Information and Computation 234 (2014) 44–56

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Effective storage capacity of labeled graphs

Dana Angluin [a,1], James Aspnes [a,*,2], Rida A. Bazzi [b], Jiang Chen [c], David Eisenstat [d], Goran Konjevod [e]

[a] Department of Computer Science, Yale University, United States
[b] Computer Science and Engineering, SCIDSE, Arizona State University, United States
[c] Google, United States
[d] Department of Computer Science, Brown University, United States
[e] Lawrence Livermore National Laboratory, United Sates

ARTICLE INFO

Article history:
Available online 21 November 2013

ABSTRACT

We consider the question of how much information can be stored by labeling the vertices of a connected undirected graph G using a constant-size set of labels, when isomorphic labelings are not distinguishable. Specifically, we are interested in the **effective capacity** of members of some class of graphs, the number of states distinguishable by a Turing machine that uses the labeled graph itself in place of the usual linear tape. We show that the effective capacity is related to the **information-theoretic capacity** which we introduce in the paper. It equals the information-theoretic capacity of the graph up to constant factors for trees, random graphs with polynomial edge probabilities, and bounded-degree graphs.

1. Introduction

We consider the question of how much information can be stored by labeling the vertices of a connected undirected graph G using a constant-size set of labels, when isomorphic labelings are not distinguishable. An exact information-theoretic bound is easily obtained by counting the number of isomorphism classes of labelings of G, which we call the **information-theoretic capacity** of the graph. More interesting is the **effective capacity** of members of some class of graphs, the number of states distinguishable by a Turing machine that uses the labeled graph itself in place of the usual linear tape.

The motivation for this model is self-organizing systems consisting of many communicating finite-state machines, where at any time, one machine (the location of the Turing machine head) takes a leadership role. Our main question is how much computing power such machines can cooperate to achieve. The answer depends on the inherent storage capacity of the communication graph, a function of its size (bigger gives more space) and symmetries (more symmetries make the space harder to exploit).

In more detail, a **graph Turing machine** consists of an undirected connected graph G, each of whose nodes holds a symbol from some finite alphabet, together with a finite-state controller that can move around the graph and update the symbols written on nodes. Because there is no built-in sense of direction on an arbitrary graph, the left and right moves of a standard Turing machine controller are replaced by moves to adjacent graph nodes with a given symbol. If there is no such adjacent graph node, the move operation fails, which allows the controller to test its immediate neighborhood for the

* Corresponding author.
E-mail addresses: angluin@cs.yale.edu (D. Angluin), aspnes@cs.yale.edu (J. Aspnes), bazzi@asu.edu (R.A. Bazzi), criver@gmail.com (J. Chen), eisenstatdavid@gmail.com (D. Eisenstat), goran@llnl.gov (G. Konjevod).

[1] Supported in part by NSF grant CCF-0916389.
[2] Supported in part by NSF grants CNS-0435201 and CCF-0916389.

0890-5401/$ – see front matter
http://dx.doi.org/10.1016/j.ic.2013.11.004

absence of particular symbols. If there is more than one such node, which node the controller moves to is chosen arbitrarily. (A more formal definition of the model is given in Section 3.)

The intent of this model is to represent what computations are feasible in various classes of simple distributed systems made up of a network of finite-state machines. Inclusion of an explicit head that can move nondeterministically to adjacent nodes (thus breaking at least local symmetries in the graph) makes the model slightly stronger than similar models from the self-stabilization literature (e.g., Dijkstra's original model in [1]) or population protocols [2]; we discuss the connection between our model and these other models in Section 2.

The main limitation on what a graph Turing machine can compute appears to be the intrinsic **storage capacity** of its graph. For some graphs (paths, for example) the storage capacity is essentially equivalent to a Turing machine tape of the same size. For others (cliques, stars, some trees), the usable storage capacity may be much less, because symmetries within the graph make it difficult to distinguish different nodes with the same labeling. We define a notion of **information-theoretic capacity** of a graph (Section 4.1) that captures the number of distinguishable classes of labelings of the graph. Essentially this comes down to counting equivalence classes of labelings under automorphisms of the graph; it is related to the notion of the **distinguishing number** of a graph, which we discuss further in Section 2.3.

The information-theoretic capacity puts an upper bound on the **effective capacity** of the graph, the amount of storage that it provides to the graph Turing machine head (defined formally in Section 4.2). Extracting usable capacity requires not only that labelings of the graph are distinguishable in principle but that they are distinguishable to the finite-state controller in a way that allows it to simulate a classic Turing machine tape. We show, in Section 6.1, that an arbitrary graph with n nodes provides at least $\Omega(\log n)$ tape cells worth of effective capacity (which matches the information-theoretic upper bound for cliques and stars, up to constant factors). For specific classes of graphs, including trees (Section 7), bounded-degree graphs (Section 8), and random graphs with polynomial edge probabilities (Section 9), we show that the effective capacity similarly matches the information-theoretic capacity.

Notably, these classes of graphs are ones for which testing graph isomorphism is easy. Whether we can extract the full capacity of a general graph is open, and appears to be related to whether graph isomorphism for arbitrary graphs can be solved in **LOGSPACE**. We discuss this issue in Section 10.

2. Related work

2.1. Self-stabilizing models

A graph Turing machine bears a strong resemblance to a network of finite-state machines, which has been the basis for numerous models of distributed computing, especially in the self-stabilization literature. Perhaps closest to the present work is the original self-stabilizing model of Dijkstra [1], where we have a collection of finite-state nodes organized as a finite connected undirected graph, and at each step some node may undergo a transition to a new state that depends on its previous state and the state of its immediate neighbors. The main difference between the graph Turing machine model and this is the existence of a unique head, and even more so, its ability to move to a single neighbor of the current node—these properties break symmetry in ways that are often difficult in classic self-stabilizing systems. A limitation of the graph Turing machine model is the restriction on what the head can sense of adjoining nodes: it cannot distinguish neighbors in the same state, or even detect whether one or many neighbors is in a particular state.

Itkis and Levin [3] give a general method for doing self-stabilizing computations in asynchronous general topology networks. Their model is stronger than ours, in that each node can maintain pointers to its neighbors (in particular, it can distinguish neighbors in the same state). Nonetheless, we have found some of the techniques in their paper useful in obtaining our current results.

2.2. Population protocols

There is also a close connection between our model and the **population protocol** model [2], in which a collection of finite-state agents interact pairwise, each member of the pair updating its state based on the prior states of both agents (see [4] for a recent survey on this and related models). This is especially true for work on population protocols with restricted communication graphs (for example, [5]). Indeed, it is *almost* possible to simulate a graph Turing machine in a population protocol, simply by moving the state of the head around as part of the state of the node it is placed on, and using interactions with neighbors to sense the local state. The missing piece in the population protocol model is that there is no mechanism for detecting the absence of a particular state in the immediate neighborhood. Although a fairness condition implies that every neighbor will make itself known eventually, the head node has no way to tell if this has happened yet. Urn automata [6], a precursor to the population protocol model in which a finite-state controller manages the population, also have some similarities to graph Turing machines, especially in the combination of a classical Turing-machine controller with an unusual data store.

The **community protocol** model of Guerraoui and Ruppert [7,8] extends population protocols by allowing agents to store a constant number of pointers to other agents that can only be used in limited ways. Despite these restrictions, Guerraoui and Ruppert show that community protocols with n agents can simulate **storage modification machines** as defined by Schönhage [9], which consist of a dynamic graph on n nodes updated by a finite-state controller. Such machines can in