Original Research Article

BIG DATA & SOCIETY

Big Data & Society
July–December 2016: 1–11

DOI: 10.1177/2053951716665128
bds.sagepub.com

SAGE

Algorithms and their others: Algorithmic culture in context

Paul Dourish

Abstract

Algorithms, once obscure objects of technical art, have lately been subject to considerable popular and scholarly scrutiny. What does it mean to adopt the algorithm as an object of analytic attention? What is in view, and out of view, when we focus on the algorithm? Using Niklaus Wirth's 1975 formulation that "algorithms + data structures = programs" as a launching-off point, this paper examines how an algorithmic lens shapes the way in which we might inquire into contemporary digital culture.

Keywords

Algorithms, practice, materiality, configurations, visibility, code

Introduction

During my time as an undergraduate student in computer science, algorithms were objects of concern in a variety of ways – as practical rubrics for the design of effective and efficient computer programs, as catalogs of ways of working, as abstract formulations in textbooks and research papers, or as mathematical conundrums that might appear in exam papers. Alongside compilers, libraries, specifications, languages, and state machines, they formed part of the intellectual furniture of that world.

From that perspective, it's rather odd to find that algorithms are now objects of public attention, arising as topics of newspaper articles and coffee shop conversations. When digital processes become more visible as elements that shape our experience, then algorithms in particular become part of the conversation about how our lives are organized. From discussions over the role that algorithms might play in hiring (Hansel, 2007) or credit scoring (Singer, 2014) to inquiries into the assumptions behind the algorithms that set the ambient temperature in office buildings (Belluck, 2015), an awareness has developed that algorithms, somehow mysterious and inevitable, are contributing to the shape of our lives in ways both big and small.

The public discussion of algorithms emerges out of (and arises at the intersection of) a series of other conversations. Some, for example, are entwined with discussions of "Big Data," with a focus on the ways that online activities create data streams from which algorithms extract patterns that guide the action of institutions, corporations and states. Others frame discussions of algorithms in terms of automation and in particular the kinds of high-speed action associated with, say, programmed trading in stock markets, high-frequency automated trades carried out by computer systems without human intervention (Buenza and Millo, 2013). Still others are concerned with the ways that algorithmic developments are transforming aspects of the labor relation, positioning human being as resources to be deployed according to programmed responses to demand, for instance in ride-sharing services like Uber (e.g. Rosenblat and Stark, 2016). Each of these is a broader or ongoing conversation into which the algorithmic has become incorporated.

Relatedly, algorithms have also become objects of academic attention in social and cultural studies, often in the context of similar concerns. Working

University of California, Irvine, CA, USA

Corresponding author:
Paul Dourish, University of California, 5086 Donald Bren Hall, Irvine, CA 92697-3440, USA.
Email: jpd@ics.uci.edu

across a number of areas, including finance, labor politics, governance, public policy, and organizational strategy, scholars such as Barocas (2014), Gillespie (2012), Glaser (2014), Manovich (2013), Pasquale (2015), Seaver (2015), and Ziewitz (2015) have turned attention to the way that algorithms are embedded within topics of academic investigation or indeed may constitute a significant new topic of attention in themselves. As with the public discussion, this academic interest in algorithms is sometimes driven by the way that algorithms are beginning to arise as objects of significance within existing academic domains; in other cases, it arises as part and parcel of a broader interest in harnessing the tools of cultural analysis to understand contemporary digital culture and its platforms (e.g. Cox, 2012; Fuller, 2008; Mackenzie, 2006; Manovich, 2001; Montford and Bogost, 2009).

However, the complex embedding of the topic of algorithms into these related concerns raises some difficulties. In particular, it requires us to be careful about the bounds and limits of algorithms and their functioning. Just what is it that we have in view when we focus on ''algorithms'' as the central object of analytic attention?

In 1975, the pioneering computer scientist Niklaus Wirth published a book entitled ''Algorithms + Data Structures = Programs'' (Wirth, 1975). Wirth was one of a group of researchers and academics who developed and advocated for the idea of ''structured programming,'' an approach to the design and engineering of software systems that emphasized the stepwise, modular decomposition of problems and a similarly structured approach to software design and construction. This approach made computer programs easier to develop (especially by teams of programmers) and easier to analyze, as well as more naturally aligning computer programs, as engineering artifacts, with the sorts of mathematical mechanisms by which they could be analyzed and assessed. Wirth did not simply cheer for this position from the sidelines; his own work in programming language design and development provided software engineers with the tools they needed to adopt the model, which soon became (and indeed, in variant forms, remains) standard industrial practice. ''Algorithms + Data Structures = Programs'' focused on the practice of software design in the structured programming tradition, setting out the case for the mutual design of algorithmic processes and the regularized data representations or ''data structures'' over which they would operate. At a time when the development and analysis of algorithms was the dominant and most prestigious area of computer science, Wirth wanted to emphasize the concomitant importance of data structures for those building effective software systems.

Wirth's formulation – algorithms + data structures = programs – highlights important concerns too for those concerned with algorithms and digital culture.

The first is that algorithms and programs are different entities, both conceptually and technically. Programs may embody or implement algorithms (correctly or incorrectly), but, as I will elaborate, programs are both more than algorithms (in the sense that programs include non-algorithmic material) and less than algorithms (in the sense that algorithms are free of the material constraints implied by reduction to particular implementations).

The second, related, observation is that since algorithms arise in practice in relation to other computational forms, such as data structures, they need to be analyzed and understood within those systems of relation that give them meaning and animate them. There is, in other words, within Wirth's formula, an analytic warrant for a relational and differential analysis of algorithm alongside data, data structure, program, process, and other analytic entities. This is not to dissolve the algorithm in a sea of relations, but rather to understand how algorithm – as a technical object, as a form of discourse, as an object of professional practice, and as a topic of public or academic concern – comes to play the particular role that it does.

The goal of this paper is to sketch just this sort of relational analysis and to place algorithm in juxtaposition with other relevant terms both in order to identify aspects of the scope and limits of ''algorithm'' as a conceptual tool, and to understand how algorithms come to act within broader digital assemblages. As Neyland (2016) notes, the danger to be guarded against here is taking an essentializing view of algorithms. Similarly, my argument here should not be read as an essentialist argument, seeking a foundational truth of the nature of algorithms as natural occurrences. No such naturalism can be sustained. Instead, the argument here is one of ethnographic responsibility and practical politics. With respect to ethnographic responsibility, I note that ''algorithm'' is a term of art within a particular professional culture – that of computer scientist, software designers, and machine learning practitioners – and I seek to understand the limits and particularlities of that term's use as a members' term, its emic character, in much the same way as we might similarly explore, respect, and analyze the consequences of members' terms within other cultural milieux. Secondly, as a matter of practical politics, I take it that the domain of Big Data is one into which social science seeks to make an intervention, and suggest that critiques of algorithmic reasoning that set their own terms of reference for key terminology are unlikely to hit home. Again, this is not to grant primacy or authority to a technical interpretation; the goal rather is to

understand what that technical interpretation is, and what consequences it might hold for social and cultural analysis. The paper takes up the question of what algorithms do within the domain of Big Data's professional practices, as "convening" objects (Ananny, 2016), and as objects that live in dynamic relations to the other material and discursive elements of software systems and the setting that produce them. In doing so, I hope to be able to identify fruitful directions for taking up the algorithm as an object of attention within software studies and allied domains.

Algorithms and their others

In computer science terms, an algorithm is an abstract, formalized description of a computational procedure. Algorithms fall into different types according to their properties or domains – combinatorial algorithms deal with counting and enumeration, numerical algorithms produce numerical (rather than symbolic) answers to equational problem, while probabilistic algorithms produce results within particular bounds of certainty. Algorithms may also vary in terms of their analytic characteristics, such as generalized performance characteristics (e.g. how their mean-time or best-time performance varies with the size of the data sets over which they operate). As part of the stock-in-trade of computer scientists and software engineers, some algorithms are known by the names of their inventors (Dijkstra's algorithm, the Viterbi algorithm, Gouraud shading, or Rivest-Shamir-Adelman) while others are known by conventional names (e.g. QuickSort, Fast Fourier Transform, Soundex, or sort-merge join).

The significance of some of these properties – formalization, abstraction, identity, and so on – becomes clearer when we look at algorithms in the context of their "others" – related but distinct phenomena that emphasize different aspects of the sociotechnical assembly. In speaking of what an algorithm "is" and "is not," I am not asserting its stable technical identity; rather, my motive is to be ethnographically true to a members' term and members' practice. As such, then, the limits of the term algorithm are determined by social engagements rather than by technological or material constraints. While social understandings and practices evolve, algorithm, as a term of technical art, nonetheless displays for members some precision and a meaning within a space of alternatives. When technical people get together, the person who says, "I do algorithms" is making a different statement than the person who says, "I study software engineering" or the one who says, "I'm a data scientist," and the nature of these differences matters to any understanding of the relationship between data, algorithms, and society. Accordingly, an investigation of the particular territory staked out by the term "algorithm", in among other related terms and phenomena, seems worthwhile, especially if the algorithm is presented as a site of particularly valuable leverage in contemporary debates.

With that caution in mind, then, we can consider the work that the term "algorithm" does and might do for social analysis contextually.

Algorithm and automation

Perhaps the most diffuse concern expressed by discussion of algorithms is that which uses the notion metonymically to address the regime of digital automation most broadly. Here, the concern is not with algorithms as such, but with a system of digital control and management achieved through sensing, large-scale data storage, and algorithmic processing within a legal, commercial, or industrial framework that lends it authority. We might point here to discussions of credit scoring (e.g. Zarsky, 2016), digitally enhanced public surveillance (e.g. Graham and Wood, 2003), or plagiarism detection (e.g. Introna, 2016) as cases where concerns with the algorithmic, in part or in whole, stand in for critiques of the larger regime of computer-based monitoring and control. To be sure, crucial issues of labor politics, social justice, personal privacy, public accountability, and democratic participation are thrown up by this technologically enabled system of management, and the expansion of the sorts of regulative, coercive, and divisive processes that are the legacy of Charles Babbage and Frederick Taylor, and algorithms play a critical role in these. Indeed, these are among the most important areas of political analysis that an understanding of "algorithm" as a term of technical art and practice can illuminate. Nonetheless, the wholesale equation of algorithm and automation makes this work more, rather than less, difficult. If we want to be able to speak of algorithms analytically in order to identify their significance as specific technical and discursive formulations then we need to be able to better identify how they operate as part of, but not as all of the larger framework.

Algorithm and code

At a greater level of specificity, we might consider the distinctions to be drawn between algorithms and code. In various forms, code has been a particular focus of attention in software studies, acting as it does as a site of material, textual, and representational production. Code is software-as-text, and particularly in the form of "source code," the human-readable expressions of program behavior that are the primary focus of programmers' productive attentions, it has perhaps been particular by those working under the umbrella of

''critical code studies'' (see, e.g., Berry, 2011; Montford et al., 2012).

In textbooks and research papers, algorithms are often expressed in what is informally called ''pseudo-code,'' a textual pastiche of conventional programming languages that embodies general ideas that most languages share without committing to the syntactic or semantic particulars of any one. Pseudo-code expresses the abstract generality of an algorithm, the idea that it can be operationalized in any programming language while transcending the particulars of each. It also expresses the promise of an algorithm, the idea that it is code-waiting-to-happen, ready to be deployed and brought to life in programs yet to be written (Introna, 2016). The idea that the relationship between the algorithm and the code is largely a temporal one is perhaps, then, not surprising, and yet there are distinctions that have a good deal of significance from an analytic perspective. I will outline four here.

First, while the transformation of an algorithm (described in mathematical terms or in pseudo-code) into code may be relatively straight-forward (although it is not necessarily so), the reverse process – to read the algorithm off the code – is not at all a simple process. There are a number of circumstances in which this need arises. Assessing whether an algorithm has been correctly implemented by a piece of code, for example, is one case of attempting to ''read off'' the algorithm (as implemented) from the code, and the complexity of this is made clear by the many cases in which errors slip through. Within the domain of Internet security, for example, there have been a number of headline cases lately where trusted code did not in fact correctly implement the algorithm that it was meant to embody, leaving systems open for attack and data breaches; the ''Heartbleed'' incident is among the best known (Durumeric et al., 2014). The difficulty of reading an algorithm off the code also lies at the heart of patent disputes (over whether a given piece of code does or does not implement a protected algorithm, for instance) as well as simply cropping up as a practical problem for a programmer charged with understanding, maintaining, modifying, or porting an existing software system written by another (or sometimes even the code we wrote ourselves).

Second, algorithms and code have different locality properties. One of the reasons, in fact, that the algorithm may not be easy to read off the code is that the algorithm may not happen all in one place. The algorithm, an apparently singular object when it appears on the page of a book, becomes many different snippets of code distributed through a large program. Even if they happen in sequence when a program is executed, they may not occur together or even nearby within the text of a program. In a program, they may be intermixed with elements of other algorithms, or they might simply be distributed between different modules, different methods, or different functions, so that they operate of the algorithm is (intentionally or unintentionally) obscured.

Third, algorithms are manifest differently on different code platforms. Object-oriented languages, procedural languages, functional languages, and declarative languages are all based on different paradigms for code expression and so will express the same algorithm quite differently. Particular examples of those language styles have different features and different sets of libraries, and will be able to rely on those in different ways to carry out some of the algorithm's operations. Different computer architectures, different data storage technologies, different arrangements of memory hierarchy, and other features of a platform mean that the code of an algorithm is highly variable and highly specific. The ''governing dynamics'' of algorithms (Ananny, 2016), then, are only in part algorithmic; they are as much platform effects.

The fourth observation is something of a corollary to the others, although one with particular consequences. One reason that an algorithm can be hard to recover from a program is that there is a lot in a program that is not ''the algorithm'' (or ''an algorithm''). The residue is machinic, for sure; it is procedural, it involves the stepwise execution of one instruction followed by another, and it follows all the rules of layout, control flow, state manipulation, and access rights that shape any piece of code. But much of it is not actually part of the – or any – algorithm. An algorithm might express, for example, how to transform one kind of data representation into another, or how to reach a numerical result for a formula, or how to transform data so that a particular constraint will hold (e.g. to sort numbers) – but actual programs that implement these algorithms need to do a lot more besides. They read files from disks, they connect to network servers, they check for error conditions, they respond to a user interrupting a process, they flash signals on the screen and play beeps, they shuffle data between different storage units, they record their progress in log files, they check for the size of a screen or the free space on a disk, and many other things besides. An algorithm may express the core of what a program is meant to do, but that core is surrounded by a vast penumbra of ancillary operations that are also a program's responsibility and also manifest themselves in the program's code. In other words, while everything that a program does and that code expresses is algorithmic in the sense that it is specified in advance by formalization, it is not algorithm, in the sense that it goes beyond things that algorithms express, or even what the term ''algorithm'' signals as a term of professional practice.

Algorithm and architecture

The third distinction that it is useful to take up is that between algorithm and architecture. This is an elaboration of part of the earlier discussion, but an elaboration that has particular relevance in the context of contemporary networked systems.

I noted above that algorithms, in the sense of particular formulations of program behavior, may not be easily localizable in code. That is, although they are often defined in terms of a ''sequence of steps'' or ''sequence of operations'', that sequence may not be laid out as a sequence of statements or sequence of lines in a program's text. The algorithm, then, is distributed or fragmented in a program.

Most contemporary programs of any complexity, however, are extremely large – often numbering in the hundreds of thousands or millions of lines of code – and must be arranged according to some organizational structure in order to help programmers and teams manage their complexity and comprehend the whole. So-called software ''architecture'' concerns the arrangement of units, modules, or elements of a larger system, and the patterns of interaction between those units. The nature of the units and the nature of the communication between them depend both on the system's architecture and on the underlying platform. Units might relate to each other as libraries, as inheritance hierarchies, as containerized components, as client/server, or in a host of related ways. The details are not of relevance to the argument here, but the point is this: first, that ''the algorithm'', to the extent that it can be treated as a unit, may not be localized even within a module, never mind within a simple extent of code; and second, that modules may be highly isolated from each other, their code unavailable to each other, perhaps written by different programmers, running on different computers, located within different administrative and management domains, and so forth.

For instance, we might talk of the algorithm by which the Internet manages the flow of data in a Transmission Control Protocol (TCP) stream. Data flow must be regulated so as to avoid congestion on transmission lines, and indeed the development of a new congestion avoidance algorithm in the late 1980s was crucial in allowing the Internet to scale to its current size (Jacobson, 1988). This ''algorithm'' though is hard to locate in practice. It is an algorithm that governs the behavior of two parties, the two end-points of a communication on a network, so they are, by definition, almost always on two different computers. Those different computers quite likely run two different implementations of the TCP/IP protocols, written by different people, and quite possibly the private, undisclosed code belonging to two different organizations. Galloway (2004) has examined protocol as a form of decentralized control, focusing on the questions of conformance and regulation that underlie networked actions, but the protocol, as an agreement or specification to which both parties must conform, obscures, to some extent, the algorithm itself. The algorithm specifies how a protocol should be implemented but it cannot be easily located *as an algorithm* in the running system, distributed as it is between different sites. More generally, the factoring of system behavior into a range of components, some of which are bound together in the same address space, some of which are distributed as different threads or processes, some of which are implemented on different computers, many of which are visible to each other only through restricted interfaces, often means that the ''algorithm'' can not only not be located within an easily delineated stretch of code, but not even within a single computer or the network of a single organization.

Given how many contemporary systems are network-based or network-backed, are designed for large-scale clusters, or even just depend on multi-core or graphics processor-based architectures common in contemporary personal platforms from desktops to wearables, the question of distribution is pervasive. Introna (2016) suggests the language of Barad's (2007) agential realism as a way of thinking about this, recognizing that the ''algorithm'' is itself an ''agential cut'', a means of constituting some semi-stable object within a dynamic and unfolding socio-technical assembly. This does not diminish the power of ''algorithm'' as a way of accounting for the operation of a digital assemblage, by any means, but it does imply that ''algorithm'' may dissolve into nothing when we drill down into the specific elements of a system that might be subject to audit or focused critical or forensic examination (c.f. Kirschenbaum, 2008). Introna's analysis shows that we should examine both what work it takes to identify certain aspects of a running system as the manifestations of an algorithm, and also what is achieved through that collective process and practice of identification.

Algorithm and materialization

The final distinction to explore here is that between the algorithm and its manifestation not just in a piece of code or even in a larger software system but in a specific instantiation – as a running system, running in a particular place, on a particular computer, connected to a particular network, with a particular hardware configuration. All of these critically shape the effect that the algorithm has.

That material configurations limit the effectiveness or reach of algorithms is no surprise; algorithmic formulations do not take into account the storage speeds,

network capacities, instruction pipelines, or memory hierarchies, each of which can have a crucial effect on algorithmic performance. More interestingly, though, the converse is also true – our experience of algorithms can change as infrastructure changes.

Consider an example taken from nuclear weapon simulation (see Dourish and Mazmanian, 2013). Due to nuclear test ban treaties, the nuclear powers have not detonated nuclear weapons in several decades. However, they continue to develop and introduce new weapons. To do so with no testing would be foolhardy, and so new designs are tested but only through simulation (Gusterson, 2001, 2008). In fact, we might argue that it was the ability to produce credible digital simulations of nuclear explosions that made test limitation treaties possible. At this point, the design of new nuclear warheads and weapons is so intrinsically tied to the technology of simulation that one could cite the technology of simulation as one of the major limits upon the production of new weapons. Advances in simulation technology make new simulations practical, and those new simulations open up new avenues for weapons design. Note that the algorithms do not need to change in this scenario; only the technologies upon which they are implemented. The simulation – the algorithm – remains unchanged, but the shifting technological base upon which an implementation of that algorithm runs means that the capacities of that algorithm and its effectiveness within a design process is changing. New technologies shift the effect and impact of an algorithm without changing the algorithm itself; they expand the bounds of algorithmic possibility.

Security infrastructures are a second area where these changes have made a difference. For instance, even simplistic so-called ''brute-force attacks'' on password systems (systematically attempting every possible password) that were once infeasibly hard with simple password technology are now trivial; more sophisticated attacks on more complicated cryptographic systems are similarly now just a matter of assembling enough computing power.

In a wide-ranging examination of algorithms that takes the Viterbi path algorithm as its key example, Mackenzie (2005) takes up some of these questions. The algorithm is powerful and has many applications, but much of what makes it effective in our world is the fact that particular implementations of the algorithm can be embodied in devices and infrastructures with specific operating capacities. Mackenzie's analysis focuses on digital temporality, and here we find a key concern with algorithms and their materialization. To speak of an algorithm like the Viterbi algorithm as ''fast'' is to speak of its complexity, its efficiency and the conditions that limit its performance, but this tells us nothing about how quickly or slowly it might actually perform in practice. The only things that have actual measurable performance (measured in seconds or fractions thereof) are implementations, in software or in silicon. The algorithm, in other words, must be understood *both* as a formalized account of computational possibilities and as a practical tool, and the relationship between these two is not fixed.

Inscrutibility

Stretching across all these discussions are a series of distinctions that seem to anchor the social analysis of algorithms. Algorithms are presented as fast, rather than slow; as automated, rather than hands-on; as machinic, rather than human. Each of these presents a series of problems when algorithms move into new domains.

Perhaps the most significant contrast, though, concerns the problems of inscrutability. The focus of several examinations has been the question of accountability and assessment thrown up by the fact that algorithms are opaque; their operation cannot be examined as easily as those of human actors, for a variety of reasons, leading us to look for new ways to make algorithmic processes visible, to render algorithms accountable, and to find within the algorithmic process some opportunity for audit, external review, and examination (e.g. Pasquale, 2015; Sandvig et al., 2014). Here, I draw on a recent article in these pages by Jenna Burrell (2016), who lays out some of the foundations for algorithmic opacity in order to trouble some of these calls for audit.

Algorithmic opacity

Burrell begins from the problems posed by opaque algorithms. For those for whom algorithmic practice potentially embodies an end-run around traditional forms of legislative accountability, this opacity is a severe problem, and some, such as Pasquale (2015), have argued that algorithms need to be available to audit. Burrell points out, though, that there are multiple different sources of algorithmic opacity, with different relations to mechanisms of redress such as audit. The first is the trade-secret protection that governs many of the algorithms that lie behind services such as Google, Facebook, and Twitter, but also those that are used by financial institutions and other corporations. Audit might have the most force here, where algorithms are held as secrets. A second source of opacity is that the ability to read or understand algorithms is a highly specialized skill, available only to a limited professional class; it depends upon particular education and training. This suggests that audit, at least under

contemporary arrangements, will always be a professionalized and specialized technical practice; with respect to audit, we might be concerned about the problems that have attended financial audit in cases like that of Enron, for example. However, most problematic is Burrell's third source of opacity. As she notes, many of the algorithms that have social and cultural significance, including those that shape the flow of information in social media, the distribution of search results in search engines, and the production of recommendations in online retail, are statistical machine learning algorithms. Operating over large amounts of data, they observe, characterize, and act on patterns that arise in the data. But these patterns are purely statistical and probabilistic phenomena – they are not human designations. A "top-down" approach might operate in terms of human-identified traits, and then seek to find them in the data; the bottom-up approach of statistical machine learning is to identify the patterns first and then see if they can be made sense of for human needs. So, for example, a "bottom-up" algorithm for handwriting recognition has no concept of the alphabet. It has not been programmed with the shape of the letters "A" or "g". It has instead exposed to thousands of examples, on the basis of which it is programmed to recognize certain arrangements of strokes as being characteristic of particular letters. Audit, in this case, has no power to reveal what the algorithm knows, because the algorithm knows only about inexpressible commonalities in millions of pieces of training data.

The questions of what we know and what we can say about the operation of machine learning or Big-Data algorithms of this sort is a key issue at stake in algorithmic analysis. During my years of computer science training, to have an algorithm was to know something. Algorithms were definitive procedures that lead to predictable results. The outcome of the algorithmic operation was known and certain. Much of the debate about "algorithms" at the moment focuses on a particular class of algorithm – statistical machine learning techniques – that produce, instead, unknowns. More accurately, they produce analyses of data that are known and understood in some terms (in terms of the formal properties of the data set – its patterns and regularities) but unknowable in others (in the terms of the domain that the data represents.) When my credit card company deems a particular purchase or stream of purchases "suspicious" and puts a security hold on my card, the company cannot explain exactly what was suspicious – they know that there's something odd but they don't know what it is.

When algorithms come to play a role in social affairs, this begins to matter. As reported by Gillespie (2011), activists in the Occupy Wall Street (OWS) movement were surprised to note that the OWS activities never became a "trending topic" on Twitter (highlighted because of user activity). Some were convinced that this must have indicated censorship; after all, how could the latest pop sensation's haircut or new tattoo be more important than this mass political action? The engineers at Twitter were adamant that no censorship had gone on, but were themselves unable to explain why OWS had not become a trending topic. They can explain the algorithm (although it's a trade secret, so they don't) – the factors that contribute, the ordering and weighting of different properties of tweets and hashtags – but that is not, in itself, enough to account for what happens in the system. To understand that, one must be able to characterize the specific dynamics of the ever-roiling mass of data – the way that people pick up ideas, the dynamics of how they repeat them, the geographical waves of interest, all going by at millions of tweets per minute. It is not just that we cannot easily recreate the circumstances and forensically figure this out (as Heraclitus 2.0 might say, you cannot step twice into the same data stream) but also that the patterns that are being analyzed are ephemeral. And yet we need to find ways to narrate them.

Although the forms of analysis in which statistical machine learning techniques are embedded are referred to with the term "Big Data," there are in fact two scalar moves at work. The first is a move from small to big – from individual data to large data sets, from one record to an accumulated mass of data (as in the Quantified Self movement – c.f. Neff and Nafus, 2016), or from one person to a large population. This is not only the scalar move from which Big Data gains both its name but also certain claims to statistical meaningfulness, and it is the move that allows statistical techniques to start to describe features of populations. The second move, though, is from big to small again, and it is the key move in narrating or accounting for the results of Big Data analysis. Machine learning techniques cluster data but humans read and narrate the clusters that arise as signaling certain categories of people – pregnant women, dual-income Minneapolis families in the market for a new car, disaffected voters, or people likely to cheat on their taxes. Each act of categorization – or more accurately, of narration – is a move from big to small, a reduction of a mass of data points to a narrative element or a defining characteristic, drawn generally from the domain of which we want to know. Electoral data is gathered in order to tell us about voters, and so we find voters in it; purchase data formulates people as consumers, and so we find consumer categories in it. And we find not only voters and consumers, but voters and consumers who can be made sense of in terms that make sense in the domain – geography, income, lifestyle, history, engagement, interests, and inclinations. Big Data analysis says

"this happens along with that" but the narratives we tell of *why* are human ones, not technical ones. We are inclined only to find things in Big Data that we expected, in some sense, to find – or at least, we find the kinds of things that we can make sense of.

It is useful here, then, to return to Wirth's formulation – algorithms + data structures = programs. It speaks to the inherent duality of algorithms and data in the production of running systems, and the problems of attempting to understand one without the other. Wirth speaks of data structures, rather than data, because algorithms are designed around data structures – about forms and regularities rather than around content. (An algorithm for sorting numbers is the same no matter what the numbers – and indeed the same algorithm should also be able to sort names, files, or dates.) Similarly, Burrell's concern with opacity also directs us to be concerned about structures and regularities in data sets and the mechanisms by which we struggle to name them. Concerns with algorithms as inscrutable and illegible may direct us instead towards the need to example the sources of the apparent legibility of data.

Some recent moves by European legislators have shifted the conversation from "audit" to "explanation," arguing that citizens who are substantively affected by the action of an algorithmic system should have a right to an explanation of how that decision was made (Goodman and Flaxman, 2016). The notion of "explanation" here reflects the duality of algorithm and data and the way that each can play a role in automated decision-making. At the same time, though, it begs other questions, including, first, what degree of explanation can successfully "explain" results, and, perhaps more pertinently, how the production of such an explanation – which must, of course be generated algorithmically – can be itself explained.

Directions

What lessons might we draw from this analysis and what directions does it suggest for future analytic work around algorithms? Should we conclude that the term "algorithm" is too beset with problems and misunderstandings to function effectively in critique, and that perhaps it is time to declare a moratorium on its use? Conceptual confusions certainly abound, but the term still carries weight and value if we can appropriately locate it within a larger analytic frame.

One consequence is to pair analyses of algorithms with analyses of the various phenomena of data – data items, data streams, and data structures – upon which they operate and in relation to which they are formulated. The rise of interest in Big Data techniques (e.g. Boellstorff and Maurer, 2015; Kitchin, 2014) is of course a significant source of interest in algorithms in the first place, but the topic of data structures – the specific representations that organize data in order to make it processable by algorithms – have been less prominent. The consequences of representational forms – of the way that data must be shaped to be processed by databases or other informational systems (e.g. Curry, 1998; Dourish, 2014), the organizing principles of data archives (e.g. Edwards et al., 2011) or the relationships between data format, data transmission, and representation (e.g. Dourish, 2015; Galloway, 2004) – is the necessary dual of algorithmic processing. While privacy discusses focus on data generation and accumulation, data organization – the data structures of Wirth's aphoristic equation – require similar scrutiny.

A second concern to which our attention might be drawn on the basis of this exploration is the question of algorithmic identity. How might we go about identifying and pinpointing algorithms in consequence of the vagaries of implementation and the flux of evolution? How can algorithms be isolated and examined, and how much sense does it even make to attempt that exercise? Calls for audit and accountability, or even the manifestation of particular algorithms in order to trace aspects of their history or movements, require some attention to the identity conditions upon which algorithmic sameness or similarity are founded. As Gillespie (2012) has noted, algorithms shift and evolve in deployment, particularly those hidden behind trade secrecy barriers; talking in any coherent way about "Google's search term prediction" algorithm, for example, is deeply problematic given the invisible shifts in implementation and strategy that lie behind the scenes. Mackenzie (2005) considers the patterns of repeatability that algorithms embody within themselves, although one might extend his analysis to consider the forms of repeatability at work in either the successive use of algorithms over different data sets, or the multiple embodiments of "the same" algorithm in different platforms and technologies. Again, the concern is not to engage in an essentializing project with the goal of laying down the criteria for algorithmic sameness; the concern is more to understand how algorithms are identified as, used as, or made to be the same in different settings, circulating as they do among platforms, institutions, corporations, and applications.

In turn, then, this might direct our attention towards a third concern, that of the temporalities of algorithms – not just the temporalities of their own processes (although those matter, because not all algorithms produce answers quickly) but also the temporalities of their evolution as implemented and deployed. Perhaps especially important here are the co-evolution of algorithms and data streams, particularly in cases where these are

mutually influential. An algorithm for, say, modeling climate data is not directly tied to the climate data itself, although it might influence the design of new sensors and data collection instruments (Edwards, 2010), but the algorithm by which Twitter determines the ''trending topics'' that it will report does exist in a feedback loop with the data over which it operates, since trending topics displayed to users of Twitter and influence their own action, including the topics they search and the postings they retweet and comment upon.

These concerns with algorithmic identity and evolution point towards an alternative approach to algorithm studies which might put aside the question of what an algorithm is as a topic of conceptual study and instead adopt a strategy of seeking out and understanding algorithms as objects of professional practice for computer scientists, software engineers, and system developers. What power does the notion of ''algorithm'' have within their conversations and collaborations, and in what way are algorithms invoked, identified, traded, performed, produced, boasted of, denigrated, and elided? What are computer scientists doing with they ''do'' algorithms, and for whom? In this approach, we might examine algorithm as a feature of the world of professional practice and as a member category. A useful model here might be Eric Livingstone's (1986) ethnographic study of the work of mathematicians and the role and nature of ''proof'' in their lived work. Focusing on the ''proof'' not as an abstract truth but as a material form, something to be written on blackboards, demonstrated in conversation, and codified into academic career narratives, Livingstone provides an account of the emergence of an object of professional practice within the everyday practical work of a scientific community. As studies by Mackenzie (2015), Neyland (2016) and Seaver (2015) begin to show, algorithms may benefit from a similar approach. Wendy Chun (2008) has argued cogently for the need to resist fetishizing technical objects such as source code or algorithm, pointing out that a capitulation to purely technical accounts risks obscuring the social and cultural practices by which those technical objects are animated in practice. While acknowledging the force of this argument, I have suggested that both ethnographic responsibility and practical politics require that the term ''algorithm'' as an analytic category must nonetheless be wielded with some precision. Clearly, its emic character is not the limit of what can be said for, with, or about it but we must nevertheless be at least conscious of where and when we make deliberate moves to invoke the term in order to do new conceptual work, and with what consequences. If the term ''algorithm'' appears in social analyses to mean just what it means emically, then it risks missing the many other elements in relation to which the algorithm arises; but by corollary, if it appears in social analyses with some new and different meaning, then it becomes difficult to imagine critiques hitting home in the places that we hope to effect change.

Finally, one of the more intriguing issues to arise in this exploration, and perhaps one that merits further attention, is the relationship between algorithmic and non-algorithmic within technological practice. That is, if algorithms are distinguishable elements of software design, delineable, identifiable, and perhaps even nameable, then we also begin to recognize that there are other elements in software systems that are machinic and programmed but not actually themselves governed by the sorts of things that are normally demarcated as ''algorithms.'' Some may be expressible algorithmically, but they are not themselves the things with which algorithm designers or algorithm analysts concern themselves. These include the happenstance interaction of different systems not necessarily designed in concert (such as the interactions between different flows on a network, different services on a server, or different modules in an application, but they also include the work ''around the edges'' of algorithms even in their most direct implementation – the housekeeping, the error-checking, the storage management, and so on. Given the easy slippage between ''algorithmic'' and ''machinic,'' or between ''algorithmic'' and ''automated,'' the emergence of a category of programmed but not algorithmic activity within computer systems – not governed by algorithms in the sense in which that term is used within computational practice – is intriguing and suggestive. Certainly, it speaks to the potential problems that software studies might have in talking to some of its potential audiences if it talks purely in terms of ''algorithms''. Further, it speaks to the disappearance within algorithm-oriented analysis of the work of making algorithms work. Perhaps, too, it suggests some useful parallels with, say, the elements of engineered systems that are not themselves outcomes of processes of design or engineering, and other gaps, holes, and rifts between systems-as-manifest and systems-as-studied.

Understanding the limits and specificities of ''algorithm,'' then, holds out the opportunity both to engage more meaningfully in interdisciplinary dialogue and to open up new areas for analysis around the edges of algorithmic systems.

Acknowledgements

I would particularly like to thank Evelyn Ruppert for providing the invitation to present the lecture on which this paper is based, and Matthew Fuller, Martin Brynskov, Lone Koefoed Hansen, Adrian Mackenzie, and others in audiences at Goldsmiths and Aarhus Universities for their feedback.

Jenna Burrell kindly shared an early copy of her paper on algorithmic opacity. Much of my thinking on this topic has developed in conversation with Tarleton Gillespie, Scott Mainwaring, Bill Maurer, Helen Nissenbaum, Phoebe Sengers, Nick Seaver, Malte Ziewitz, and other collaborators in the Intel Science and Technology Center for Social Computing. Anonymous reviewers for the journal provided invaluable feedback that has improved the paper considerably.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: in part by the National Science Foundation under awards 1525861 and 1556091.

References

Ananny M (2016) Toward an ethics of algorithms: Convening, observation, probability, and timeliness. *Science, Technology & Human Values* 41(1): 93–117.

Barad K (2007) *Meeting the Universe Halfway: Quantum Physics and the Entanglement of Matter and Meaning*. Durham, NC: Duke University Press.

Barocas S (2014) *Panic Inducing: Data Mining, Fairness, and Privacy,* PhD Thesis, New York University, NY.

Belluck P (2015) Chilly at work? Office formula was devised for men. *New York Times*, 3 August. Available at: http://www.nytimes.com/2015/08/04/science/chilly-at-work-a-decades-old-formula-may-be-to-blame.html (accessed 5 December 2015).

Berry D (2011) *The Philosophy of Software: Code and Mediation in the Digital Age*. Basingstoke, UK: Palgrave Macmillan.

Boellstorff T and Maurer B (eds) (2015) *Data, Now Bigger and Better!* Chicago, IL: Prickly Paradigm Press.

Buenza D and Millo Y (2013) *Folding: Integrating algorithms into the floor of the New York Stock Exchange*. Working paper, Social Science Research Network (SSRN).

Burrell J (2016) How the machine 'thinks': Understanding opacity in machine learning algorithms. *Big Data and Society* 3(1): 1–12.

Chun W (2008) On "sourcery", or code as fetish. *Configurations* 16(3): 299–324.

Cox G (2012) *Speaking Code: Coding as Aesthetic and Political Expression*. Cambridge, MA: MIT Press.

Curry M (1998) *Digital Places: Living with Geographical Information Technologies*. London, UK: Routledge.

Dourish P (2014) NoSQL: The shifting materialities of database technology. *Computational Culture* 4.

Dourish P (2015) Packets, protocols, and proximity: The materialities of internet routing. In: Parks and Starosielski (eds) *Signal Traffic: Critical Studies of Media Infrastructures*. Champaign, IL: University of Illinois Press, pp. 183–204.

Dourish P and Mazmanian M (2013) Media as material: Information representations as material foundations for organizational practice. In: Carlile, Nicolini, Langley, et al. (eds) *How Matter Matters: Objects, Artifacts, and Materiality in Organization Studies*. Oxford, UK: Oxford University Press, pp. 92–118.

Durumeric Z, Kasten J, Adrian D, et al. (2014) The matter of heartbleed. In: *Proceedings of ACM internet measurement conference IMC'14*, Vancouver, BC, Canada, pp. 475–488.

Edwards P (2010) *A Vast Machine: Computer Models, Climate Data, and the Politics of Global Warming*. Cambridge, MA: MIT Press.

Edwards P, Mayernik M, Batcheller A, et al. (2011) Science friction: Data, metadata, and collaboration. *Social Studies of Science* 41(5): 667–690.

Fuller M (2008) *Software Studies: A Lexicon*. Cambridge, MA: MIT Press.

Galloway A (2004) *Protocol: How Control Exists after Decentralization*. Cambridge, MA: MIT Press.

Gillespie T (2011) Can an algorithm be wrong? Available at: http://culturedigitally.org/2011/10/can-an-algorithm-be-wrong/ (accessed 5 December 2015).

Gillespie T (2012) The relevance of algorithms. In: Gillespie T, Boczkowski P and Foot K (eds) *Media Technologies: Essays on Communication, Materiality, and Society*. Cambridge, MA: The MIT Press.

Glaser V (2014) Enchanted algorithms: How organizations use algorithms to automate decision-making routines. In: *Proceedings of the Annual Meeting of the Academy of Management*, Philadelphia, PA.

Goodman B and Flaxman S (2016) EU regulations on algorithmic decision-making and a "Right to Explanation". In: *International conference on machine learning workshop on human interpretability in machine learning (WHI 2016)*, June, New York, NY, pp. 26–30.

Graham SDN and Wood D (2003) Digitizing surveillance: Categorization, space, inequality. *Critical Social Policy* 23(2): 227–248.

Gusterson H (2001) The virtual nuclear weapons laboratory in the new world order. *American Ethnologist* 28(2): 417–437.

Gusterson H (2008) Nuclear futures: Anticipating knowledge, expert judgment and the lack that cannot be filled. *Science and Public Policy* 35(8): 551–560.

Hansel S (2007) Google answer to filling jobs is an algorithm. *New York Times*, 3 January. Available at: http://www.nytimes.com/2007/01/03/technology/03google.html (accessed 5 December 2015).

Introna LD (2016) Algorithms, governance, and governmentality: On governing academic writing. *Science, Technology & Human Values* 41(1): 17–49.

Jacobson V (1988) Congestion avoidance and control. In: *Proceedings of ACM symposium on communications architectures and protocols SIGCOMM'88*, Stanford, CA, pp. 314–329.

Kirschenbaum M (2008) *Mechanisms: New Media and the Forensic Imagination*. Cambridge, MA: MIT Press.

Kitchin R (2014) *The Data Revolution: Big Data, Open Data, Data Infrastructures and Their Consequences*. London, UK: Sage.

Livingstone E (1986) *The Ethnomethodological Foundations of Mathematics*. Boston, MA: Routledge & Kegan Paul.

Mackenzie A (2005) Protocols and the irreducible traces of embodiment: The Viterbi algorithm and the mosaic of machine time. In: Hassan (ed) *24/7: Time and Temporality in the Network Society*. Stanford, CA: Stanford University Press, pp. 89–108.

Mackenzie A (2006) *Cutting Code: Software and Sociality*. Pieterlen, Switzerland: Peter Lang International Academic Publishers.

Mackenzie A (2015) The production of prediction: What does machine learning want? *European Journal of Cultural Studies* 18(4–5): 429–445.

Manovich L (2001) *The Language of New Media*. Cambridge, MA: MIT Press.

Manovich L (2013) *Software Takes Command*. London, UK: Bloomsbury.

Montford N and Bogost I (2009) *Racing the Beam: The Atari Video Computer System*. Cambridge, MA: MIT Press.

Montford N, Baudoin P, Bell J, et al. (2012) *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*. Cambridge, MA: MIT Press.

Neff G and Nafus D (2016) *The Quantified Self*. Cambridge, MA: MIT Press.

Neyland D (2016) Bearing account-able witness to the ethical algorithmic system. *Science, Technology & Human Values* 41(1): 50–76.

Pasquale F (2015) *The Black Box Society: The Secret Algorithms that Control Money and Information*. Cambridge, MA: Harvard University Press.

Rosenblat A and Stark L (2016) Uber's drivers: Information asymmetries and control in dynamic work. *International Journal of Communication* 10: 3758–3784.

Sandvig C, Hamilton K, Karahalios K, et al. (2014) Auditing algorithms: Research methods for detecting discrimination on internet platforms. In: *Annual Meeting of the International Communication Association*. Seattle, WA, pp. 1–23.

Seaver N (2015) Working with algorithms: Plans and mess. In: Kai Franz (ed) *Serial Nature*. Stuttgart: Edition Solitude.

Singer N (2014) The scoreboards where you can't see your score. *New York Times*, 27 December. Available at: http://www.nytimes.com/2014/12/28/technology/the-scoreboards-where-you-cant-see-your-score.html (accessed 5 December 2015).

Wirth N (1975) *Algorithms + Data Structures = Programs*. Englewood Cliffs, NJ: Prentice-Hall.

Zarsky T (2016) The trouble with algorithmic decisions: An analytic road map to examine efficiency and fairness in automated and opaque decision making. *Science, Technology & Human Values* 41(1): 118–132.

Ziewitz M (2015) Governing algorithms: Myth, mess, and methods. *Science, Technology & Human Values* 41(4): 3–16.