Computational Models for Learning to Code

Ara Vartanian, Xiaojin Zhu

March 3, 2017

Overview

Preliminaries

Please interrupt me.

Let's begin with the most perilous part of any talk

https://aravart.github.io/speech-games/

Live Demo

(Google Chrome only, probably)

Behavior

A simple language

Consider a simple language that can be used to control a “turtle” program. We have three instructions:

A simple language

Consider a simple language that can be used to control a "turtle" program. We have three instructions:

- `Move();`

A simple language

Consider a simple language that can be used to control a “turtle” program. We have three instructions:

- `Move();`
- `TurnLeft();`

A simple language

Consider a simple language that can be used to control a “turtle” program. We have three instructions:

- `Move();`
- `TurnLeft();`
- `TurnRight();`

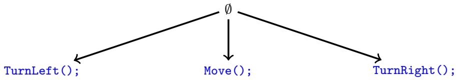
Program graph

Imagine the student traversing a *state space* of the possible programs in an editor.

$\emptyset$

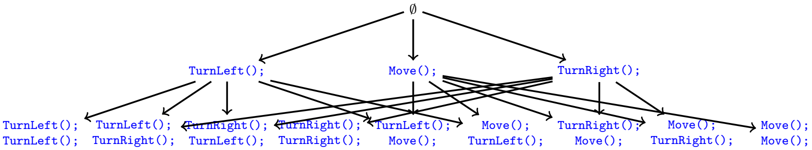
Program graph

Imagine the student traversing a *state space* of the possible programs in an editor.



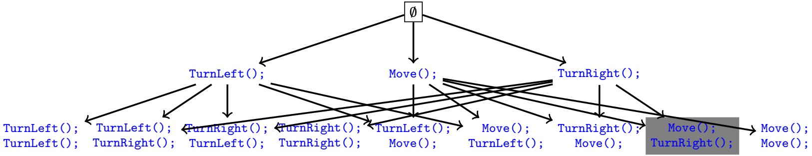
Program graph

Imagine the student traversing a *state space* of the possible programs in an editor.



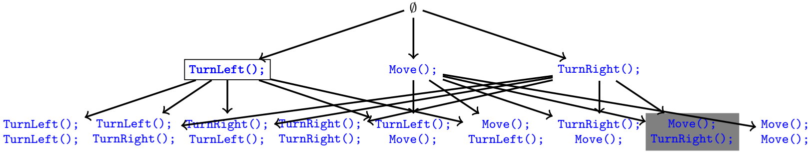
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



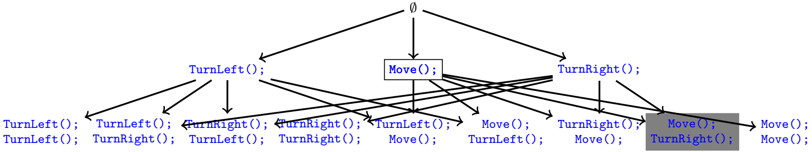
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



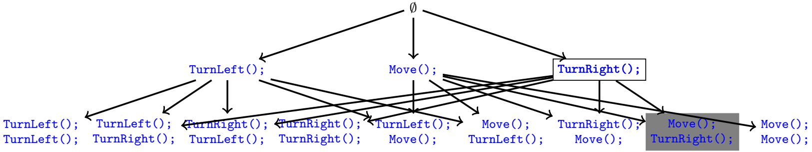
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



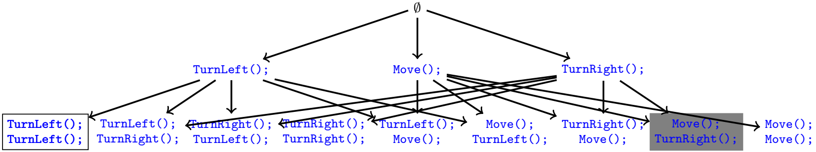
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



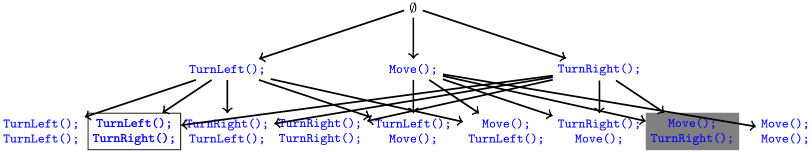
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



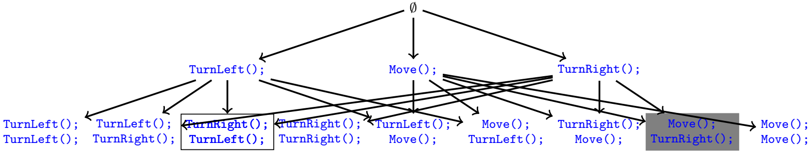
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



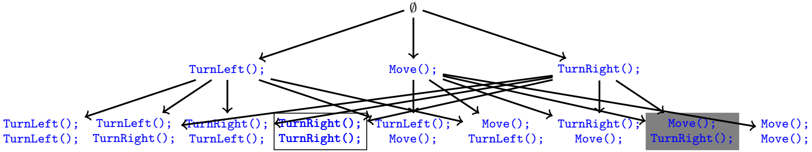
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



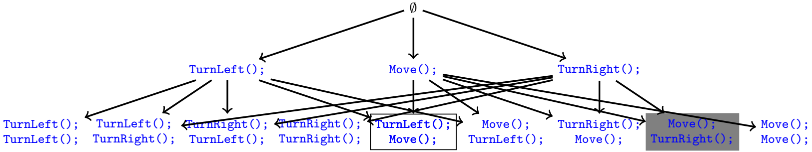
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



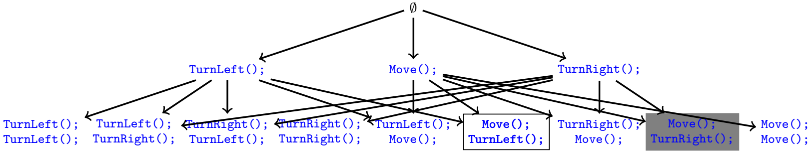
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



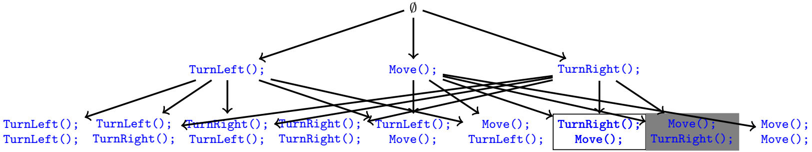
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



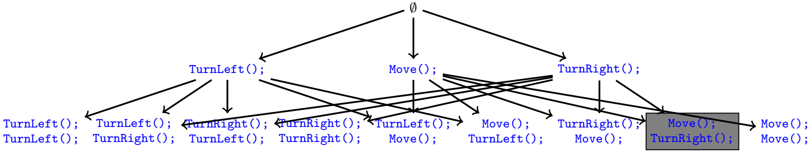
Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



Programming as graph search

Given a goal node v and a graph G, we can model the task of programming as a search over G for v. Breadth-first search would look like:



Now imagine a perfect programmer...

A perfect programmer performs the operation above flawlessly, terminating after finding the goal node (with probability 1).

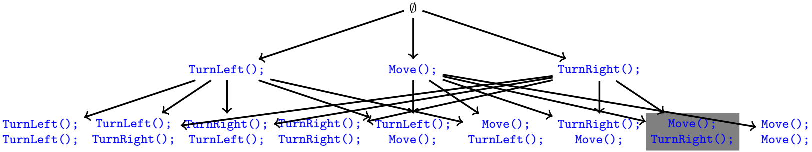
Now imagine a buggy programmer...

Now imagine a buggy programmer...

Some of the edges might be missing.

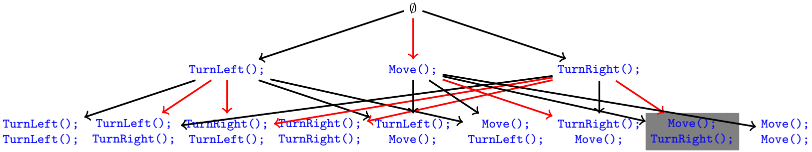
Now imagine a buggy programmer...

Some of the edges might be missing.



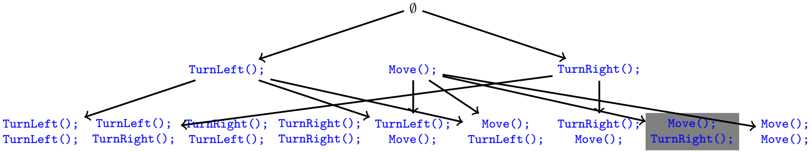
Now imagine a buggy programmer...

Some of the edges might be missing.



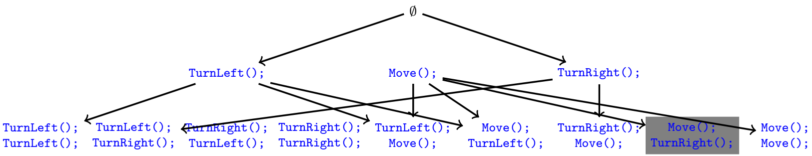
Now imagine a buggy programmer...

Some of the edges might be missing.



Now imagine a buggy programmer...

Some of the edges might be missing.



And there is no path to the goal. Search comes up empty.

Intuition

If we knock out each edge with some probability p, then for any goal node v we have some (hard to compute) probability that v is reachable from the root of G. Let's call this probability of r_v.

Two ways to think about this

Here's plain old breadth-first search...

Algorithm 1 Breadth-First Search

procedure BFS
Input: Program graph G, source node u, goal node v.
Output: Whether v was found.
    Create a queue Q; Enqueue u onto Q
    Create a set V; Add u to V
    while Q is not empty **do**
        Dequeue an item from Q into n
        If n is u **then return** True
        for each edge e incident on n **do**
            Let m be the other end of e
            if m not in V **then**
                Add m to V
                Enqueue m onto Q
    return False

Two ways to think about this

Here's breadth-first search with-forgetting...

Algorithm 2 Breadth-First Search With Forgetting

procedure BFSWithForgetting
Input: Program graph G, source node u, goal node v, forgetting probability p.
Output: Whether v was found.

```
    Create a queue Q; Enqueue u onto Q
    Create a set V; Add u to V
    while Q is not empty do
        Dequeue an item from Q into n
        If n is u then return True
        for each edge e incident on n do
            Let m be the other end of e
            if m not in V then
                Add m to V
                With probability 1 − p, enqueue m onto Q
    return False
```

Two ways to think about this

Alternatively, we can remove the edges from G and then pass this modified G into plain old breadth-first search.

Estimating the probability of finding v

Algorithm 3 Monte Carlo Estimation of r_v

```
procedure MonteCarloR
Input: Program graph G, goal node v, forgetting probability p, budget b.
Output: Estimate r̂_v.
    Set r to 0
    for i in 1 to b do
        Copy G into G'
        for each edge e in G' do
            Remove e from G' with probability p
        Call Procedure: BFS(G',∅,v) [Algorithm 1]
        Increment r if BFS found v
    Return: r/b
```

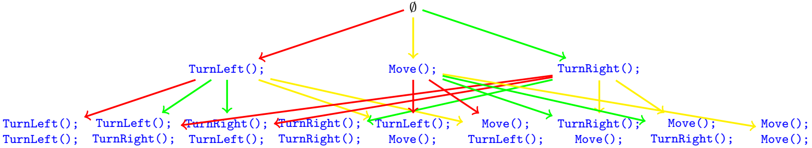
A tangent: Plato was a computer scientist?

A tangent: Plato was a computer scientist?

Meno: And how are you going to search for [the nature of virtue] when you don't know at all what it is, Socrates? Which of all the things you don't know will you set up as target for your search? And even if you actually come across it, how will you know that it is that thing which you don't know?

Making things more interesting: edge types

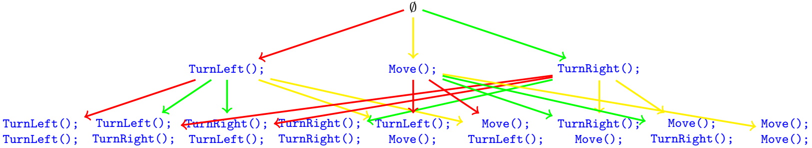
In our story so far, we have one probability p for forgetting an edge, but different operations (edges) might be more difficult to follow:



Here we've colored the edges based on which instruction *type* was inserted.

Making things more interesting: edge types

In our story so far, we have one probability p for forgetting an edge, but different operations (edges) might be more difficult to follow:

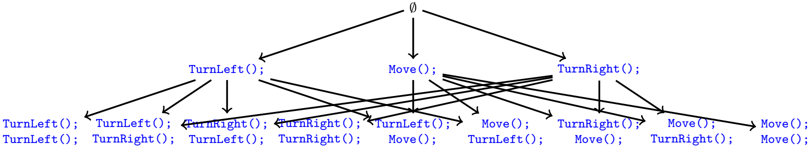


Here we've colored the edges based on which instruction *type* was inserted. More generally:

$$p : \mathit{SourceFeatures} \times \mathit{InsertionFeatures} \times \mathit{InsertionPosition} \rightarrow [0, 1]$$

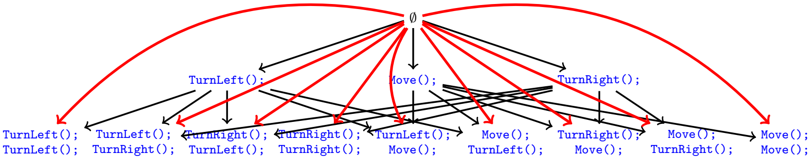
Modularity

How can we express the idea that the student learns reusable fragments or idioms?



Modularity

How can we express the idea that the student learns reusable fragments or idioms?



Think of these as composite edges.

Modularity

Just where do we add edges? A couple of ideas.

Modularity

Just where do we add edges? A couple of ideas.

- The most promiscuous construction would fix some basic graph G and then add an edge between any pair of vertices u, v where there is a path from u to v in G.

Modularity

Just where do we add edges? A couple of ideas.

- The most promiscuous construction would fix some basic graph G and then add an edge between any pair of vertices u, v where there is a path from u to v in G.
- More conservatively, fix some basic graph G and then add an edge between any pair of vertices u, v where v is the result of inserting some program w into u at a single position.

Learning

- So far we have a model of the student's *behavior* given some fixed (vector or function) p, but how does the student learn how to improve?

- So far we have a model of the student's *behavior* given some fixed (vector or function) p, but how does the student learn how to improve?
- Consider a formal learning perspective...

- So far we have a model of the student's *behavior* given some fixed (vector or function) p, but how does the student learn how to improve?
- Consider a formal learning perspective...
- We have a learner A.

- So far we have a model of the student's *behavior* given some fixed (vector or function) p, but how does the student learn how to improve?
- Consider a formal learning perspective...
- We have a learner A.
- We have some set of experiences (or training items) $S = \{X_1, \ldots, X_n\} \in \mathcal{S}$.

- So far we have a model of the student's *behavior* given some fixed (vector or function) p, but how does the student learn how to improve?
- Consider a formal learning perspective...
- We have a learner A.
- We have some set of experiences (or training items) $S = \{X_1, \ldots, X_n\} \in \mathcal{S}$.
- Let's be notationally awkward and think of p as living in some space Θ.

- So far we have a model of the student's *behavior* given some fixed (vector or function) p, but how does the student learn how to improve?
- Consider a formal learning perspective...
- We have a learner A.
- We have some set of experiences (or training items) $S = \{X_1, \ldots, X_n\} \in \mathcal{S}$.
- Let's be notationally awkward and think of p as living in some space Θ.
- So A is some function $A : \mathcal{S} \to \Theta$.

Experiences?

An experience $X_i = \{X_{i1}, \ldots, X_{im}\}$ is a particular path through the program graph.

$\emptyset$

```
TurnLeft();
```

```
TurnLeft();
Move();
```

```
TurnLeft();
Move();
TurnRight();
```

```
TurnLeft();
Move();
TurnRight();
Move();
```

Learning

A learns by reducing its probability of forgetting in proportion to the number of edges of that type it has seen in training.

Algorithm 4 Learn

```
procedure LEARN
    Input: Example programs X_1, X_2, ..., X_n, learning rate 0 ≤ γ ≤ 1.
    Output: A learner characterized by p.
    Initialize p(t) for each t in the domain of p to 1
    for i in 1 to n do
        for each subseqence l = {X_ij, ..., X_ik} of {X_i1, ..., X_im} do
            if t(l) is defined then
                p(t(l)) ← γp(t(l))
    return p
```

Here t is a type function that takes an edge to its type.

Teaching

Teaching

- Let's take the point of view that the performance of learner A on some search problem defines *testing*.

Teaching

- Let's take the point of view that the performance of learner A on some search problem defines *testing*.
- We then defined *learning*.

Teaching

- Let's take the point of view that the performance of learner A on some search problem defines *testing*.
- We then defined *learning*.
- So what's teaching?

Intuition

- Let's say we wanted to increase the chance of the student succeeding at reaching a particular program v. What would we do?

Intuition

- Let's say we wanted to increase the chance of the student succeeding at reaching a particular program v. What would we do?
- We would show that program (probably over and over again).

Intuition

- Let's say we wanted to increase the chance of the student succeeding at reaching a particular program v. What would we do?
- We would show that program (probably over and over again).
- Think of this as a *cheating* strategy of teaching.

Intuition

- But now let's say there isn't a single v state but a large set of states $v_1, \ldots, v_n$, all of which we want to teach reasonably well and say we had a limited budget for the number of items we could teach with. What would we do?

Intuition

- But now let's say there isn't a single v state but a large set of states $v_1, \ldots, v_n$, all of which we want to teach reasonably well and say we had a limited budget for the number of items we could teach with. What would we do?
- Conjecture: we'd extract common, re-usable patterns and teach those.

Intuition

- But now let's say there isn't a single v state but a large set of states $v_1, \ldots, v_n$, all of which we want to teach reasonably well and say we had a limited budget for the number of items we could teach with. What would we do?
- Conjecture: we'd extract common, re-usable patterns and teach those.
- Think of this as a *curriculum* strategy of teaching.

Formalize this!

For a single v:

$$\begin{aligned} \min_{S \in \mathcal{S}} \quad & \epsilon(S) \\ \text{st} \quad & \mathbb{P}(R(v, A(S)) = 1) \geq \alpha. \end{aligned}$$

Formalize this!

For a single v:

$$\begin{aligned} \min_{S \in \mathcal{S}} \quad & \epsilon(S) \\ \text{st} \quad & \mathbb{P}(R(v, A(S)) = 1) \geq \alpha. \end{aligned}$$

But if $V \sim F_V$:

$$\begin{aligned} \min_{S \in \mathcal{S}} \quad & \epsilon(S) \\ \text{st} \quad & \mathbb{E}[R(V, A(S))] \geq \alpha. \end{aligned}$$

- Here ϵ is a *teacher's effort* function.
- And R is the Bernoulli random variable representing the success of $A(S)$ at finding V.

Teacher's effort

What's that teacher's effort $\epsilon(S)$?

Teacher's effort

What's that teacher's effort $\epsilon(S)$?

- We can take $\epsilon(S) = \sum_{X \in S} |X|$ to express a preference for the smallest sequence of paths.

Teacher's effort

What's that teacher's effort $\epsilon(S)$?

- We can take $\epsilon(S) = \sum_{X \in S} |X|$ to express a preference for the smallest sequence of paths.
- Over a graph with no composite edges, this is equivalent to the number of instructions in a program.

Teacher's effort

What's that teacher's effort $\epsilon(S)$?

- We can take $\epsilon(S) = \sum_{X \in S} |X|$ to express a preference for the smallest sequence of paths.
- Over a graph with no composite edges, this is equivalent to the number of instructions in a program.
- But we can be creative here...

Questions?