peer-reviewed pre-print for the 2014 Johns Hopkins Guide to Digital Media, Ryan et al, eds.

“Algorithm”

Bethany Nowviskie

The term *algorithm*, most commonly associated with computer science, may be used for any effective procedure that reduces the solution of a problem to a predetermined sequence of actions. In software, algorithms are used for performing calculations, conducting automated reasoning, and processing data (including digital texts) – but algorithms may also be implemented in mathematical models, mechanical devices, biological networks, electrical circuitry, and in practices resulting in generative or procedural art (see CODE, COMPUTATIONAL LINGUISTICS, PROCEDURAL).

In common usage, *algorithm* typically references a deterministic algorithm, formally defined as a finite and generalizable sequence of instructions, rules, or linear steps designed to guarantee that the agent performing the sequence will reach a particular, pre-defined goal or establish incontrovertibly that the goal is unreachable. The “guarantee” part of this description is important, as it differentiates algorithms from heuristics, which commonly proceed by “rules of thumb.” Like algorithms, heuristic methods can be used iteratively to reach a desired end-state and may be responsive to feedback from external sources. However, the heuristic process is fundamentally one of informal trial and error rather than of constrained, formally algorithmic activity according to a set of pre-defined rules. (Non-deterministic algorithms are a class of algorithm that attempts to solve harder problems by finding the best solution available with a given set of constraints. They do not guarantee to find a single, best solution and may, on repetition, present radically different outcomes.)

Almost any everyday problem can be solved heuristically or algorithmically. We proceed by heuristics when, for example, we’ve lost our car keys: *I look in my bag. I look in my* *bag again. I search my jacket pockets. I check the front door, because I left them dangling there last week.* The weak point of the heuristic method becomes evident when its user needs to shift gears. I'm not finding my keys in the usual places. Should *I peer with trepidation into the locked car or check the washing machine*? *Is it possible someone has taken them? Should I keep looking, or is it time to give up and call a cab?* In formal, graph-based problem-solving, heuristics are sometimes used to guide the search for solutions by identifying the most promising branches of a tree for further exploration, or by cutting out unpromising branches altogether (See GRAPH THEORY). The basic "problem with heuristics" – which could lead to the inadvertent elimination of the entire branch of a desired outcome branch from the search tree – "is how to decide half-way what would be an appropriate next action, i.e. how to design heuristic rules that lead to good solutions instead of bad ones" (Lagus). Tellingly, we often attribute decisions in successful heuristic processes to intuition and those that result in undesirable outcomes to confusion and bad luck.

If heuristics fail or prove too unsystematic for comfort, we can shift to algorithmic problem-solving (expressed here in pseudocode):

```
For each room in the house,
and for each item in the room;
        pick up and examine the item.
            If the item appears by objective criteria to be the missing object,
                    terminate the search.
            If not, put down the item and continue this loop
                    until all items in all rooms have been tested.
```

Eventually, if this so-called "brute force" algorithm is executed perfectly, we will either find our keys or determine conclusively that they are not in the house. It requires no ingenuity on the part of the performer, and there's a kind of predestination or special providence embedded in the process. That is to say, we know to expect one of two prescribed outcomes before even undertaking the search. And – as its strict definition requires – this algorithm is almost wholly generalizable. If you suspect you have left your keys at a friend's house, you can run the process there. If the misplaced object is a watch, or a hat, these steps are equally applicable. (Admittedly, it isn't a very efficient algorithm, because it requires us, for example, to pick up and examine heavy furnishings, and to re-examine the housecat every time it saunters into a new room, but more elegant versions could be designed.)

At present, advancements in programming frameworks and computing hardware (specifically, the increasing speed and power of processors) are driving new work in concurrency and parallelization of algorithms. This is an active, and even pressing, research area in computer science, with implications for machine learning, a subfield of artificial intelligence (Bekkerman et al; see ARTIFICIAL INTELLIGENCE). But the development and theorization of algorithms has a long history. The word itself stems from the name of 9th-century mathematician Abū Jaʿfar Muhammad ibn Mūsa, al-Ḵwārizmī, and was first applied (as *algorism*) to any arithmetical operation using Arabic numerals, before shifting in meaning to its current sense in the late 19th and early 20th century. The works of Llull, Leibnitz, Babbage, Lovelace, and Boole (among others) are infused with procedural and proto-algorithmic notions of language, logic, and calculation, and raise many of the same questions about the applicability of algorithm to interpretation that underlie present-day concerns about hermeneutics in digital media and the digital humanities (Nowviskie). Jean-Luc Chabert and Évelyne Barbin offer in-depth discussion of the many ways "algorithms were in use long before it was necessary to give a clear definition" of the term, and describe how "problems concerning the foundations of mathematical logic" prompted clarification of the concept. This work crystallized largely outside the realm of literature and linguistics in the 1930s, as thinkers like Kurt Gödel, Alonzo Church, Stephen Kleene, Alan Turing, and Emil Post responded to formalist mathematical questions posed by David Hilbert's *Entscheidungsproblem* and worked to demonstrate the inability of algorithmic processes to establish a truth-value for certain classes of mathematical statements. Chabert and Barbin note with some irony that "the intuitive idea of an algorithm played an important heuristic role in all those works" which challenged Hilbert's mathematical formalism (455-458). Donald Knuth's 1968 *Art of Computer Programming* helped to codify five widely-accepted properties of algorithms: that they are finite in length, definite or unambiguous, have zero or more inputs and one or more outputs, and are composed of "effective" steps, sufficiently basic as to be executable. Knuth also asserted, "in some loosely-defined aesthetic sense," the desirability of "good" algorithms, citing criteria of efficiency, "adaptability of the algorithm to computers," simplicity, and elegance (2-9). All of these refinements to the concept of the algorithm are relevant both to artistic production and to the interpretation of texts and artifacts, two domains generally predicated on ambiguity, subjectivity, and flux (see COMBINATORY AND AUTOMATIC TEXT GENERATION, DIGITAL HUMANITIES).

As presently understood, algorithms are expected to be both perfectly precise and entirely implementable. An old bubblegum wrapper joke helps to make this point: How do you fit four elephants into a Volkswagen? The algorithmic answer is that you put two in the front seat and two in the back. Although those steps are clearly unambiguous, they are impossible to implement. In contrast is an algorithm for writing encyclopedia articles: *Step 1. Write a paragraph. Step 2: Repeat Step 1 until the article is complete.* The procedure is clearly implementable – it was performed more than 150 times in the present volume – but it is far too ambiguous to be a "textbook," or even a useful, algorithm. What constitutes a paragraph? What criteria indicate completion? How does the algorithm know that you're writing an article and not a monograph, a novel, or a comic book? And how might a human agent's interpretation and performance of an algorithmic process alter it? That is to say, with what assumptions do we approach algorithmic or procedural activities and how might those assumptions both shape and be shaped by action within systems of constraint? (See WRITING UNDER CONSTRAINT.) In other words, algorithmic methods are productive not only of new texts, but of new readings. In "Algorithmic Criticism," Stephen Ramsay argues that "critical reading practices already contain elements of the algorithmic" and that algorithms "can be made to conform to the methodological project of *inventio* without transforming the nature of computation or limiting the rhetorical range of critical inquiry" (2008: 489). It is also important to acknowledge that even the most clinically perfect and formally unambiguous algorithms embed their designers' theoretical stances toward problems, conditions, and solutions.

Repositioning closed, mechanical or computational operations as participatory or playful algorithms requires acknowledgement of a primary definition, derived from the studies of game theorist Martin Shubik (See GAME THEORY). Shubik concludes a survey of "the scope of gaming" with the simple statement that "all games call for an explicit consideration of the role of the rules." He understands this "consideration" not only as adherence by players to a set of constraints, but also as appreciation of the impact of rules on the whole scope of play. The ruleset or constraining algorithm in any ludic or hermeneutic system becomes another participant in the process and, in the course of execution or play, can seem to open itself to interpretation and subjective response – in some cases, to real, iterative or turn-based modification (Suber). In considering the "role of the rules" we must follow C. S. Peirce, and interpret algorithmic specifications “in the sense in which we speak of the ‘rules’ of algebra; that is, as a permission under strictly defined conditions” (4.361). The permission granted here is not only to perform but also to reimagine and reconfigure.

SEE ALSO: randomness; sampling; search; Turing test.

References and further reading

Bekkerman, Ron, Mikhail Bilenko, and John Langford, eds. 2011. *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press.

Chabert, Jean-Luc and Évelyne Barbin. 1999. *A History of Algorithms: from the Pebble to the Microchip.* Springer Verlag.

Knuth, Donald E. 1968. *The Art of Computer Programming*. Volume 1. Boston: Addison-Wesley Professional.

Lagus, Krista. 1995. *Automated Pagination of the Generalized Newspaper Using Simulated Annealing*. Master’s thesis, Helsinki University of Technology.

Peirce, C.S. 1933. *Collected Papers*. Cambridge, MA: Harvard UP.

Nowviskie, Bethany. 2013. “Ludic Algorithms.” in *PastPlay: History, Technology, and the Return to Playfulness*. Kevin Kee, ed. University of British Columbia Press.

Ramsay, Stephen. 2008. “Algorithmic Criticism.” In *A Companion to Digital Literary Studies*, Susan Schreibman and Ray Siemens, eds. Oxford: Blackwell. 478 – 491.

Shubik, Martin. 1972. "On the Scope of Gaming," *Management Science* 18 (5): 34.

Suber, Peter. 2003. "Nomic: A Game of Self-Amendment". Earlham College. Available: http://www.earlham.edu/~peters/nomic.htm