PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A

*Phil. Trans. R. Soc. A* (2006) **364**, 3199–3214
doi:10.1098/rsta.2006.1885
*Published online* 19 October 2006

# Powerline communication and the 36 officers problem

By Sophie Huczynska*

*School of Mathematics and Statistics, University of St Andrews, St Andrews, Fife KY16 9SS, UK*

In this survey paper, we explore the interactions between mathematics and engineering inspired by the challenge of transmitting data along powerlines. In particular, we focus on how combinatorial objects called *permutation arrays* offer a way of encoding data which allows the noise problems experienced in powerline communications (PLCs) to be overcome. The first study of permutation codes was carried out in the 1970s, but the preference in traditional information theory for codes with small alphabet size meant that permutation codes were largely ignored until recently. Their rediscovery for use in PLCs has brought about a resurgence of interest in the construction and properties of permutation arrays. We survey previous and current work in this area, and discuss future developments.

**Keywords: permutation arrays; coding theory; powerline communication**

## 1. Introduction

What is the connection between technology which allows the Internet to be received through electric power sockets, and a problem posed in the eighteenth century about arranging soldiers on parade? The answer lies in the application of combinatorics, a branch of mathematics that studies the counting and arrangement of collections of objects, to the engineering challenges of transmitting data along powerlines.

As demonstrated by the party game 'Chinese Whispers', the process of sending a message is not always 100% reliable. This is equally true in the communication of digitally encoded information, since various forms of interference (technically referred to as *noise*) often cause data to be distorted en route. To overcome this problem, it is necessary to invent a way of representing and transmitting the data, so that errors in transmission can be identified and, if possible, corrected, when the (possibly corrupted) message is received. Claude Shannon's paper *A mathematical theory of communication*, written in 1948, effectively founded the branch of electronic engineering called *information theory* to meet this challenge. A key role in this theory is played by *error-correcting codes*, pioneered by Richard W. Hamming. Since then, coding theory has developed many links with the mathematical disciplines of algebra and combinatorics. Techniques and

*sophieh@mcs.st-and.ac.uk

One contribution of 23 to a Triennial Issue 'Mathematics and physics'.

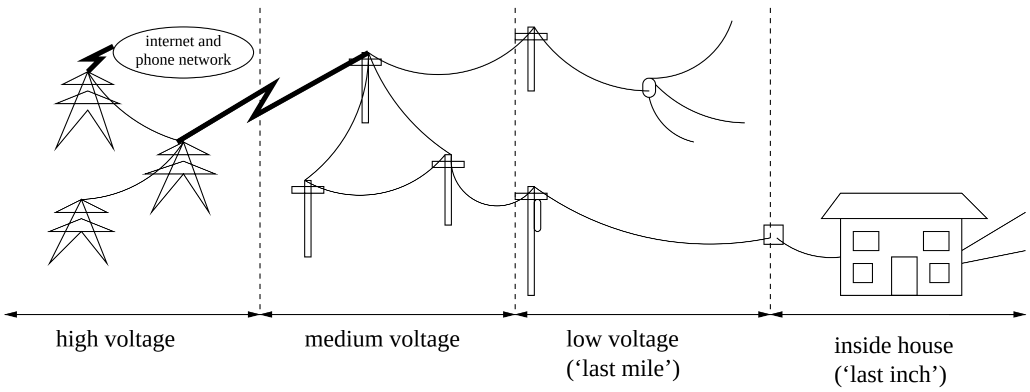
internet and phone network
high voltage
medium voltage
low voltage ('last mile')
inside house ('last inch')

Figure 1. A model for PLC, after Sankar (2004).

results from these disciplines have been successfully applied to solve coding problems, while the problems have inspired new areas of mathematics. The interplay between coding theory and parts of mathematics traditionally labelled 'pure' has led to some surprising breakthroughs in areas as diverse as sphere packing and simple groups. We provide an introduction to coding in §2.

Recently, powerline communication (PLC), the technology which allows the transmission of data over the same lines used to transmit electric power, has been heralded as the 'next big thing' in communications. The potential advantages of providing data services by exploiting the power grid are clear. The vast infrastructure already in place for power distribution means that these services will be available to more users than any other alternative (in particular, helping to overcome the urban/rural 'digital divide'), with no disruption upon installation. Moreover, in-building powerlines could be reused to create local area networks in homes or offices, and facilities such as automated meter reading could be offered by utility companies. Since most devices which process data and access the Internet are normally plugged into electrical sockets, combining the two networks is an elegant and appealing development (figure 1).

However, despite the enormous potential of PLCs, there is still much work to be done before the technology is widely adopted. The *IEEE Journal on Selected Areas in Communications* recently put out a call for papers to 'lay the foundation for a new generation of communication technology for powerline data transmission'. One of the major technological problems to be overcome is that of noise. Powerlines have been described as a 'rough and tumble' environment for data compared to traditional data transmission lines such as coaxial cable, and they are subject to unpredictable sources of interference. Recently, the combinatorial topic of *permutation arrays*, last studied in the 1970s, has been found to offer a solution to the major problem of noise in powerlines, and is consequently enjoying a mathematical resurgence. Researchers have recently shown (see Han Vinck 2000; Pavlidou *et al.* 2003) that a new modulation/coding scheme, using permutation arrays, can robustly handle all forms of frequency disturbances and background noise. We discuss this work in §3.

In the rest of the paper, we discuss the history, theory and construction of permutation arrays, explore their application to error-correcting codes and PLC, and consider the exciting challenges which PLC presents to both mathematicians

Table 1. Representing letters of the alphabet (A, B, C etc.) as binary strings of length 5.

| letter | corresponding binary string |
|---|---|
| $A$ | 00000 |
| $B$ | 00001 |
| $C$ | 00010 |
| $D$ | 00011 |
| $E$ | 00100 |
| $\vdots$ | $\vdots$ |

and engineers. The theory of permutation arrays uses a wide range of mathematical objects and techniques, from mutually orthogonal latin squares (MOLS; introduced by Euler (1782) in the 36 officers problem), to polynomials over finite fields, to sharply transitive groups. We discuss recent generalizations of permutation arrays, by the author and others, and speculate on future research directions.

## 2. An introduction to coding theory

Suppose that we want to send a message from $A$ to $B$. In any real-life setting, there is a chance that the message will be corrupted during transmission, so that the message received at $B$ will not be the same as that which was sent from $A$. In order to overcome this problem, it would be helpful to have a means of communication that makes it possible to recognize whether or not a received message has been corrupted. Moreover, it would be extremely helpful to be able to recover the original message from a received, corrupted message.

By ‘sending messages’, we shall generally mean transmitting data within or between machines, and we will represent each piece of data by a string of digits of a fixed length. The set of digits which we use will be called the *alphabet*, and we shall denote the length of a string by $n$. For example, all 26 letters of our normal alphabet $A$–$Z$ may be represented using strings of length 5 and symbols $\{0,1\}$, since there are $2\times2\times2\times2\times2=32$ such strings (we have a choice of putting a 0 or a 1 in each of the five positions). For practical reasons, an alphabet consisting of two symbols is a popular choice (since it allows a representation as on–off or high–low). We show in table 1 how the first few letters could be represented in this way.

The set of strings of length $n$ which we choose to use is called a *code*; each string in the code is called a *codeword*, and a *message* consists of a sequence of codewords. Given some data which we wish to send from $A$ to $B$, we *encode* it as a sequence of codewords and transmit this sequence (along a potentially ‘noisy’ channel); the sequence is then received, decoded and finally arrives at its destination (see figure 2).

Now suppose we transmit such a message, and an error occurs, changing one symbol to another. For example, using the code in table 1, we might send the letter $A$, but the last digit could be corrupted from a 0 to a 1, so that the received

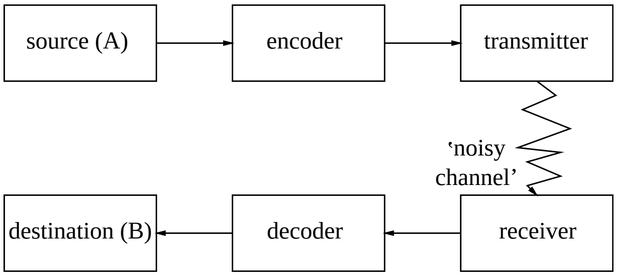

Figure 2. Transmitting data from $A$ to $B$.

message looks like $B$. At the receiving end, we cannot tell that $B$ was not the original message sent; even if we do know that a digit has been changed, was the original message $A$, or $D$, or some other letter?

In order to be able to recognize and correct errors, it is helpful to choose our codewords to be sufficiently different from each other, so that we can identify the original codeword even after it has suffered some corruption. This idea of 'difference' is formally expressed in terms of *Hamming distance* (after Richard W. Hamming). For any two codewords $x = a_1 a_2 \dots a_n$ and $y = b_1 b_2 \dots b_n$, each comprising $n$ digits from some alphabet, the distance $D(x,y)$ between them is the number of positions in which their symbols differ, i.e. the number of $k$ such that $a_k \neq b_k$. Thus, in the code above, $D(00000, 00001) = 1$ while $D(10101, 01110) = 4$. The smallest distance between any two distinct codewords in a code C is called the *minimum distance* of the code; for example, the code above has minimum distance 1.

A code of minimum distance $d$ can detect up to $d-1$ errors, since at least $d$ errors must occur to turn one codeword into another. So, if all codewords differ from each other in at least two positions, then the changing of a single digit is detectable. The code {001,010,100,111} (alphabet {0,1}, length 3, minimum distance 2) has this property: if a single digit of any codeword is changed, it ceases to be a valid codeword. Does this minimum distance of $d$ allow us to correct $d-1$ errors? No—in the example, if we received the string 000, we would know it was not a valid codeword, but not whether it had come from 001 or 010.

We find that a code with minimum distance $d = 2e + 1$ can correct up to $e$ errors. This is because, for each codeword C, the set $S_C$ of strings obtained from C by up to $e$ errors has no elements in common with the corresponding set from any other codeword. Thus, we can always correctly decode an element of $S_C$ as C. However, if we wish to perform error correction and detection simultaneously, then care is needed; in particular, we can no longer detect as many errors as before. For, if a codeword suffers more than $e$ errors, it may be sent into the set $S_{C'}$ for some other codeword $C'$, and hence wrongly decoded as $C'$. Thus, a code with distance $d = 2e + 1$ will correct $e$ errors, but cannot simultaneously detect more errors. (The situation when $d = 2e$ is similar.) As an example, consider the code {00010,01001,10100,11111} with minimum distance 3. It can correct one error. Thus, if 00010 is corrupted by one digit to 10010, it is identifiable as invalid, and it can be successfully decoded to 00010. However, if 00010 is corrupted by two digits to 11010, then we can tell that at least two errors have occurred, but we have no way of determining which of the valid codewords it originated from.

## 3. Coding and powerline communication

The idea of using electric powerlines to transmit data creates various challenges for information theory. In a traditional data transmission setting, encoded messages are sent along dedicated channels, e.g. copper wire or coaxial cable. The possible noise problems are well understood, the main problem being *white Gaussian noise* (also called *background noise*), which has the effect of corrupting individual symbols of codewords independently at random. Traditional information theory has been developed to deal effectively with this problem. However, in a line whose primary use is the transmission of electric power, the types of noise are more varied. The three main types are:

— permanent narrow-band noise, which affects some frequency over a period of time (e.g. from television sets or computer terminals),
— impulse noise, i.e. noise of short duration which affects many/all timeslots, and
— white Gaussian/background noise.

In PLC, the first two kinds are the most important.

Usually, engineers aim to send their signal using as small a *bandwidth*, or range of frequencies, as possible. One way of dealing with narrow-band noise is to use a wider-than-usual range of frequencies to transmit the data, in order to avoid bad parts of the spectrum. By sacrificing small bandwidth, we can increase the probability that the received data is correct. This *frequency spreading* can be performed using *frequency modulation* (the same technique used in FM radio), in which a carrier wave has its frequency (number of cycles per second) varied to represent information. In our setting, rather than varying the frequency of the carrier wave in direct proportion to changes in some input signal, as with radio, we make the carrier frequency jump between a set of $M$ discrete values. This technique is called $M$-ary *frequency shift keying* (FSK). To deal effectively with impulse noise, more research is needed about its statistical behaviour, in particular impulse duration and inter-arrival times. However, just as frequency spreading can help overcome narrow-band noise, it is known that using many timeslots can help deal with impulse noise. Therefore, a scheme for modulation and coding which offers both frequency and time diversity, such as the following, should be robust against the two main types of noise affecting PLC.

We begin by modulating the signal as described above to form a family of $M$ distinct frequencies, which we shall call $f_1, \dots, f_M$. We use a code C with an alphabet of size $M$, which we write as $\{1, \dots, M\}$. The message to be sent is encoded as a codeword of C of length $n$, and the symbols of the codeword are transmitted in time as the corresponding frequencies. Whereas many traditional applications favour a small alphabet size, here it is desirable to have a wide spread of frequencies, so a large alphabet size is preferred. For practical reasons, a constant power envelope is also desired; this suggests the use of codewords in which every symbol occurs a fixed number of times in each codeword (a *constant composition code*). Combining these requirements naturally leads us to the consideration of a *permutation code*, which has alphabet $\{1, 2, \dots, n\}$ and so possesses the property that each symbol occurs precisely once in each codeword. Another way of saying this is that each codeword is a *permutation*, or rearrangement, of the symbols $\{1, 2, \ldots, n\}$. The set of all possible permutations on $n$ symbols is a well-studied mathematical object, called the *symmetric group* and denoted by $S_n$.

If we form an array by taking, as the rows of the array, the codewords of a permutation code C of length $n$ and minimum distance $d$, then the array possesses the following properties:

— each row contains the symbols $\{1, 2, \ldots, n\}$ in some order, and
— any two rows disagree in at least $d$ columns, i.e. agree in at most $n-d$ columns.

Such an array is called a *permutation array*, written as PA($n$, $d$). If the distance between any two rows is *precisely* $d$, we have an *equidistant permutation array*. An example of a PA(5,5), i.e. the codewords of a permutation code C with length and minimum distance both 5, is shown as follows (it is in fact equidistant):

$$\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \\ 3 & 4 & 5 & 1 & 2 \\ 4 & 5 & 1 & 2 & 3 \\ 5 & 1 & 2 & 3 & 4 \end{array}.$$

The first codeword would be transmitted in timeslots $t_1, t_2, t_3, t_4, t_5$ as the frequencies $\{f_1, f_2, f_3, f_4, f_5\}$.

Once the encoded message is transmitted, the process of *demodulation* must occur at the receiving end. The simplest demodulator detects $n$ envelopes and outputs, as estimate for the transmitted symbol (frequency), the one which corresponds to the largest envelope. However, this approach is not ideal for powerline channels; in particular, the broad-band nature of impulse noise can lead to many large envelopes being detected. A better approach, which corresponds naturally to the decoding process for permutation codes, is to use *threshold demodulation*. For each sub-channel, a threshold is set, and the demodulator outputs all symbols (frequencies) which correspond to envelopes greater than the threshold. This means, of course, that we may obtain more than one candidate symbol for a given position in our message. Decoding is then performed by giving, as output, the codeword in C, which has the maximum number of agreements with the demodulator output. This scheme allows the correction of up to $d-1$ incorrect demodulator outputs, caused by any of the possible sources of noise.

For example, take the code C above, with length 5 and minimum distance 5. We send the codeword 45123, which is transmitted as $\{f_4, f_5, f_1, f_2, f_3\}$. Suppose that it suffers narrow-band noise at the sub-channels for frequencies $f_2, f_3$ and $f_4$, leading to demodulator output

$$\{(f_2, f_3, f_4), (f_2, f_3, f_4, f_5), (f_1, f_2, f_3, f_4), (f_2, f_3, f_4), (f_2, f_3, f_4)\}.$$

Hence, the candidate codeword has the following symbols in its five positions:

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

Comparing this with the five codewords of C, there is agreement in all five positions with 45123 and in three positions with the other codewords, so it will be correctly decoded. Now suppose that this transmission also suffers impulse noise at the fifth timeslot, causing all symbols (frequencies) to appear in the fifth position. We obtain output

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

This agrees with the codewords of C in 4,4,3,5 and three positions, respectively, and so the output is again correctly decoded to the sent codeword.

We next ask: how can we find such permutation codes or, equivalently, permutation arrays? In particular, how can we find 'best possible' codes, with maximum size for given length and distance?

## 4. Building permutation arrays

The number of permutations on $n$ points is given by $n\times(n-1)\times(n-2)\times\cdots\times2\times1$, which we write as $n!$ (and say as '$n$ factorial'). This number arises because we have $n$ choices for the entry in the first position, then $n-1$ choices for the entry in the second position, and so on, until there is only one possibility for the final position.

If we take a permutation and try to make a new permutation by changing a single entry, we find that this is impossible; we must change at least two entries. For example, for $n=5$, try this with the permutation 12345. If we change the first entry from 1 to 3, we get 32345; this is not a permutation, and to convert it into a new permutation, we must change the original 3 into a 1, to get 32145. Hence, any two permutations of $\{1, \ldots, n\}$ which are not identical must disagree in at least two positions, i.e. have minimum distance 2. Thus, the set $S_n$ of all permutations of length $n$ is a PA($n$, 2).

This code can deal with at most a single error, so it should be possible to do better; however, the trade-off is that we will have to reduce the number of codewords.

Recall the code C from §3, with five codewords, each of length 5 and minimum distance 5. The PA($n$, $n$) is given by

$$\begin{array}{ccccc}1&2&3&4&5\\2&3&4&5&1\\3&4&5&1&2\\4&5&1&2&3\\5&1&2&3&4\end{array}.$$

An array of this type, consisting of $n$ rows and $n$ columns, in which each of $n$ symbols occurs exactly once in each row and column, is called a *latin square* of order $n$. Such squares were first systematically studied by the mathematician Leonhard Euler (1707–1783), shown in figure 3, who introduced them in 1783 as 'une nouvelle espéce de quarrès magiques', a new kind of magic square.

Since Euler's time, latin squares have been extensively studied; they play a key role in the mathematical discipline of combinatorics and have important applications to the design of experiments in statistics. One useful result known about latin squares is that if we choose any permutation for the first row, and add

Figure 3. Leonhard Euler (1707–1783).

new rows one-at-a-time subject only to the condition that they *could* form the rows of a latin square, then they *do*, i.e. we can always complete $k$ such rows (for $k$ less than $n$) to a full latin square of order $n$.

A latin square of order $n$ is a PA($n$, $n$). Clearly, we cannot have more than $n$ rows in a PA($n$, $n$): for, if we try to make a new row, what symbol would we put in the first position? Thus, a latin square gives a permutation code which can be used in the previous scheme to correct up to $n-1$ errors, although the trade-off is the fairly small number of codewords ($n$ out of a possible $n!$).

For permutation arrays PA($n$, $d$) whose minimum distance $d$ lies between 2 and $n$, the situation is much less clear-cut. In many cases, the maximum possible size of a PA($n$, $d$) is not known and, even in cases where it is known, we do not always know how to build a permutation array which attains the bound. Using a range of different tools, mathematicians and engineers have been able to construct PA($n$, $d$)s for various values of $n$ and $d$, and obtain some estimates for the maximum number of codewords that such codes can have. In the rest of this section, we discuss a variety of these results, including some very recent work.

## (*a*) *How big can a permutation code be?*

We begin by asking: for a PA($n$, $d$) with $d$ between 2 and $n$, how many rows (equivalently, how many codewords) can it have? We shall denote the maximum possible size of a PA($n$, $d$) by M($n$, $d$). Various bounds are known; one of the most useful is the following (e.g. Blake *et al.* 1979):

$$\mathrm{M}(n, d) \le \frac{n!}{(d-1)!}.$$

In particular, this tells us that a PA($n$, 2) has maximum size $n!$, while a PA($n$, $n$) has maximum size $n$; we have just seen how to make codes which achieve these bounds. A permutation code C, or the corresponding PA($n$, $d$), is called *sharp* if its size equals this upper bound. Various examples of sharp permutation arrays are known, many (but not all) arising from *permutation groups* (see Blake *et al.* 1979). Here is a sharp PA(4,3) of size $4!/2! = 12$,

$$\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & 1 & 4 & 3 \\
3 & 4 & 1 & 2 \\
4 & 3 & 2 & 1 \\
1 & 3 & 4 & 2 \\
2 & 4 & 3 & 1 \\
3 & 1 & 2 & 4 \\
4 & 2 & 1 & 3 \\
1 & 4 & 2 & 3 \\
2 & 3 & 1 & 4 \\
3 & 2 & 4 & 1 \\
4 & 1 & 3 & 2
\end{array}.$$

Thinking in a more geometric way, we can obtain further bounds. Given a sheet of paper and a pair of compasses, we can describe the set of all points at distance $d$ or less from a point $P$, in two-dimensional space, by drawing the circle with centre $P$ and radius $d$. In a similar way, given a permutation $\sigma$ in $\mathrm{S}_n$, we can consider all the permutations which have Hamming distance $d$ or less from $\sigma$. We call the set of all such permutations the *sphere of radius d* with centre $\sigma$, and denote the size (*volume*) of this set by $\mathrm{V}_d$. Just as a circle with radius $d$ will have the same area, regardless of which point $P$ we choose for its centre, the number of permutations in a sphere of radius $d$ in $\mathrm{S}_n$ is independent of our choice of $\sigma$.

We can now use the ideas of 'covering' and 'packing' using spheres. For a code C whose codewords form a PA($n$, $d$) of maximum size, every permutation of $\mathrm{S}_n$ not in C must be within distance $d-1$ of a member of C. This idea of 'covering' the space with spheres of appropriate radius leads to the lower bound,

$$M(n, d) \ge \frac{n!}{\mathrm{V}_{d-1}}.$$

Moreover (broadly speaking), spheres of radius $(d-1)/2$ around the codewords cannot overlap. Using this idea to 'pack' as many spheres as possible into the space yields the upper bound,

$$M(n,d) \le \frac{n!}{\max((d-1)!,\, T(n,d))},$$

where $T(n,d)$ depends on $n$ and $d$ and is essentially $\mathrm{V}_{(d-1)/2}$ with a 'correction term' for even $d$. More details can be found in Deza & Frankl (1977). The bounds are the analogues of the Gilbert–Varshamov bound and the Hamming bound, well known in coding theory. The techniques used to prove these results can also be used to improve the performance of computer searches for permutation arrays, as described in the paper of Chu *et al.* (2004).

In general, if we take a code C and consider the non-overlapping spheres with radius $(d-1)/2$ centred on its codewords, as above, we will find that the spheres do not cover all of $\mathrm{S}_n$. In other words, there will be permutations of $\mathrm{S}_n$ which are not contained in any of these spheres. In the special case when every permutation of $\mathrm{S}_n$ does lie within one of these spheres, the permutation code, and its corresponding $\mathrm{PA}(n,d)$, is said to be *perfect*. Such codes satisfy the upper 'packing' bound with equality. In the setting of traditional coding theory, there is an analogous concept of perfect codes, and some famous examples are known. However, the existence of a (non-trivial) perfect permutation code is still an open question. Based on arithmetic conditions, it was thought for some time that a good candidate might be a PA(11,5), but in fact work of Rothaus & Thompson (1966) shows that there cannot exist a perfect permutation array with these parameters.

Pursuit of upper bounds for the size of permutation arrays remains a fertile area for research, using a variety of ingenious methods. For example, Tarnanen (1999) has applied the technique of *linear programming* to this problem.

### (*b*) *Building codes using theory*

Permutation codes can be obtained in two ways: by using purely theoretical methods or by searching with a computer (perhaps using some theory to enhance the search strategy). To give a flavour of the first approach, we focus on three examples: using groups; polynomials; and latin squares. Other methods, which could have been explored had space allowed, use *balanced incomplete block designs*, *transversal packings* and *distance preserving mappings*.

#### (i) *Codes from groups*

A *group* is a set $G$ of objects, together with an *operation* '$*$' which allows us to take two objects $x$, $y$ from $G$ and combine them to make a new object $x*y$ which is also in $G$. To qualify as a group, the set and operation must also satisfy a set of *axioms*, which guarantee that they are 'well behaved'. For example, the set of all positive, zero and negative numbers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (the *integers*), with the operation of addition, form a group.

As was mentioned earlier, the set of permutations of $\{1, 2, \ldots, n\}$ forms a group, $\mathrm{S}_n$. When working with permutations in this setting, it is useful to think of them as mappings from the set $\{1, 2, \ldots, n\}$ to itself. Specifically, we view the permutation $\sigma = a_1a_2 \ldots a_n$ as a mapping which sends $i$ to $a_i$, the symbol in the $i$th position. Hence, the permutation 3214 in $\mathrm{S}_4$ is the mapping which sends $1 \mapsto 3$,

$2 \mapsto 2$, $3 \mapsto 1$ and $4 \mapsto 4$. A set of permutations is said to be *k-transitive* if, for any two ordered sets of $k$ symbols from $\{1, 2, \ldots, n\}$, there is some permutation in the set which sends one set to the other. If the mapping which does this is always unique, the set is called *sharply k-transitive.* We can show that a sharply $k$-transitive group is a sharp PA$(n, d)$, with $d = n - k + 1$ (see Blake *et al.* 1979). Such groups are well known and understood by mathematicians, so this provides a useful source of permutation codes.

### (ii) *Codes from polynomials*

Consider a *polynomial*, which we express as

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

with *coefficients* $a_n$. If $a_n$ is non-zero, then $n$ is called the *degree* of $f$. The number $\alpha$ is said to be a *root* of $f$ if $f(\alpha) = 0$. For example, working with the integers, the polynomial $f(x) = x^2 - 1$ has two roots 1 and 1, since $f(1) = 1^2 - 1 = 0$ and $f(-1) = (-1)^2 - 1 = 0$. A well-known mathematical result tells us that if the coefficients are taken from a well-behaved set of numbers called a *field*, then $f(x)$ cannot have more than $n$ roots in the field. This fact allows us to construct permutation arrays by using polynomials over so-called *finite fields*, i.e. fields with a finite number of elements. Each row corresponds to a different *permutation polynomial* of the finite field (a polynomial which, as a function, permutes the elements of the field), and the limit on the number of roots can be used to guarantee a certain minimum distance between the rows.

Recall that a prime number $p > 1$ is a number which is divisible only by itself and 1; the first few primes are $2, 3, 5, 7, 11, \ldots$. It can be proved that a finite field with $n$ elements exists precisely if $n$ is a *prime power*, i.e. $n = p \times p \times \cdots \times p$ for some prime number $p$.

**Theorem 4.1.** *Suppose $n$ is a prime power and that there are $E$ permutation polynomials over the finite field of $n$ elements with degree less than or equal to $d$. Then, we can construct a PA$(n, n - d)$ of size $E$.*

More details may be found in Chu *et al.* (2004).

### (iii) *Codes from latin squares*

We saw earlier how the eighteenth century mathematician Leonhard Euler introduced latin squares. One problem which particularly interested him was the so-called '36 officers problem', which he stated as follows:

> A very curious problem, which has exercised for some time the ingenuity of many people, has involved me in the following studies, which seem to open a new field of analysis, in particular the study of combinations. The question revolves around arranging 36 officers to be drawn from 6 different ranks and also from 6 different regiments so that they are ranged in a square so that in each line (both horizontal and vertical) there are 6 officers of different ranks and different regiments.
>
> (Euler 1782)

This problem can be restated in terms of latin squares as follows. Let the six ranks be represented by the symbols $\{1,2,3,4,5,6\}$ and the six regiments by the symbols $\{A,B,C,D,E,F\}$. We want to form two latin squares, one using the symbols $\{1,2,3,4,5,6\}$ and the other using $\{A,B,C,D,E,F\}$, such that when we superimpose them, each possible number–letter pair occur precisely once. Two latin squares of order $n$, which give each of the possible $n^2$ ordered pairs when superimposed, are said to be *orthogonal*. A set of latin squares of order $n$ is said to be a set of MOLS if any two squares in the set are orthogonal to each other. For example, two MOLS of order 4 are

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{matrix} \quad \text{and} \quad \begin{matrix} A & B & C & D \\ C & D & A & B \\ D & C & B & A \\ B & A & D & C \end{matrix}.$$

When superimposed, they yield each of the 16 possible pairs

$$\begin{matrix} (1,A) & (2,B) & (3,C) & (4,D) \\ (2,C) & (1,D) & (4,A) & (3,B) \\ (3,D) & (4,C) & (1,B) & (2,A) \\ (4,B) & (3,A) & (2,D) & (1,C) \end{matrix}.$$

Euler could not find a pair of MOLS of order 6, and conjectured that no such pair can exist for any $n$ of the form $n=4k+2$. This conjecture remained unresolved until early in the twentieth century, when it was confirmed that no pair of order 6 exist, but that a pair can be found for every other value of $n$ greater than 2. The largest possible size for a set of MOLS of order $n$ is $n-1$; a set of this size is a *complete set*. In particular, we know how to construct such a set when $n$ is a prime power.

The following result has recently been proved in Colbourn *et al.* (2004).

**Theorem 4.2.** *If there are m MOLS of order n, then a PA(n, n−1) exists of order mn. In particular, when n is a prime power, a sharp PA(n, n−1) exists, of order n(n−1).*

Essentially, the orthogonality property of the latin squares is exploited to obtain the necessary minimum distance in the array.

### (c) *Building codes by computer search*

For some values of $n$ and $d$, it is not easy to create a PA($n$, $d$) by the methods described above. In these cases, we can use a computer to search among the permutations of $S_n$ for an appropriate set of codewords. Perhaps the simplest way to build a permutation code by computer is to start with an empty set of codewords, run through all permutations of $S_n$, and add a permutation to the set if it has distance $d$ or more from all permutations already in the set. This is called the *greedy algorithm*. Since the number of permutations of length $n$ increases very quickly as $n$ increases, this method is practical only for $n$ up to 10 or 11 (11! is close to 40 million). By modifying the algorithm in various ways, and applying it repeatedly, we can make it work more effectively; however, it remains essentially a 'brute force' approach.

A more sophisticated version of the above is to insist that the permutation array we are searching for possesses certain structural properties (e.g. possesses a certain *automorphism group*). Although, at first sight, imposing an extra condition may seem counter-productive, it is in fact helpful; the array can then be specified by a smaller set of representatives, allowing us to reduce the size of the search space.

Another approach is to use *graph theory*. In combinatorics, a *graph* consists of a set of *vertices* (points) and *edges* (lines between points). A *clique* in a graph is a set of points with the property that any pair of points in the set is joined by an edge. We can make a graph whose vertices are all the permutations of length $n$, with an edge between two vertices if the Hamming distance between the corresponding permutations is $d$ or more. Then, finding a PA($n$, $d$) is equivalent to finding a clique in the graph; search techniques developed for graph theory can then be used to search for as large a clique as possible.

## 5. Future developments: where next?

So, what does the future hold for permutation arrays and PLCs? How will this promising relationship between pure mathematics and electronic engineering develop?

We foresee more research, especially by mathematicians themselves, on effective encoding and decoding schemes for permutation codes. For example, recent work by Bailey (2005) takes permutation codes which are permutation groups and develops a novel decoding algorithm for them. This decoding process uses a so-called *uncovering by bases*, which exploits group-theoretic properties of the chosen codes and uses combinatorial designs. There is a rich mathematical literature pertaining to permutations, and we envisage other aspects of permutation theory being used to develop increasingly sophisticated coding applications.

We foresee the development and investigation of related codes and structures. To be useful in PLC, as described earlier, we want a code in which each symbol occurs a fixed number of times per codeword (to ensure a constant power envelope) and which uses a relatively large alphabet size (for frequency spreading). Such codes are called *constant composition codes*. Permutation codes possess an important extra property not possessed by most constant composition codes: the ‘fixed number of occurrences’ of a symbol in a codeword is, in fact, the same for each symbol (namely 1). This consistency seems to be a key reason why so many theoretical constructions can be developed. Hence, a natural step is to consider the larger class of constant composition codes, for which the fixed number of occurrences is the same for each symbol, but where we now allow this number to be greater than 1. Several very recent papers have started to address this topic, from both a coding theory perspective (Ding & Yin 2005; Chu *et al.* 2006) and a more combinatorial viewpoint (Huczynska & Mullen 2006). This last paper introduces the concept of a *frequency permutation array*, the natural generalization of a permutation array.

At the most general level, the PLC requirements outlined above suggest undertaking an investigation of constant composition codes with larger-than-usual alphabet sizes. Until very recently, almost all work on this topic has focused on alphabets of size 2 or 3 (so-called *binary* or *ternary* codes). Motivated by this, several researchers have recently begun to consider constant composition codes with arbitrary alphabet size (see, for example, recent work of Luo *et al.* 2003; Ding & Yin

2005). Compared to the extensive literature already existing for binary and ternary constant composition codes, this is still an emerging area, with much to be discovered.

This research also presents a challenge to computer scientists. We touched briefly on the construction of codes by computer search, describing some algorithms which have been employed by mathematicians. However, efficient search constitutes an entire research area in theoretical computer science, and it is probably that more sophisticated search techniques from this area, such as *backtrack search*, will be able to discover close-to-optimal codes for a wide range of parameters. This would be particularly helpful for constant composition codes, where the number of symbol occurrences is non-uniform.

Finally, there still remain many engineering challenges which must be overcome before the promise of PLC can become a reality. We need to obtain a better understanding of the nature of noise problems which can occur. It is desirable to undertake research into other modulation/coding methods, to identify other schemes which are robust against these problems. Beyond the realms of research, there are agreements to be made, by industry and government, on issues such as standards.

Yet, although there is still much work to be done before the dream of 'Internet through power sockets' becomes a reality, the pursuit of this dream is already having a tangible positive impact on the research community. The needs and demands of PLCs are stimulating new connections between, and new directions within, pure mathematics and electronic engineering. Novel engineering approaches are being developed, for which previously untapped seams of mathematics are being mined, and this new interest is encouraging theoreticians to explore new aspects of their field, or to revisit and develop old areas. In the future, we anticipate the two disciplines working evermore closely, towards the development of sophisticated, application-led encoding/decoding strategies which overcome the technical problems of PLC implementation. This is an exciting and timely opportunity for interdisciplinary research. And when, one day in the future, we are downloading data through power sockets, we might spare a thought for Euler and the 36 officers.

The author is supported by a Royal Society Dorothy Hodgkin Fellowship. Thanks to John O'Connor for kindly supplying the picture of Euler and to Gary Mullen for his encouragement. Thanks to the anonymous referees for their helpful comments.

## References

Bailey, R. F. 2005 Permutation groups, error-correcting codes and uncoverings. Ph.D. thesis, Queen Mary, University of London.

Blake, I. F., Cohen, G. & Deza, M. 1979 Coding with permutations. *Inf. Control* **43**, 1–19. (doi:10.1016/S0019-9958(79)90076-7)

Chu, W., Colbourn, C. J. & Dukes, P. 2004 Constructions for permutation codes in powerline communications. *Des. Codes Cryptogr.* **32**, 51–64. (doi:10.1023/B:DESI.0000029212.52214.71)

Chu, W., Colbourn, C. J. & Dukes, P. 2006 On constant composition codes. *Discrete Appl. Math.* **154**, 912–929. (doi:10.1016/j.dam.2005.09.009)

Colbourn, C. J., Kløve, T. & Ling, A. C. H. 2004 Permutation arrays for powerline communication and mutually orthogonal latin squares. *IEEE Trans. Inf. Theory* **50**, 1289–1291. (doi:10.1109/TIT.2004.828150)

Deza, M. & Frankl, P. 1977 On maximal numbers of permutations with given maximal or minimal distance. *J. Comb. Theory A* **22**, 352–360. (doi:10.1016/0097-3165(77)90009-7)

Ding, C. & Yin, J. 2005 Combinatorial constructions of constant composition codes. *IEEE Trans. Inf. Theory* **51**, 3671–3674. (doi:10.1109/TIT.2005.855612)

Euler, L. 1782 Recherches sur une nouvelle espéce de quarrès magiques. *Verh. Zeeuwsch. Genootsch. Wetensch. Vlissengen* **9**, 85–239.

Han Vinck, A. J. 2000 Coded modulation for powerline communications. *AEÜ J*, 45–49.

Huczynska, S. & Mullen, G. L. 2006 Frequency permutation arrays. *J. Comb. Designs* **14**, 463–478. (doi:10.1002/jcd.20096)

Luo, Y., Fu, F.-W., Han Vinck, A. J. & Chen, W. 2003 On constant-composition codes over $\mathbb{Z}_q$. *IEEE Trans. Inf. Theory* **49**, 3010–3016. (doi:10.1109/TIT.2003.819339)

Pavlidou, N., Han Vinck, A. J., Yazdani, J. & Honary, B. 2003 (April) Power lines communications: state of the art and future trends. *IEEE Commun. Mag.* **41**, 34–40. (doi:10.1109/MCOM.2003.1193972)

Rothaus, O. & Thompson, J. G. 1966 A combinatorial problem in the symmetric group. *Pac. J. Math.* **18**, 175–178.

Sankar, J. 2004 An assessment of power line communications for the delivery of broadband network access, Technical Report, UKERNA.

Tarnanen, H. 1999 Upper bounds on permutation codes via linear programming. *Eur. J. Comb.* **20**, 101–114. (doi:10.1006/eujc.1998.0272)

## AUTHOR PROFILE

**Sophie Huczynska**

Sophie Huczynska was born in 1978. She obtained an MSc degree in mathematics (with first class honours) from the University of Glasgow, graduating in 1999. She continued in the Department of Mathematics and Statistics at the University of Glasgow for her PhD, which was in the area of finite fields. Upon completing her PhD in December 2002, she became a research associate in the Mathematical Reasoning Group in the Division of Informatics at the University of Edinburgh. Since September 2003, she has been working at the School of Mathematics and Statistics at the University of St Andrews, initially as a teaching fellow and currently (since October 2004) as a Royal Society Dorothy Hodgkin Research Fellow. Her research interests lie in the areas of number theory, combinatorics, information theory and theoretical computer science; she is particularly interested in the intersections between these areas. Non-scientific enthusiasms include art-house cinema, yoga and cooking for friends.