Formal Characterisations of *Algorithm*

Declan Thompson
Stanford University

IHPST, 27 February 2020

Talk Overview

1. Computability theory
2. Difficulties in accounting for algorithms
3. Sketch of a new approach

Algorithm as ‘effective procedure’

- Study of algorithms didn’t start in earnest until the 20th century
- The primary methodology has involved analysing ‘algorithm’ as ‘effective procedure’

Algorithm as 'effective procedure'

- Study of algorithms didn't start in earnest until the 20th century
- The primary methodology has involved analysing 'algorithm' as 'effective procedure'
  1. Isolate a set of primitive basic operations
  2. Specify simple (deterministic!) ways to combine these operations
  3. We get a class $\mathfrak{M}$ of machines/programs/function specifications

Algorithm as 'effective procedure'

- Study of algorithms didn't start in earnest until the 20th century
- The primary methodology has involved analysing 'algorithm' as 'effective procedure'
  1. Isolate a set of primitive basic operations
  2. Specify simple (deterministic!) ways to combine these operations
  3. We get a class $\mathfrak{M}$ of machines/programs/function specifications

$$\begin{aligned} A \text{ is computable} \quad &\Leftrightarrow \quad \text{some effective procedure decides } A \\ &\Leftrightarrow \quad \text{some } M \in \mathfrak{M} \text{ decides } A \end{aligned}$$

Algorithm as 'effective procedure'

- Study of algorithms didn't start in earnest until the 20th century
- The primary methodology has involved analysing 'algorithm' as 'effective procedure'
  1. Isolate a set of primitive basic operations
  2. Specify simple (deterministic!) ways to combine these operations
  3. We get a class $\mathfrak{M}$ of machines/programs/function specifications

$$\begin{aligned} A \text{ is computable} \quad &\Leftrightarrow \quad \text{some effective procedure decides } A \\ &\Leftrightarrow \quad \text{some } M \in \mathfrak{M} \text{ decides } A \end{aligned}$$

- This approach has been very effective!

Computability theory doesn't deal with algorithms

Many of the given definitions [of algorithm] are of the form 'An algorithm is a program in this language/system/machine'. This does not really conform to the current usage of the word 'algorithm'. Rather, this is more in tune with the modern use of the word 'program'. They all have a feel of being a specific implementation of an algorithm on a specific system. (Yanofsky 2011, pp. 253–4)

Computability theory doesn't deal with algorithms

> Many of the given definitions [of algorithm] are of the form 'An algorithm is a program in this language/system/machine'. This does not really conform to the current usage of the word 'algorithm'. Rather, this is more in tune with the modern use of the word 'program'. They all have a feel of being a specific implementation of an algorithm on a specific system. (Yanofsky 2011, pp. 253–4)

- The implementation/algorithm distinction is completely irrelevant for computability theory
  - The distinction is also irrelevant for complexity theory
- Both these disciplines deal with the existence of effective/efficient solutions to *problems*
- Formally, we can reduce talk of algorithms in computability/complexity theory to talk of programs

Modern computer science practice marks an intuitive distinction between algorithms and implementations.

- Can we make this precise?
- What claims do we make about *algorithms*?

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

- These claims suggest adherence to *algorithmic realism* (Dean 2016)
- How do we formalise these claims?

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

- These claims suggest adherence to *algorithmic realism* (Dean 2016)
- How do we formalise these claims?
  - Direct Algorithmic Realism

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

- These claims suggest adherence to *algorithmic realism* (Dean 2016)
- How do we formalise these claims?
  - Direct Algorithmic Realism
  - Strong Church's Thesis

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

- These claims suggest adherence to *algorithmic realism* (Dean 2016)
- How do we formalise these claims?
  - Direct Algorithmic Realism
  - Strong Church's Thesis
  - Algorithms-as-Abstracts

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

Goal of project

Find a mathematical object which can adequately play the role of *algorithm*, as it's used in statements like the above.

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

Goal of project

Find a mathematical object which can adequately play the role of *algorithm*, as it's used in statements like the above.

- Standard philosophy of mathematics questions: out of scope

Prim's algorithm

Prim's algorithm (Khoussainov and Khoussainova 2012, p. 172)

$Prim(G, v)$:

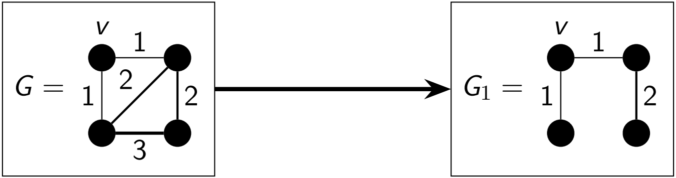
1. Intialize $V_1 = \{v\}$, $E_1 = \emptyset$, and set $G_1 = (V_1, E_1)$.
2. *While* there is an edge that connects a vertex in V_1 to a vertex not in V_1 *do*
   a. Find an edge $e = \{u, v'\}$ with smallest weight $w(e)$ such that $u \in V_1$ and $v' \notin V_1$.
   b. Set $V_1 = V_1 \cup \{v'\}$, $E_1 = E_1 \cup \{e\}$, and $G_1 = (V_1, E_1)$.
3. Output $G_1 = (V_1, E_1)$.

Prim's algorithm

Prim's algorithm (Khoussainov and Khoussainova 2012, p. 172)

$Prim(G, v)$:

1. Intialize $V_1 = \{v\}$, $E_1 = \emptyset$, and set $G_1 = (V_1, E_1)$.
2. *While* there is an edge that connects a vertex in V_1 to a vertex not in V_1 *do*
   a. Find an edge $e = \{u, v'\}$ with smallest weight $w(e)$ such that $u \in V_1$ and $v' \notin V_1$.
   b. Set $V_1 = V_1 \cup \{v'\}$, $E_1 = E_1 \cup \{e\}$, and $G_1 = (V_1, E_1)$.
3. Output $G_1 = (V_1, E_1)$.



Prim's algorithm

Prim's algorithm (Khoussainov and Khoussainova 2012, p. 172)

$Prim(G, v)$:

1. Intialize $V_1 = \{v\}$, $E_1 = \emptyset$, and set $G_1 = (V_1, E_1)$.
2. *While* there is an edge that connects a vertex in V_1 to a vertex not in V_1 *do*
    a. Find an edge $e = \{u, v'\}$ with smallest weight $w(e)$ such that $u \in V_1$ and $v' \notin V_1$.
    b. Set $V_1 = V_1 \cup \{v'\}$, $E_1 = E_1 \cup \{e\}$, and $G_1 = (V_1, E_1)$.
3. Output $G_1 = (V_1, E_1)$.

Testing Strong Church's Thesis

Fix a reasonable model of computation, and let $\mathfrak{M}$ be the class of machines under that model. An *algorithm* is a machine $M \in \mathfrak{M}$.

Prim's algorithm

Prim's algorithm (Khoussainov and Khoussainova 2012, p. 172)

$Prim(G, v)$:

1. Intialize $V_1 = \{v\}$, $E_1 = \emptyset$, and set $G_1 = (V_1, E_1)$.
2. *While* there is an edge that connects a vertex in V_1 to a vertex not in V_1 *do*
   a. Find an edge $e = \{u, v'\}$ with smallest weight $w(e)$ such that $u \in V_1$ and $v' \notin V_1$.
   b. Set $V_1 = V_1 \cup \{v'\}$, $E_1 = E_1 \cup \{e\}$, and $G_1 = (V_1, E_1)$.
3. Output $G_1 = (V_1, E_1)$.

- Which $M \in \mathfrak{M}$ is Prim's algorithm? Many issues

Prim's algorithm

Prim's algorithm (Khoussainov and Khoussainova 2012, p. 172)

Prim(G, v):

1. Intialize $V_1 = \{v\}$, $E_1 = \emptyset$, and set $G_1 = (V_1, E_1)$.
2. *While* there is an edge that connects a vertex in V_1 to a vertex not in V_1 *do*
   a. Find an edge $e = \{u, v'\}$ with smallest weight $w(e)$ such that $u \in V_1$ and $v' \notin V_1$.
   b. Set $V_1 = V_1 \cup \{v'\}$, $E_1 = E_1 \cup \{e\}$, and $G_1 = (V_1, E_1)$.
3. Output $G_1 = (V_1, E_1)$.

- Which $M \in \mathfrak{M}$ is Prim's algorithm? Many issues
- Step 2a is under-determined

Prim's algorithm

Prim's algorithm (Khoussainov and Khoussainova 2012, p. 172)

Prim(G, v):

1. Intialize $V_1 = \{v\}$, $E_1 = \emptyset$, and set $G_1 = (V_1, E_1)$.
2. *While* there is an edge that connects a vertex in V_1 to a vertex not in V_1 *do*
   a. Find an edge $e = \{u, v'\}$ with smallest weight $w(e)$ such that $u \in V_1$ and $v' \notin V_1$.
   b. Set $V_1 = V_1 \cup \{v'\}$, $E_1 = E_1 \cup \{e\}$, and $G_1 = (V_1, E_1)$.
3. Output $G_1 = (V_1, E_1)$.

- Which $M \in \mathfrak{M}$ is Prim's algorithm? Many issues
- Step 2a is under-determined
  - Standard models of computation force us to choose a tie-breaker

Algorithms-as-Abstracts

- Single machines are too concrete.
- Take an equivalence relation on $\mathfrak{M}$ - algorithms are equivalence classes
- See Dean (2007, 2016) for details on why this is dubious

Out of options?

Goal of project

Find a mathematical object which can adequately play the role of *algorithm*, as it's used in modern computer science practice.

- ~~Direct Algorithmic Realism~~
- ~~Strong Church's Thesis~~
- ~~Algorithms-as-Abstracts~~

Out of options?

Goal of project

Find a mathematical object which can adequately play the role of *algorithm*, as it's used in modern computer science practice.

- ~~Direct Algorithmic Realism~~
- ~~Strong Church's Thesis~~
- ~~Algorithms-as-Abstracts~~

Two questions:

1. Computability theory and complexity theory don't need algorithms. Why are we assuming algorithms need them?
2. What do algorithms actually do?

Reconsidering effectiveness

Effectiveness has a strong case in computability theory. What about *algorithmic* claims?

Reconsidering effectiveness

Effectiveness has a strong case in computability theory. What about *algorithmic* claims?

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

Reconsidering effectiveness

Effectiveness has a strong case in computability theory. What about *algorithmic* claims?

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

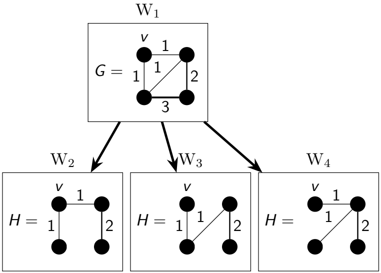
- Let's sequester effectiveness to a requirement on *implementations*
- This matches practice - we can describe uncomputable algorithms!

What do algorithms do?

- Algorithms, whenever presented, are presented as (potential) solutions to a given *task*

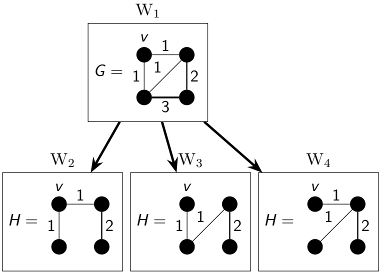
What do algorithms do?

- Algorithms, whenever presented, are presented as (potential) solutions to a given *task*
- *Tasks are not functions*



What do algorithms do?

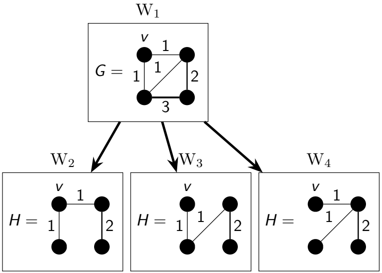
- Algorithms, whenever presented, are presented as (potential) solutions to a given *task*
- *Tasks are not functions*



We need an account of tasks.

What do algorithms do?

- Algorithms, whenever presented, are presented as (potential) solutions to a given *task*
- *Tasks are not functions*



We need an account of tasks.

- An algorithm gives you a method for achieving a task
  - Essentially, an algorithm breaks a task down into sub-tasks, or *basic operations*

New questions

New questions

1. Why take algorithms' basic operations to be an antecedently fixed set of functions?

New questions

1. Why take algorithms' basic operations to be an antecedently fixed set of functions?
   - Yiannis Moschovakis (2001) and Yuri Gurevich (2000), among others, suggest the basic operations of an algorithm should be arbitrary functions
   - We can represent an algorithm "on its natural level of abstraction"

New questions

1. Why take algorithms' basic operations to be an antecedently fixed set of functions?
   - Yiannis Moschovakis (2001) and Yuri Gurevich (2000), among others, suggest the basic operations of an algorithm should be arbitrary functions
   - We can represent an algorithm "on its natural level of abstraction"
2. Why take algorithms' basic operations to be functions at all?
   - The tasks algorithms solve are not functions
   - If we renounce effectiveness, why stick with functions?

Key observation

The natural basic steps of an algorithm are themselves tasks

The natural basic steps of an algorithm are tasks

Prim's algorithm (Khoussainov and Khoussainova 2012, p. 172)

$Prim(G, v)$:

1. Intialize $V_1 = \{v\}$, $E_1 = \emptyset$, and set $G_1 = (V_1, E_1)$.
2. *While* there is an edge that connects a vertex in V_1 to a vertex not in V_1 *do*
   a. Find an edge $e = \{u, v'\}$ with smallest weight $w(e)$ such that $u \in V_1$ and $v' \notin V_1$.
   b. Set $V_1 = V_1 \cup \{v'\}$, $E_1 = E_1 \cup \{e\}$, and $G_1 = (V_1, E_1)$.
3. Output $G_1 = (V_1, E_1)$.

The natural basic steps of an algorithm are tasks

Prim's algorithm (Khoussainov and Khoussainova 2012, p. 172)

$Prim(G, v)$:

1. Intialize $V_1 = \{v\}$, $E_1 = \emptyset$, and set $G_1 = (V_1, E_1)$.
2. *While* there is an edge that connects a vertex in V_1 to a vertex not in V_1 *do*
    a. Find an edge $e = \{u, v'\}$ with smallest weight $w(e)$ such that $u \in V_1$ and $v' \notin V_1$.
    b. Set $V_1 = V_1 \cup \{v'\}$, $E_1 = E_1 \cup \{e\}$, and $G_1 = (V_1, E_1)$.
3. Output $G_1 = (V_1, E_1)$.

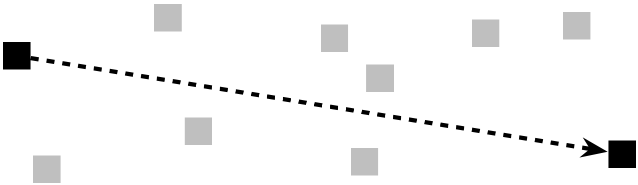
Treat tasks and basic operations as the same type of mathematical object:

1. We need an account of a task
2. We need to explain how subtasks can be combined into an algorithm
3. We need to explain how the resulting notion of algorithm relates to implementations

Tasks

A task is a specification of desired behaviour.

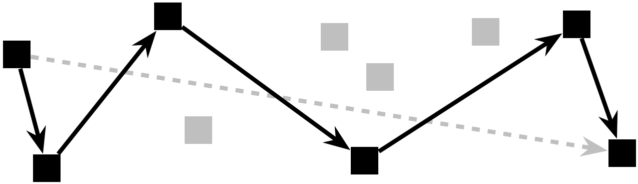
- Formally, a task is a set of sequences of first order models
- MST is a set of pairs of first order models
- Enumerate Primes is a set of infinite sequences of first order models, each corresponding to a different enumeration of primes



Algorithms

An algorithm is a specification of how to solve a task, assuming the ability to solve other tasks (i.e. sub-tasks/basic operations)

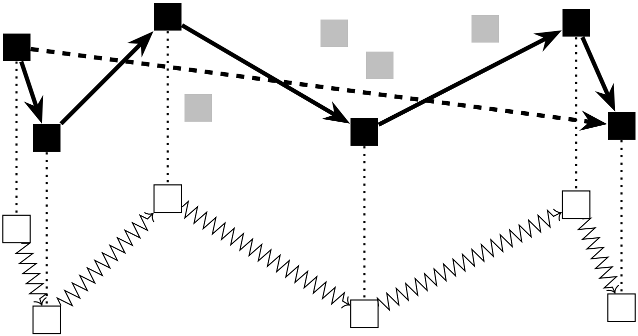
- An algorithm specifies which sub-tasks to carry out, in what order(s)
- We take an algorithm as the set of its own traces
- Formally, an algorithm is a set of sequences of first order models



Implementations

An implementation is a simulation of the algorithm on one of the standard machine models

- Each execution trace $s_1, s_2, \dots s_n$ of M can be split into sub-traces corresponding to sub-tasks of the algorithm
- The sub-traces in each execution trace match the sequence of basic operations in some sequence of the algorithm



Analysis

Claims about algorithms

1. Program X implements Prim's algorithm.
2. MergeSort and QuickSort are different sorting algorithms.
3. The Euclidean algorithm is correct for finding the greatest common divisor of two positive integers.
4. BubbleSort runs in $O(n^2)$ time.
5. Shor's algorithm is a quantum algorithm.

Summary

- Computability theory and complexity theory have no (formal) use for algorithms
- Effectiveness is a desideratum more properly suited to implementations, rather than algorithms
- If we renounce the drive for effectiveness, we can avoid problems facing many extant accounts of *algorithm*, while still maintaining compatibility with computability theory

Thank you

Declan Thompson

declan@stanford.edu
www.stanford.edu/~declan

- Walter Dean. "What Algorithms Could Not Be". Rutgers University - Graduate School - New Brunswick, 2007.
- Walter Dean. "Algorithms and the Mathematical Foundations of Computer Science". In: *Gödel's Disjunction: The Scope and Limits of Mathematical Knowledge*. First edition. Oxford, United Kingdom: Oxford University Press, 11 Aug. 2016, pp. 19–66. ISBN: 978-0-19-182037-3.
- Yuri Gurevich. "Sequential Abstract-State Machines Capture Sequential Algorithms". In: *ACM Transactions on Computational Logic* 1.1 (1 July 2000), pp. 77–111.
- Bakhadyr Khoussainov and Nodira Khoussainova. *Lectures on Discrete Mathematics for Computer Science*. Algebra and Discrete Mathematics v. 3. OCLC: ocn697260707. New Jersey: World Scientific, 2012. 346 pp. ISBN: 978-981-4340-50-2.

- Yiannis N Moschovakis. "What Is an Algorithm?" In: *Mathematics Unlimited — 2001 and Beyond*. Ed. by Björn Engquist and Wilfried Schmid. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 919–936. ISBN: 978-3-642-56478-9.
- N. S. Yanofsky. "Towards a Definition of an Algorithm". In: *Journal of Logic and Computation* 21.2 (1 Apr. 2011), pp. 253–286.