Routledge
Taylor & Francis Group

Information, Communication & Society

ISSN: 1369-118X (Print) 1468-4462 (Online) Journal homepage: https://www.tandfonline.com/loi/rics20

Thinking critically about and researching algorithms

Rob Kitchin

To cite this article: Rob Kitchin (2017) Thinking critically about and researching algorithms, Information, Communication & Society, 20:1, 14-29, DOI: 10.1080/1369118X.2016.1154087

To link to this article: https://doi.org/10.1080/1369118X.2016.1154087


Published online: 25 Feb 2016.


Submit your article to this journal


Article views: 7289


View Crossmark data


Citing articles: 61 View citing articles

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=rics20

INFORMATION, COMMUNICATION & SOCIETY, 2017
VOL. 20, NO. 1, 14–29
http://dx.doi.org/10.1080/1369118X.2016.1154087

Routledge
Taylor & Francis Group

Thinking critically about and researching algorithms

Rob Kitchin

NIRSA, National University of Ireland Maynooth, County Kildare, Ireland

ABSTRACT

More and more aspects of our everyday lives are being mediated, augmented, produced and regulated by software-enabled technologies. Software is fundamentally composed of algorithms: sets of defined steps structured to process instructions/data to produce an output. This paper synthesises and extends emerging critical thinking about algorithms and considers how best to research them in practice. Four main arguments are developed. First, there is a pressing need to focus critical and empirical attention on algorithms and the work that they do given their increasing importance in shaping social and economic life. Second, algorithms can be conceived in a number of ways – technically, computationally, mathematically, politically, culturally, economically, contextually, materially, philosophically, ethically – but are best understood as being contingent, ontogenetic and performative in nature, and embedded in wider socio-technical assemblages. Third, there are three main challenges that hinder research about algorithms (gaining access to their formulation; they are heterogeneous and embedded in wider systems; their work unfolds contextually and contingently), which require practical and epistemological attention. Fourth, the constitution and work of algorithms can be empirically studied in a number of ways, each of which has strengths and weaknesses that need to be systematically evaluated. Six methodological approaches designed to produce insights into the nature and work of algorithms are critically appraised. It is contended that these methods are best used in combination in order to help overcome epistemological and practical challenges.

ARTICLE HISTORY
Received 30 September 2015
Accepted 10 February 2016

KEYWORDS
Algorithm; code; epistemology; methodology; research

Introduction: why study algorithms?

The era of ubiquitous computing and big data is now firmly established, with more and more aspects of our everyday lives – play, consumption, work, travel, communication, domestic tasks, security, etc. – being mediated, augmented, produced and regulated by digital devices and networked systems powered by software (Greenfield, 2006; Kitchin & Dodge, 2011; Manovich, 2013; Steiner, 2012). Software is fundamentally composed of algorithms – sets of defined steps structured to process instructions/data to produce an output – with all digital technologies thus constituting 'algorithm machines' (Gillespie, 2014a). These 'algorithm machines' enable extensive and complex tasks to be tackled that would be all but impossible by hand or analogue machines. They can perform millions

CONTACT Rob Kitchin rob.kitchin@nuim.ie
© 2016 Informa UK Limited, trading as Taylor & Francis Group

of operations per second; minimise human error and bias in how a task is performed; and can significantly reduce costs and increase turnover and profit through automation and creating new services/products (Kitchin & Dodge, 2011). As such, dozens of key sets of algorithms are shaping everyday practices and tasks, including those that perform search, secure encrypted exchange, recommendation, pattern recognition, data compression, auto-correction, routing, predicting, profiling, simulation and optimisation (MacCormick, 2013).

As Diakopoulos (2013, p. 2) argues: 'We're living in a world now where algorithms adjudicate more and more consequential decisions in our lives. … Algorithms, driven by vast troves of data, are the new power brokers in society.' Steiner (2012, p. 214) thus contends:

> algorithms already have control of your money market funds, your stocks, and your retirement accounts. They'll soon decide who you talk to on phone calls; they will control the music that reaches your radio; they will decide your chances of getting lifesaving organs transplant; and for millions of people, algorithms will make perhaps the largest decision of in their life: choosing a spouse.

Similarly, Lenglet (2011), MacKenzie (2014), Arnoldi (2016), Pasquale (2015) document how algorithms have deeply and pervasively restructured how all aspects of the finance sector operate, from how funds are traded to how credit agencies assess risk and sort customers. Amoore (2006, 2009) details how algorithms are used to assess security risks in the 'war on terror' through the profiling passengers and citizens. With respect to the creation of Wikipedia, Geiger (2014, p. 345) notes how algorithms 'help create new articles, edit existing articles, enforce rules and standards, patrol for spam and vandalism, and generally work to support encyclopaedic or administrative work.' Likewise, Anderson (2011) details how algorithms are playing an increasingly important role in producing content and mediating the relationships between journalists, audiences, newsrooms and media products.

In whatever domain algorithms are deployed they appear to be having disruptive and transformative effect, both to how that domain is organised and operates, and to the labour market associated with it. Steiner (2012) provides numerous examples of how algorithms and computation have led to widespread job losses in some industries through automation. He concludes

> programmers now scout new industries for soft spots where algorithms might render old paradigms extinct, and in the process make mountains of money … Determining the next field to be invaded by bots [automated algorithms] is the sum of two simple functions: the potential to disrupt plus the reward for disruption. (Steiner, 2012, pp. 6, 119)

Such conclusions have led a number of commentators to argue that we are now entering an era of widespread algorithmic governance, wherein algorithms will play an ever-increasing role in the exercise of power, a means through which to automate the disciplining and controlling of societies and to increase the efficiency of capital accumulation. However, Diakopoulos (2013, p. 2, original emphasis) warns that: 'What we generally lack as a public is *clarity about how algorithms exercise their power over us.*' Such clarity is absent because although algorithms are imbued with the power to act upon data and make consequential decisions (such as to issue fines or block travel or approve a loan) they are largely black boxed and beyond query or question. What is at stake

then with the rise of 'algorithm machines' is new forms of algorithmic power that are reshaping how social and economic systems work.

In response, over the past decade or so, a growing number of scholars have started to focus critical attention on software code and algorithms, drawing on and contributing to science and technology studies, new media studies and software studies, in order to unpack the nature of algorithms and their power and work. Their analyses typically take one of three forms: a detailed case study of a single algorithm, or class of algorithms, to examine the nature of algorithms more generally (e.g., Bucher, 2012; Geiger, 2014; Mackenzie, 2007; Montfort et al., 2012); a detailed examination of the use of algorithms in one domain, such as journalism (Anderson, 2011), security (Amoore, 2006, 2009) or finance (Pasquale, 2014, 2015); or a more general, critical account of algorithms, their nature and how they perform work (e.g., Cox, 2013; Gillespie, 2014a, 2014b; Seaver, 2013).

This paper synthesises, critiques and extends these studies. Divided into two main sections – thinking critically about and researching algorithms – the paper makes four key arguments. First, as already noted, there is a pressing need to focus critical and empirical attention on algorithms and the work that they do in the world. Second, it is most productive to conceive of algorithms as being contingent, ontogenetic, performative in nature and embedded in wider socio-technical assemblages. Third, there are three main challenges that hinder research about algorithms (gaining access to their formulation; they are heterogeneous and embedded in wider systems; their work unfolds contextually and contingently), which require practical and epistemological attention. Fourth, the constitution and work of algorithms can be empirically studied in a number of ways, each of which has strengths and weaknesses that need to be systematically evaluated. With respect to the latter, the paper provides a critical appraisal of six methodological approaches that might profitably be used to produce insights into the nature and work of algorithms.

Thinking critically about algorithms

While an algorithm is commonly understood as a set of defined steps to produce particular outputs it is important to note that this is somewhat of a simplification. What constitutes an algorithm has changed over time and they can be thought about in a number of ways: technically, computationally, mathematically, politically, culturally, economically, contextually, materially, philosophically, ethically and so on.

Miyazaki (2012) traces the term 'algorithm' to twelfth-century Spain when the scripts of the Arabian mathematician Muḥammad ibn Mūsā al-Khwārizmī were translated into Latin. These scripts describe methods of addition, subtraction, multiplication and division using numbers. Thereafter, 'algorism' meant 'the specific step-by-step method of performing written elementary arithmetic' (Miyazaki, 2012, p. 2) and 'came to describe any method of systematic or automatic calculation' (Steiner, 2012, p. 55). By the mid-twentieth century and the development of scientific computation and early high level programming languages, such as Algol 58 and its derivatives (short for ALGOrithmic Language), an algorithm was understood to be a set of defined steps that if followed in the correct order will computationally process input (instructions and/or data) to produce a desired outcome (Miyazaki, 2012).

From a computational and programming perspective an 'Algorithm = Logic + Control'; where the logic is the problem domain-specific component and specifies the abstract

formulation and expression of a solution (what is to be done) and the control component is the problem-solving strategy and the instructions for processing the logic under different scenarios (how it should be done) (Kowalski, 1979). The efficiency of an algorithm can be enhanced by either refining the logic component or by improving the control over its use, including altering data structures (input) to improve efficiency (Kowalski, 1979). As reasoned logic, the formulation of an algorithm is, in theory at least, independent of programming languages and the machines that execute them; 'it has an autonomous existence independent of "implementation details"' (Goffey, 2008, p. 15).

Some ideas explicitly take the form of an algorithm. Mathematical formulae, for example, are expressed as precise algorithms in the form of equations. In other cases problems have to be abstracted and structured into a set of instructions (pseudo-code) which can then be coded (Goffey, 2008). A computer programme structures lots of relatively simple algorithms together to form large, often complex, recursive decision trees (Neyland, 2015; Steiner, 2012). The methods of guiding and calculating decisions are largely based on Boolean logic (e.g., if this, then that) and the mathematical formulae and equations of calculus, graph theory and probability theory. Coding thus consists of two key translation challenges centred on producing algorithms. The first is translating a task or problem into a structured formula with an appropriate rule set (pseudo-code). The second is translating this pseudo-code into source code that when compiled will perform the task or solve the problem. Both translations can be challenging, requiring the precise definition of what a task/problem is (logic), then breaking that down into a precise set of instructions, factoring in any contingencies such as how the algorithm should perform under different conditions (control). The consequences of mistranslating the problem and/or solution are erroneous outcomes and random uncertainties (Drucker, 2013).

The processes of translation are often portrayed as technical, benign and commonsensical. This is how algorithms are mostly presented by computer scientists and technology companies: that they are 'purely formal beings of reason' (Goffey, 2008, p. 16). Thus, as Seaver (2013) notes, in computer science texts the focus is centred on how to design an algorithm, determine its efficiency and prove its optimality from a purely technical perspective. If there is discussion of the work algorithms do in real-world contexts this concentrates on how algorithms function in practice to perform a specific task. In other words, algorithms are understood 'to be strictly rational concerns, marrying the certainties of mathematics with the objectivity of technology' (Seaver, 2013, p. 2). 'Other knowledge about algorithms – such as their applications, effects, and circulation – is strictly out of frame' (Seaver, 2013, pp. 1–2). As are the complex set of decision-making processes and practices, and the wider assemblage of systems of thought, finance, politics, legal codes and regulations, materialities and infrastructures, institutions, inter-personal relations, which shape their production (Kitchin, 2014).

Far from being objective, impartial, reliable and legitimate, critical scholars argue that algorithms possess none of these qualities except as carefully crafted fictions (Gillespie, 2014a). As Montfort et al. (2012, p. 3) note, '[c]ode is not purely abstract and mathematical; it has significant social, political, and aesthetic dimensions,' inherently framed and shaped by all kinds of decisions, politics, ideology and the materialities of hardware and infrastructure that enact its instruction. Whilst programmers might seek to maintain a high degree of mechanical objectivity – being distant, detached and impartial in how

they work and thus acting independent of local customs, culture, knowledge and context (Porter, 1995) – in the process of translating a task or process or calculation into an algorithm they can never fully escape these. Nor can they escape factors such as available resources and the choice and quality of training data; requirements relating to standards, protocols and the law; and choices and conditionalities relating to hardware, platforms, bandwidth and languages (Diakopoulos, 2013; Drucker, 2013; Kitchin & Dodge, 2011; Neyland, 2015). In reality then, a great deal of expertise, judgement, choice and constraints are exercised in producing algorithms (Gillespie, 2014a). Moreover, algorithms are created for purposes that are often far from neutral: to create value and capital; to nudge behaviour and structure preferences in a certain way; and to identify, sort and classify people.

At the same time, 'programming is … a live process of engagement between thinking with and working on materials and the problem space that emerges' (Fuller, 2008, p. 10) and it 'is not a dry technical exercise but an exploration of aesthetic, material, and formal qualities' (Montfort et al., 2012, p. 266). In other words, creating an algorithm unfolds in context through processes such as trial and error, play, collaboration, discussion and negotiation. They are ontogenetic in nature (always in a state of becoming), teased into being: edited, revised, deleted and restarted, shared with others, passing through multiple iterations stretched out over time and space (Kitchin & Dodge, 2011). As a result, they are always somewhat uncertain, provisional and messy fragile accomplishments (Gillespie, 2014a; Neyland, 2015). And such practices are complemented by many others, such as researching the concept, selecting and cleaning data, tuning parameters, selling the idea and product, building coding teams, raising finance and so on. These practices are framed by systems of thought and forms of knowledge, modes of political economy, organisational and institutional cultures and politics, governmentalities and legalities, subjectivities and communities. As Seaver (2013, p. 10) notes, 'algorithmic systems are not standalone little boxes, but massive, networked ones with hundreds of hands reaching into them, tweaking and tuning, swapping out parts and experimenting with new arrangements.'

Creating algorithms thus sits at the 'intersection of dozens of … social and material practices' that are culturally, historically and institutionally situated (Montfort et al., 2012, p. 262; Napoli, 2013; Takhteyev, 2012). As such, as Mackenzie (2007, p. 93) argues treating algorithms simply 'as a general expression of mental effort, or, perhaps even more abstractly, as process of abstraction, is to lose track of proximities and relationalities that algorithms articulate.' Algorithms cannot be divorced from the conditions under which they are developed and deployed (Geiger, 2014). What this means is that algorithms need to be understood as relational, contingent, contextual in nature, framed within the wider context of their socio-technical assemblage. From this perspective, 'algorithm' is one element in a broader apparatus which means it can never be understood as a technical, objective, impartial form of knowledge or mode of operation.

Beyond thinking critically about the nature of algorithms, there is also a need to consider their work, effects and power. Just as algorithms are not neutral, impartial expressions of knowledge, their work is not impassive and apolitical. Algorithms search, collate, sort, categorise, group, match, analyse, profile, model, simulate, visualise and regulate people, processes and places. They shape how we understand the world and they do work in and make the world through their execution as software, with profound consequences (Kitchin & Dodge, 2011). In this sense, they are profoundly performative

as they cause things to happen (Mackenzie & Vurdubakis, 2011). And while the creators of these algorithms might argue that they 'replace, displace, or reduce the role of biased or self-serving intermediaries' and remove subjectivity from decision-making, computation often deepens and accelerates processes of sorting, classifying and differentially treating, and reifying traditional pathologies, rather than reforming them (Pasquale, 2014, p. 5).

Far from being neutral in nature, algorithms construct and implement regimes of power and knowledge (Kushner, 2013) and their use has normative implications (Anderson, 2011). Algorithms are used to seduce, coerce, discipline, regulate and control: to guide and reshape how people, animals and objects interact with and pass through various systems. This is the same for systems designed to empower, entertain and enlighten, as they are also predicated on defined rule-sets about how a system behaves at any one time and situation. Algorithms thus claim and express algorithmic authority (Shirky, 2009) or algorithmic governance (Beer, 2009; Musiani, 2013), often through what Dodge and Kitchin (2007) term 'automated management' (decision-making processes that are automated, automatic and autonomous; outside of human oversight). The consequence for Lash (2007) is that society now has a new rule set to live by to complement constitutive and regulative rules: algorithmic, generative rules. He explains that such rules are embedded within computation, an expression of 'power through the algorithm'; they are 'virtuals that generate a whole variety of actuals. They are compressed and hidden and we do not encounter them in the way that we encounter constitutive and regulative rules. … They are … pathways through which capitalist power works' (Lash, 2007, p. 71).

It should be noted, however, that the effects of algorithms or their power is not always linear or always predictable for three reasons. First, algorithms act as part of a wider network of relations which mediate and refract their work, for example, poor input data will lead to weak outcomes (Goffey, 2008; Pasquale, 2014). Second, the performance of algorithms can have side effects and unintended consequences, and left unattended or unsupervised they can perform unanticipated acts (Steiner, 2012). Third, algorithms can have biases or make mistakes due to bugs or miscoding (Diakopoulos, 2013; Drucker, 2013). Moreover, once computation is made public it undergoes a process of domestication, with users embedding the technology in their lives in all kinds of alternative ways and using it for different means, or resisting, subverting and reworking the algorithms' intent (consider the ways in which users try to game Google's PageRank algorithm). In this sense, algorithms are not just what programmers create, or the effects they create based on certain input, they are also what users make of them on a daily basis (Gillespie, 2014a).

Steiner's (2012, p. 218) solution to living with the power of algorithms is to suggest that we '[g]et friendly with bots.' He argues that the way to thrive in the algorithmic future is to learn to 'build, maintain, and improve upon code and algorithms,' as if knowing how to produce algorithms protects oneself from their diverse and pernicious effects across multiple domains. Instead, I would argue, there is a need to focus more critical attention on the production, deployment and effects of algorithms in order to understand and contest the various ways that they can overtly and covertly shape life chances. However, such a programme of research is not as straightforward as one might hope, as the next section details.

Researching algorithms

The logical way to flesh out our understanding of algorithms and the work they do in the world is to conduct detailed empirical research centrally focused on algorithms. Such research could approach algorithms from a number of perspectives:

> a technical approach that studies algorithms as computer science; a sociological approach that studies algorithms as the product of interactions among programmers and designers; a legal approach that studies algorithms as a figure and agent in law; a philosophical approach that studies the ethics of algorithms, (Barocas, Hood, & Ziewitz, 2013, p. 3)

and a code/software studies' perspective that studies the politics and power embedded in algorithms, their framing within a wider socio-technical assemblage and how they reshape particular domains. There are a number of methodological approaches that can be used to operationalise such research, six of which are critically appraised below. Before doing so, however, it is important to acknowledge that there are three significant challenges to researching algorithms that require epistemological and practical attention.

Challenges

Access/black boxed

Many of the most important algorithms that people encounter on a regular basis and which (re)shape how they perform tasks or the services they receive are created in environments that are not open to scrutiny and their source code is hidden inside impenetrable executable files. Coding often happens in private settings, such as within companies or state agencies, and it can be difficult to negotiate access to coding teams to observe them work, interview programmers or analyse the source code they produce. This is unsurprising since it is often a company's algorithms that provide it with a competitive advantage and they are reluctant to expose their intellectual property even with non-disclosure agreements in place. They also want to limit the ability of users to game the algorithm to unfairly gain a competitive edge. Access is a little easier in the case of open-source programming teams and open-source programmes through repositories such as Github, but while they provide access to much code, this is limited in scope and does not include key proprietary algorithms that might be of more interest with respect to holding forms of algorithmic governance to account.

Heterogeneous and embedded

If access is gained, algorithms, as Seaver (2013) notes, are rarely straightforward to deconstruct. Within code algorithms are usually woven together with hundreds of other algorithms to create algorithmic systems. It is the workings of these algorithmic systems that we are mostly interested in, not specific algorithms, many of which are quite benign and procedural. Algorithmic systems are most often 'works of collective authorship, made, maintained, and revised by many people with different goals at different times' (Seaver, 2013, p. 10). They can consist of original formulations mashed together with those sourced from code libraries, including stock algorithms that are re-used in multiple instances. Moreover, they are embedded within complex socio-technical assemblages made up of a heterogeneous set of relations including potentially thousands of individuals, data sets, objects, apparatus, elements, protocols, standards, laws, etc. that frame their development.

Their construction, therefore, is often quite messy, full of 'flux, revisability, and negotiation' (p. 10), making unpacking the logic and rationality behind their formulation difficult in practice. Indeed, it is unlikely that any one programmer has a complete understanding of a system, especially large, complex ones that are built by many teams of programmers, some of whom may be distributed all over the planet or may have only had sight of smaller outsourced segments. Getting access to a credit rating agency's algorithmic system then might give an insight into its formula for assessing and sorting individuals, its underlying logics and principles, and how it was created and works in practice, but will not necessarily provide full transparency as to its full reasoning, workings or the choices made in its construction (Bucher, 2012; Chun, 2011).

Ontogenetic, performative and contingent

As well as being heterogeneous and embedded, algorithms are rarely fixed in form and their work in practice unfolds in multifarious ways. As such, algorithms need to be recognised as being ontogenetic, performative and contingent: that is, they are never fixed in nature, but are emergent and constantly unfolding. In cases where an algorithm is static, for example, in firmware that is not patched, its work unfolds contextually, reactive to input, interaction and situation. In other cases, algorithms and their instantiation in code are often being refined, reworked, extended and patched, iterating through various versions (Miyazaki, 2012). Companies such as Google and Facebook might be live running dozens of different versions of an algorithm to assess their relative merits, with no guarantee that the version a user interacts with at one moment in time is the same as five seconds later. In some cases, the code has been programmed to evolve, re-writing its algorithms as it observes, experiments and learns independently of its creators (Steiner, 2012).

Similarly, many algorithms are designed to be reactive and mutable to inputs. As Bucher (2012) notes, Facebook's EdgeRank algorithm (that determines what posts and in what order are fed into each users' timeline) does not act from above in a static, fixed manner, but rather works in concert with the each individual user, ordering posts dependent on how one interacts with 'friends.' Its parameters then are contextually weighted and fluid. In other cases, randomness might be built into an algorithm's design meaning its outcomes can never be perfectly predicted. What this means is that the outcomes for users inputting the same data might vary for contextual reasons (e.g., Mahnke and Uprichard (2014) examined Google's autocomplete search algorithm by typing in the same terms from two locations and comparing the results, finding differences in the suggestions the algorithm gave), and the same algorithms might be being used in quite varied and mutable ways (e.g., for work or for play). Examining one version of an algorithm will then provide a snapshot reading that fails to acknowledge or account for the mutable and often multiple natures of algorithms and their work (Bucher, 2012).

Algorithms then are often 'out of control' in the sense that their outcomes are sometimes not easily anticipated, producing unexpected effects in terms of their work in the world (Mackenzie, 2005). As such, understanding the work and effects of algorithms needs to be sensitive to their contextual, contingent unfolding across situation, time and space. What this means in practice is that single or limited engagements with algorithms cannot be simply extrapolated to all cases and that a set of comparative case studies

need to be employed, or a series of experiments performed with the same algorithm operating under different conditions.

Approaches

Keeping in mind these challenges, this final section critically appraises six methodological approaches for researching algorithms that I believe present the most promise for shedding light on the nature and workings of algorithms, their embedding in socio-technical systems, their effects and power, and dealing with and overcoming the difficulties of gaining access to source code. Each approach has its strengths and drawbacks and their use is not mutually exclusive. Indeed, I would argue that there would be much to be gained by using two or more of the approaches in combination to compensate for the drawbacks of employing them in isolation. Nor are they the only possible approaches, with ethnomethodologies, surveys and historical analysis using archives and oral histories offering other possible avenues of analysis and insight.

Examining pseudo-code/source code

Perhaps the most obvious way to try and understand an algorithm is to examine its pseudo-code (how a task or puzzle is translated into a model or recipe) and/or its construction in source code. There are three ways in which this can be undertaken in practice. The first is to carefully deconstruct the pseudo-code and/or source code, teasing apart the rule set to determine how the algorithm works to translate input to produce an outcome (Krysa & Sedek, 2008). In practice this means carefully sifting through documentation, code and programmer comments, tracing out how the algorithm works to process data and calculate outcomes, and decoding the translation process undertaken to construct the algorithm. The second is to map out a genealogy of how an algorithm mutates and evolves over time as it is tweaked and rewritten across different versions of code. For example, one might deconstruct how an algorithm is re-scripted in multiple instantiations of a programme within a code library such as github. Such a genealogy would reveal how thinking with respect to a problem is refined and transformed with respect to how the algorithm/code performs 'in the wild' and in relation to new technologies, situations and contexts (such as new platforms or regulations being introduced). The third is to examine how the same task is translated into various software languages and how it runs across different platforms. This is an approach used by Montfort et al. (2012) in their exploration of the '10 PRINT' algorithm, where they scripted code to perform the same task in multiple languages and ran it on different hardware, and also tweaked the parameters, to observe the specific contingencies and affordances this introduced.

While these methods do offer the promise of providing valuable insights into the ways in which algorithms are built, how power is vested in them through their various parameters and rules, and how they process data in abstract and material terms to complete a task, there are three significant issues with their deployment. First, as noted by Chandra (2013), deconstructing and tracing how an algorithm is constructed in code and mutates over time is not straightforward. Code often takes the form of a 'Big Ball of Mud': '[a] haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle' (Foote & Yoder, 1997; cited in Chandra, 2013, p. 126). Even those that have produced it can find it very difficult to unpack its algorithms and routines; those unfamiliar with its

development can often find that the ball of mud remains just that. Second, it requires that the researcher is both an expert in the domain to which the algorithm refers and possesses sufficient skill and knowledge as a programmer that they can make sense of a 'Big Ball of Mud'; a pairing that few social scientists and humanities scholars possess. Third, these approaches largely decontextualise the algorithm from its wider socio-technical assemblage and its use.

Reflexively producing code

A related approach is to conduct auto-ethnographies of translating tasks into pseudo-code and the practices of producing algorithms in code. Here, rather than studying an algorithm created by others, a researcher reflects on and critically interrogates their own experiences of translating and formulating an algorithm. This would include an analysis of not only the practices of exploring and translating a task, originating and developing ideas, writing and revising code, but also how these practices are situated within and shaped by wider socio-technical factors such as regulatory and legal frameworks, form of knowledge, institutional arrangements, financial terms and conditions, and anticipated users and market. Ziewitz (2011) employed this kind of approach to reflect on producing a random routing algorithm for directing a walking path through a city, reflecting on the ontological uncertainty in the task itself (that there is often an ontological gerrymandering effect at work as the task itself is re-thought and re-defined while the process of producing an algorithm is undertaken), and the messy, contingent process of creating the rule set and parameters in practice and how these also kept shifting through deferred accountability. Similarly, Ullman (1997) uses such an approach to consider the practices of developing software and how this changed over her career.

While this approach will provide useful insights into how algorithms are created, it also has a couple of limitations. The first is the inherent subjectivities involved in doing an auto-ethnography and the difficulties of detaching oneself and gaining critical distance to be able to give clear insight into what is unfolding. Moreover, there is the possibility that in seeking to be reflexive what would usually take place is inflected in unknown ways. Further, it excludes any non-representational, unconscious acts from analysis. Second, one generally wants to study algorithms and code that have real concrete effects on peoples' everyday lives, such as those used in algorithmic governance. One way to try and achieve this is to contribute to open-source projects where the code is incorporated into products that others use, or to seek access to a commercial project as a programmer (on an overt, approved basis with non-disclosure agreements in place). The benefit here is that the method can be complemented with the sixth approach set out below, examining and reflecting on the relationship between the production of an algorithm and any associated ambitions and expectations vis-à-vis how it actually does work in the world.

Reverse engineering

In cases where the code remains black boxed, a researcher interested in the algorithm at the heart of its workings is left with the option of trying to reverse engineer the compiled software. Diakopoulos (2013, p. 13) explains that '[r]everse engineering is the process of articulating the specifications of a system through a rigorous examination drawing on domain knowledge, observation, and deduction to unearth a model of how that system

works.' While software producers might desire their products to remain opaque, each programme inherently has two openings that enable lines of enquiry: input and output. By examining what data are fed into an algorithm and what output is produced it is possible to start to reverse engineer how the recipe of the algorithm is composed (how it weights and preferences some criteria) and what it does.

The main way this is attempted is by using carefully selected dummy data and seeing what is outputted under different scenarios. For example, researchers might search Google using the same terms on multiple computers in multiple jurisdictions to get a sense of how its PageRank algorithm is constructed and works in practice (Mahnke & Uprichard, 2014), or they might experiment with posting and interacting with posts on Facebook to try and determine how its EdgeRank algorithm positions and prioritises posts in user time lines (Bucher, 2012), or they might use proxy servers and feed dummy user profiles into e-commerce systems to see how prices might vary across users and locales (*Wall Street Journal*, detailed in Diakopoulos, 2013). One can also get a sense of an algorithm by 'looking closely at how information must be oriented to face them, how it is made algorithm-ready'; how the input data are delineated in terms of what input variables are sought and structured, and the associated meta-data (Gillespie, 2014a). Another possibility is to follow debates on online forums by users about how they perceive an algorithm works or has changed, or interview marketers, media strategists, and public relations firms that seek to game an algorithm to optimise an outcome for a client (Bucher, 2012).

While reverse engineering can give some indication of the factors and conditions embedded into an algorithm, they generally cannot do so with any specificity (Seaver, 2013). As such, they usually only provide fuzzy glimpses of how an algorithm works in practice but not its actual constitution (Diakopoulos, 2013). One solution to try and enhance clarity has been to employ bots, which posing as users, can more systematically engage with a system, running dummy data and interactions. However, as Seaver (2013) notes, many proprietary systems are aware that many people are seeking to determine and game their algorithm, and thus seek to identify and block bot users.

Interviewing designers or conducting an ethnography of a coding team

While deconstructing or reverse engineering code might provide some insights into the workings of an algorithm, they provide little more than conjecture as to the intent of the algorithm designers, and examining that and how and why an algorithm was produced requires a different approach. Interviewing designers and coders, or conducting an ethnography of a coding team, provides a means of uncovering the story behind the production of an algorithm and to interrogate its purpose and assumptions.

In the first case, respondents are questioned as to how they framed objectives, created pseudo-code and translated this into code, and quizzed about design decisions and choices with respect to languages and technologies, practices, influences, constraints, debates within a team or with clients, institutional politics and major changes in direction over time (Diakopoulos, 2013; MacKenzie, 2014; Mager, 2012). In the second case, a researcher seeks to spend time within a coding team, either observing the work of the coders, discussing it with them, and attending associated events such as team meetings, or working in situ as part of the team, taking an active role in producing code. An example of the former is Rosenberg's (2007) study of one company's attempt to produce a new product conducted over a three-year period in which he was given full access to the company, including

observing and talking to coders, and having access to team chat rooms and phone conferences. An example of the latter is Takhteyev's (2012) study of an open-source coding project in Rio de Janeiro where he actively worked on developing the code, as well as taking part in the social life of the team. In both cases, Rosenberg and Takhteyev generate much insight into the contingent, relational and contextual way in which algorithms and software are produced, though in neither case are the specificities of algorithms and their work unpacked and detailed.

Unpacking the full socio-technical assemblage of algorithms

As already noted, algorithms are not formulated or do not work in isolation, but form part of a technological stack that includes infrastructure/hardware, code platforms, data and interfaces, and are framed and conditions by forms of knowledge, legalities, governmentalities, institutions, marketplaces, finance and so on. A wider understanding of algorithms then requires their full socio-technical assemblage to be examined, including an analysis of the reasons for subjecting the system to the logic of computation in the first place. Examining algorithms without considering their wider assemblage is, as Geiger (2014) argues, like considering a law without reference to the debate for its introduction, legal institutions, infrastructures such as courts, implementers such as the police, and the operating and business practices of the legal profession. It also risks fetishising the algorithm and code at the expense of the rest of the assemblage (Chun, 2011).

Interviews and ethnographies of coding projects, and the wider institutional apparatus surrounding them (e.g., management and institutional collaboration), start to produce such knowledge, but they need to supplemented with other approaches, such as a discursive analysis of company documents, promotional/industry material, procurement tenders and legal and standards frameworks; attending trade fairs and other inter-company interactions; examining the practices, structures and behaviour of institutions; and documenting the biographies of key actors and the histories of projects (Montfort et al., 2012; Napoli, 2013). Such a discursive analysis will also help to reveal how algorithms are imagined and narrated, illuminate the discourses surrounding and promoting them, and how they are understood by those that create and promote them. Gaining access to such a wider range of elements, and being able to gather data and interlink them to be able to unpack a socio-technical assemblage, is no easy task but it is manageable as a large case study, especially if undertaken by a research team rather than a single individual.

Examining how algorithms do work in the world

Given that algorithms do active work in the world it is important not only to focus on the construction of algorithms, and their production within a wider assemblage, but also to examine how they are deployed within different domains to perform a multitude of tasks. This cannot be simply denoted from an examination of the algorithm/code alone for two reasons. First, what an algorithm is designed to do in theory and what it actually does in practice do not always correspond due to a lack of refinement, miscodings, errors and bugs. Second, algorithms perform in context – in collaboration with data, technologies, people, etc. under varying conditions – and therefore their effects unfold in contingent and relational ways, producing localised and situated outcomes. When users employ an algorithm, say for play or work, they are not simply playing or working in conjunction with the algorithm, rather they are 'learning, internalizing, and becoming intimate with' it

(Galloway, 2006, p. 90); how they behave is subtly reshaped through the engagement, but at the same time what the algorithm does is conditional on the input it receives from the user. We can therefore only know how algorithms make a different to everyday life by observing their work in the world under different conditions.

One way to undertake such research is to conduct ethnographies of how people engage with and are conditioned by algorithmic systems and how such systems reshape how organisations conduct their endeavours and are structured (e.g., Lenglet, 2011). It would also explore the ways in which people resist, subvert and transgress against the work of algorithms, and re-purpose and re-deploy them for purposes they were not originally intended. For example, examining the ways in which various mobile and web applications were re-purposed in the aftermath of the Haiti earthquake to coordinate disaster response, remap the nation and provide donations (Al-Akkad et al., 2013). Such research requires detailed observation and interviews focused on the use of particular systems and technologies by different populations and within different scenarios, and how individuals interfaced with the algorithm through software, including their assessments as to their intentions, sense of what is occurring and associated consequences, tactics of engagement, feelings, concerns and so on. In cases where an algorithm is black boxed, such research is also likely to shed some light on the constitution of the algorithm itself.

Conclusion

On an average day, people around the world come into contact with hundreds of algorithms embedded into the software that operates communications, utilities and transport infrastructure, and powers all kinds of digital devices used for work, play and consumption. These algorithms have disruptive and transformative effect, reconfiguring how systems operate, enacting new forms of algorithmic governance and enabling new forms of capital accumulation. Yet, despite their increasing pervasiveness, utility and the power vested in them to act in autonomous, automatic and automated ways, to date there has been limited critical attention paid to algorithms in contrast to the vast literature that approaches algorithms from a more technical perspective. This imbalance in how algorithms are thought about and intellectually engaged with is perhaps somewhat surprising given what is at stake in a computationally rich world. As such, there is a pressing need for critical attention across the social sciences and humanities to be focused on algorithms and forms of algorithmic governance. The contribution of this paper to this endeavour has been to: advance an understanding of algorithms as contingent, ontogenetic, performative in nature and embedded in wider socio-technical assemblages; to detail the epistemological and practical challenges facing algorithm scholars; and to critically appraise six promising methodological options to empirically research and make sense of algorithms. It is apparent from the studies conducted to date that there is a range of different ways of making sense of algorithms and the intention of the paper has not been to foreclose this diversity, but rather to encourage synthesis, comparison and evaluation of different positions and to create new ones. Indeed, the more angles taken to uncover and debate the nature and work of algorithms the better we will come to know them.

Likewise, the six approaches appraised were selected because I believe they hold the most promise in exposing how algorithms are constructed, how they work within socio-

technical assemblages and how they perform actions and make a difference in particular domains, but they are by no means the only approaches that might be profitably pursued. My contention is, given each approach's varying strengths and weaknesses, that how they reveal the nature and work of algorithms needs to be systematically evaluated through methodologically focused research. Studies that have access to the pseudo-code, code and coders may well be the most illuminating, though they still face a number of challenges, such as deciphering how the algorithm works in practice. Moreover, there is a need to assess: (1) how they might be profitably used in conjunction with each other to overcome epistemological and practical challenges; (2) what other methods might be beneficially deployed in order to better understand the nature, production and use of algorithms? With respect to the latter, such methods might include ethnomethodologies, surveys, historical analysis using archives and oral histories, and comparative case studies. As such, while the approaches and foci I have detailed provide a useful starting set that others can apply, critique, refine and extend, there are others that can potentially emerge as critical research and thinking on algorithms develops and matures.

Acknowledgements

Many thanks to Tracey Lauriault, Sung-Yueh Perng and the referees for comments on earlier versions of this paper.

Disclosure statement

No potential conflict of interest was reported by the author.

Funding

The research for this paper was funded by a European Research Council Advanced Investigator award [ERC-2012-AdG-323636-SOFTCITY].

Notes on contributor

Rob Kitchin is a professor and ERC Advanced Investigator at the National University of Ireland Maynooth. He is currently a principal investigator on the Programmable City project, the Digital Repository of Ireland, the All-Island Research Observatory and the Dublin Dashboard. [email: rob.kitchin@nuim.ie]

References

Al-Akkad, A., Ramirez, L., Denef, S., Boden, A., Wood, L., Buscher, M., & Zimmermann, A. (2013). '*Reconstructing normality*': *The use of infrastructure leftovers in crisis situations as inspiration for the design of resilient technology*. Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration (pp. 457–466). New York, NY: ACM. Retrieved October 16, 2014, from http://dl.acm.org/citation.cfm?doid=2541016.2541051

Amoore, L. (2006). Biometric borders: Governing mobilities in the war on terror. *Political Geography*, *25*, 336–351.

Amoore, L. (2009). Algorithmic war: Everyday geographies of the war on terror. *Antipode*, *41*, 49–69.
Anderson C. W. (2011). Deliberative, agonistic, and algorithmic audiences: Journalism's vision of its public in an age of audience. *Journal of Communication*, *5*, 529–547.
Arnoldi, J. (2016). Computer algorithms, market manipulation and the institutionalization of high frequency trading. *Theory, Culture & Society*, *33*(1), 29–52.
Barocas, S., Hood, S., & Ziewitz, M. (2013). Governing algorithms: A provocation piece. Retrieved October 16, 2014, from http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2245322
Beer, D. (2009). Power through the algorithm? Participatory Web cultures and the technological unconscious. *New Media and Society*, *11*(6), 985–1002.
Bucher, T. (2012). 'Want to be on the top?' Algorithmic power and the threat of invisibility on Facebook. *New Media and Society*, *14*(7), 1164–1180.
Chandra, V. (2013). *Geek sublime: Writing fiction, coding software*. London: Faber.
Chun, W. H. K. (2011). *Programmed visions*. Cambridge: MIT Press.
Cox, G. (2013). *Speaking code: Coding as aesthetic and political expression*. Cambridge: MIT Press.
Diakopoulos, N. (2013). *Algorithmic accountability reporting: On the investigation of black boxes*. A Tow/Knight Brief. Tow Center for Digital Journalism, Columbia Journalism School. Retrieved August 21, 2014, from http://towcenter.org/algorithmic-accountability-2/
Dodge, M., & Kitchin, R. (2007). The automatic management of drivers and driving spaces. *Geoforum*, *38*(2), 264–275.
Drucker, J. (2013). Performative materiality and theoretical approaches to interface. *Digital Humanities Quarterly*, *7*(1). Retrieved June 5, 2014, from http://www.digitalhumanities.org/dhq/vol/7/1/000143/000143.html
Foote, B., & Yoder, J. (1997). Big Ball of Mud. *Pattern Languages of Program Design*, *4*, 654–692.
Fuller, M. (2008). Introduction. In M. Fuller (Ed.), *Software studies – A lexicon* (pp. 1–14). Cambridge: MIT Press.
Galloway, A. R. (2006). *Gaming: Essays on algorithmic culture*. Minneapolis: University of Minnesota Press.
Geiger, S. R. (2014). Bots, bespoke, code and the materiality of software platforms. *Information, Communication & Society*, *17*(3), 342–356.
Gillespie, T. (2014a). The relevance of algorithms. In T. Gillespie, P. J. Boczkowski, & K. A. Foot (Eds.), *Media technologies: Essays on communication, materiality, and society* (pp. 167–193). Cambridge: MIT Press.
Gillespie, T. (2014b, June 25). Algorithm [draft] [#digitalkeyword]. *Culture Digitally*. Retrieved October 16, 2014, from http://culturedigitally.org/2014/06/algorithm-draft-digitalkeyword/
Goffey, A. (2008). Algorithm. In M. Fuller (Ed.), *Software studies – A lexicon* (pp. 15–20). Cambridge: MIT Press.
Greenfield, A. (2006). *Everyware: The dawning age of ubiquitous computing*. Boston, MA: New Riders.
Kitchin, R. (2014). *The data revolution: Big data, open data, data infrastructures and their consequences*. London: Sage.
Kitchin, R., & Dodge, M. (2011). *Code/space: Software and everyday life*. Cambridge: MIT Press.
Kowalski, R. (1979). Algorithm = Logic + Control. *Communications of the ACM*, *22*(7), 424–436.
Krysa, J., & Sedek, G. (2008). Source code. In M. Fuller (Ed.), *Software studies – A lexicon* (pp. 236–242). Cambridge: MIT Press.
Kushner, S. (2013). The freelance translation machine: Algorithmic culture and the invisible industry. *New Media & Society*, *15*(8), 1241–1258.
Lash, S. (2007). Power after hegemony: Cultural studies in mutation. *Theory, Culture & Society*, *24*(3), 55–78.
Lenglet, M. (2011). Conflicting codes and codings: How algorithmic trading is reshaping financial regulation. *Theory, Culture & Society*, *28*(6), 44–66.
MacCormick, J. (2013). *Nine algorithms that changed the future: The ingenious ideas that drive today's computers*. Princeton, NJ: Princeton University Press.

Mackenzie, A. (2005). The performativity of code: Software and cultures of circulation. *Theory, Culture & Society*, *22*(1), 71–92.
Mackenzie, A. (2007). Protocols and the irreducible traces of embodiment: The Viterbi algorithm and the mosaic of machine time. In R. Hassan & R. E. Purser (Eds.), *24/7: Time and temporality in the network society* (pp. 89–106). Stanford, CA: Stanford University Press.
Mackenzie, A., & Vurdubakis, T. (2011). Code and codings in Crisis: Signification, performativity and excess. *Theory, Culture & Society*, *28*(6), 3–23.
MacKenzie, D. (2014). *A sociology of algorithms: High-frequency trading and the shaping of markets*. Working paper, University of Edinburgh. Retrieved July 6, 2015, from http://www.sps.ed.ac.uk/__data/assets/pdf_file/0004/156298/Algorithms25.pdf
Mager, A. (2012). Algorithmic ideology: How capitalist society shapes search engines. *Information, Communication, & Society*, *15*(5), 769–787.
Mahnke, M., & Uprichard, E. (2014). Algorithming the algorithm. In R. König & M. Rasch (Eds.), *Society of the query reader: Reflections on web search* (pp. 256–270). Amsterdam: Institute of Network Cultures.
Manovich, L. (2013). *Software takes control*. New York, NY: Bloomsbury.
Miyazaki, S. (2012). Algorhythmics: Understanding micro-temporality in computational cultures. *Computational Culture*, Issue 2. Retrieved June 25, 2014, from http://computationalculture.net/article/algorhythmics-understanding-micro-temporality-in-computational-cultures
Montfort, N., Baudoin, P., Bell, J., Bogost, I., Douglass, J., Marino, M. C., … Vawter, N. (2012). *10 PRINT CHR$ (205.5 + RND (1)): GOTO 10*. Cambridge: MIT Press.
Musiani, F. (2013). Governance by algorithms. *Internet Policy Review*, *2*(3). Retrieved October 7, 2014, from http://policyreview.info/articles/analysis/governance-algorithms
Napoli, P. M. (2013, May). *The algorithm as institution: Toward a theoretical framework for automated media production and consumption*. Paper presented at the Media in Transition Conference, Massachusetts Institute of Technology, Cambridge, MA. Retrieved from ssrn.com/abstract = 2260923
Neyland, D. (2015). On organizing algorithms. *Theory, Culture & Society*, *32*(1), 119–132.
Pasquale, F. (2014). *The emperor's new codes: Reputation and search algorithms in the finance sector*. Draft for discussion at the NYU 'Governing Algorithms' conference. Retrieved October 16, 2014, from http://governingalgorithms.org/wp-content/uploads/2013/05/2-paper-pasquale.pdf
Pasquale, F. (2015). *The black box society: The secret algorithms that control money and information*. Cambridge, MA: Harvard University Press.
Porter, T. M. (1995). *Trust in numbers: The pursuit of objectivity in science and public life*. Princeton, NJ: Princeton University Press.
Rosenberg, S. (2007). *Dreaming in code: Two dozen programmers, three years, 4,732 bugs, and one quest for transcendent software*. New York: Three Rivers Press.
Seaver, N. (2013). *Knowing Algorithms*. Media in Transition 8, Cambridge, MA. Retrieved August 21, 2014, from http://nickseaver.net/papers/seaverMiT8.pdf
Shirky, C. (2009). *A speculative post on the idea of algorithmic authority*. Shirky.com. Retrieved October 7, 2014, from http://www.shirky.com/weblog/2009/11/a-speculative-post-on-the-idea-of-algorithmic-authority/
Steiner, C. (2012). *Automate this: How algorithms took over our markets, our jobs, and the world*. New York, NY: Portfolio.
Takhteyev, Y. (2012). *Coding places: Software practice in a South American City*. Cambridge: MIT Press.
Ullman, E. (1997). *Close to the machine*. San Francisco, CA: City Lights Books.
Ziewitz, M. (2011, September 29). *How to think about an algorithm? Notes from a not quite random walk*. Discussion paper for Symposium on 'Knowledge Machines between Freedom and Control'. Retrieved August 21, 2014, from http://ziewitz.org/papers/ziewitz_algorithm.pdf