Thinking critically about and researching algorithms

Rob Kitchin, National University of Ireland Maynooth, County Kildare, Ireland

The Programmable City Working Paper 5

http://www.nuim.ie/progcity/

28th October 2014

Abstract

The era of ubiquitous computing and big data is now firmly established, with more and more aspects of our everyday lives being mediated, augmented, produced and regulated by digital devices and networked systems powered by software. Software is fundamentally composed of algorithms -- sets of defined steps structured to process instructions/data to produce an output. And yet, to date, there has been little critical reflection on algorithms, nor empirical research into their nature and work. This paper synthesises and extends initial critical thinking about algorithms and considers how best to research them in practice. It makes a case for thinking about algorithms in ways that extend far beyond a technical understanding and approach. It then details four key challenges in conducting research on the specificities of algorithms -- they are often: 'black boxed'; heterogeneous, contingent on hundreds of other algorithms, and are embedded in complex socio-technical assemblages; ontogenetic and performative; and 'out of control' in their work. Finally, it considers six approaches to empirically research algorithms: examining source code (both deconstructing code and producing genealogies of production); reflexively producing code; reverse engineering; interviewing designers and conducting ethnographies of coding teams; unpacking the wider socio-technical assemblages framing algorithms; and examining how algorithms do work in the world.

Key words: algorithm, code, epistemology, research

Introduction

> "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. ... What we generally lack as a public is *clarity about how algorithms exercise their power over us*. With that clarity comes an increased ability to publicly debate and dialogue the merits of any particular algorithmic power. While legal codes are available for us to read, algorithmic codes are more opaque, hidden behind layers of technical complexity" (Diakopoulos 2013: 2, original emphasis).

> "algorithms ... are rarely discussed in themselves and rarely attended to as objects of analysis" (Mackenzie 2007: 93).

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 2007; Kitchin and Dodge 2011; Steiner 2012; Manovich 2013). 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). 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).

In response, the field of software studies has emerged over the past decade taking software as its object of critical analysis, considering how it is produced and deployed, and how it does work in the world (see Fuller 2008; Chun 2011; Manovich 2013). In such work there has been an emphasis of shifting attention from examining the effects of digital technologies on society to unpacking the nature of the software that makes them function. Kitchin and Dodge (2011) liken this to the difference between studying the effects of ill-health on the world and charting the epidemiology of ill-health and the aetiology of different illnesses and diseases. The former consists of examining how systems such as telematic networks shape traffic management, but does so without considering in detail how such

effects are manifestly the result of how telematic systems are constituted and configured, with rules and procedures formalised within algorithms and code. That said, software studies itself has largely focused on the production of software and how its use reshapes how different domains function. As a result, most studies tend to concentrate on compiled code, rather than unpacking and making sense of its constitute code and the algorithms this expresses, as lamented by Mackenzie (2007) and Diakopoulos (2013).

However, in the past couple of years, a small number of critical scholars have started to focus attention on code and algorithms, drawing on ideas from science and technology studies and new media studies. This paper synthesises and extends these studies and critical reflections developing, in particular, critical thinking with respect to algorithms and considers how best to research them in practice. The first section sets out what an algorithm is and how they relate to software. The second section makes a case for thinking about algorithms in ways that extend beyond a technical understanding and approach. The third details four key challenges in conducting research on the specificities of algorithms -- they are: often 'black boxed'; heterogeneous, often contingent on hundreds of other algorithms, and are embedded in complex socio-technical assemblages; ontogenetic and performative; and their work is often 'out of control'. This is followed a discussion of six approaches to empirically researching algorithms: examining source code (both deconstructing code and producing genealogies of production); reflexively producing code; reverse engineering; interviewing designers and conducting ethnographies of coding teams; unpacking the wider socio-technical assemblage framing and supporting algorithms; and examining how algorithms do work in the world.

What is an algorithm?

According to Miyazaki (2012), the term 'algorithm' can be traced to 12th 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: 2) and "came to describe any method of systematic or automatic calculation" (Steiner 2012: 55). In the mid-20th 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).

As a set of defined steps, an algorithm can be relatively easily codified; that is, turned into code that if executed will perform the algorithm (Knuth 1968; Chun 2008; Goffey 2008). Code is "the uncompiled, non-executable code of a computer program stored in source files"; a set of commands written in either a low level language (such as machine or assembly language, which provide specific instructions direct to the microprocessor) or a higher level programming language that has some characteristics of natural language, for example, rules of syntax (Krysa and Sedek 2008: 237). When compiled, the code is converted into machine executable code, a string of binary processor commands that when run performs the algorithm(s).

From a 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) (Kowakski 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 (Kowakski 1979). As Goffey (2008: 17) states, "[a]lgorithms do things, and their syntax embodies a command structure to enable this to happen". 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: 15).

Some ideas explicitly take the form of an algorithm. Mathematical formula, for example, are expressed as precise algorithms in the form of equations. For example, Pythagorean theory states that for a right-angle triangle the square of the hypotenuse (the side opposite the right angle; c) is equal to the sum of the square of the other two sides ($a + b$), commonly expressed as $a^2 + b^2 = c^2$. This equation can be easily turned into a set of instructions for calculating a, b or c, given different known information. For example, if we know the value of a and b we can calculate c using the formula:

$$c = \sqrt{a^2} + b^2$$

This is straightforward to express as code. For example, using PHP script[1] as:

```
<?php
 $a = 3;
 $b = 4;
 $c = sqrt($a*$a + $b*$b);
 echo "a = " .$a. "<br>";
 echo "b = " .$b. "<br>";
 echo "c = " .$c. "<br>";
 ?>
```

This code will produce *c* for a triangle where side *a* is 3 units and side *b* is 4 units (to calculate for other lengths those inputs just need to be changed appropriately). Line 4 performs the calculation, and lines 5 to 7 print the results on the screen. It should be noted that algebraic notion is "a symbolic abstract statement that is not bound to any form of concrete execution" and thus can be expressed in different ways (e.g., $a^2 + b^2 = c^2$ or $c = \sqrt{a^2} + b^2$), however in code it is a definitive statement and is non-reversible ($a^2 + b^2 \Rightarrow c^2$) (Miyazaki 2012: 3).

In other cases problems do not explicitly take the form of an algorithm but have to be abstracted and structured into a set of instructions which can then be coded (Goffey 2008). For example, we might want to calculate the number of ghost estates[2] in Ireland using a database of all the properties in the country that details their occupancy and construction status. There is no readily defined algorithm for such a calculation so one needs to be created[3]. First, we need to define what is a ghost estate in terms of (a) how many houses grouped together constitute an estate (e.g., 5, 10, 20, etc)?; (b) what proportion of these houses have to be empty or under-construction for that estate to be labelled a ghost estate (e.g., 10%, 20%, 50%, etc)? We can then combine these rules into a simple formula -- "a ghost estate is an estate of 10 or more houses where over 50% of houses are vacant or under-construction". Next we can write a program that searches and sifts the property database to find estates that meet our criteria and totals up the overall number. We could extend the algorithm to also record the coordinates of each qualifying estate and use another set of algorithms to plot them onto a digital map. In this way lots of relatively simple algorithms are structured together to form large, often complex, recursive decision trees (Steiner 2012;

[1] http://php.about.com/od/finishedphp1/ss/hypotenuse_2.htm#step-heading
[2] In the wake of the 2008 global financial crash the Irish property sector collapsed leaving the landscape scared with unfinished or unoccupied housing estates that were dubbed 'ghost estates'.
[3] The discussion of the creation of an algorithm is a real one, with the process of formulating the rule set discussed in detail in Kitchin *et al*. (2013).

Neyland 2014). 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. First, translating a task or problem into a structured formula with an appropriate rule set (what is sometimes called pseudo-code). Second, translating this recipe into code that when compiled will perform the task or solve the problem. Both translations can be challenging. On the one hand, there is the challenge of defining precisely 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). This requires understanding a process so well that "you can explain it to something as stonily stupid as a computer" (Fuller 2008: 10). There are many tasks and problems that are extremely difficult or impossible to translate into algorithms (MacCormick 2013) or, if they are, they are hugely oversimplified. Also, the answer to a task/problem might be indeterminate or fuzzy, meaning that an algorithm might be seeking the most appropriate, approximate or relevant result without necessarily knowing what they are (e.g., search algorithms do not know precisely what a user is seeking and using a handful of key words aim to provide a list of relevant results) (Gillespie 2014a). The consequence of mistranslating the problem and/or solution will lead to erroneous outcomes and random uncertainties (Drucker 2013). On the other hand, there are a set of constraints that have to be considered and accounted for, such as limitations/foibles with respect to input data or the hardware processing the program/data or responding to its outputs. Consequently, there may be many algorithms that might provide a workable solution or reach the same result, with programmers choosing between them based on how quickly, efficiently or elegantly they perform (Gillespie 2014b). And the same solution might be expressed within code in quite different ways depending on the idiosyncrasies of how a programmer translates the solution and constructs code (Cox 2013).

Nonetheless, while programmers might vary in how they formulate code, the processes of translation is 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: 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: 2). "Other knowledge about algorithms — such as their applications, effects, and circulation — is strictly out of frame" (Seaver 2013: 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, that shape their production (Kitchin 2014). As a consequence, how algorithms are most often understood is very narrowly framed and lacking in critical reflection. The next section sets out a more critical reading of the creation and use of algorithms.

Thinking critically about algorithms

Given the increasing role of algorithms in mediating everyday life, following Seaver (2013) and others (e.g., Bucher 2012, Gillespie 2014a, Goffey 2008, Montfort *et al*. 2012), it is vital that we develop more critical ways of thinking about them that does not keep particular viewpoints 'strictly out of frame' and which situate them within their wider socio-technical assemblages. This requires technical approaches to be complemented by perspectives that consider: the discursive logic driving the propensity to translate practices and systems into computation; how the practices of coding algorithms are thoroughly social, cultural, political and economic in nature; and how algorithms perform diverse tasks, much of which raises political, economic and ethical concerns.

(1) The logic of computation

As noted, it is undoubtedly the case that more and more of the tasks of everyday life, and the work previously undertaken by people or analogue machines, are being subject to computation; that is, translated into psuedo-code and sets of algorithms that are compiled into software that then mediate those tasks. The premise is that "[a]lmost everything we do, from driving a car to trading a stock to picking a spouse, can be broken down into a string of binary decisions based on binary input" (Steiner 2012: 25). That is, there is a logic to all action and all tasks and systems can be disassembled into component parts and reassembled as a set of interwoven algorithms that calculate and perform a task based on appropriate input.

Computation is seen as a way of dealing with extensive and complex tasks effectively and efficiently, as well as creating new ways of defining and tackling tasks by enabling new

approaches that would have been all but impossible by hand or analogue machine due to the difficulties and time-consuming nature of calculation. Digital systems formalize rule sets and can operate in automated, automatic, and autonomous ways; they can perform millions of operations per second; they largely eliminate human error or subjectivity in how a task is performed; and they can significantly reduce costs and increase turnover and profit (Kitchin and Dodge 2011). In so doing, computation is seen to make economic activity and public administration more competitive, productive, efficient and effective. As a result, over the past half century millions of ideas and tasks have been translated into algorithms and code (logic + control), including multiple social systems (e.g., education, health, welfare, dating, entertainment, communication, policing). The resulting software are thus not just "applications, but also models for life itself," and their work "shapes the possibilities of life" (Kushner 2013: 1243).

This logic of computation is compelling because, on the one hand, it asserts a coherent logic and brings order and rationality to complex systems, and on the other it produces profit and enables the exercise of power at the same time as it seemingly improves everyday life. The former is seen to produce systems that are technical, objective, impartial, commonsensical, pragmatic, and reliable. On this basis they assert to be credible, trustworthy, practical and "legitimate brokers of relevant knowledge" and action (Gillespie 2014a). With respect to the latter, code induces a process of interpellation, wherein people willingly and voluntarily subscribe to and desire its logic, trading potential negative effects such as the erosion of privacy or new forms of regulation against benefits gained such as convenience, pleasure, knowledge, creativity, productivity, and so on (Kitchin and Dodge 2011; Cox 2013).

For all the benefits the logic of computation asserts, questions remain as to what is lost in a society that whole-heartedly embraces the logic of computation? The notion that nearly everything we do can be broken down into and processed through algorithms is inherently highly reductionist. It assumes that complex, often fuzzy, relational, and contextual social and economic interactions and modes of being can be logically disassembled, modelled and translated, whilst only losing a minimum amount of tacit knowledge and situational contingencies. Moreover, as Gillespie (2014a) notes: "Algorithms do not just process data, they produce, affirm and certify knowledge through a particular logic built on specific assumptions. Computation asserts and prioritizes a particular epistemological way of making sense of and acting in the world; of codifying practices and knowledges and processing them using algorithms." It thus prioritizes a certain mode of

sense making and an instrumental rationality based on a combined and narrowly framed 'episteme (scientific knowledge) and teche (practical instrumental knowledge)', which works to marginalise and replace 'phronesis (knowledge derived from practice and deliberation) and metis (knowledge based on experience)' (Parsons 2004: 49).

The logic of computation thus works to trump and close off other kinds of knowing and acting, narrowing fields of view and action. In so doing, it changes how organisations perceive themselves, their objectives, problems, tasks, consumers, and so on. For example, algorithmic journalism (wherein algorithms are used to create and distribute news stories) changes what is considered news, how the audience is understood and treated, and shapes what news they engage with (Anderson 2011). Both content and audience is atomised, quantified, computed. And as algorithms are increasingly used they become legitimated, endeavours are fitted to them, reifying choices made and forcing additional ones, and the logic of computation becomes reinforced and promoted (Gillespie 2014a).

But to what extent does the epistemology of algorithms and computation, and its claims to objectivity, impartiality and legitimacy, hold up to critical scrutiny? It is to this question the paper now turns.

(2) The framing and production of algorithms

> "[T]here is an important tension emerging between what we expect these algorithms to be, and what they in fact are" (Gillespie 2011).

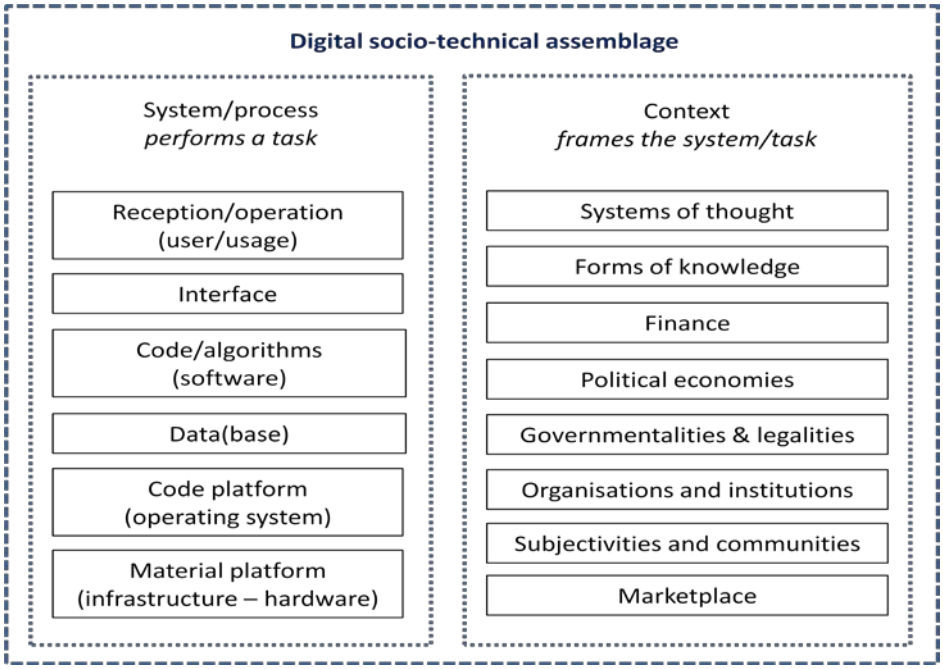
The tension that Gillespie (2011) describes is the variance between the epistemology detailed in the last section, and the general desire to perceive algorithms as being objective, impartial, reliable, and legitimate, and the reality that they possess none of these qualities except as carefully crafted fictions (Gillespie 2014a). As Montfort *et al*. (2013: 3) note, "[c]ode is not purely abstract and mathematical; it has significant social, political, and aesthetic dimensions," being the products of complex socio-technical assemblages, inherently framed and shaped by all kinds of decisions, politics and ideology. 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 (Kitchin and Dodge 2011; Diakopoulos 2013; Drucker 2014; Neyland 2014). 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, 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: 10) and it "is not a dry technical exercise but an exploration of aesthetic, material, and formal qualities" (Montfort *et al.* 2012: 266). In other words, creating an algorithm unfolds in context through processes such as trial and error, play, collaboration, discussion, and negotiation. They are teased into being: edited, revised, deleted and restarted, shared with others, passing through multiple iterations stretched out over time and space. As a result, they are always somewhat uncertain, provisional and messy fragile accomplishments (Gillespie 2014a; Neyland 2014). 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. And 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: 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: 262; Takhteyev 2012, Napoli 2013). As such, as Mackenzie (2007) 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 set of apparatus (see Figure 1) -- admittedly, along with data, a crucial element that the other elements support and facilitate --which means it can never be understood as a simply a technical, objective, impartial form of knowledge or mode of operation.

Figure 1: The socio-technical assemblage surrounding algorithms



(3) The work, effects and power of algorithms

Beyond thinking critically about the nature of algorithms and their epistemology, 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, analyze, profile, model, simulate, visualize 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 and Dodge 2011). In this sense, they are profoundly performative as they cause things to happen (Mackenzie and Vurdubakis 2011); they are engines not cameras (Mackenzie 2008).

Algorithms dictate how the myriad of software systems, digital devices, networked infrastructure that now permeate everyday life operate. As Steiner (2012: 214) details:

> "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, Pasquale (2014) documents 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. With respect to the creation of Wikipedia, Geiger (2014) 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." Anderson (2011) details how algorithms are playing an increasingly important role in mediating between journalists, audiences, newsrooms, and media products.

Indeed, whatever the domain algorithms are deployed in they are 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. 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: 6, 119). 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 simply deepens and accelerates processes of sorting, classifying and differentially treating, reifying traditional pathologies, rather than reforming them (Pasquale (2014: 5).

Far from being neutral then 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: 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). 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: 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 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. Indeed, despite the brief observations noted above, there has been few in-depth empirical studies about the work, effects and power of algorithms in different spheres. To rectify this situation requires that we more proactively and systematically empirically research algorithms, a task that 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 *et al*. 2013: 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 different ways that such research could be operationalized, six of which are detailed below. Before doing so, however, it is important to acknowledge that there are four significant challenges to researching algorithms that require consideration and, where possible, solutions.

Challenges

(1) 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 analyze the source code they produce. This is unsurprising since it is often a company's algorithms that provide them with a competitive edge 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 competitive edge. Access is a little easier in the case of open-source programming teams and open-source programs through repositories such as Github, but while they provide access to much code, this is limited in scope and does not include key propriety algorithms produced and deployed by companies that might of more interest. For some, such as Pasquale (2014) the fact that the rules and power of many crucial algorithms are hidden from view and are not transparent, accountable and contestable, is a major source of concern, especially in systems that regulate and shape life chances. It is only by resolving the issue of open scrutiny that algorithmic governance can be held to account. Nonetheless, it is likely that negotiating access to algorithms and their construction will continue to thwart key research taking place.

(2) Heterogeneous and embedded

If access is gained, some argue that algorithms and code can be decoded in relatively straightforward ways. For example, Montfort *et al*. (2012: 7) state:

> "code is ultimately understandable. Programs cause a computer to operate in a particular way, and there is some reason for this operation that is grounded in the design and material reality of the computer, the programming language, and the particular program. This reason can be found. The way code works is not a divine mystery or an imponderable. ... The working of code is knowable." (Montfort *et al*: 2012: 7)

In contrast, others such as Seaver (2013) contest this assertion. He argues that algorithms, even when their workings are revealed, are rarely straightforwardly deconstructed. Algorithms, he rightly notes, 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: 10). They can consist of original formulations mashed together sourced from code libraries with stock algorithms that are re-used in multiple instances. They are embedded within complex socio-technical assemblages made up of a heterogeneous set of relations including potentially thousands of individuals, datasets, 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, but not necessarily its full logic, workings or choices made in its construction. While this complexity and embeddedness means that it is all but impossible to fully know an algorithmic system or provide straightforward, face-value evaluations (Chun 2011), it is nevertheless possible to understand some of the logics and principles and to critically engage with how such systems are created and work in practice (Bucher 2012).

(3) Ontogenetic and performative

As well as being heterogeneous and embedded, algorithms are rarely fixed in form and their work in practice unfolds in multifarious ways, reactive to input, interaction and situation. As such, algorithms need to be recognized as being ontogenetic and performative: that is, they are never fixed in nature, but are emergent and constantly unfolding. 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. Similarly, the socio-technical assemblages surrounding algorithms are never static, but are also are constantly unfolding. Examining one version of an algorithm will then provide a snapshot reading that fails to acknowledge or account for the mutable and often multiple nature of algorithms and their work (Bucher 2012).

(4) Out of control

Algorithms are often 'out of control' in the sense that their outcomes are sometimes not easily anticipated, producing unexpected outcomes in terms of their work in the world (Mackenzie 2005). Moreover, they might proliferate at a rate that is not easily accounted for by innovation diffusion models, spreading in use and how they are deployed in unexpected ways (Mackenzie 2005). What this means is that the outcomes for users inputting the same data might vary for contextual reasons (for example, Mahnke and Uprichard (2014) examined Google's autocomplete search algorithm by typing in the same terms from two locations and comparing the results, finding difference 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). Making sense then of 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, and the discussion concerning the need to think critically about algorithms, in this final section six ways to research algorithms are discussed. Each approach has its strengths and drawbacks, and are designed to shed light on the various concerns relating to algorithms, such as their nature and workings, their embedding in socio-technical systems, their effects and power, and dealing with and overcoming the difficulties of gaining access/black-boxing. They are not the only approaches that could be undertaken, but rather are a selection which seem to hold promise in advancing our critical understanding of algorithms. Moreover, their use is not mutually exclusive and there would be much to be gained by using two or more of the approaches in combination to compensate for the drawbacks of using them in isolation.

(1) Examining pseudo-code/source code

Perhaps the most obvious way to try and understand an algorithm is to examine its pseudo-code and/or its construction in code. In the first case, this would consist of examining how a task or puzzle is translated into a model or recipe. In the second, the focus would concentrate on the expression of the algorithm as 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 and Sedek 2008). In practice this means carefully sifting through documentation, code and programmer comments and 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 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 introduces. What this revealed was that soft and hard media make subtle differences to the outcomes of what was essentially the same, simple one line algorithm.

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 and Yoder 1997, cited in Chandra 2013: 126). Code has often been created by many individuals, it is complex, bodged and gerry-rigged. Those that 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 to possess 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 assemblage and its use.

(2) 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 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. The aim is to tease out the various practices, interactions and politics of creating algorithms and trace how they unfold in contingent and relational ways that often differ quite markedly to the process as set out in computer science texts and manuals. 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 how 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-a-vis how it actually does work in the world.

(3) 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: 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 program inherently each has two openings that enable lines of enquiry: input and output. By examining what data is 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 and 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, PR firms that seek to game an algorithm to optimize 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 propriety systems are aware that many people are seeking to determine and game their algorithm, and thus seek to identify and block bot users.

(4) 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, 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. Rosenberg was given full access to the company, spending time with senior management, in project meetings both internal and external to the company, 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.

(5) Unpacking the full socio-technical assemblage of algorithms

As already discussed, algorithms are not formulated or work in isolation, but form part of a wider technological stack that includes infrastructure/hardware, code platforms, data, and interfaces, and are framed and conditions by forms of knowledge, legalities, governmentalities, institutions, marketplace, finance, and so on (see Figure 1). 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 fetizishing 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, 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, behaviour of institutions; and documenting the biographies of key actors and the histories of projects (Montfort *et al*. 2012; Napoli 2013). 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.

(6) Examining how algorithms do work in the world

Algorithms deserve critical attention because they do active work in the world. It is important then 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: 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. 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, 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. Yet, despite their pervasiveness, utility, and the power vested in them to act in autonomous, automatic and automated ways that have profound effects on individuals and institutions, to date there has been very little critical attention paid to algorithms. Instead, the focus has been on how to create algorithms from a more technical perspective. This imbalance in how algorithms are thought about and intellectually engaged with is perhaps somewhat surprising, but as discussed algorithms are not straightforward to access, often being black-boxed, or to make sense of given their qualities of being complex, contingent, ontogenetic, embedded in wider socio-technical assemblages, and being 'out of control'. Nonetheless, it is clear that much more critical thinking and empirical research needs to be conducted with respect to algorithms and their work.

This paper has drawn on the handful of existing essays and empirical studies to provide a synoptic overview of critical thought with respect to algorithms and to advance an argument as to their nature, how to make sense of their construction and work, and how to research them in practice. It has contended that a broad approach is required that recognizes that algorithms are fragile accomplishments (Gillespie 2014a); the result of diverse set of actors, institutions, technologies, practices, structures, governmentalities and knowledges, and recognizes that their use and effects are situated and contextual. To provide empirical evidence for such thinking, six different approaches have been detailed, setting out a number of specific foci and suggesting methodologies which might be profitably employed and their pros and cons.

Given the limitations of each foci and approach, and the need for a more inclusive approach to analysis and assessment, they need to be used in conjunction with each other in order to provide both breadth and depth of understanding. Moreover, they are not the only foci or approaches that could be deployed and they could be complemented by others. For example, one could examine algorithms through an ethical lens (Kraemer *et al*., 2011), or with respect to their political economy and the reproduction of capitalism (Mager 2012), or from a legal or regulatory perspective, and other methods might be deployed such as ethnomethodology or surveys. They do, however, provide a useful starting set that hopefully others will add to as critical research and thinking on algorithms develops and matures.

Acknowledgements

The research for this paper was funded by a European Research Council Advanced Investigator award (ERC-2012-AdG-323636-SOFTCITY). Many thanks to Tracey Lauriault and Sung-Yueh Perng for comments on a draft of this paper.

References

Al-Akkad, A., Ramirez, L., Denef, S., Boden, A., Wood, L., Buscher, M. and 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*, ACM, New York, NY, pp. 457-466. http://dl.acm.org/citation.cfm?doid=2541016.2541051 (last accessed 16 October 2014)

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.

Barocas, S., Hood, S. and Ziewitz, M. (2013) Governing Algorithms: A Provocation Piece. *Available at SSRN 2245322*. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2245322 (last accessed 16 October 2014)

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*. Faber, London.

Chun, W.H.K, (2008) Programmability, in *Software Studies – a Lexicon*, ed. Matthew Fuller. Cambridge, MA: MIT Press, pp. 224-28

Chun, W.H.K. (2011) *Programmed Visions*. MIT Press, Cambridge, MA.

Cox, G. (2013) *Speaking Code: Coding as Aesthetic and Political Expression*. MIT Press, Cambridge, MA.

Diakopoulos, N. (2013) *Algorithmic Accountability Reporting: On the Investigation of Black Boxes*. A Tow/Knight Brief. Tow Center for Digital Journalism, Columbia Journalism School. http://towcenter.org/algorithmic-accountability-2/ (last accessed 21st August 2014)

Dodge, M. and 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) http://www.digitalhumanities.org/dhq/vol/7/1/000143/000143.html (last accessed 5 June 2014).

Foote, B. and Yoder, J. (1997) Big Ball of Mud. *Pattern Languages of Program Design* 4: 654-92

Fuller, M. (2008) Introduction, in *Software Studies – a Lexicon*, ed. Matthew Fuller. Cambridge, MA: MIT Press, pp. 1-14.

Galloway, A.R. (2006) *Gaming: Essays on Algorithmic Culture*. Minneapolis, MN: University of Minnesota Press.

Geiger, S.R. (2014) Bots, bespoke, code and the materiality of software platforms. *Information, Communication & Society* 17(3): 342-56.

Gillespie, T. (2011) Can an algorithm be wrong? Twitter Trends, the specter of censorship, and our faith in the algorithms around us. *Culture Digitally*, October 19th http://culturedigitally.org/2011/10/can-an-algorithm-be-wrong/ (last accessed 27 June 2014)

Gillespie, T. (2014a) The relevance of algorithms, in *Media Technologies: Essays on Communication, Materiality, and Society*, ed. by Gillespie, T., Boczkowski, P.J. and Foot, K.A. Cambridge, MA: MIT Press, pp.167-93.

Gillespie, T. (2014b). Algorithm [draft] [#digitalkeyword]. *Culture Digitally*, June 25th. http://culturedigitally.org/2014/06/algorithm-draft-digitalkeyword/ (last accessed 16 October 2014)

Goffey, A. (2008) Algorithm, in *Software Studies – a Lexicon*, ed. Matthew Fuller. Cambridge, MA: MIT Press, pp. 15-20.

Greenfield, A. (2006) *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders, Boston.

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

Kitchin, R. and Dodge, M. (2011) *Code/Space: Software and Everyday Life*. MIT Press, Cambridge, MA.

Kitchin, R., Gleeson, J. and Dodge, M. (2013) Unfolding mapping practices: A new epistemology for cartography. *Transactions of the Institute of British Geographers* 38(3): 480–496

Knuth, D. (1968) *The Art of Computer Programming: Volume 1 Fundamental Algorithms*. Addison-Wesley, Reading, MA.

Kowalski, R. (1979) Algorithm = Logic + Control. *Communications of the ACM* 22 (7): 424-36.

Kraemer, F., van Overveld, K. and Peterson, M. (2011) Is there an ethics of algorithms? *Ethics of Information Technology* 13: 251–260.

Krysa, J. and Sedek, G. (2008) Source code, in *Software Studies – a Lexicon*, ed. M. Fuller, Cambridge, MA: MIT Press, pp. 236-242

Kushner, S. (2013) The freelance translation machine: Algorithmic culture and the invisible industry. *New Media & Society* 15(8): 1241-58.

Lash, S. (2007) Power after hegemony: Cultural studies in mutation', *Theory, Culture & Society* 24(3): 55–78.

MacCormick, J. (2013) *Nine Algorithms That Changed the Future: The Ingenious Ideas That Drive Today's Computers*. Princeton University Press, Princeton, NJ.

Mackenzie, A. (2005) The performativity of code: Software and cultures of circulation. *Theory, Culture and 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 *24/7: Time and Temporality in the Network Society* eds., R. Hassan and R.E. Purser. Stanford, CA: Stanford University Press, 89–106.

Mackenzie, A. and Vurdubakis, T. (2011) Code and codings in Crisis: Signification, performativity and excess. *Theory, Culture and Society* 28(6): 3-23

MacKenzie, D. (2008) *An Engine, Not a Camera. How Financial Models Shape Markets*. MIT Press, Cambridge, MA.

Mager, A. (2012) Algorithmic ideology: How capitalist society shapes search engines. *Information, Communication, and Society* 15 (5): 769-787.

Mahnke, M and Uprichard, E. (2014) Algorithming the algorithm. In *Society of the Query Reader: Reflections on Web Search,* eds R. König and M. Rasch. Amsterdam: Institute of Network Cultures, pp. 256-270.

Manovich, L. (2013) *Software Takes Control*. Bloomsbury, New York.

Miyazaki, S. (2012) Algorhythmics: Understanding micro-temporality in computational cultures. *Computational Culture* http://computationalculture.net/article/algorhythmics-understanding-micro-temporality-in-computational-cultures (last accessed 25 June 2014)

Montfort, N., Baudoin, P., Bell, J., Bogost, I., Douglass, J., Marino, M.C., Mateas, M., Reas, C., Sample, M. and Vawter, N. (2012) *10 PRINT CHR$ (205.5 + RND (1)); : GOTO 10*. MIT Press, Cambridge, MA.

Musiani, F. (2013) Governance by algorithms. *Internet Policy Review*, 2(3) http://policyreview.info/articles/analysis/governance-algorithms) (last accessed 7th October 2014)

Napoli, P.M. (2013) *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, May, 2013. ssrn.com/abstract=2260923

Neyland, D. (2014) On organizing algorithms. *Theory, Culture and Society*, online first.

Parsons, W. (2004) Not just steering but weaving: relevant knowledge and the craft of building policy capacity and coherence. *Australian Journal of Public Administration* 63 (1): 43–57.

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.* http://governingalgorithms.org/wp-content/uploads/2013/05/2-paper-pasquale.pdf (last accessed 16 October 2014)

Porter, T.M. (1995) *Trust in Numbers: The Pursuit of Objectivity in Science and Public Life.* Princeton University Press, Princeton, NJ.

Seaver, N. (2013) Knowing Algorithms, in *Media in Transition 8*, Cambridge, MA. http://nickseaver.net/papers/seaverMiT8.pdf (last accessed 21st August 2014)

Shirky, C. (2009) A speculative post on the idea of algorithmic authority. Shirky.com http://www.shirky.com/weblog/2009/11/a-speculative-post-on-the-idea-of-algorithmic-authority/ (last accessed 7th October 2014)

Steiner, C. (2012) *Automate This: How Algorithms Took Over Our Markets, Our Jobs, and the World*. Portfolio, New York.

Takhteyev, Y. (2012) *Coding Places: Software Practice in a South American City*. MIT Press, Cambridge, MA.

Ullman, E. (1997) *Close to the Machine*. City Lights Books, San Francisco.

Ziewitz, M. (2011) *How to think about an algorithm? Notes from a not quite random walk.* Discussion paper for Symposium on 'Knowledge Machines between Freedom and Control', September 29th. http://ziewitz.org/papers/ziewitz_algorithm.pdf (last accessed 21st August 2014)