uses of the technique will produce more useful results.

Better Heuristic

It is predicted that the greatest gain in performance will come from finding an improved heuristic function. Unfortunately it is also predicted that improvement in this area will be the most difficult to achieve, mainly due to the nature of the field the syndrome decoding problem operates over.

If a more accurate heuristic function can be found then the associations, such as those noticed in figures 4.2 and 4.3 will become more pronounced, which could potentially start to leak out useful information about the problem instance being attacked.

Chapter 6

Conclusions

6.1 What Was Shown

1. In its current state heuristic optimisation is not up to the task of solving even the smaller sizes of problem which would be considered secure.

2. The hamming distance between current image and expected image is not a good heuristic. The data in all results showed extremely low levels of association. A better heuristic must definitely be devised if this mode of attack is to make progress.

3. To design a good heuristic for this problem is hard because of the manner of the information output by unsuccessfully attacks. With **PPP**, for example, a heuristic could be designed which could produce a relatively 'continuous' scalar field. With **SD**, the image vector is over a two-element field making the concept of 'continuous' almost meaningless. Additionally a single bit change in the secret vector can potentially change every bit in the image vector.

4. Direct simulated annealing attacks can successfully solve problems with key sizes of up to 48 bits.

5. Smaller problem instances often have multiple solutions. It is not clear if this is the case for larger problems.

6. Even with the limited heuristic used in this project, there is definitely an association between the mean correctness and sticking times of bits over a large number of **SA** runs.

A new insight has been given into the difficulty of solving the general case of the syndrome decoding problem. It is extremely likely, when using key sizes of 512 bits and above, that an identification scheme built upon this problem will remain secure for years to come.

6.2 Applications To Other Problems

Although non-standard optimisation had very limited success in attacking the syndrome decoding problem, the same technique could also be used to attack other problems such as the permuted kernel problem and constrained linear equations.

Bibliography

[Cha93] F. Chaubard. Asymptotic analysis of probabilistic algorithms for finding short codewords. In *Proceedings of EUROCODE '92*, volume 339 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[CJ02] J. A. Clark and J. L. Jacob. Fault injection and a timing channel on an analysis technique. In *Advances in Cryptology — Proceedings of EUROCRYPT '02*, pages 181–196. Springer-Verlag, 2002.

[DBL97] R. A. DeMillo D. Boneh and R. J. Lipton. On the importance of checking cryptographic systems for faults. In W. Funny, editor, *Advances in Cryptology — Proceedings of EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 35–51, Berlin, Heidelberg, 1997. Springer-Verlag.

[DG03] A. Dimovski and D. Gligoroski. Attacks on the transposition ciphers using optimization heuristics. In *Proceedings of ICEST 2003*, Sofia, Bulgaria, 2003.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology — Proceedings of EUROCRYPT '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa-Barbara, California, 1987. Springer-Verlag.

[Geo92] J. Georgiades. Some remarks on the security of the identification scheme based on permuted kernels. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 5(2):133–137, 1992.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *17th ACM Symposium on the Theory of Computing — STOC*, pages 291–304, Providence, Rhode Island, U.S.A., 1985. ACM Press.

[GQ88] L. C. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In C. G. Günter, editor, *Advances in Cryptology — Proceedings of EUROCRYPT '88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128, Davos, Switzerland, 1988. Springer-Verlag.

[Jak95] T. Jakobsen. A fast method for cryptanalysis of substitution ciphers. In *Cryptologia*, volume 19(3), pages 265–274, 1995.

[KM99] L. R. Knudsen and W. Meier. Cryptanalysis of an identification scheme based on the permuted perceptron problem. In J. Stern, editor, *Advances in Cryptology — Proceedings of EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 363–374, Berlin, Heidelberg, 1999. Springer-Verlag.

[Koc96] P. C. Kocher. Timing attacks on implementations of of diffe-hellman, rsa, dss and other systems. In N. Koblitz, editor, *Advances in Cryptology — Proceedings of EUROCRYPT '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Berlin, Heidelberg, 1996. Springer-Verlag.

[MRG79] D. S. Johnson M. R. Garey. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman Company, San Francisco, 1979.

[OO89] K. Ohta and T. Okamoto. A modification of the Fiat-Shamir scheme. In S. Goldwasser, editor, *Advances in Cryptology — Proceedings of EUROCRYPT '88*, volume 403 of *Lecture Notes in Computer Science*, pages 232–243, Santa-Barbara, California, 1989. Springer-Verlag.

[OS91] H. Ong and C.P. Shnorr. Fast signature generation with a Fiat-Shamir-like scheme. In I. B. Damgard, editor, *Advances in Cryptology — Proceedings of EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 432–440, Aarhus, Denmark, 1991. Springer-Verlag.

[PC94] J. Patarin and P. Chauvaud. Improved algorithms for the permuted kernel problem. In D. R. Stinson, editor, *Advances in Cryptology — Proceedings of CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 391–??, Berlin, Heidelberg, 1994. Springer-Verlag.

[Poi94] D. Pointcheval. Neural networks and their cryptographic applications. In P. Charpin, editor, *Livre des résumés - EUROCODE '94*, volume 435 of *Lecture Notes in Computer Science*, pages 183–193, La Bussière sur Ouche, France, 1994. INRIA.

[Poi95] D. Pointcheval. A new identification scheme based on the perceptrons problem. In L.C. Guillou and J.-J. Quisquater, editors, *Advances in Cryptology — Proceedings of EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 319–328. Springer-Verlag, 1995.

[Sha90] A. Shamir. An efficient identification scheme based on permuted kernels. In G. Brassard, editor, *Advances in Cryptology — Proceedings of EUROCRYPT '89*, volume 435 of *Lecture Notes in Computer Science*, pages 606–609, Santa-Barbara, California, 1990. Springer-Verlag.

[Ste90] J. Stern. An alternative to the Fiat-Shamir protocol. In *Advances in Cryptology — Proceedings of EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 173–180, 1990.

[Ste94] J. Stern. A new identification scheme based on syndrome decoding. In *Advances in Cryptology — Proceedings of EUROCRYPT '93*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer-Verlag, 1994.

[Ste97] J. Stern. Designing identification schemes with keys of short size. In *Advances in Cryptology — Proceedings of EUROCRYPT '93*, volume 839 of *Lecture Notes in Computer Science*, pages 164–173. Springer-Verlag, 1997.

[TBG92] P. Chauvard T. Baritaud, M. Campana and H. Gilbert. on the security of the permuted kernel identification scheme. In E. F. Brickell, editor, *Advances in Cryptology — Proceedings of CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, California, USA, 1992. Springer-Verlag.

Appendix A

Statistical Data For Larger Problems

A.1 Problem 1 (32×64, $p = 8$)

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure A.1: Secret Vector From Problem 1

$$\begin{pmatrix} 0 & 12 & 20 & 29 & 43 & 45 & 49 & 53 \end{pmatrix}$$

Figure A.2: Indices of Ones From Problem 1

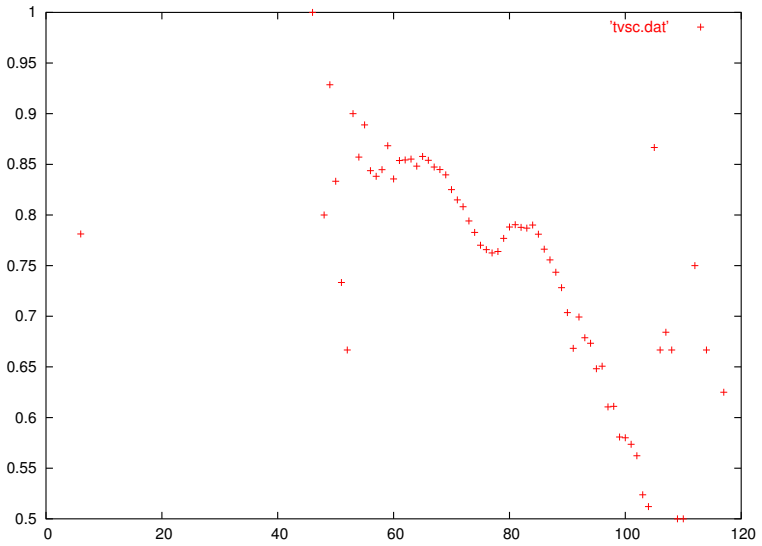

Figure A.3: plot of (correctness, stick time)

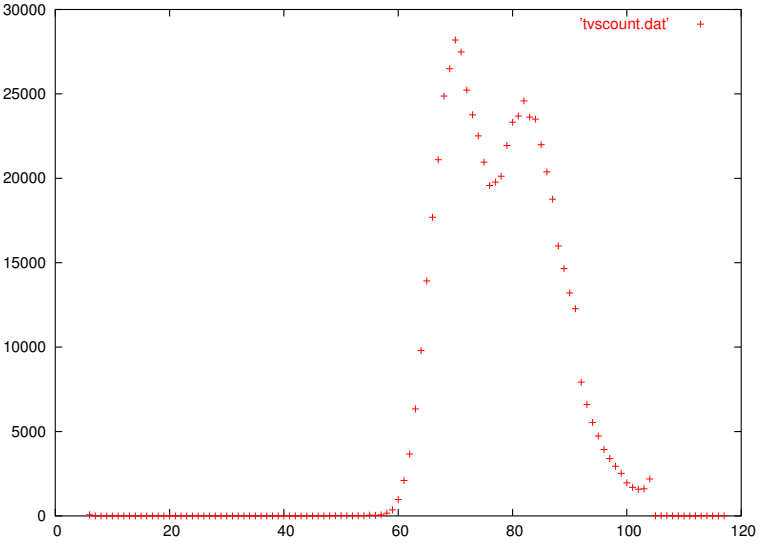

Figure A.4: plot of (#bits, stick time)

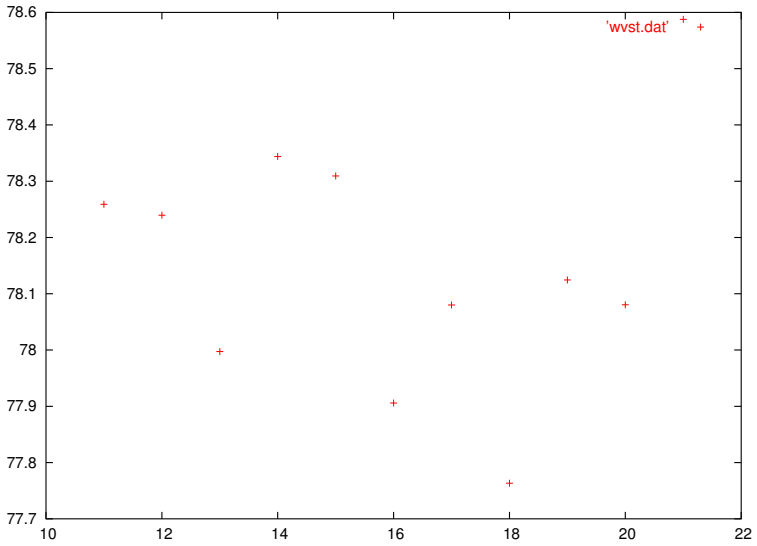

Figure A.5: plot of (av. stick time, bit weight)

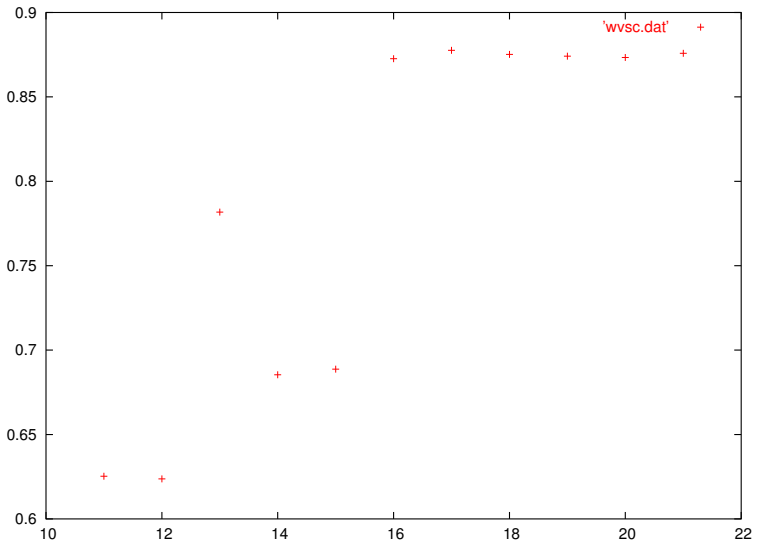

Figure A.6: plot of (correctness, bit weight)

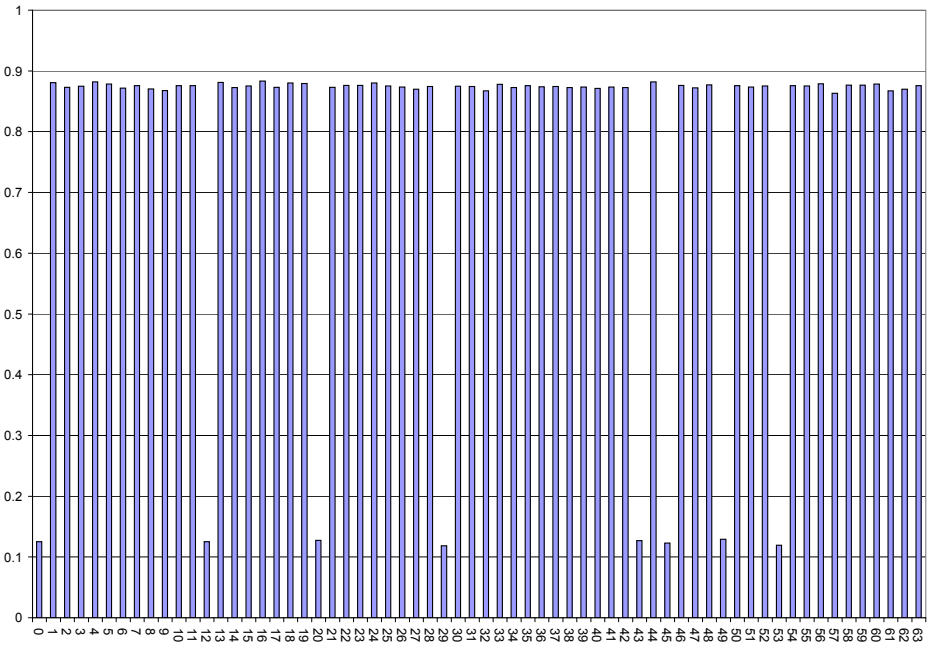

Figure A.7: plot of (correctness, bit#)

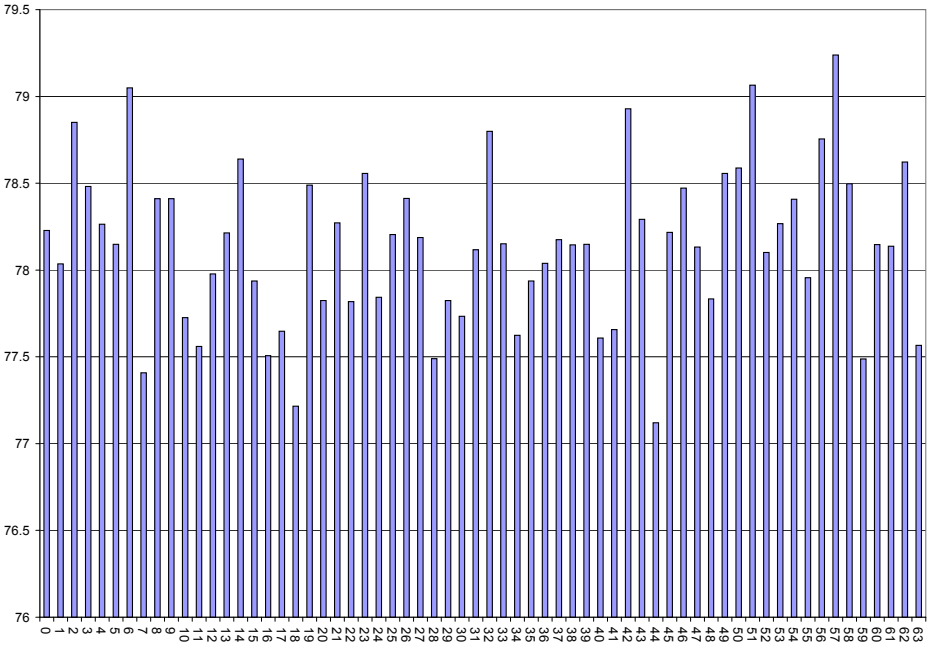

Figure A.8: plot of (stick, bit#)

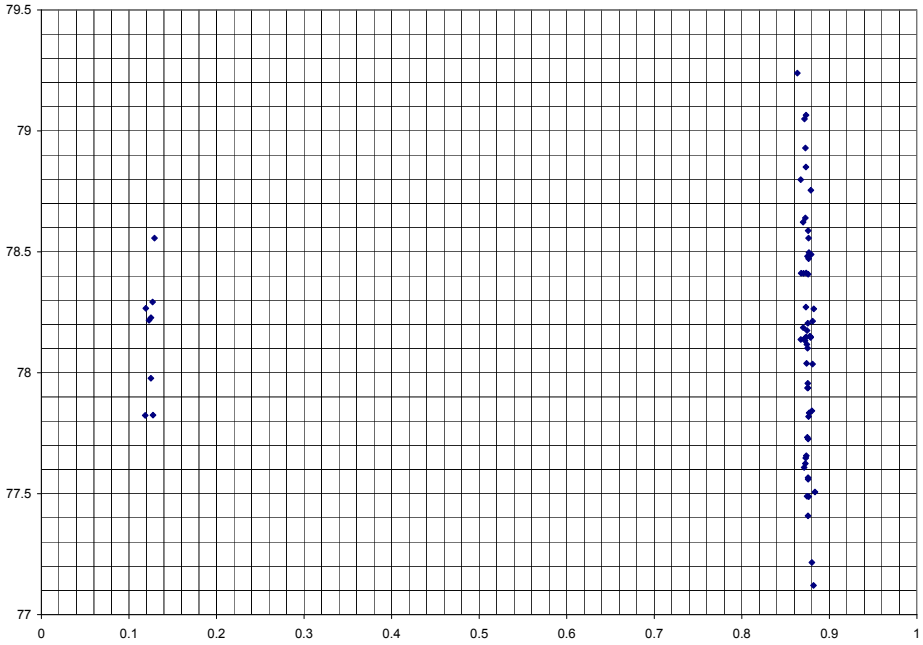

Figure A.9: plot of (average stick time, correctness) for each bit

A.2 Problem 2 (64×128, $p = 16$)

$$S = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure A.10: Secret Vector From Problem 2

$$\begin{pmatrix} 1 & 2 & 28 & 34 & 40 & 41 & 48 & 52 & 69 & 76 & 98 & 100 & 110 & 112 & 122 & 127 \end{pmatrix}$$

Figure A.11: Indices of Ones From Problem 2

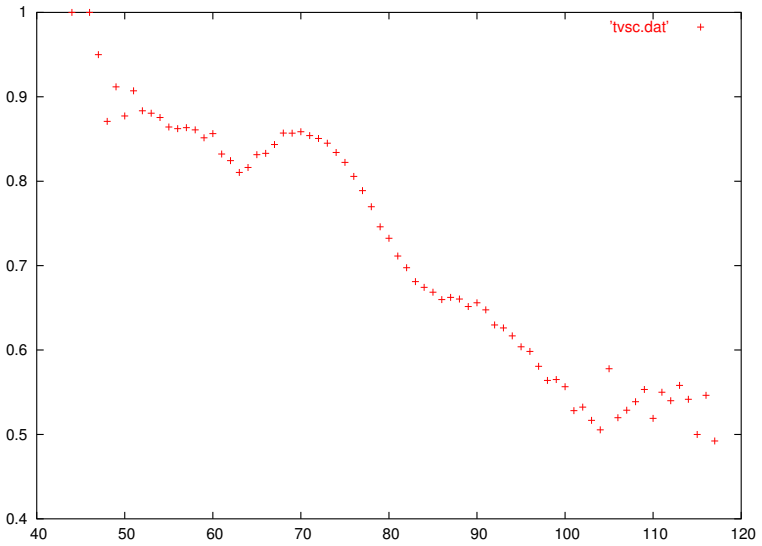

Figure A.12: Plot of (correctness, stick time)

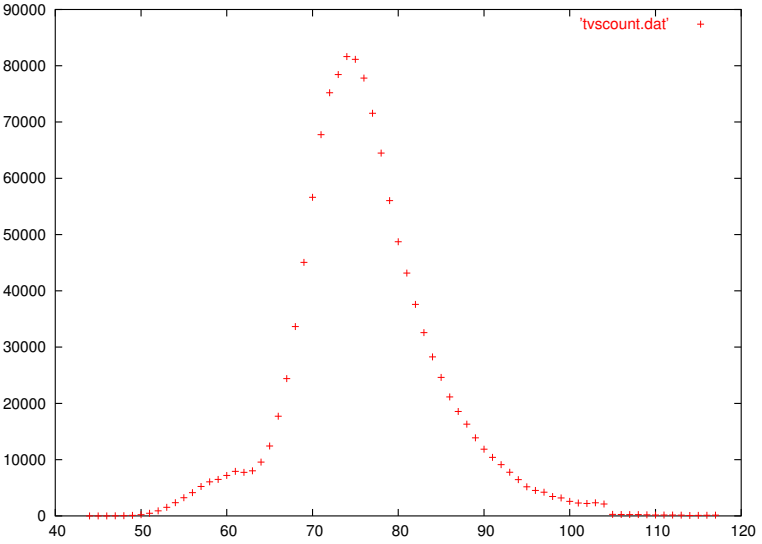

Figure A.13: plot of (#bits, stick time)

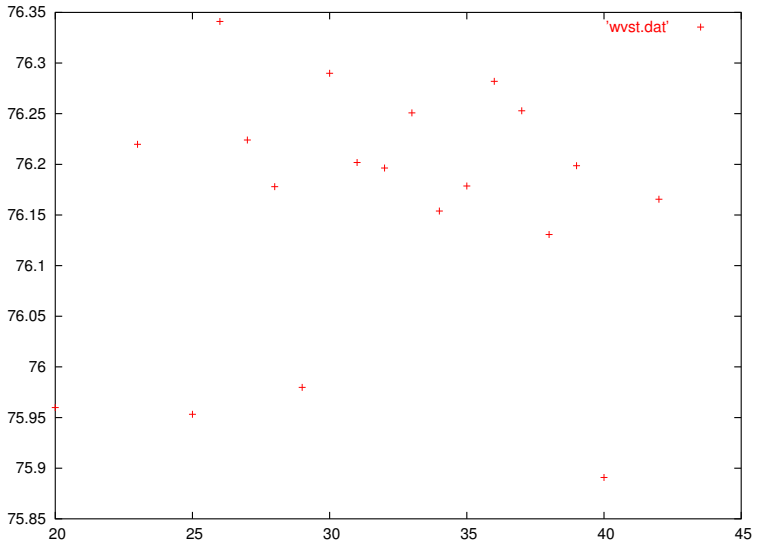

Figure A.14: plot of (av. stick time, bit weight)

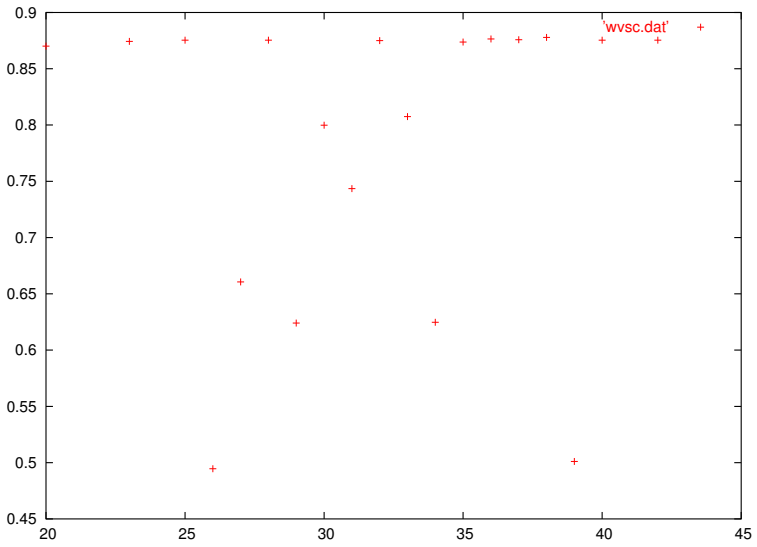

Figure A.15: plot of (correctness, bit weight)

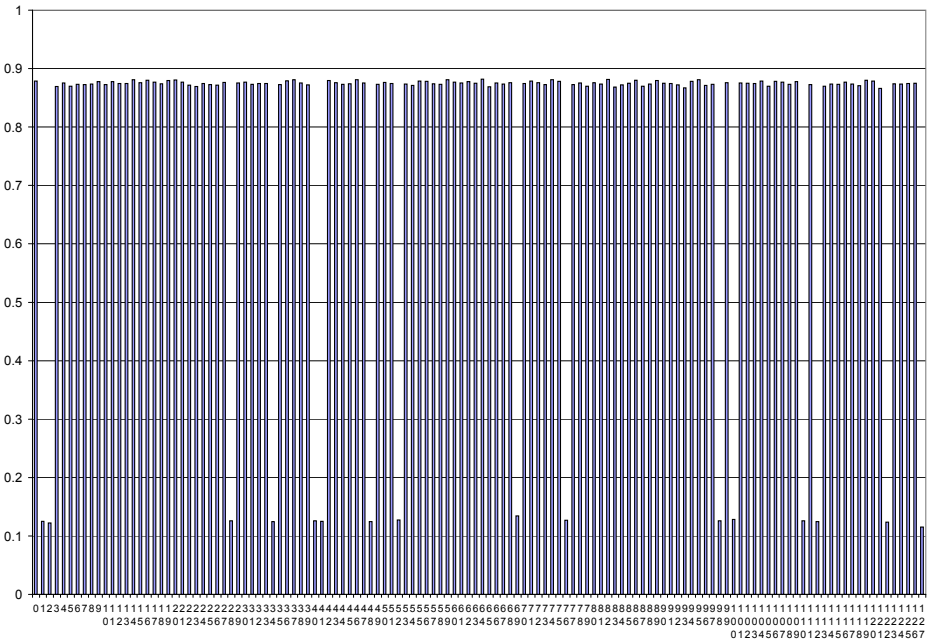

Figure A.16: plot of (correctness, bit#)

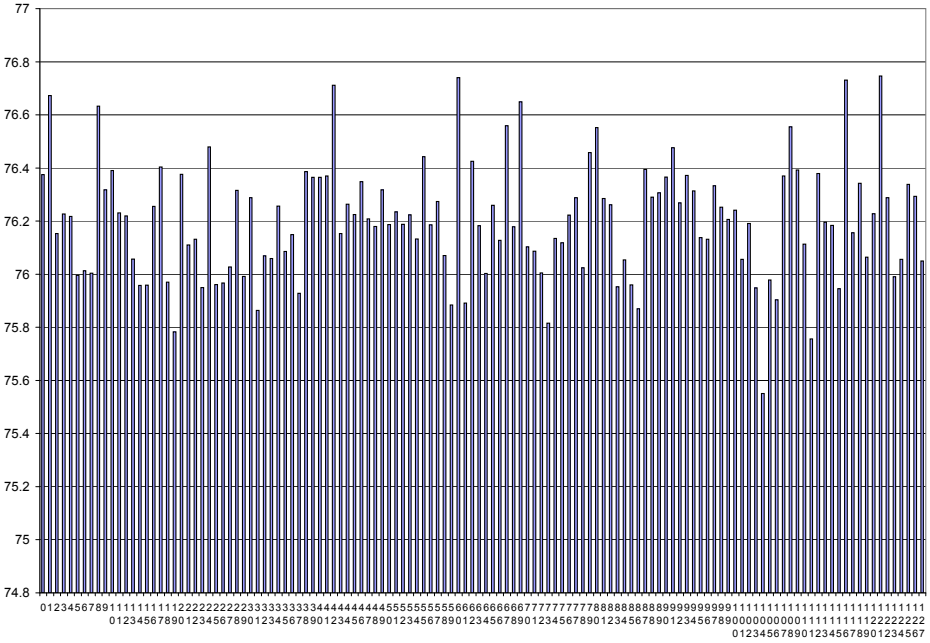

Figure A.17: plot of (stick, bit#)

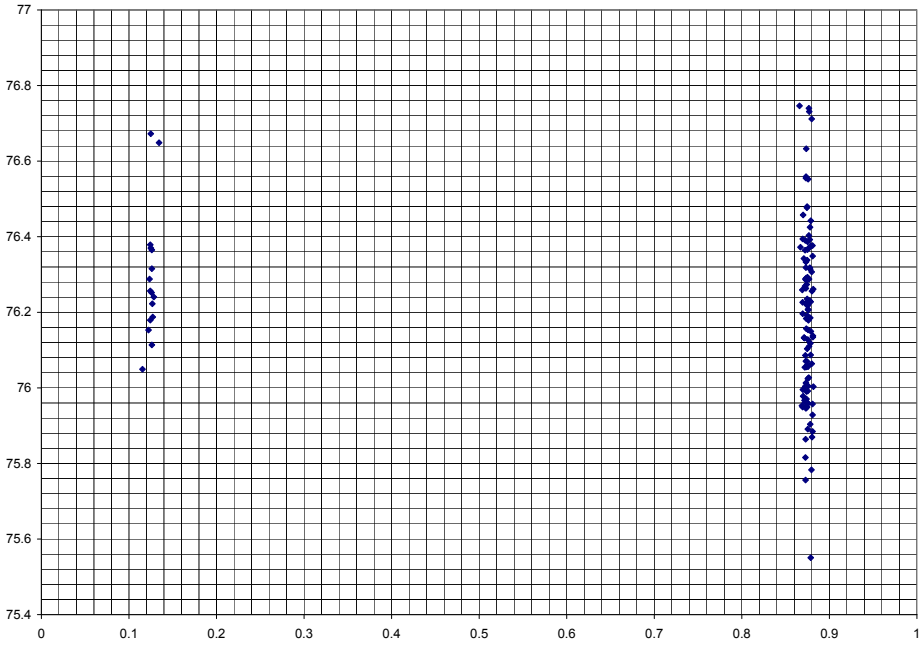

Figure A.18: plot of (average stick time, correctness) for each bit

A.3 Problem 3 (128×256, $p = 32$)

$$S = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Figure A.19: Secret Vector From Problem 3

$$\begin{pmatrix}
8 & 43 & 51 & 53 & 57 & 66 & 68 & 71 & 73 & 79 & 86 & 88 & 90 & 92 & 100 & 125 \\
132 & 137 & 139 & 147 & 149 & 151 & 167 & 171 & 175 & 176 & 181 & 186 & 217 & 234 & 239 & 249
\end{pmatrix}$$

Figure A.20: Indices of Ones From Problem 3

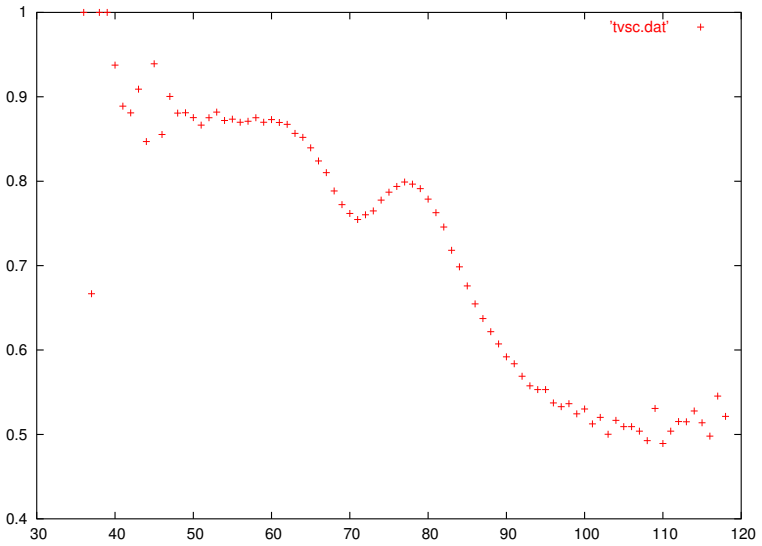

Figure A.21: plot of (correctness, stick time)

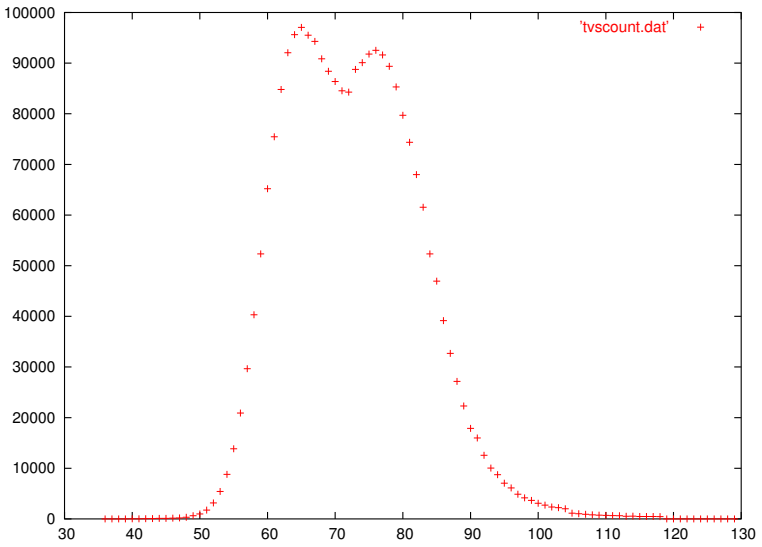

Figure A.22: plot of (#bits, stick time)

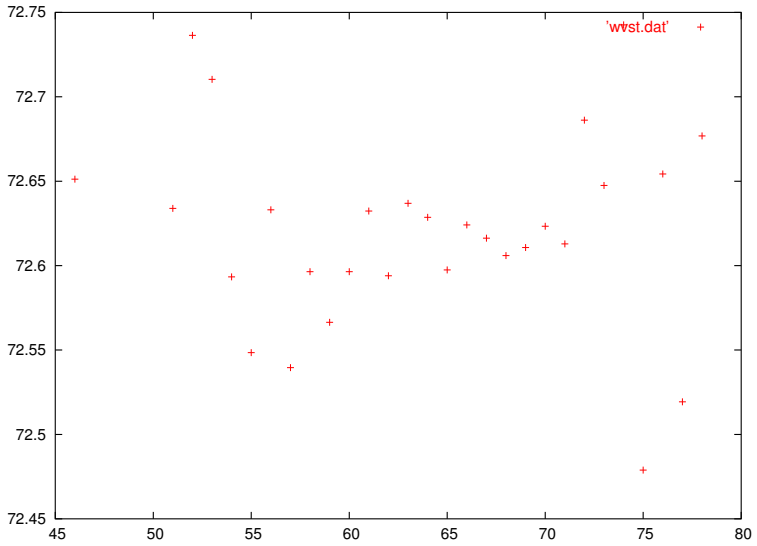

Figure A.23: plot of (av. stick time, bit weight)

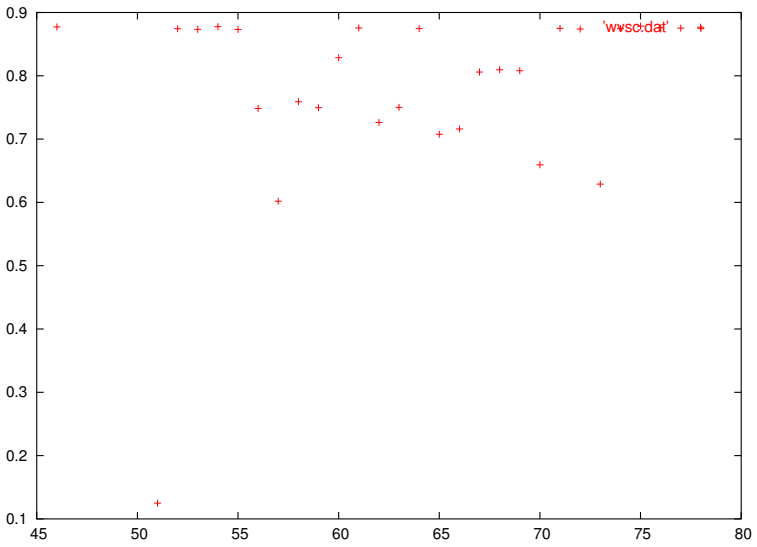

Figure A.24: plot of (correctness, bit weight)

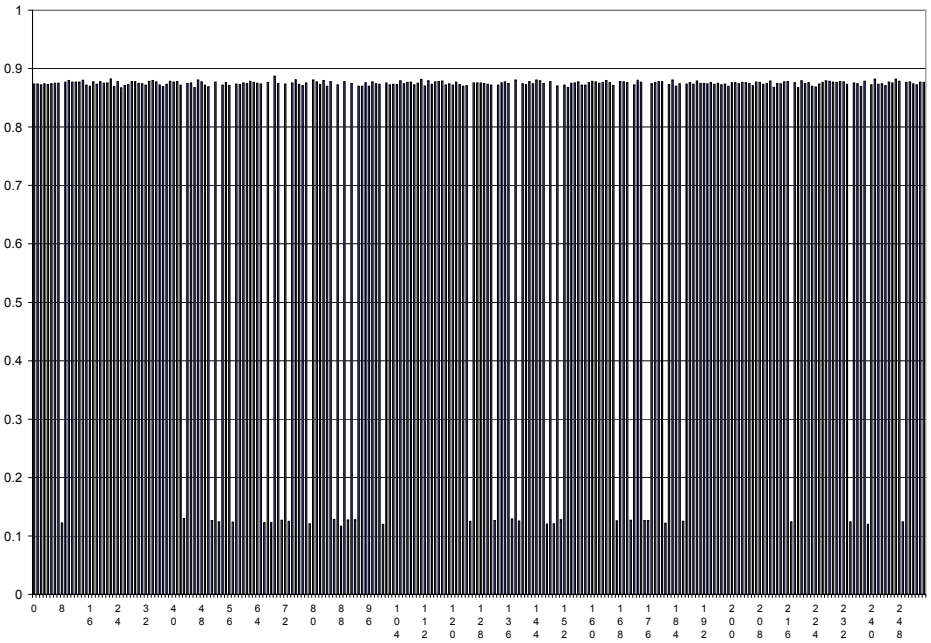

Figure A.25: plot of (correctness, bit#)

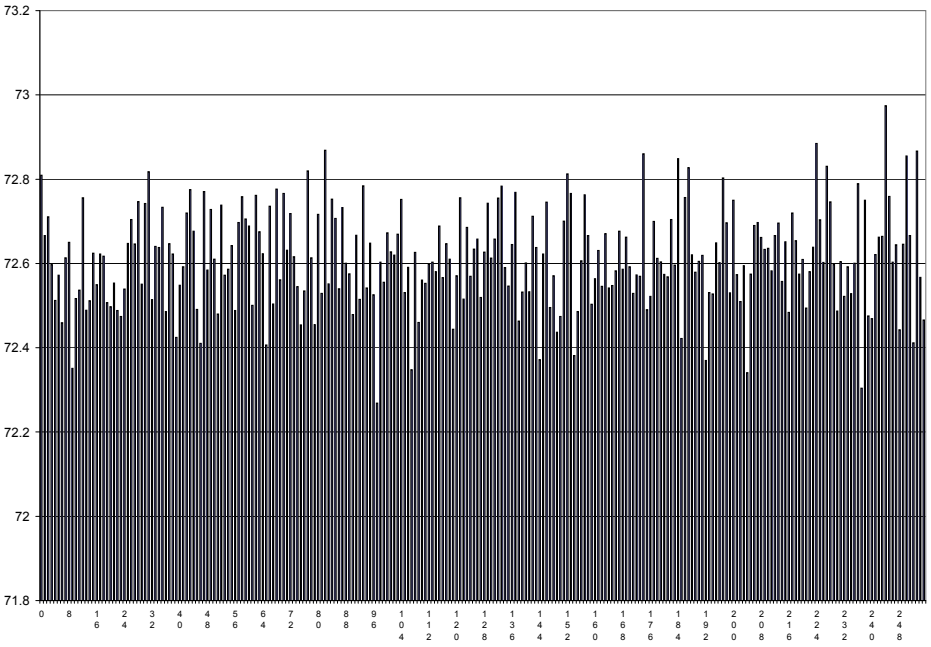

Figure A.26: plot of (stick, bit#)

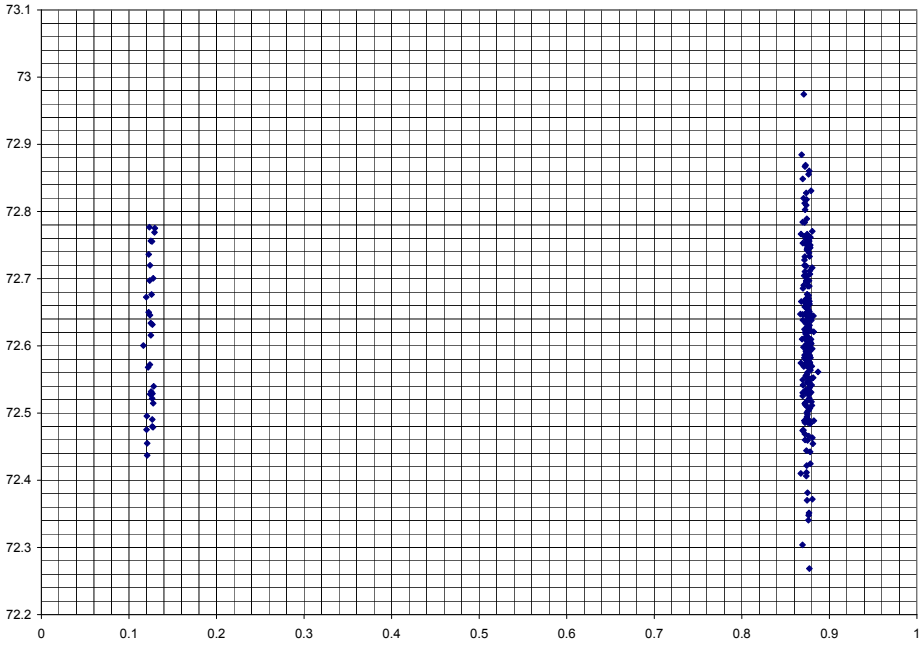

Figure A.27: plot of (average stick time, correctness) for each bit