ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Prof. R. Wattenhofer

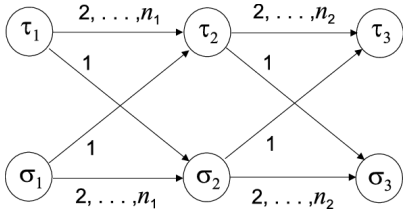
Algorithm Learning

Learning to Induce Graph-Walking Programs

Choose your favourite graph data sample where all nodes are labelled according to some scheme. Given a starting point and a set of endpoints from that graph, how can we identify the traversal query (algorithm) that was used to reach the endpoints?

The general problem is quite hard. So, make the assumption that every step in the sought traversal is decided only by the labels of the neighbours of the current node. This simplifies the problem greatly and makes it very similar to the problem of regular expression learning, though in this case we benefit implicitly from the additional information coming from all the paths that we *did not take* to reach the endpoints.

Graph (network) data is ubiquitous and often very complex. Unlike relational database data (that can too be viewed as graph data), it usually does not benefit from a fixed, explicit structure and is therefore somewhat more difficult to work with.



The following are some candidate sub-problems that we identified as reasonable starting points for the project

1. Consider a graph G in which every node has a unique identifier (say an integer) and is assigned one of finitely many colours $\{1, 2, \ldots, k\}$. Given pairs of the form $\langle$*startpoints*, *endpoints*$\rangle$, how do we find walking programs $c_1 c_2 \cdots c_l$ that, when followed, reach *endpoints* from *startpoints* in G?

2. Consider a graph G just like the above, but with real numbers, strings, or even both serving as labels instead of colours. Assume that we are given the pairs of the form $\langle$*startpoints*, *endpoints*$\rangle$ as before. How do we find walking programs of a special filtering domain-specific language of our own making that can compare real numbers and test some basic properties of strings to decide to which nodes to proceed?

The first of the problem can be approached by brute force (but might be too time-expensive), Monte Carlo Tree Search (if formulated as a game, though the lack of an objective function or adversary makes the approach a bit less appropriate), pre-processing the graph in advance to speed up the program search (could be both time- and memory-expensive if not done well), and by gathering information about various parts of the graph with random walks.

A possible solution to the second challenge would likely use the core approach chosen to tackle the previous problem, with a modification allowing it to learn the filtering criteria. Something of a similar nature has been done before both by training neural networks on a

large artificial dataset, and by resorting to logical machinery to learn from programs that were proposed but did not fit the training data.

Who is this for? Bachelor's but preferably master's students interested in diving into the above-mentioned problem. Some familiarity with graphs, graph terminology, or finite automata will make for a speedier start. Interest in research would certainly be put to a good use.

Interested? Please reach out to us for more details.

Point of contact: Peter Belcak, email pbelcak@ethz.ch, ETZ G63.