Data: The Past, The Present and The Future
Professor Richard Harvey FBCS

24 October 2020

This is the second in series of lectures called “Great Ideas from Computer Science. In the first lecture I looked at algorithms and we discovered that some rather fantastical things – algorithms which went from being completely tractable to intractable as the problem increased in size. We also discovered that often the nub of a successful algorithm is a novel way of representing the data. That is what I want to look at here.

Data as a concept seems to quite antique and one of the earlier Gresham Professors, Samuel Pepys, would have used the word as we would today[1]. Of course the origins are as the plural of *datum* meaning a geographical point or reference point. And generations of undergraduates are put the wheel for writing “the data is” in the singular[2]. For the purposes of this lecture, I am interested in the meaning of the word as “digital stuff” and I shall not worry too much about the latinate origins of the word not whether it is plural or singular.

As far as early computers went, the main problem was jamming data into their puny memories. To see this in detail it is helpful to consider how a basic computer works. In Figure 1, the computational part is called the CPU and the part that stores most of the data is called memory[3]. When the CPU wants to read some data it sets the read/write line to high or low (depending on who designed the computer) which instructs the memory to be in read-only mode. The CPU then sets the, let’s say sixteen, lines on the address bus to a binary number and that number then forms an address. The memory then switches to that address and sets the data-bus lines to be whatever is stored at that address. A few nano-seconds later, the CPU reads the value. The number of data lines is called the “width” of the memory and in early devices it tended to be eight because the CPU was designed to handle 8-bit numbers.

[1] The Oxford English dictionary gives the earliest use of data in “A most plaine and easie way of finding the Sunnes Amplitude and Azimuth” 1630 and credits Admiral William Batten who was a colleague of Samuel Pepys. Something that Pepys would have found infuriating since he disliked Batten. Given that Pepys was a Gresham Professor and Batten was not, loyalty demanded that I searched hard to find an earlier reference to data or at least an earlier reference to data in the sense of numerical evidence. More information on the etymology of data see [7].

[2] A colleague has speculated that we should have a rubber stamp marked “Data are a plural” which we can stamp in the margins of work by the uneducated or possibly on their foreheads.

[3] Memory is rather loose term and a few years ago we would have been quite pedantic about differentiating between memory that we could only read from (Read Only Memory or ROM) or memory from which we could read and write from (confusingly called Random Access Memory even though there is nothing random about it) and disc memory (or disk if you are American). Disc memory stores information even after the power is turned off, unlike RAM, but is incredibly slow compared to RAM. But nowadays these technologies have all merged into one so let’s park it for a bit.

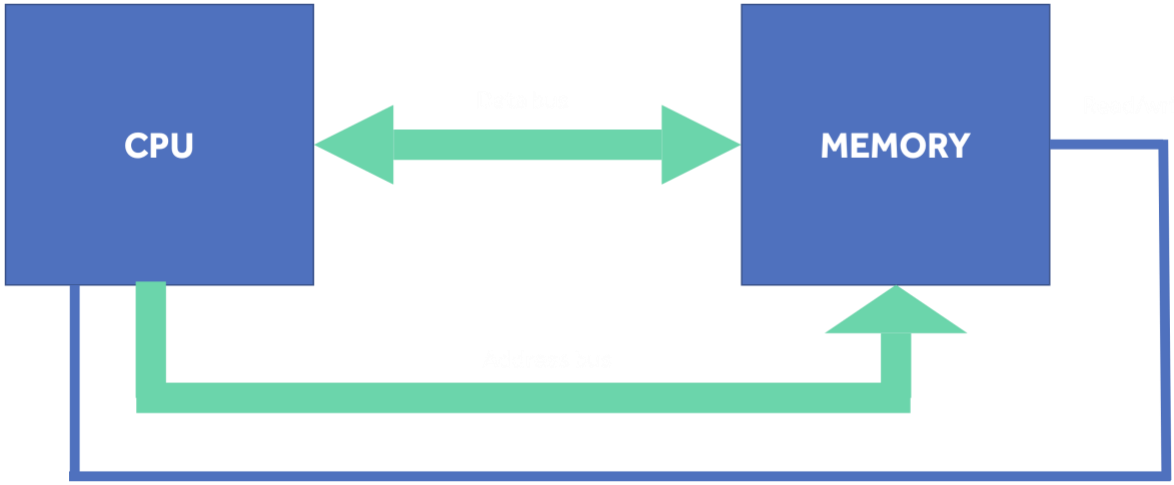

Figure 1: *Showing a central processor unit (CPU) and some storage or memory*

So, the challenge for software designers was how to fit things into 8-bits or, if not 8-bits, then multiples of 8-bits. An 8-bit word can hold $2^8 = 256$ combinations which is not very many! Thus, 8-bit *words* can be used to hold yes/no quantities (often called *logicals* by programmers)[4]. 256 combinations are also just enough to represent the English character set and a few symbols which can be used to indicate other things – a fact that was discovered much earlier with the invention of the teleprinter[5].

But what about numbers? Clearly you would not find it very congenial if your spreadsheet could only hold numbers (0, …, 255) which would be the implication of using 8-bit words. So, computer designers realised that then had to concatenate bytes to hold numerical values. Each CPU has its own standard for representing things. And for speed and convenience, software that runs on that type of computer will usually adopt the "native" date types used by that CPU. This means that translating binary data stored on one machine to another machine can be a hassle and in the 60s, 70s and even 80s many a PhD student wasted hours writing conversion routines[6]. But what about numbers that are not whole numbers? Non-integers. Computers have two ways of representing those: *fixed-point* numbers which are now not used very much and *floating-point* numbers which are ubiquitous. Floating-point numbers were another *bête noir* of graduate students because there were many early computers, each with their own definition and, frankly, it was soon discovered that the numerical analysis of several computers was lousy[7].

The idea behind floating point numbers is to represent all real numbers as

$$\pm M \times 10^E$$

[4] In fact, yes/no quantities only use two states, so an 8-bit word can hold eight logicals although modern systems often map a logical to the whole word to save the time associated with unpacking the bits.

[5] If you have ever wondered why those extra characters are called "control characters" then I am happy to tell you that they were used for controlling the flow of data down teleprinters ("flow control" became "control" and hence control characters). So that typists could type control characters, early keyboards contained a control-key (marked CTRL on the PC) – one held-down the control key and hit the appropriate letter and a control symbol was sent. Try opening a terminal on your Mac or PC and hit control-G. You should hear the bell associated with an old teletype.

[6] And I have not mentioned endianness. If a computer stores the least significant bit of a word at a lower address than the most significant bit, then it is known as little-endian machine. Big endian mutatis mutandis. If you have an Intel PC then you are little-endian, if you have a PowerPC Macintosh then you are big-endian. If you have an ARM processor then you are bi-endian. How modern! If you are racking your brains – the Endians were two tribes in Gulliver's Travels who disagreed most violently over a very trivial thing.

[7] It was commonplace to run a program called paranoia that ran many floating point calculations and then told you the disappointing news that your computer, which cost many, many, thousands of dollars, was not rounding properly.

Where *M* is a number called the *mantissa* and *E* is the *exponent*. The more general form replaces 10 with *B* a chosen *base*.

Thus, the road distance from UEA to Barnard's Inn which is 186.6839 km would be represented as 1.866839×10^3. Sometimes computer floating point numbers use base ten but more often base two. Clearly when we convert a number like 1.866839×10^3 to 1.8231×2^{10}, truncations can happen but, with good numerical analysis one can keep these under control. To see this in action I created an Excel spreadsheet with a thousand random numbers and added them up. It should make difference whether I sort the numbers from smallest to largest, or visa versa, or leave as it. But it does. In my example the sums differed by roughly 5×10^{-7}. Not much, but annoying if you are calculating a moonshot trajectory[8][9]. Fortunately, some of the madness of multiple formats for floating point numbers has dissipated and there is now an international standard known as IEEE 754 which specifies how floating-point numbers should be stored, rounded and processed.

Early computers had to be programmed at an incredibly low level: move whatever you find at this memory location to register A; move whatever you find at this other memory location to register B; apply a logical OR operation to registers A and B and store the result in A; store register A to some other memory location. This sort of program was very time consuming and irksome to construct since common operations, such as adding two floating-point numbers together, would take many hundreds of instructions which had to be typed-in (or input on cards or tape). As computers grew in power, people realised that it would be a lot easier if we could program them in a *high-level* language and that it was feasible to write programs called *compilers* that converted the high-level language into *machine code*[10]. These high-level compilers tended to insulate programmers from the tiresome low-level stuff which meant it was possible to be more productive but, like a driver who neither knows nor cares how an automobile works, there are potential difficulties when things go wrong.

High-level languages introduced the second age of computer data which is when the data are structured to suit the problem rather than the hardware. This was the time when we started to refer to *data-structures* which were semantically related groups of data, usually stored in contiguous memory locations. The simplest one I can think of is an *array,* which is a collection of numbers, like a column of data in Excel and stored one after another in memory. Accessing arrays is fast because, if we know the address of the first element, then we need only add the offset, which is the width of the word in each array element, to get to the next element. If we want, say, the fifth element then we add to the *base address*, five times the offset and bingo! Arrays are fine if we know in advance how many elements we want to hold, as we often do in scientific computing, but if we have variable amounts of data then it's a bit of faff to check that we have not overrun the end of the end of the array and, if we have, re-dimension a new array. The solution is a *linked list*, which is a staple data structure of undergraduate computer science courses. In a linked list each element holds the data and a pointer to the next element in the list[11]. It's a bit bulky, because of all those addresses, but it allows complete flexibility. Data structures that can change as the program runs are called *dynamic*

[8] There are summation algorithms that attempt to estimate the error as you go along the list and subtract it at the end [5]. I'm afraid I did not have time to work out if Microsoft Excel is using one of these compensated summation algorithms. Given the huge number of people using Excel to add floating point numbers then I should jolly well hope so!

[9] I should also point out that spreadsheets are notorious for creating errors. There is a whole literature on spreadsheet errors. Some of it is summarised here [8].

[10] Actually, there is a zoo of exotic programs that create code: loaders, assemblers; interpreters; cross-compilers and compilers. Which is to say nothing of the menagerie of programs that allow you to build such things. One of my favourites is yacc which is unix tool which allows you to compile compilers. Yacc is an acronym, Yet Another Compiler Compiler, which implies that computer scientists spend rather too long writing compilers for exotic languages.

[11] A linked list with two addresses to objects is a *tree* or *binary tree* (binary because each node has two children rather than because its holds binary data).

data structures[12]. In the slides I show how choosing the right data structure can make huge differences to the performance of algorithms. The example I picked is search, which I also looked at in the previous lecture on algorithms. I picked search partly so there is a link between the two lectures but also as a segue to databases.

A word of warning on databases. In the previous lecture on algorithms [1] you may have got the impression, via Cobham-Edmonds thesis, that pretty much any algorithm which had polynomial complexity or less was trivial to implement as quickly as we want. That's not true, and the topic of databases brings that untruth to the fore: many of the things we need to do to databases look computationally very tractable, but when we are handling gazzillions of data items, and thousands of irate Amazon customers trying to search for a present days before Christmas, speed is critical.

Is a database anything more than a great big file of data? All the products on the Amazon website for example? Well, yes and no. It's true that when you search for something at Amazon, you are issuing queries to a database[13] but those data need to be stored very carefully on Amazon's servers otherwise your query might take hours. The usual first step to creating a database is *normalisation* which is a fancy way of splitting the data into tables which might be easier to update. Database specialists have a horror of inconsistency and errors (*data integrity* is Queen) and since humans enter data into databases, errors are rife[14].

To make it more concrete I created a simple *flat-file* database that contains the Gresham College lectures. A segment is shown in Table 1.
Table 1: *A section of an example flat-file database*

Title	Date	Time	Lecturer	Slides	Tech support	Publicity and liaison	Transcript length	Series
Snow white: evil witches	19/11/2020	18:00	Joanna Bourke	powerpoint	James	Lucia	10	Evil women
Understanding the universe with AI	23/11/2020	13:00	Roberto Trotta	powerpoint	James	Lucia	6	The unexpected universe
Data: the past, the present and the future	18/11/2020	18:00	Richard Harvey	keynote	James	Claire	4	Great Ideas from Computer Science
The changing geography of ill health	25/11/2020	18:00	Chris Witty	keynote	Richard, James	Lucia	8	Major debates in public health

[12] It's a bit of fine point as to whether a data structure is *dynamic* or *static*. On the face of it, it is a simple categorisation – if the data structure can change size at run time then it is dynamic otherwise it is static. Some early languages, such as Fortran77 do not easily support dynamic data structures. That said, it is fairly simple to write a Fortran program that simulates a linked list using a static array for the storage thus creating a dynamic structure from a static one.
[13] https://www.amazon.co.uk/s?k=gresham+college is asking the Amazon database to search through all the titles of items looking for the words "Gresham" or "college" and return them ranked by, well who knows how they are ranked – by profitability for Amazon I should think.
[14] The study of databases is all about detail. For example, one of the common problem in databases is the problem of duplicates. Thus, almost all databases have "deduping" algorithms. If those work incorrectly then you will be on the receiving end of multiple pieces of identical junk mail – the mailing database being realise that Richard Harvey, Richard William Harvey, R W Harvey, R Hervey and R Hrvy are the same person (the latter version now being how cool kids now spell my name).

It all looks very innocuous until one realises that there are at least 2000 lectures online, and each year there are another 130 to cope with. Normalisations appear to, at first look, make the data structures look more complicated but they are designed to remove classic "gotchas" of database design such as the problem of having to update several tables when, for example, a customer's address changes. The First Normal Form (1NF) forbids columns that contain multiple values (Tech support in our example). This seems sensible enough – search is more complex when a column might return multiple items. Tables 2.1 and 2.2 are my reworking of the data – this time in 1NF. You might note that I have now introduced something called a *key*. A key is a unique identifier for each item in the database: keys are key feature of most databases.

Table 2.1: *A section of the database in Table 1 put into 1NF (Table 1)*

Key	Title	Date	Time	Lecturer	Slides	Publicity and liaison	Transcript length	Series
JB32020	Snow white: evil witches	19/11/2020	18:00	Joanna Bourke	powerpoint	Lucia	10	Evil women
RT22020	Understanding the universe with AI	23/11/2020	13:00	Roberto Trotta	powerpoint	Lucia	6	The unexpected universe
RH22020	Data: the past, the present and the future	18/11/2020	18:00	Richard Harvey	keynote	Claire	4	Great Ideas from Computer Science
CW12020	The changing geography of ill health	25/11/2020	18:00	Chris Witty	keynote	Lucia	8	Major debates in public health

Table 2.2 *A section of the database in Table 1 put into 1NF(Table 2)*

Tech support	Key
James	JB32020
James	RT22020
James	RH22020
Richard	CW12020

In the Second Normal Form (2NF), we aim to discover attributes that are not dependent directly on the key and remove them into separate tables. For example, the lecture series: we do not need to store the lecture series for every lecture and certainly knowing a lecture series does not uniquely identify the lecture. So, we would store it in separate tables: one for each series probably.

In the interests of expediency, I will avoid telling you about all the normal forms: there at least six of them. However, it is worth noting that these ideas all spring from a mathematical model of databases called the *relational* model which itself is based on predicate calculus. Databases[15] so formed are known as relational databases and nowadays all databases are relational (they may be other things too).

[15] Ted Cobb, who invented relational databases while at IBM Research Santa Clara, actually used the term data bank [6] but banks are never popular so we now have database.

In terms of practical databases, the field seems to have started in around the 1960s. One of the early systems was Sabre – the airline reservation system [2]. The system has become so well cited that a variety of apocryphal stories have grown up around the system including a meeting between an IBM Sales Executive and CR Smith, the founder of American Airlines on an aircraft in 1953 when the system was conceived. What is certainly true is that manual booking system was creaking at the seams – each hub booked seats using card-based boards or swivelling carousels. If you needed to book a flight you called the hub. What is less well talked about was, at the time, the system was regarded as a business failure: IBM were mainly interested in selling hardware at very high margins, and dramatically undercosted the system. The effect of this investment was that by the 1970s American Airlines was able to provide travel agents with electronic booking systems with the inevitable accusation, from other airlines, that the system was unfairly prioritising flights from American Airlines. Lawsuits followed and American Airlines vigorously defended themselves on the grounds that they had made a risky early investment in technology and were entitled to reap the rewards. Despite these ups and downs, Sabre is still going strong and remains a market-leader in airline reservation systems.

Relational databases came after the first large systems and in the late 70s lead to the formation of one of the software industry's behemoths, Oracle. Oracle was a champion of a new language, SQL (usually pronounced by the cognoscenti as "sequel"), that allowed programmers and operators to frame queries without the pain of having to write loops. For example, the SQL command SELECT TOP 1 Salary FROM Employee means please go through every row of the database and look at the entries in the column marked Salary. Find the highest value of Salary and then return the name in the Employee column.

A few years later the concept of object orientated (OO) programming became popular and there was much recherché debate about the extent to which relational databases were, or were not, object orientated. Marketers were very keen for databases to be object oriented but anyone who knew anything about programming, especially anyone who was trying to teach first-year undergraduates object oriented programming, realised that OO was over-hyped and indeed for many situations positively unhelpful (more on this in later lectures). However, one technique that has proved useful is the so-called Entity Relationship Diagram or ERD. The original graphical presentation proposed by Chen in 1976 [3] is now no longer common[16] but the ideas are gained traction and ERDs form one of a long list of graphical modelling tools that are now commonplace, not only in computer science, but in management. If you work in a large organisation you may well have a Systems architect or an Information architect or possibly a Data architect (possibly you have all three[17]). Such people spend hours locked in front of ERDs (DFDs – data flow diagrams) and other traditional instruments of torture in the hope that they can re-engineer your organization to keep the right data in the right place at the right time.

As we moved into the 1990s the internet became prevalent and it soon become evident that websites were great fun for individuals to create but allowed for a lot of individualism, whimsy and required considerable effort to maintain. And that was an anathema to any organisation that was trying to sell something. Hence the rise of Content Management Systems (CMS). In a CMS the webserver serves the page that may either itself be stored in a database or, more likely, builds the page from elements contained in the database. Because building pages in "on the fly" can lead to slow response time, it is normal to pre-build common pages and only change them when one of the components changes. At the heart of almost every substantial website is a database and it is true to say that, without the foundational work on databases in the 1970s, e-commerce as we know it today would not exist.

[16] In my slides I use a form known as the Information Engineering model.
[17] All three is clearly best as then they can have meetings with each other and not bother anyone else.

One of the enabling technologies of the web was the use of a *markup language*. Markup here is an analogy to the publishing process in which an copy editor "marks up" how a document is to be typeset. Hypertext Markup Language (HTML), which itself was a variant of a more general markup language, SGML (Standardised Generalised Markup Language), allowed authors to label parts of a webpage so that the web browser could render them in an appropriate manner. For example, the HTML this means render the word "this" in a way that denotes emphasis. For most browsers this would be italic, but for browsers that do not have an italic setting (a speech-reader for example) they would be able to do something else. HTML was the ultimate expression of substance over style – fancy graphic design was ditched in place of semantic *tags* such as which were read by the browser but not passed onto the user. An obvious extension was to realise that the same idea could be used to label data.

```
<!DOCTYPE memo
[
<!ELEMENT memo
(to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subj (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE memo []>

<memo>
<to>Simon Thurley</to>
<from>Richard Harvey</from>
<subj>Your lecture</subj>
<body>I thought your lecture on the Tudors
was excellent Simon. Not as excellent as
mine on data though!</body>
</memo>
```

Figure 2: *XML description of a simple memorandum*

EXtensible Markup Language (XML) generalises the idea of semantic markup and is frequently used to avoid the horror of data becoming separated from its meaning. To see how this works, consider Figure 2 in which I have defined two files. The first is the Document Type Definition (DTD) and is a description of what a memo should look like. In this case it must have a to-field, a from-field, a subject and a body. Each one of those elements can contain #PCDATA which is XML for parsed character data. On the right-hand side is the memo. There is some gobbledegook at the start, but otherwise it looks like a human readable description of a memo. This is powerful idea and when you are watching this lecture you might like to ponder the visual scene in front of you is created by a coder in a box in Barnard's Inn London which is analysing the scene and writing an XML description of what it sees. Your decoder in your PC, phone or Smart TV has the MPEG DTD and is able to render a reasonable approximation of the scene. Depending on how much money you spent on your decoder, and your bandwidth, your version will be more or less accurate.

XML means that the old problem of discs and discs of meaningless data should a thing of the past: the *metadata* sticks with the data[18]. When we were all working on MPEG-7, the retrieval standard, it was fashionable to state that data processing was all about the metadata and the metadata were more valuable than the data. That is clearly an exaggeration but the converse may well be true: data without metadata is meaningless. One advantage of XML is, as we can see on the slides where I give an example of how a computer stores a Powerpoint slide (it's compressed XML), it is human readable. This has also led to something of a movement in data science which is to keep data in human readable formats with XML encoding. Given that disc storage is cheap this is understandable, but it can lead to very uncomfortable amounts of data bloat especially as we enter the decade of "big data".

[18] Data as metadata is one of the senses of the word pre-1630 – Medieval scribes used to refer to their annotations as data.

Big data is a phrase coined by a team of consultants at the McKinsey Global Institute which is the research arm of McKinsey & Company. Since the original authors declined to provide a definition of "big" since then a number of less than satisfactory descriptions have been used. The three "v"s of big data state that big is characterised by high volumes of data, that may be rapidly changing (velocity) and coming from more sources than before (variety). Given the annoying imprecision about what big data are or what the three "v"s mean, yt is very difficult to be confident that big data is anything other than a meaningless buzzword useful only for corporate powerpoint slides and grant proposals. However, what is true is that the prevalence of machine learning and the cheapness of collecting and storing data can lead to surprising insights. In a previous lecture on Higher Education [4], I cited a result from a London university that had two entrances to their building. They discovered that students who came in one door were less likely to drop-out than those who came in the other door. That is a typical big data application: the automated gates were connected to a large database so each entry and exit could be labelled and connected, via a relational database, to the table of student grades. The analysts then ran a clustering algorithm and out popped the result[19].

Thus, the story of computer data has come full circle. We started with flat files of unstructured data, showed have the strictures of computers forced us to jam the data into structures. As computers progressed, data structures morphed from being a straightjacket to a tool for algorithm design and thought. The further discipline of a database allows very fast processing of data and encourages yet more analysis of organisations, their processes and data flows. Such analyses are improved by a range of formal and semi-formal tools for data analysis which in turn leads to improved ability to handle unstructured data and the connecting of large data sources and the formulation of the next chapter in data which is data science.

Bibliography

[1] R. Harvey, "An introduction to Algorithms," 20 October 2020. [Online]. Available: https://www.gresham.ac.uk/lectures-and-events/algorithms-intro. [Accessed 17 Nov 2020].

[2] R. V. Head, "Getting sabre off the Ground," *IEEE Annals of the Hisotry of Computing,* no. October-December, pp. 32--39, 2002.

[3] P. P.-S. Chen, "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transactions on Database Systems,* vol. 1, no. 1, pp. 9--36, 1976.

[4] R. Harvey, "The Digital University and Other Mythical Creatures," 11 Feb 2020. [Online]. Available: https://www.gresham.ac.uk/lectures-and-events/digital-university. [Accessed 17 Nov 2020].

[5] N. J. Higham, "The accuracy of floating point summation," *SIAM Journal on Scientific Computing,* vol. 4, no. 4, pp. 783--799, 1993.

[6] E. F. Cobb, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* vol. 13, no. 6, pp. 377--387, 1970.

[7] J. Furner, ""Data": the data," in *Information Cultures in the Digital Age: a Festschrift in Honor of Rafael Capurro*, Wiesbaden, Springer VS, 2016, pp. 287--306.

[8] J. Borwein and D. H. Bailey, "The Reinhart-Rogoff error - or how not to Excel at economics," 22 April 2013. [Online]. Available: https://theconversation.com/the-reinhart-rogoff-error-or-how-not-to-excel-at-economics-13646. [Accessed 17 Nov 2020].

[19] If you want to know *why* one set of students performs better than the other … well you will have to watch the lecture!