Setting Geodemographics in the Context of Developments in Computing and Artificial Intelligence

Peter Prescott

I. The Increasing Influence of Computers in Geography

It would be naive to consider the development of geodemographics without also attending to the technological progress that has enabled that development. Indeed, calls for an integration of compututional techniques and technologies with academic geographic study have come in periodic bursts over the last half-century or more, and from several slightly different angles, which we shall briefly summarize here.

Geography's first *quantitative revolution* took place in the 1950s and '60s (Burton, 1963); Haggett (2008) pictures the movement as a diffusion process of key figures carrying the ideas from department to department (Fig. 1). One of these key figures was Waldo Tobler, whose 'Computer Movie Simulating Urban Growth' (Tobler, 1970) is notable not just as pioneering work of computational geography, but for being the subject of the paper in which Tobler coined his now famous 'first law of geography' (Miller, 2004).

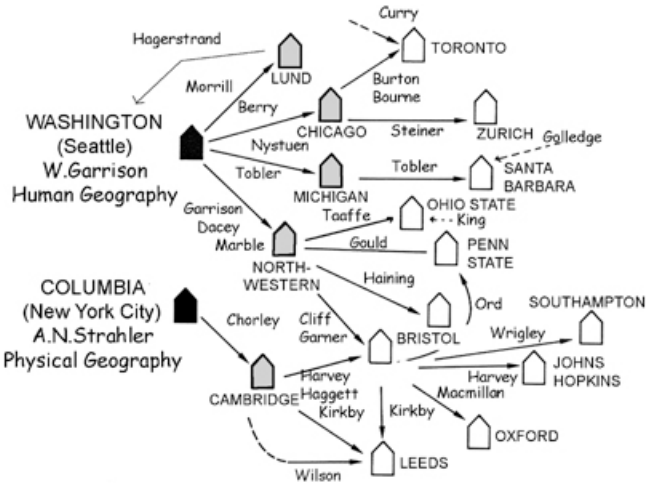

Figure 1: *Diffusion of Quantitative Geographical Revolution across Institutions*

The term *Geographical Information System* (GIS) was first coined by Tomlinson (1969). The meaning of the 'S' later evolved from *Systems* to *Science*. The latter term (*Geographic Information Science*) was first defined by Goodchild (1992), who argued that the focus of the field should be "thinking about science rather than systems" (p.44). This shift was reflected in the renaming in 1995 of the *International Journal of Geographical Information Systems* (Goodchild, 1995). (One consequence of this shift means that it is often unclear in subsequent writing whether an author's reference to 'GIS' is intended to mean G.I. Systems or G. I. Science – I will therefore try explicitly to refer to GISys. or GISci.)

In 1996, Openshaw & Abrahart (1996) opened the first conference on the freshly defined area of 'GeoComputation,' with a manifesto suggesting eight different directions for potential research. GeoComputation was defined as including "the constituent technologies of high performance computing, artificial intelligence, and GIS" – on the one hand, one might say it was therefore intended to be broader than GIS; on the other hand, it is more narrowly concerned with 'technologies' than Goodchild's call to prioritize more general questions of underlying 'science.' The conference led to a book by the same name edited by Longley et al. (1998); a substantially new but functionally equivalent second edition has been produced by Brunsdon & Singleton (2015). Openshaw & Openshaw (1977) advocated the utility and use of 'Artificial Intelligence in Geography,' explaining the principles and relevance of neural networks, genetic algorithms, agent-based modelling, and fuzzy logic. Gahegan remarks that machine learning and AI have seen better progress in the last two decades than the other directions suggested for GeoComputation (Harris et al., 2017).

Most recently, the term 'Data Science' has emerged (Donoho, 2017), with some suggesting that it represents a whole new paradigm for scientific research (Hey et al., 2009). More than fifty years ago, Tukey (1962) suggested that his work in data analysis be recognized as a science in its own right, rather than just an aspect of statistics, since statistics had become established as a branch of mathematics rather than a science as such. But it is only since the 'data deluge' (Hey & Trefethen, 2003) of the twenty-first century that such an idea has become mainstream. While every social science is unavoidably affected by the shift, responses vary in whether 'data science' is seen as a genuine paradigm-shift to be positively embraced, or more of a trend and trope whose challenge requires critique from a grounded theoretical disciplinary perspective. Singleton & Arribas-Bel (2019), for example, advocate 'Geographic Data Science' as a necessary development for "sustaining the relevance of Geography and subdisciplinary approaches within a rapidly changing

socio-technological landscape." Kang et al. (2019) on the other hand argue that the need is the opposite way around: "a successful UDS ['Urban Data Science'] is impossible without a broader and more successful urban studies."

II. Precursors to the Computer Age

The history of computing should begin with the invention of the Analytical Engine developed by Charles Babbage (Bromley, 1982) and explained and promoted by his assistant, the mathematician Ada Lovelace (Menabrea & Lovelace, 1843). Physical devices to aid arithmetic calculation had existed since (at least) the creation of the abacus several centuries before the birth of Christ (Sugden, 1981). Shickard's 'calculating clock' of 1623 was probably the first true mechanical calculator; Pascal, Morland, Liebnitz, and Hahn all improved on this in various ways, before De Colmar's *arithmometer* became the first mechanical calculator put into general production (Solla Price, 1984).

Babbage's first machine, the *Difference Engine*, while considerably more sophisticated in its conception, was still essentially also an arithmetical calculator. However, his *Analytical Engine* (Fig. 2) was a true programmable Turing-complete computer, and as such deserves to be recognized as the beginning of a new era in technology. Unfortunately, although Lovelace explicitly recognized that the machine could be adapted to work with "letters or any other general symboly; and in fact it might bring out its results in algebraical notation" (Menabrea & Lovelace, 1843), Babbage himself does not seem to have recognized the ground-breaking implications of automated symbolic manipulation as opposed to mere numerical calculation (Swade, 2012). At any rate, neither of Babbage's machines were ever completed – at least not until almost two hundred years later (Swade & Babbage, 2001) – and although the Analytical Engine preempted many of the features of twentieth-century digital computers, it does not seem to have actually been a direct influence.

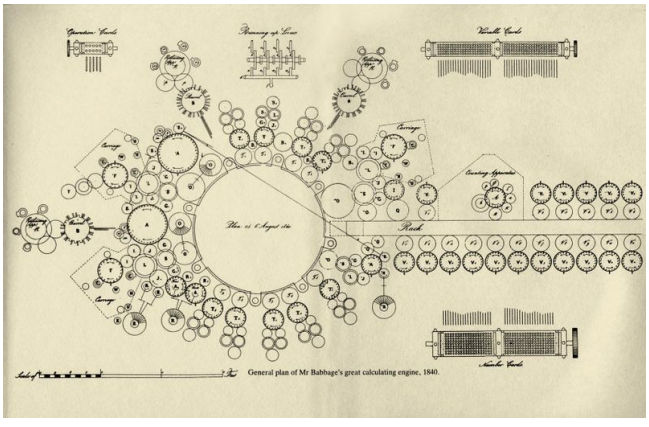

Figure 2: *Diagram of Babbage's Analytic Engine*

A more successful technological breakthrough can be seen with Hermann Hollerith's punch-card tabulating machine that was designed and used to process the results of the American census of 1890. Babbage's machines had also used punch-cards; both he and Hollerith seem to have been inspired by their use in the weavers' hand-looms created by the Frenchman Joseph-Marie Jacquard (Essinger, 2004). Hollerith's Tabulating Machine was nothing like as ambitious in its functionality as Babbage's Analytical Engine, but unlike Babbage's, Hollerith's Machine swiftly achieved its intended purpose. The 1880 American census had taken seven years to process; Hollerith's solution permitted the data for sixty-two million citizens to be processed in just two and a half years (Biles et al., 1989). In 1896 Hollerith incorporated the 'Tabulating Machine Company'; in 1924 its name was changed to the 'International Business Machines Corporation,' now better known by its acronymn IBM (O'Regan, 2018).

III. Foundational Theoretical Breakthroughs

Several theoretical breakthroughs were necessary before the true power of an automatic computing machine could become apparent: in particular, the theory of computability developed by Turing (1936); the theory of functional logic developed by Church (1936); and the mathematical theory of communication developed by Shannon (1948).

Both Turing and Church developed their theories in attempting to solve the famous *Entscheidungsproblem* ('decision problem') posed by Hilbert in 1928 (Hilbert & Ackermann, 1928), which asked for an algorithm able to decide whether a given statement is provable from a set of axioms – they showed that such an algorithm is impossible, and therefore some decision problems are undecidable.

This is related to the problem of proving the consistency of the axioms of arithmetic, which had been included in Hilbert's original 1900 list of significant unsolved mathematical problems (Thiele, 2005), and which was solved by Gödel (1931), with his now famous Incompleteness Theorems. These effectively dashed the hopes of the logical positivists that all problems could be solved if only they could be framed in formal mathematics (Mancosu et al., 2009); this is ironic, considering how frequently accusations of 'positivism' have been directed at quantitative geographers (Sheppard, 2014).

Church (1936), who was the first to publish his solution, framed his approach by proposing "a definition of effective calculability [...corresponding] to the somewhat vague intuitive notion in terms of which problems of this class are often stated" (p.346). He first explained the operation of the *lambda calculus*, a language for manipulating logical formulae, consisting of the symbols *{*, *}*, *(*, *)*, λ, *[*, *]*, and other characters representing variables. This enabled him to identify the notion of an *effectively calculable* function with a *lambda-definable* function, and drawing on the work of Godel and others he was then able to show that there exist functions that are not lambda-definable.

The approach of Turing (1936) was quite different.

While Church's approach was language-oriented, Turing's was machine-oriented: he demonstrated that "it is possible to invent a single machine which can be used to compute any computable sequence" (p.241). He did this by a detailed description of the operation of such a 'universal computing machine,' requiring only that it have a 'tape' which it could write to, read from, and move along, and a small set of inbuilt default instructions (which Turing called '*m*-configurations'). The solution to Hilbert's *Entscheidungsproblem* then becomes a question of showing that the machine is unable to ever compute the solution to the problem of whether an uncomputable number is computable; or equivalently, to compute whether an algorithm will eventually halt or not.

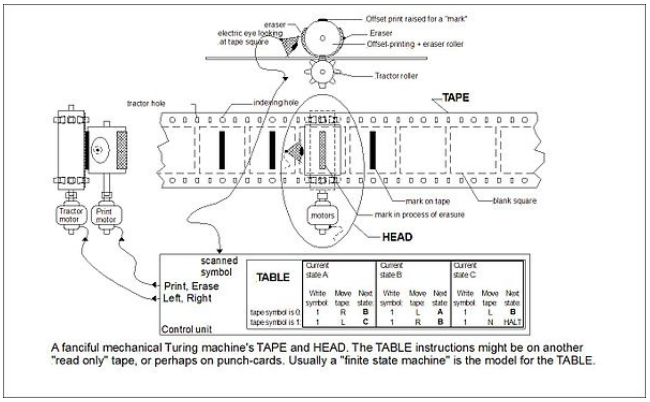

Figure 3: *Diagram of Turing Machine*

The work of Turing and Church showed that if a language or a machine was capable of simple manipulations of data, it would also be capable of much more complex manipulation of data. Negatively they showed that some functions/algorithms cannot be computed, but this was done by showing positively that any that can be computed can be composed of just a few simple constituent parts. But it was Shannon (1948) who developed the *Mathematical Theory of Communication* which would allow for similar sophistication in considering what was required to conserve and communicate the information contained in any particular set of data.

Shannon approaches the subject of communication as essentially *an engineering problem*, and shows how by considering the flow of information source as a stochastic process, it is possible to quantify how certain we are of the way in which a sequence of information will continue, given what we already know. By thus considering information's *entropy* (the concept is borrowed from statistical themodynamics), it becomes possible to quantify the channel capacity necessary to transmit a given message. Further, Shannon shows that it is even possible to reduce information loss across a 'noisy' loss-inducing channel to as small a level as desired, by deliberately encoding the necessary level of redundancy into the message before transmission.

Shannon's paper is also notable for being the first published instance of the concept of 'binary digits,' or *bits*, though he credits Tukey with the invention of the term. He notes that any "device with two stable positions, such as a relay or a flip-flop circuit, can store one bit of information" (p.380).

IV. Key Developments in Computer Hardware Engineering

By the time Shannon's paper was published, computers had already been created that used electromechanical relays and electronic circuits. Aiken's Harvard Mark I (Cohen, 1990), put into operation in 1944, used the first; the ENIAC (Burks, 1947), completed in 1945, used the latter, in the form of vacuum tubes (also known as thermionic valves). While the latter was an improvement on the former, both were large, cumbersome and fragile.

However, in 1947 a Bell Labs team researching the nature of semiconductors made a discovery that led to the invention of transistors (Nelson, 1962). The key theoretical breakthrough was made by Shockley (1949), who left Bell Labs in 1955 to start his own semiconductor company in California – thus marking the beginning of 'Silicon Valley' (Brock, 2012). But while Shockley was a brilliant scientist he was a poor business manager, and in 1957 an alienated group of his scientists – who came to be known as 'The Traitorous Eight' (Kassing, 2011) – left and founded their own company, Fairchild Semiconductor, which produced the first commercially viable integrated circuit (Moore, 1998).

Among those eight scientists was Gordon Moore (Thackray et al., 2015), who in 1965 observed that the number of transistor components fit onto an integrated circuit was roughly doubling each year, and predicted that "over the short term this rate can be expected to continue, if not to increase" (Moore, 1965). In fact, the trend – now known as *Moore's Law*, though it is an observed correspondence and not a physical law – has held remarkably well for fifty years (Fig. 4), although we are now perhaps approaching its necessary physical limits (Mack, 2011).

This exponential improvement in transistor density is significant because of its correlation with computers' processing power and economic affordability. *Bell's Law* (Bell, n.d.) explains how this progress creates new classes of computers, as each development offers four options: (1) maintaining constant price and increasing performance; (2) lowering price and performance to extend range; (3) extending performance with new supercomputers; (4) minimizing price with newly viable classes of microcomputer. With each new class of computer there comes a corresponding increase in their ubiquity (Arribas-Bel & Reades, 2018).

V. *Software* and Programming Languages

To take advantage of the increasing processing power offered by computer hardware required corresponding developments in *software* – another word apparently coined by Tukey (Anscombe, 2003) in an article for the American Mathematical Monthly (Tukey, 1958). In par-

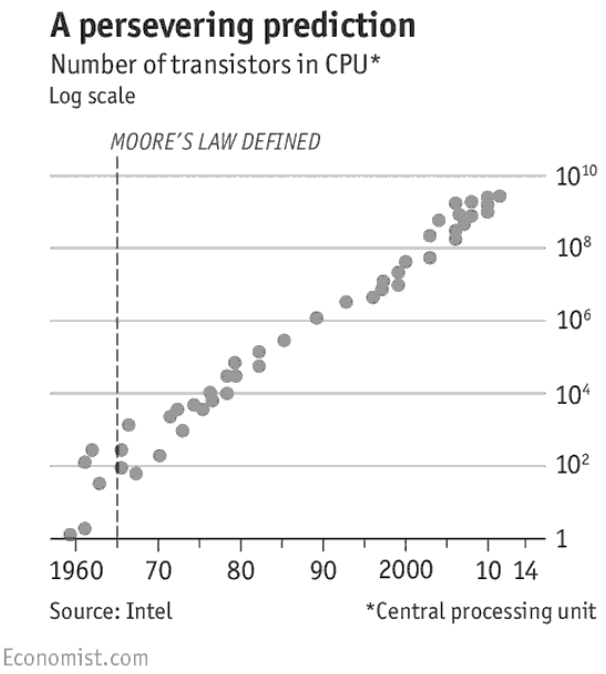

Figure 4: *Moore's Law sustained over Fifty Years*

ticular, what is needed is a way of clearly and concisely describing the *algorithms* to be performed by the computer.

Mathematical algorithms have existed since ancient times, with Euclid's algorithm for finding a greatest common divisor being the most famous (Knuth, 2014). But such processes were described informally, with imprecise natural language. The rise of modern mathematics led to highly developed notation for describing static functional relations, but not for recursive dynamic processes (Knuth & Pardo, 1980) – perhaps because before the invention of the computer there was no need. Perhaps the most accessible way of describing an algorithmic process, is with a *flow diagram*, using boxes and arrows to illustrate conditional instructions. This technique was used by Goldstine & von Neumann (1948), but its visual nature means that while it is suitable for communicating to humans how a machine should be made to function, it is of no use for directly programming the machine.

The first high-level programming language (Wegner, 1976) was FORTRAN (FORmula TRANslating system), the first version of which was released in 1954 (Backus, 1978), and which is still used for scientific computing today (Kermode, 2020). This built on previous work on assemblers and compilers which were able to convert symbolic programming instructions into machine code (Allen, 1981). Wegner suggests that the '50s was a decade of discovering and describing the basic concepts of programming languages, and that the '60s was a decade of elaborating and analyzing them. Much could be said about the many programming languages that have emerged and evolved since then, and about their underlying principles (Reynolds, 1998), but here I will focus on three issues that I see as key to the scientific utility of computation: platform independence, open source code, and readability.

Fortran was first implemented for a specific machine – the IBM 704 (Backus et al., 1957). For Fortran code to be written on a new type of machine, a completely new compiler had to be written specific to that machine. Fortunately, this is no longer necessary due to the concept of a machine-independent *operating system* (OS), which are now considered "an essential part of any computer system" (Silberschatz et al., 2011). The first successful portable operating system was Unix, another product of Bell Labs (Ritchie & Thompson, 1978). Unix was developed together with the C programming language in which it was written. C is notable in that, for a human-readable computer language, it is 'rather low-level,' dealing with 'the same sorts of objects that most computers do' (Ritchie et al., 1978). Portability was not initially the primary goal, but as the success of the system became apparent, efforts were made to refine and extend the C language so as to remove difficulties in functioning on a wide variety of machines (Johnson & Ritchie, 1978).

But just because it is technically possible to run code on a machine, or to adapt it so that it will run, does not mean it is legally permissible. Using and adapting Unix required a license from AT&T, the proprietary owner (Raymond, 2000). Refusing to submit to such a situation, Richard Stallman, an artificial intelligence researcher at MIT, announced his intention to "Free Unix!" by creating a compatible software system that would be free to adapt and copy, recursively named GNU (GNU's not Unix!) (Stallman, 1983). The achievement of creating a functional free OS kernel actually belongs though to Linus Torvalds (1991)'s *Linux*. Raymond (2001) attributes Torvalds' success to the way that he was able to mobilize a large number of beta-testers and co-developers to rapidly eliminate bugs and create a functioning product. He calls this *Linus' Law*: "Given enough eyeballs, all bugs are shallow."

If co-developers are to be able to easily spot bugs in others' code, it helps if a programming language is easily readable. Knuth (1984) suggested that programmers shift their focus in writing programs from instructing the computer what to do, to explaining to a human what the computer is meant to do – he called this *literate programming*. A prime example of this is the Python programming language (Van Rossum & Drake Jr, 1995), which has even been described as 'executable pseudocode' (Dierbach, 2014).

VI. Artificial Intelligence and Machine Learning

While programming allows a human to specifically instruct a machine how to process given input, a different paradigm asks whether and how it is possible to enable a machine to learn what to do by giving it examples rather than specific instructions. Turing (1950) considered the grand prospect of whether a computer might be able to convincingly imitate a human being (the 'Turing Test'). *Artificial Intelligence* as a field of research is generally

considered to have begun with the Dartmouth Summer Research Project proposal put forward by McCarthy et al. (1955). The practical question concerns the possibility of *machine learning*, a term first popularized by Samuel (1959), who created a program that was able to learn "to play a better game of checkers than can be played by the person who wrote the program."

Regardless of how far off successful *imitation* of general human intelligence might be (Müller & Bostrom, 2016), substantial breakthroughs have been made in drawing *inspiration* from human cognitive architecture. McCulloch & Pitts (1943) showed that there exists a straightforward correspondence between the binary logic of propositional calculus, and the physiological model of of the nervous system as a network of neurons which each have some threshold over which excitation triggers a synaptic impulse. Hebb (1949) attempted to account for the synaptic plasticity necessary to explain neural learning in such terms. The *Perceptron Algorithm* of Rosenblatt (1958) was the first demonstration of how an *Artificial Neural Network* could be trained to learn. However, a single-layered perceptron network is only able to distinguish linearly-separable data, and theoretical doubts about the value of multi-layer neural networks (Minsky & Papert, 1969), as well as the technical issue of how much computational power is required, meant that it is only really in the last decade that the power of deep learning has become apparent (Sejnowski, 2018).

Rosenblatt's Perceptron is an example of *supervised* learning, in that the algorithm requires that each point in its training dataset be labelled; in contrast, *unsupervised* machine learning requires no labels, and is useful for *pattern recognition* (Bishop, 2006) in exploratory data analysis. The clearest examples of unsupervised machine learning are clustering algorithms. The earliest research into cluster analysis was done by R.C. Tryon (Blashfield, 1980), who sought to improve on the deficiencies of factor analysis (Tryon, 1958). While his primary field of study was psychology, he applied his pioneering clustering methodology to a social area analysis of San Francisco (Tryon, 1955). Thus we can claim that geodemographic classification was one of the earliest applications of artificial intelligence, and certainly of A.I. in geography.

References

Allen, F. E. (1981). The History of Language Processor Technology in IBM. *IBM Journal of Research and Development*, *25*(5), 535–548. https://doi.org/10.1147/rd.255.0535

Anscombe, F. R. (2003). Quiet Contributor: The Civic Career and Times of John W. Tukey. *Statistical Science*, *18*(3), 287–310.

Arribas-Bel, D., & Reades, J. (2018). Geography and computers: Past, present, and future. *Geography Compass*, *12*(10), e12403. https://doi.org/10.1111/gec3.12403

Backus, J. (1978). THE HISTORY OF FORTRAN I, II, AND III. *Association for Computing Machinery*, *13*(8).

Backus, J. W., Beeber, R. J., Best, S., Goldberg, R., Haibt, L. M., Herrick, H. L., … Nutt, R. (1957). The FORTRAN automatic coding system. In *Papers presented at the February 26-28, 1957, western joint computer conference: Techniques for reliability* (pp. 188–198). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1455567.1455599

Bell, G. (n.d.). Bell's Law for the Birth and Death of Computer Classes: A theory of the Computer's Evolution. *IEEE Solid-State Circuits Society Newsletter*, *13*(4), 8–19. https://doi.org/10.1109/N-SSC.2008.4785818

Biles, G. E., Bolton, A. A., & DiRe, B. M. (1989). Herman Hollerith: Inventor, Manager, EntrepreneurA Centennial Remembrance. *Journal of Management*, *15*(4), 603–615. https://doi.org/10.1177/014920638901500409

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Blashfield, R. K. (1980). The Growth Of Cluster Analysis: Tryon, Ward, And Johnson. *Multivariate Behavioral Research*, *15*(4), 439–458. https://doi.org/10.1207/s15327906mbr1504_4

Brock, D. C. (2012). From automation to Silicon Valley: The automation movement of the 1950s, Arnold Beckman, and William Shockley. *History and Technology*, *28*(4), 375–401. https://doi.org/10.1080/07341512.2012.756236

Bromley, A. G. (1982). Charles Babbage's Analytical Engine, 1838. *Annals of the History of Computing*, *4*(3), 196–217.

Brunsdon, C., & Singleton, A. (2015). *Geocomputation: A Practical Primer*. 1 Oliver's Yard, 55 City Road London EC1Y 1SP: SAGE Publications, Inc. https://doi.org/10.4135/9781473916432

Burks, A. W. (1947). Electronic Computing Circuits of the ENIAC. *Proceedings of the IRE*, *35*(8), 756–767. https://doi.org/10.1109/JRPROC.1947.234265

Burton, I. (1963). THE QUANTITATIVE REVOLUTION AND THEORETICAL GEOGRAPHY. *The Canadian Geographer/Le Géographe Canadien*, *7*(4), 151–162. https://doi.org/10.1111/j.1541-0064.1963.tb00796.x

Church, A. (1936). An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, *58*(2), 345–363. https://doi.org/10.2307/2371045

Cohen, I. B. (1990). Howard H. Aiken and the computer. In S. G. Nash (Ed.), *A history of scientific computing* (pp. 41–52). New York, NY, USA: ACM. https://doi.org/10.1145/87252.88066

Dierbach, C. (2014). Python as a first programming language. *Journal of Computing Sciences in Colleges*, *29*(3), 73.

Donoho, D. (2017). 50 Years of Data Science. *Journal of Computational and Graphical Statistics*, *26*(4), 745–766. https://doi.org/10.1080/10618600.2017.1384734

Essinger, J. (2004). *Jacquard's Web: How a hand-loom led to the birth of the information age*. OUP Oxford.

Goldstine, H. H., & von Neumann, J. (1948). Planning and Coding of Problems for an Electronic Computing Instrument.

Goodchild, M. F. (1992). Geographical information science. *International Journal of Geographical Information Systems*, *6*(1), 31–45. https://doi.org/10.1080/02693799208901893

Goodchild, M. F. (1995). Future Directions for Geographic Information Science. *International Journal of Geographical Information Science*, *1*(1), 1–7.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für mathematik und physik.

Haggett, P. (2008). The Local Shape of Revolution: Reflections on Quantitative Geography at Cambridge in the 1950s and 1960s. *Geographical Analysis*, *40*(3), 336–352. https://doi.org/10.1111/j.1538-4632.2008.00731.x

Harris, R., O'Sullivan, D., Gahegan, M., Charlton, M., Comber, L., Longley, P., … Evans, A. (2017). More bark than bytes? Reflections on 21+ years of geocomputation. *Environment and Planning B: Urban Analytics and City Science*, *44*(4), 598–617. https://doi.org/10.1177/2399808317710132

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. New York, NY, USA: Wiley.

Hey, T., Tansley, S., & Tolle, K. (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*.

Hey, T., & Trefethen, A. (2003). The Data Deluge: An e-Science Perspective. In F. Berman, G. Fox, & T. Hey (Eds.), *Wiley Series in Communications Networking & Distributed Systems* (pp. 809–824). Chichester, UK: John Wiley & Sons, Ltd. https://doi.org/10.1002/0470867167.ch36

Hilbert, D., & Ackermann, W. (1928). *Grundzüge der theoretischen Logik*. J. Springer.

Johnson, S. C., & Ritchie, D. M. (1978). UNIX time-sharing system: Portability of C Programs and the UNIX system. *The Bell System Technical Journal*, *57*(6), 2021–2048. https://doi.org/10.1002/j.1538-7305.1978.tb02141.x

Kang, W., Oshan, T., Wolf, L. J., Boeing, G., Frias-Martinez, V., Gao, S., … Xu, W. (2019). A roundtable discussion: Defining urban data science. *Environment and Planning B: Urban Analytics and City Science*, *46*(9), 1756–1768. https://doi.org/10.1177/2399808319882826

Kassing, J. (2011). *Dissent in Organizations*. Polity.

Kermode, J. R. (2020). F90wrap: An automated tool for constructing deep Python interfaces to modern Fortran codes. *Journal of Physics: Condensed Matter*, *32*(30), 305901. https://doi.org/10.1088/1361-648X/ab82d2

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, *27*(2), 97—111.

Knuth, D. E. (2014). *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Professional.

Knuth, D. E., & Pardo, L. T. (1980). The Early Development of Programming Languages. In *A History of Computing in the Twentieth Century* (pp. 197–273). Elsevier. https://doi.org/10.1016/B978-0-12-491650-0.50019-8

Longley, P. A., Brooks, S., Macmillan, W., & McDonnell, R. A. (1998). *Geocomputation: A primer*. (P. A. Longley, S. Brooks, W. Macmillan, & R. A. McDonnell, Eds.). Wiley.

Mack, C. A. (2011). Fifty Years of Moore's Law. *IEEE Transactions on Semiconductor Manufacturing*, *24*(2), 202–207. https://doi.org/10.1109/TSM.2010.2096437

Mancosu, P., Zach, R., & Badesa, C. (2009). The Development of Mathematical Logic from Russell to Tarski, 19001935. In L. Haaparanta (Ed.), *The Development of Modern Logic* (pp. 318–470). Oxford University Press. https://doi.org/10.1093/acprof:oso/9780195137316.003.0029

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (1955). A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE.

McCulloch, W. S., & Pitts, W. (1943). A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. *Bulletin of Mathematical Biophysics*, 17.

Menabrea, L., & Lovelace, A. (1843). Sketch of the Analytical Engine Invented by Charles Babbage (English Translation). *Scientific Memoirs*, (3).

Miller, H. J. (2004). Tobler's First Law and Spatial Analysis. *Annals of the Association of American Geographers*, *94*(2), 284–289. https://doi.org/10.1111/j.1467-8306.2004.09402005.x

Minsky, M. L., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.

Moore, G. E. (1965). Cramming More Components Onto Integrated Circuits. *Proceedings of the IEEE*, *86*(1), 82–85. https://doi.org/10.1109/JPROC.1998.658762

Moore, G. E. (1998). The role of Fairchild in silicon technology in the early days of "Silicon Valley". *Proceedings of the IEEE*, *86*(1), 53–62. https://doi.org/10.1109/5.658759

Müller, V. C., & Bostrom, N. (2016). Future Progress in Artificial Intelligence: A Survey of Expert Opinion. In V. C. Müller (Ed.), *Fundamental Issues of Artificial Intelligence* (pp. 555–572). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-26485-1_33

Nelson, R. (1962). The Link Between Science and Invention: The Case of the Transistor. In *The Rate and Direction of Inventive Activity: Economic and Social Factors*. Princeton: Princeton University Press. https://doi.org/10.1515/9781400879762

O'Regan, G. (2018). Hollerith's Tabulating Machines and the Birth of IBM. In G. O'Regan (Ed.), *The Innovation in Computing Companion: A Compendium of Select, Pivotal Inventions* (pp. 151–153). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-02619-6_31

Openshaw, S., & Abrahart, R. (1996). Geocomputation.

Openshaw, S., & Openshaw, C. (1977). *Artificial Intelligence in Geography*. Wiley.

Raymond, E. S. (2000). A Brief History of Hackerdom. Thyrsus Enterprises.

Raymond, E. S. (2001). *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary* (1 edition). O'Reilly Media.

Reynolds, J. C. (1998). *Theories of Programming Languages*. Cambridge University Press.

Ritchie, D. M., Johnson, S. C., Lesk, M. E., & Kernighan, B. W. (1978). UNIX time-sharing system: The C programming language. *The Bell System Technical Journal*, *57*(6), 1991–2019. https://doi.org/10.1002/j.1538-7305.1978.tb02140.x

Ritchie, D. M., & Thompson, K. (1978). The UNIX Time-Sharing System. *Bell System Technical Journal*, *57*(6), 1905–1929. https://doi.org/10.1002/j.1538-7305.1978.tb02136.x

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, *3*(3), 210–229. https://doi.org/10.1147/rd.33.0210

Sejnowski, T. J. (2018). *The Deep Learning Revolution*. MIT Press.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Sheppard, E. (2014). We have never been positivist. *Urban Geography*, *35*(5), 636–644. https://doi.org/10.1080/02723638.2014.916907

Shockley, W. (1949). The Theory of p-n Junctions in Semiconductors and p-n Junction Transistors. *Bell System Technical Journal*, *28*(3), 435–489. https://doi.org/10.1002/j.1538-7305.1949.tb03645.x

Silberschatz, A., Galvin, P. B., & Galvin, G. (2011). *Operating System Concepts Essentials*. Wiley.

Singleton, A., & Arribas-Bel, D. (2019). Geographic Data Science. *Geographical Analysis*, *2019*(0), 1–15. https://doi.org/10.1111/gean.12194

Solla Price, D. (1984). A History of Calculating Machines. *IEEE Micro*, *4*(1), 22–52. https://doi.org/10.1109/MM.1984.291305

Stallman, R. (1983). New UNIX implementation. *net.unix-wizards*.

Sugden, K. F. (1981). A HISTORY OF THE ABACUS. *Accounting Historians Journal*, *8*(2), 1–22. https://doi.org/10.2308/0148-4184.8.2.1

Swade, D. (2012). Origins of Digital Computing: Alan Turing, Charles Babbage, and Ada Lovelace. In *A Computable Universe* (pp. 23–43). WORLD SCIENTIFIC. https://doi.org/10.1142/9789814374309_0002

Swade, D., & Babbage, C. (2001). *Difference Engine: Charles Babbage and the Quest to Build the First Computer*. Viking Penguin.

Thackray, A., Brock, D. C., & Jones, R. (2015). *Moore's Law: The Life of Gordon Moore, Silicon Valley's Quiet Revolutionary*. Hachette UK.

Thiele, R. (2005). Hilbert and his Twenty-Four Problems. In J. Borwein, K. Dilcher, G. Van Brummelen, & M. Kinyon (Eds.), *Mathematics and the Historian's Craft* (pp. 243–295). New York, NY: Springer New York. https://doi.org/10.1007/0-387-28272-6_11

Tobler, W. R. (1970). A Computer Movie Simulating Urban Growth in the Detroit Region. *Economic Geography*, *46*(sup1), 234–240. https://doi.org/10.2307/143141

Tomlinson, R. (1969). A Geographic Information System for Regional Planning. *Journal of Geography*, *78*(1), 45–48. https://doi.org/10.5026/jgeography.78.45

Torvalds, L. (1991). Linuxa free unix-386 kernel.

Tryon, R. C. (1955). *Identification of Social Areas by Cluster Analysis: A general method with an application to the San Francisco Bay Area*. University of California Press.

Tryon, R. C. (1958). General Dimensions of Individual Differences: Cluster Analysis Vs. Multiple Factor Analysis. *Educational and Psychological Measurement*, *18*(3), 477–495. https://doi.org/10.1177/001316445801800304

Tukey, J. W. (1958). The Teaching of Concrete Mathematics. *The American Mathematical Monthly*, *65*(1), 1–9. https://doi.org/10.1080/00029890.1958.11989128

Tukey, J. W. (1962). The Future of Data Analysis. *The Annals of Mathematical Statistics*, *33*(1), 1–67. https://doi.org/10.1214/aoms/1177704711

Turing, A. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *2*(42).

Turing, A. (1950). Computing Machinery and Intelligence. *Mind*.

Van Rossum, G., & Drake Jr, F. L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Wegner, P. (1976). Programming LanguagesThe First 25 Years. *IEEE Transactions on Computers*, *C-25*(12), 1207–1225. https://doi.org/10.1109/TC.1976.1674589