What Can Be Computed Locally?

Moni Naor*
Larry Stockmeyer†

Abstract. The purpose of this paper is a study of computation that can be done locally in a distributed network, where "locally" means within time (or distance) independent of the size of the network. *Locally Checkable Labeling* (*LCL*) problems are considered, where the legality of a labeling can be checked locally (e.g., coloring). The results include the following:

- There are non-trivial LCL problems that have local algorithms.
- There is a variant of the dining philosophers problem that can be solved locally.
- Randomization cannot make an LCL problem local; i.e., if a problem has a local randomized algorithm then it has a local deterministic algorithm.
- It is undecidable, in general, whether a given LCL has a local algorithm.
- However, it is decidable whether a given LCL has an algorithm that operates in a given time t.
- Any LCL problem that has a local algorithm has one that is order-invariant (the algorithm depends only on the order of the processor id's).

Keywords: distributed computation, local computation, graph labeling problem, resource allocation, dining philosophers problem, randomized algorithms

This paper appeared in the SIAM Journal on Computing, Vol. 24, No. 6, 1995, pp. 1259–1277.

*Dept. of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel.
Partly supported by a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences.
Work performed while at the IBM Almaden Research Center. E-mail: naor@wisdom.weizmann.ac.il.

†IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120. E-mail: stock@almaden.ibm.com.

1 Introduction

A property of distributed computational systems is *locality.* Each processor is directly connected to at most some fixed number of others. Despite the locality of connections, we may want to perform some computation such that the values computed at different nodes must fit together in some global way. The purpose of this paper is to attempt to understand what can be computed when algorithms must satisfy a strong requirement of locality, namely, that the algorithm must run in *constant time* independent of the size of the network. A processor running in constant time t must base its output solely on the information it can collect from processors located within radius t from it in the network. Apart from the obvious advantage of constant time (that constant time takes less time than non-constant time), another advantage is improved fault-tolerance: if the algorithm runs in constant time, a failure at a processor p can only affect processors in some bounded region around p. Another motivation for locality is in recent work on self-stabilizing distributed algorithms; for example, Afek, Kutten and Yung [2] introduced the idea of detecting an illegal global configuration by checking local conditions.

Our work has three goals: first, to lay some groundwork for studying the question of what can and cannot be computed locally; second, to establish some basic, general results; and third, to study particular examples.

A network is modeled as an undirected graph where each node represents a processor and edges represent direct connections between processors. We consider only networks of bounded degree. Our main focus is on computational problems of producing "labelings" of the network. Since our subject is constant time algorithms, it makes sense to restrict to labelings such that the validity of a labeling can be checked locally (i.e., by checking within some fixed radius from the node). We call these *locally checkable labelings* (*LCL*'s). Familiar examples of LCL's are vertex coloring, edge coloring, and maximal independent set (MIS). In the case of MIS, for example, one local constraint says that if vertex v is in the MIS then no neighbor of v is in the MIS; another constraint says that if v is not in the MIS then v has at least one neighbor in the MIS. In general, the output labeling might depend on some initial input labeling, and most of our general results hold in this case. If all processors are identical, it is already known (by familiar symmetry arguments) that the types of labeling problems that can be solved deterministically are very limited. So we assume that processors are given unique numerical id's. If an algorithm runs in time t then, for each vertex v, the processor at v can collect information about the structure of the network, including processor id's (and possibly input labels), in the region of radius t around v. Then the processor must choose its output label based on this information. The algorithm must be correct, that is, the entire output labeling must be valid, regardless of how the processors are numbered with unique id's.

Several recent papers have given improved time algorithms for certain LCL's such as MIS and vertex coloring, for example, Awerbuch, Goldberg, Luby and Plotkin [3], Goldberg, Plotkin and Shannon [8], Linial [10], and Panconesi and Srinivasan [15]. However, these papers do not consider constant time; the running time of the algorithms grows with the size of the network. Indeed the time must grow. In the first paper to establish the limitations of locality in this context, Linial [10] proved that, even on ring networks, an MIS or a 3-coloring of vertices cannot be found in constant time.[1]

In light of previous work on locality, two questions come to mind:

- Can any nontrivial LCL problem be solved in constant time?
- If the answer to the first question is "yes", can we characterize the LCL's that can be solved in constant time?

One of our results is that the answer to the first question is "yes". Define a *weak c-coloring* of a graph to be a coloring of the vertices with c colors such that each non-isolated vertex has at least one neighbor colored differently. It is easy to see that a weak 2-coloring exists for every graph. We show the following for every fixed d.

- Consider the class of graphs of maximum degree d where every vertex has odd degree. There is a $c = c(d)$ and an algorithm that finds a weak c-coloring in time 2 for any graph in this class. Here c is exponential in d, but in an additional time $O(\log^* d)$ the number of colors can be reduced to 2.

This result is the best possible in three senses:

- For d-regular graphs where d is even, for no constant $c = c(d)$ is there a constant time algorithm that finds a weak c-coloring.
- The time bound 2 cannot be reduced to 1.
- If we change the definition of a coloring so that every vertex v must have at least two neighbors colored differently than v, then even for d-regular graphs with d odd, a coloring cannot be found in constant time.

Although a weak coloring might seem a strange concept, we have used it as a basis for a solution to a certain resource allocation problem. A well-known paradigm for resource allocation problems is Dijkstra's Dining Philosophers Problem, which was later generalized from a ring to arbitrary graphs (see, e.g., [4, 11]). In the version of the problem we consider, there is a given *conflict graph* where each node represents a processor and each edge represents a resource (a "fork") which is shared by the two endpoint processors. It is assumed that if two processors share a resource, then they are also close in the communication network. At any time, a fork can be "owned" by at most one of the processors that share it. Each processor can be in one of three states: resting, hungry, or eating. The processors operate asynchronously. A resting processor can become hungry at any time. In order to eat, a processor must obtain certain forks; we get different types of problems depending on precisely what "certain" means. A processor eats for at most a bounded time, after which it returns to the resting state. A processor p can attempt to "grab" a certain fork, and can release an owned fork. The grab operation will fail if the fork is currently owned by the other processor q; if this occurs, p may decide to wait for q to release the fork. We require a solution that

[1] Actually, Linial gives a lower bound of $\Omega(\log^* n)$ on oriented rings of size n, which matches an upper bound of Cole and Vishkin [6] to within a constant factor.

is starvation free, meaning that a hungry processor will eventually be able to eat. An important measure of the goodness of a solution is the maximum length of a *waiting chain* that can develop. As pointed out by Choy and Singh [5], a difficulty with long waiting chains is that if a processor p fails while holding a fork, the failure will affect every processor behind p in the waiting chain.

In the traditional version of this problem, if a processor shares d forks (has d incident edges in the conflict graph), it can eat only when it has obtained all d forks. In this case, Lynch [11] gave a solution with waiting chains of length $O(c)$ assuming that the conflict graph is edge colored with c colors. The maximum length was reduced to $O(\log c)$ by Styer and Peterson [17], again assuming that an edge coloring is given. Choy and Singh [5] give a solution with waiting chains of length at most 3, assuming that a certain vertex coloring with $d+1$ colors is given. All of these solutions require that the conflict graph be initially colored in some way. Such colorings (provably) cannot be found in constant time. It is therefore natural to ask whether there is any *purely local* solution to this problem, i.e., a solution with waiting chains bounded by a constant, and which does not assume any initial coloring of the conflict graph. In fact, it can be shown that there is no local solution to this problem by reducing the MIS problem to it. However, we show that there is a purely local solution to a relaxed version of the problem. In this version, a processor can eat when it has obtained *any two* forks. This can be viewed as a threshold condition: a processor can proceed when it has two units of resource. We call this problem the *formal-dining philosophers* problem. Imagine that dining is formal and in order to eat a philosopher must dress formally and in particular wear cuff links. We assume that the resource on each edge is a cuff link. In order to dress formally (in the western male tradition) and eat, the philosopher must get any two cuff links. Our solution works in any bounded degree conflict graph of minimum degree 3, i.e., every vertex has at least 3 incident edges. (If the degree is 2, then we have Dijkstra's original version on a ring, for which it is impossible to find a local solution.) To our knowledge, this is the first nontrivial resource allocation problem that has been solved in a purely local fashion.

Returning to the second question above (Can we characterize the LCL's that can be solved in constant time?), another result shows that this will be difficult – it is undecidable. Fix any $d \geq 3$, and let $\mathcal{G}$ be the class of d-regular graphs or the class of graphs of maximum degree d. Even if we restrict attention to LCL's such that every graph in $\mathcal{G}$ has a legal labeling, we show that it is undecidable, given an LCL $\mathcal{L}$, whether there is a constant-time algorithm that solves $\mathcal{L}$ for every graph in $\mathcal{G}$. If $d = 2$, however, the problem becomes decidable. The problem is also decidable if we are given a specific time t and would like to know whether there is a t-time algorithm for the given LCL instance.

We close this Introduction by mentioning two additional "general" results. The first states that there is no loss of generality in restricting attention to algorithms that do not use the actual values of the processor id's, but only their relative order. This result is useful in proving some of our other results. The proof is by a Ramsey theory argument similar to ones in [18, 7, 13]. This is in contrast to the non-constant-time case, where for instance an order-invariant algorithm for 3-coloring the ring would take time $\Theta(n)$, but the Cole-Vishkin [6] method (which uses the actual values of the id's) takes time $O(\log^* n)$.

Another result states that randomization does not help in solving LCL's in constant time. For the class $\mathcal{G}$ of d-regular graphs or the graphs of maximum degree d for any fixed $d \geq 2$, if there is a randomized algorithm that runs in time t and that solves the LCL $\mathcal{L}$ with error probability $\varepsilon < 1$ on any graph in $\mathcal{G}$, then there is a deterministic algorithm that runs in time t and solves $\mathcal{L}$ on any graph in $\mathcal{G}$.

We now outline the remainder of the paper. Section 2 gives our definitions of LCL's and local algorithms. In Section 3 we show that every local algorithm can be replaced with an order-invariant one. The subject of Section 4 is undecidability and decidability of questions about local solvability. In Section 5 we show that randomization does not help in solving LCL's locally. The subject of Section 6 is weak coloring. In Section 7, the local algorithm for weak coloring is used, together with other ideas, to give a local solution to the formal-dining philosophers problem. In Section 8, we suggest some open questions raised by our work. For readers interested mainly in the results for weak coloring and formal-dining philosophers, we should point out that Sections 6 and 7 are completely independent from Sections 4 and 5. In addition, the local algorithms for weak coloring and formal-dining philosophers do not depend on anything from Sections 3, 4, or 5, although the impossibility results for weak coloring and formal-dining philosophers use the order-invariance result from Section 3.

2 Definitions

We first give some definitions and notations concerning graphs. All graphs in this paper are simple and undirected. For a graph $G = (V, E)$ and vertices $u, v \in V$, let $dist_G(u, v)$ be the distance (length of a shortest path) in G from u to v. If $u \in V$ and $e \in E$, and if the endpoints of e are v and w, then $dist_G(u, e) = \min\{dist_G(u, v), dist_G(u, w)\} + 1$. For a vertex u and a nonnegative integer r, let $B_G(u, r)$ denote the subgraph of G consisting of all vertices v and edges e such that $dist_G(u, v) \leq r$ and $dist_G(u, e) \leq r$. The subscript G is omitted when G is clear from context. A *centered graph* is a pair (H, s) where H is a graph and s is a vertex of H. The *radius* of (H, s) is the maximum distance from s to any vertex or edge of H.

We now define the notion of a "locally checkable labeling" (LCL). For simplicity, we give the definition only for vertex labelings. A similar definition can be given for edge labelings (e.g., edge colorings or edge orientations). To make the definition somewhat more general, we allow the vertices of the graph to be initially labeled with "input labels". Formally, then, an *LCL* $\mathcal{L}$ consists of a positive integer r (called the *radius* of $\mathcal{L}$), a finite set Σ of *input labels*, a finite set Γ of *output labels*, and a finite set $\mathcal{C}$ of *locally consistent labelings*. Each element of $\mathcal{C}$ is a centered graph of radius at most r where each vertex is labeled with a pair from $\Sigma \times \Gamma$. Given a graph $G = (V, E)$ and a labeling $\lambda : V \to \Sigma \times \Gamma$, the labeling λ is *$\mathcal{L}$-legal* if, for every $u \in V$, there is a $(H, s) \in \mathcal{C}$ and an isomorphism π mapping $B_G(u, r)$ to H such that $\pi(u) = s$ and such that π respects the labeling, i.e., for every w, the label-pair of w equals the label-pair of $\pi(w)$. Although certain types of labelings, such as the usual definition of vertex coloring, are more naturally expressed in terms of *forbidden conditions* instead of allowed conditions, it is easy to see that the definition above captures such labelings. Essentially, the set $\mathcal{C}$ gives a "truth table" of all locally consistent labelings. Many of our specific examples of LCL's do not have input. Such LCL's are a special case of the definition above simply by taking $|\Sigma| = 1$.

We consider distributed algorithms which operate on graphs G that are initially input-labeled and where each vertex is also numbered with a unique positive integer *id*. If the algorithm produces an output label for each vertex within t steps, we can assume that, for each vertex u, the part of the algorithm running at u collects information about the structure, input labels, and id's of $B_G(u,t)$, and chooses an output label for u based on this information (although particular algorithms might not actually "use" all this information). Suppose that the algorithm is to be run on graphs of maximum degree d. For a constant t, a *local algorithm with time bound t* is a function A; the input to A is a centered graph (H,s) of radius at most t and degree at most d whose vertices are labeled with (input, id) pairs; the value of $A((H,s))$ is some $\gamma \in \Gamma$. The local algorithm A is applied to an input-labeled and id-numbered graph G by applying A independently at each vertex of G; that is, for each vertex u, the output label of u is $A(B(u,t))$ where $B(u,t)$ is viewed as a centered graph with center u. For a local algorithm A, an LCL $\mathcal{L}$, and a class $\mathcal{G}$ of graphs, we say that A *solves $\mathcal{L}$ for $\mathcal{G}$* if, for every $G \in \mathcal{G}$, every input labeling of G, and every numbering of the vertices of G with unique id's, A produces an $\mathcal{L}$-legal labeling, i.e., the combination of the output labeling produced by A with the initial input labeling is $\mathcal{L}$-legal.

Since the subject of the paper is locality, we largely restrict attention to (infinite) classes of graphs for which membership in the class can be checked locally. Examples are d-regular graphs and graphs of maximum degree d, for any constant d. Note that if membership in $\mathcal{G}$ can be checked locally, then $\mathcal{G}$ is *closed under disjoint union*; i.e., for every $G, G' \in \mathcal{G}$, the graph consisting of the disjoint union of G and G' belongs to $\mathcal{G}$. We consider only classes with some constant upper bound on degree.

Remark. Although it might be more natural to assume that the id's for an n-vertex graph are drawn from $\{1,2,\ldots,n\}$, there is no harm in requiring algorithms to handle arbitrary id numberings. For suppose that A incorrectly labels G when id's are arbitrary. Form a new graph G' with n' vertices consisting of the disjoint union of G with a large enough graph so that the vertices of G' can be numbered from $\{1,2,\ldots,n'\}$ while keeping the numbering of G the same. Then A labels G' incorrectly.

3 Order-invariant algorithms

In what follows, it is sometimes useful to restrict attention to algorithms that do not use the actual values of the id's, but only their relative order. Two id numberings η and η' of a graph H are *order-equivalent* if, for every pair of vertices u and v, $\eta(u) < \eta(v)$ iff $\eta'(u) < \eta'(v)$. A local algorithm A is *order-invariant* if for every (H,s) in the domain of A, if we obtain H' from H by changing the id numbering η to any other η' such that η and η' are order-equivalent, then $A((H,s)) = A((H',s))$.

Using Ramsey theory, we show that there is no loss of generality in restricting attention to order-invariant algorithms. This type of application of Ramsey theory is hardly new: starting with Yao's celebrated paper on searching tables [18], through Frederickson and Lynch's [7] paper on a problem in distributed computing and Moran, Snir and Manber's [13] work on decision trees, and many other papers.

For a set S and an integer $p \le |S|$, let $[S]^p$ denote the set of subsets $A \subseteq S$ with $|A| = p$. We use the following theorem due to Ramsey [16]. (For information on Ramsey Theory see [9].)

Theorem 3.1 (Ramsey) *For any p, m and c, there is a number $R(p, m, c)$ such that the following holds. Let S be a set of size at least $R(p, m, c)$. For any coloring of $[S]^p$ with at most c colors, there is a $T \subseteq S$ with $|T| = m$ such that all of $[T]^p$ is colored the same.*

We first state and prove the order-invariance result in a stronger form which will be useful later.

Lemma 3.2 *Fix an LCL $\mathcal{L}$ (with or without input), a class $\mathcal{G}$ of graphs, and a time bound t. Let d be the maximum degree of a graph in $\mathcal{G}$. There is a number R, depending only on d, t, and $\mathcal{L}$, such that the following holds. For every local algorithm A with time bound t and every set S of id's with $|S| \geq R$, there is an order-invariant local algorithm A' with time bound t such that, for every $G \in \mathcal{G}$ and every input labeling of G, if A labels G correctly for every id numbering drawn from S then A' labels G correctly for every id numbering.*

Proof. We show how to convert any A to an order-invariant A' such that the last sentence of the theorem is satisfied. Let $(K_1, s_1), \ldots, (K_z, s_z)$ be the set of input-labeled centered graphs (K, s) such that id numberings of (K, s) appear in the domain of A. Let p be the maximum number of vertices in any K_i. Note that p and z depend only on d, t, and $\mathcal{L}$.

Given any set S of id's, define an equivalence relation on $[S]^p$ as follows. For $X, X' \in [S]^p$, let

$$X = \{x_1, x_2, \ldots, x_p\} \text{ and } X' = \{x'_1, x'_2, \ldots, x'_p\}$$

be the elements of X and X' indexed in increasing order. Viewing $K_1, \ldots, K_z$ as graphs on disjoint sets of vertices, let V be the union of all these vertex sets. For $\sigma : V \to \{1, 2, \ldots, p\}$, let $K_j(\sigma)$ (resp., $K'_j(\sigma)$) be the graph K_j where each vertex v is numbered with the id $x_{\sigma(v)}$ (resp., $x'_{\sigma(v)}$). We restrict attention to those σ's such that, for every K_j, no two vertices of K_j are numbered the same. Now $X \equiv X'$ iff $A((K_j(\sigma), s_j)) = A((K'_j(\sigma), s_j))$ for all σ and all j. It is easy to see that this is an equivalence relation. It is also clear that there is an upper bound c on the number of equivalence classes. This bound depends only on p, z, and $\mathcal{L}$, so it depends only on d, t, and $\mathcal{L}$. Let r be the radius of $\mathcal{L}$ and let m equal p plus the maximum number of vertices in any centered graph of degree at most d and radius at most $r + t$. Again, m depends only on d, t, and $\mathcal{L}$. Let $R = R(p, m, c)$, so R depends only on d, t and $\mathcal{L}$.

Carrying out the above for any S with $|S| \geq R$, Theorem 3.1 implies that there is a set of id's $T \subseteq S$ with $|T| = m$ such that all members of $[T]^p$ are equivalent. Let U equal T minus the p largest members of T. By choice of m, $|U|$ is as large as the maximum number of vertices in any centered graph of degree at most d and radius at most $r + t$. We claim that A is order-invariant when id's are drawn from U. Let (H, s) be any centered graph in the domain of A with id numbering η mapping its vertices to U. Let (H', s) be this graph with the id numbering η' mapping to U, where η, η' are order-equivalent. Let X (resp., X') be a member of $[T]^p$ containing all the id's of H (resp., H') such that, for every vertex u, if $\eta(u) = x_i$ then $\eta'(u) = x'_i$ (where, as above, the elements of X and X' are indexed in increasing order). This is possible because T contains p "extra" elements not belonging to U, so in the case that H has fewer than p vertices, we can use the extra elements to pad the sets X and X' to be of size exactly p. Let σ and j be such that $(H, s) = (K_j(\sigma), s_j)$. By choice of X and X', $(H', s) = (K'_j(\sigma), s_j)$. So $A((H, s)) = A((H', s))$ since $X \equiv X'$.

The algorithm A' works as follows. On a centered graph (H, s) numbered with id's, A' first changes the id's in an order-equivalent way to id's in U. (Since U is large enough, this is always possible.) Then A' answers the same as A on the newly numbered H. Note that since A is order-invariant on U, it does not matter exactly how A' does the renumbering, provided that it is order-equivalent. Clearly A' is order-invariant.

It remains to show that A' has the required correctness property. Let $G \in \mathcal{G}$ and fix some input labeling. Suppose that A' does not label G correctly. This means that there is some vertex u of G such that $B(u, r)$ is not labeled correctly. Obtain a new id-numbered graph by changing the id's to id's in S in such a way that (i) each vertex of $B(u, r + t)$ has a new id in U, and (ii) the new id-numbering of $B(u, r+t)$ is order-equivalent with the old one. But since A' is order-invariant and A is order-invariant when id's are drawn from U, it follows from the definition of A' that, for each vertex v of $B(u, r)$, the output label given to v by A' under the original id numbering is the same as the output label given to v by A under the new id numbering. This contradicts the assumption that A correctly labels G when id's are drawn from S. $\square$

The following is now immediate.

Theorem 3.3 *Fix an LCL $\mathcal{L}$ and a class $\mathcal{G}$ of graphs. If there is a local algorithm A with time bound t that solves $\mathcal{L}$ for $\mathcal{G}$ then there is an order-invariant local algorithm A' with time bound t that solves $\mathcal{L}$ for $\mathcal{G}$.*

4 Undecidability

In this section we consider the problem, for a fixed class $\mathcal{G}$ of graphs, of deciding whether a given LCL $\mathcal{L}$ can be solved in constant time for $\mathcal{G}$. The answer could be "no" for an uninteresting reason, namely, that there is some $G \in \mathcal{G}$ that has no $\mathcal{L}$-legal labeling. Therefore we restrict attention to $\mathcal{L}$'s for which every $G \in \mathcal{G}$ has an $\mathcal{L}$-legal labeling. We also restrict attention to LCL's without input; since our main result is an undecidability result, this just makes the result stronger. Define $Y(\mathcal{G})$ (resp., $N(\mathcal{G})$) to be the set of LCL's $\mathcal{L}$ without input such that every $G \in \mathcal{G}$ has an $\mathcal{L}$-legal labeling and there is (resp., is not) a constant t such that some local algorithm with time bound t solves $\mathcal{L}$ for $\mathcal{G}$. Recall that sets Y and N are *recursively separable* if there is a Turing machine that answers "yes" on every input from Y and answers "no" on every input from N (and we do not care about its answer otherwise).

Theorem 4.1 *Fix any $d \geq 3$, and let $\mathcal{G}$ be the class of d-regular graphs or the class of graphs of maximum degree d. Then $Y(\mathcal{G})$ and $N(\mathcal{G})$ are not recursively separable.*

Proof. We show that if $Y(\mathcal{G})$ and $N(\mathcal{G})$ are recursively separable, then it can be decided for a given Turing machine M whether M halts on blank tape. We first describe the proof for the class of 4-regular graphs. We begin by proving the result for a different class of graphs, and then work in several steps towards 4-regular graphs.

(1). Consider first the class of 2-dimensional grid graphs where one corner of the graph is marked as "special", say by having an extra edge which connects it to a new vertex of degree 1.

(A 2-dimensional grid graph has vertices $\{1,\ldots,k\}\times\{1,\ldots,l\}$ for some k and l, and two vertices are connected by an edge if the L_1-distance between them is 1.) Imagine that the special corner is the upper left corner. Let M be a given Turing machine with states Q and tape alphabet T. Modify M if necessary so that (i) if M does not halt then M visits an infinite amount of tape, and (ii) the head never moves left of its initial position. The idea is to have the LCL $\mathcal{L}$ force the labeling to be a computation of M started on blank tape, where the ith row of the grid contains the configuration (tape contents, state, and head position) at the ith step. The head position is given by writing the state symbol just to the left of the scanned symbol. Since a computation has two senses of direction, left versus right on the tape and up (past) versus down (future) in the time dimension, the LCL will also force consistent senses of direction on the grid, at least in the part of the grid that contains the computation. We imagine the senses of direction as giving direction to the edges of the grid, from left to right, and from up to down (i.e., from past to future).

The construction of the labeling problem is not difficult conceptually, since it is well-known that the validity of a Turing machine computation can be checked locally. For definiteness, we describe one way of carrying out the details. A vertex label has the form $\langle\sigma,i,j\rangle$ where $\sigma\in Q\cup T\cup\{I\}$, $0\leq i\leq 2$, and $0\leq j\leq 1$. σ is called the *s-label* of the vertex (where s stands for "symbol"). Let v and v' be adjacent vertices with labels $\langle\sigma,i,j\rangle$ and $\langle\sigma',i',j'\rangle$. If $j=j'$ then there is a "horizontal" (left-to-right) edge from v to v' iff $i'=i+1 \bmod 3$. If $j\neq j'$ then there is a "vertical" (up-to-down) edge from v to v' iff $i'=i+1 \bmod 3$. The s-label I means that the vertex is "inactive", i.e., it is not in the part of the grid that contains the computation.

The LCL $\mathcal{L}$ enforces the following constraints:

1. The s-label of the special corner must be the initial state of M, and the two senses of direction must be directed away from this corner.

2. The senses of direction propagate correctly. This can be done, for example, by requiring the senses of direction to be consistent on every 3×3 subgrid. However, we do not require any sense of direction between two adjacent inactive vertices.

3. Each vertex on the upper boundary of the grid, other than the special corner, has s-label either I or the blank tape symbol.

4. In the vicinity of a state symbol, the computation must proceed according to the transition rules of M. However, we allow the state symbol to disappear if the head attempts to move off the right boundary or the bottom boundary of the grid.

5. In a neighborhood that does not contain a state symbol, each row must be identical to the row above it, except that vertices can become inactive. That is, if there are left-to-right edges from v_1 to v_2 and from v_2 to v_3, if there is an up-to-down edge from v_2 to v_4, and if none of v_1,v_2,v_3 are s-labeled by a state symbol, then the s-label of v_4 must be either the s-label of v_2 or I.

6. There is no up-to-down edge from an inactive vertex to an active vertex (i.e., once a tape cell becomes inactive, it cannot become active at a later time).

7. A non-halting state symbol cannot be adjacent to an inactive vertex.

Suppose that M halts in t steps when started on blank tape. Assume for the moment that the grid is $k \times l$ where $k, l \geq t+1$. Then there is a legal labeling where the s-labeling of the upper left $(t+1)\times(t+1)$ subgrid describes a halting computation of M on blank tape, and the other vertices are inactive. This labeling can be found by a local algorithm with time bound $2t+3$. This algorithm works as follows at a vertex v. If v lies within the $(t+2)\times(t+2)$ subgrid having the special corner as one of its corners, then the position of v in this subgrid is known, and the label of v can be found since it depends only on this position. Otherwise, v is labeled $\langle I, 0, 0\rangle$ (recall that the senses of direction do not have to be maintained within an inactive region). The argument for a smaller grid is similar (recall that the head can "move off" the grid at the right and bottom boundaries, so a legal labeling exists).

Suppose now that M does not halt. An argument similar to the one just given shows that a legal labeling exists (in particular, all vertices are active). Assume that a local algorithm A with time bound t finds a legal labeling for any id-numbering of any grid. Using Theorem 3.3, convert A to an order-invariant algorithm A' with time bound t. For a given grid, consider the id-numbering where each row is labeled from left to right in increasing order, and the id's used for row i are all smaller than those used for row $i+1$. Since A' is order invariant, A' will assign the same label to any two vertices v and w that are both farther than distance t from any boundary of the grid, since any two such points look the same to A'. Since M uses an infinite amount of tape, it is clear that the labeling produced by A' is not legal if the grid is sufficiently large.

(2). We now consider 2-dimensional grid graphs where no corner is "special", so all four corners look identical locally. A problem with the previous construction is that now there can be four computations, one starting at each corner. If M does not halt, or if the grid is too small, the senses of direction of these computations will conflict. The problem is solved by having four "levels". Now a label has the form $\langle \lambda_1, \lambda_2, \lambda_3, \lambda_4\rangle$ where each λ_i is a label as in part (1). The constraints that must hold at a special corner in part (1) now must hold at each corner, but only on one level. The arguments that a legal labeling always exists, and that M halts iff a legal labeling can be found in constant time, are essentially identical to those of part (1). In the case that M halts and the grid is so small that 2–4 computations overlap, the constant time algorithm uses the order of the id's at the relevant corners to decide which computation to put on which level.

(3). The next step is to consider a class of 4-regular graphs. These are grid graphs with extra edges added around the boundary to make the graph 4-regular. This can be done in such a way that the boundary vertices and the corner vertices can be identified locally. Call these graphs *4-regular grids.*

(4). Finally we consider the entire class of 4-regular graphs. Call a vertex a *defect* if it does not look locally like a 4-regular grid, i.e., for some suitably large (but constant) c, $B(v, c)$ is not consistent with a 4-regular grid. One problem with the previous construction is that now there can be many vertices that look locally like the corner of a 4-regular grid, so many different computations will be started and might conflict with one another, e.g., turn a halting computation into a non-halting one. This can occur, however, only if the graph has defects. The new idea is to propagate a chain of "erasing symbols" E from the defect back to the corner, so that the computation does not have to start.

More precisely, the s-symbols now include also E. For two adjacent vertices with s-symbol E, we use the component i of a label to give a direction to the edge connecting them (call these *E-edges*). We have the following constraints on vertices with s-label E: a defect can be labeled E; a corner can be labeled E iff it has exactly one E-edge directed in; any other vertex can be labeled E iff it has exactly one E-edge directed in and exactly one E-edge directed out. If a corner has s-label E, then its neighborhood does not have to have senses of direction.

Suppose that M does not halt and that the graph is a sufficiently large 4-regular grid. It is clear that no corner can be labeled E since the graph has no defects. The labeling could contain cycles of vertices labeled E, but this will violate other constraints if it occurs in the active region. It then follows as above that a legal labeling exists, but cannot be found in constant time.

If the graph is not a 4-regular grid, then it must have a defect. In this case, it can be seen that the entire graph can be s-labeled with E's and I's. Say that the defect u *kills* the corner w if there is a directed path of E-labeled vertices from u to w. By choosing the constant c above large enough, it can be seen that there is an E-labeling such that each defect kills at most one corner and E-labeled vertices in different paths are not adjacent.

Suppose that M halts in t steps. Assume for the moment that no two corners of the graph are within distance $4t + 4$ of each other. Consider the following labeling on one level. Fix some corner w. Say that w is a *good corner* if $B(w, 2(t+1))$ contains no defects (i.e., looks like part of a 4-regular grid). If w is good, then the s-labeling of the appropriate $(t+1)\times(t+1)$ subgrid describes a computation of M as in part (1). If $B(w, 2(t+1))$ contains a defect u, then we choose in some systematic way a path labeled E from one such defect u to w. The rest of $B(w, 2(t+1))$ is labeled I. Any vertex not within distance $2(t+1)$ from some corner is labeled I. Such a labeling can be found in time $O(t)$. If corners can be close together, it must be checked that four levels are enough to do the labeling. Define a graph where there is a vertex for every good corner, and an edge connecting two vertices if the computations started at the corresponding good corners overlap. It can be checked that no component of this graph has more than four vertices, so the labeling can be done on four levels, and a local algorithm can determine an assignment of good corners to levels.

For the case $d = 3$ we use, instead of 2-dimensional grids, degree-3 "honeycomb" graphs; these look like a tiling of the plane with hexagons. For $d \geq 5$, we can use $d - 3$ copies of the same grid graph, where corresponding vertices in different copies are connected as a clique. $\Box$

Remark. The LCL's constructed in the proof above have an upper bound r_0 on their radius where r_0 is a constant, i.e., it does not depend on the machine M. It follows that there is an infinite time hierarchy of LCL's with some fixed radius r_0. That is, for every time t, there is an LCL of radius r_0 that cannot be solved by any local algorithm with time bound t, but that can be solved by some local algorithm with some time bound $t' > t$.

In contrast, the following holds for degree 2.

Theorem 4.2 *Let $\mathcal{G}$ be the class of 2-regular graphs or the class of graphs of maximum degree 2. $Y(\mathcal{G})$ and $N(\mathcal{G})$ are recursively separable. Moreover, this can be done in time polynomial in the size of the input $\mathcal{L}$.*

Proof. Consider first the case of 2-regular graphs. Let $\mathcal{L}$ be a given LCL, and let r be its radius. If (H, s) belongs to the set $\mathcal{C}$ of locally consistent labelings, then either H is a simple path of $2r+1$ vertices with s at the center, or H is a ring having at most $2r$ vertices. Assuming that every graph in $\mathcal{G}$ has an $\mathcal{L}$-legal labeling, we claim that there is a local algorithm with some constant time bound t that solves $\mathcal{L}$ for $\mathcal{G}$ iff $\mathcal{C}$ contains a line segment in which all vertices are labeled the same, say α. The "if" direction is obvious, since any ring having at least $2r+1$ vertices has an $\mathcal{L}$-legal labeling where all vertices are labeled α. (In this case, a local algorithm with time bound r checks whether it is working on a cycle of size at least $2r+1$. If so, it produces the label α. If not, it knows the entire graph so it can use the rank of the id of the vertex to find a label for the vertex by looking in a table, where the table contains an $\mathcal{L}$-legal labeling for each cycle of size less than $2r+1$.) For the "only if" direction, by Theorem 3.3 there is an order-invariant A' with time bound t that solves $\mathcal{L}$ for $\mathcal{G}$. Consider the id numbering of a ring where the order of the id's increase around the ring, except at one point where the ordering wraps around. If the ring is sufficiently large, there will be a segment of length $2r+1$ such that A' gives the same label to every vertex of the segment.

The case of maximum degree 2 is a little more complicated. It is still necessary that $\mathcal{C}$ contain a line segment labeled the same, but this is no longer sufficient. Since the graph could be a line, we must also check that there is an $\mathcal{L}$-legal labeling in which all vertices, except possibly vertices within some bounded distance from the endpoints of the line, are labeled the same. This is easily reduced to a reachability problem on a certain graph K. Fix a left-to-right orientation of a line. For every member of $\mathcal{C}$ that is a line segment, there are two vertices in K, one for each left-to-right orientation of the segment. There is an edge directed from v to w iff there is a labeling of the oriented line in which the center of v is just to the left of the center of w. For example, if $r=2$, if v is the locally consistent labeling B–C–D where the center is the vertex labeled B and if w is B–C–D–X where the center is labeled C, then there is an edge from v to w. Any vertex such as v, for which the center is the leftmost endpoint of the segment, is called a *source*. A goal vertex is any vertex corresponding to a line segment containing $2r+1$ vertices all labeled the same. Then there is a local algorithm with some constant time bound t iff there is a directed path from some source to some goal. Note that if there is such a path then there is one of length $O(|\mathcal{C}|)$, so $t = O(\max\{|\mathcal{C}|, r\})$ suffices. $\square$

Remark. It can also be shown, for the graph classes $\mathcal{G}$ in Theorem 4.2, that it is decidable for a given LCL $\mathcal{L}$ whether every graph in $\mathcal{G}$ has an $\mathcal{L}$-legal labeling.

The final result of this section is an easy consequence of Theorem 3.3.

Theorem 4.3 *Fix any $d \geq 2$, and let $\mathcal{G}$ be the class of d-regular graphs or graphs of maximum degree d. It is decidable, given $\mathcal{L}$ and t, whether there is a local algorithm with time bound t that solves $\mathcal{L}$ for $\mathcal{G}$.*

Proof. By Theorem 3.3 we can restrict attention to order-invariant algorithms with time bound t. There are only a finite number of such algorithms, and for each algorithm A we can test whether it solves $\mathcal{L}$ for $\mathcal{G}$ as follows. Let r be the radius of $\mathcal{L}$. Let $\mathcal{H}$ be the set of centered graphs (H, s) such that, for some $G \in \mathcal{G}$ and some vertex v of G, H is isomorphic to $B_G(v, r+t)$ under an isomorphism mapping s to v. For the classes $\mathcal{G}$ under consideration, there are a finite number of such graphs for each r and t, and there is an algorithm that lists them given r, t. For each $(H, s) \in \mathcal{H}$ and each order-inequivalent id numbering of the vertices of H, we apply A to each vertex in $B_H(s, r)$ to obtain a labeling of $B_H(s, r)$. If this labeling is not consistent according to $\mathcal{L}$, then A is not correct, since A will fail at some vertex v of some $G \in \mathcal{G}$ where $B_G(v, r+t)$ is isomorphic to H; by "fail" we mean that the labeling of $B_G(v, r)$ is not consistent according to $\mathcal{L}$. On the other hand, if A always labels $B_H(s, r)$ correctly for every H and every numbering, then A is correct. For if A fails at some vertex v of some $G \in \mathcal{G}$, then A fails at v in $B_G(v, r+t)$ which is isomorphic to some H. □

5 Randomized algorithms

We now turn to randomized algorithms and show, for certain classes of graphs, that randomization does not help in solving LCL's in constant time. A *randomized local algorithm* P with time bound t is specified by a deterministic local algorithm A with time bound t and a function $b(n)$ to positive integers called the *randomization bound*. In this case, A expects each vertex to be labeled with an input label (if the LCL has input labels), an id number, and a random number. To run P on a graph G that is id-numbered and input-labeled, first randomly and independently choose for each vertex a random number in the range $[1, b(l)]$, where l is the largest id in G; then run A on the resulting graph. We assume no upper bound on the growth rate of $b(n)$. We say that P *solves* $\mathcal{L}$ *for* $\mathcal{G}$ *with error probability* ε if, for every input-labeled and id-numbered $G \in \mathcal{G}$, P produces an $\mathcal{L}$-legal labeling with probability at least $1 - \varepsilon$.

Remark. The above definition of a randomized local algorithm might seem too liberal, since it allows the range of randomization at a particular vertex v to depend on the largest id in the entire graph. It would be more reasonable to have the range of randomization at v depend only on v's id. But since our result is that randomization does not help in solving labeling problems locally, there is no harm in using the more liberal definition. On the other hand, this definition may seem too restrictive, since the definition of success is global. However, in the deterministic case we wanted the labeling to be legal everywhere, not just in most vertices. Indeed, if all we require is that, for each vertex v, the probability that v is legally labeled be at least $1 - \varepsilon$, then randomization does help: consider the problem of 3-coloring in a ring. This problem has no deterministic local algorithm [10] nor a probabilistic one [14] (with the global correctness requirement). Suppose that we start with all vertices uncolored and at every step each vertex that is not permanently colored chooses a random color. If the vertex chose a color different from the colors of its two neighbors, then this color is considered permanent. If this algorithm is executed for t steps, then we can say that, for each vertex v, the probability that v is legally colored is at least $1 - \varepsilon$, where ε decreases exponentially in t.

Theorem 5.1 *Fix an LCL $\mathcal{L}$ and a class $\mathcal{G}$ of graphs closed under disjoint union. If there is a randomized local algorithm P with time bound t that solves $\mathcal{L}$ for $\mathcal{G}$ with error probability ε for some $\varepsilon < 1$, then there is a deterministic local algorithm A with time bound t that solves $\mathcal{L}$ for $\mathcal{G}$.*

Proof. Suppose for contradiction that there is no deterministic local algorithm with time bound t that solves $\mathcal{L}$ for $\mathcal{G}$. In particular, there is no order-invariant algorithm with this property. There is an upper bound on the number of order-invariant local algorithms with time bound t (where the upper bound depends only on $\mathcal{L}$, t, and the degree bound d). This immediately proves the following:

Claim 5.1 *There is a number N such that every order-invariant local algorithm A' with time bound t fails on some particular input-labeled and id-numbered graph G having at most N vertices, where by "fail" we mean that A does not produce an $\mathcal{L}$-legal output labeling.*

Let R be the number given by Lemma 3.2. Let m be the minimum number of vertices in a graph in $\mathcal{G}$. If $R < N + m$, then take $R = N + m$ to ensure that $R \geq N + m$. For $j \geq 1$, let $S_j = \{(j-1)R+1, \ldots, jR\}$. Let $\mathcal{I}_j$ be the set of graphs $G \in \mathcal{G}$ having at most N vertices that are input-labeled and have id numbering drawn from S_j. If Σ is the input alphabet, an upper bound on the cardinality of $\mathcal{I}_j$ is

$$k = 2^{\binom{N}{2}} R^N |\Sigma|^N.$$

Choose q large enough that $(1 - \frac{1}{k})^q < 1 - \varepsilon$.

The key to the proof is the following:

Claim 5.2 *For every j with $1 \leq j \leq q$ there is a graph $G_j \in \mathcal{I}_j$ such that, if P is run on G_j with randomization bound $b(qR)$, then the probability that P fails on G_j is at least $1/k$.*

To prove the claim, suppose it is false, i.e., that for every $G \in \mathcal{I}_j$, P fails with probability strictly less than $1/k$. We can view a random choice of P as a sequence of random numbers $\rho_1, \ldots, \rho_R$ in the interval $[1, qR]$, where ρ_i is the random number chosen for the vertex with id $(j-1)R + i$. Since the error probability is less than the reciprocal of the number of graphs in $\mathcal{I}_j$, it follows by a standard argument (e.g., [1]) that there must be a particular choice $\hat{\rho}_1, \ldots, \hat{\rho}_R$ such that P is correct on all of $\mathcal{I}_j$ when this particular random choice is made. We can then obtain a deterministic algorithm A that works on $\mathcal{I}_j$. This algorithm first chooses the random number $\hat{\rho}_{i-(j-1)R}$ at the vertex with id i for every i, and then simulates P. By Lemma 3.2, there is an order-invariant A' that is correct on all of $\mathcal{I}_j$. But this contradicts Claim 5.1, and so proves Claim 5.2.

It is now easy to complete the proof of Theorem 5.1. Run P on the graph G consisting of the disjoint union of the G_j for $1 \leq j \leq q$. (If no vertex of this graph has label qR, then add another component with m vertices and maximum id label qR. This is possible since $qN + m \leq qR$.) Since P fails independently with probability at least $1/k$ on each G_j, it follows from the choice of q that P fails on G with probability strictly greater than ε. This contradiction proves the theorem. □

A version of Theorem 5.1 holds also for certain classes of connected graphs, for example, connected d-regular graphs and connected graphs of maximum degree d, for any fixed $d \geq 2$. All we need is the ability to connect together the graphs $G_1, \ldots, G_q$ into a single graph in the class in such a way that P's error probability on each piece does not decrease when the pieces are connected. For example, the following holds.

Theorem 5.2 *Fix an LCL $\mathcal{L}$ and a $d \geq 2$, and let $\mathcal{G}$ be the class of connected d-regular graphs or the class of connected graphs of maximum degree d. If there is a randomized local algorithm*

P with time bound t that solves $\mathcal{L}$ for $\mathcal{G}$ with error probability ε for some $\varepsilon < 1$, then there is a deterministic local algorithm A with time bound $2(t+r)+1$ that solves $\mathcal{L}$ for $\mathcal{G}$, where r is the radius of $\mathcal{L}$.

Proof. The proof is very similar to the previous one, and we only sketch the differences. Claim 5.1 is modified to state that there is an N such that every order-invariant local algorithm with time bound t fails on some connected graph with radius at least $t+r+1$ and with at most N vertices. For if not, there would be a local algorithm with time bound $2(t+r)+1$ that solves $\mathcal{L}$ for $\mathcal{G}$. This algorithm first checks whether it is working on a graph of radius at most $t+r$ by trying to inspect the entire graph. If so, it can produce a labeling because it knows the entire graph. If not, it recourses to an algorithm with time bound t that works on every connected graph of radius at least $t+r+1$. The set $\mathcal{I}_j$ now contains pairs (G, v) where G has at most N vertices and radius at least $t+r+1$ and where v is a vertex of G, so the bound k increases by a factor of N. The conclusion of Claim 5.2 is now that for every j there is a $(G_j, v_j) \in \mathcal{I}_j$ such that, with probability at least $1/k$, P fails on G_j at the particular vertex v_j, meaning that $B(v_j, r)$ is not labeled correctly. Since the radius of G_j is at least $t+r+1$, there is some edge e such that removing e does not affect the behavior of P when labeling $B(v_j, r)$. Let G'_j be the graph with this e removed. We can now connect together $G'_1, \ldots, G'_q$ to a graph in the class $\mathcal{G}$, using the endpoints of the removed edges as connection points. $\square$

Remark. An alternate conclusion in Theorem 5.2 is that there is a deterministic local algorithm A with time bound t that solves $\mathcal{L}$ for all graphs in $\mathcal{G}$ having radius at least $t+r+1$.

6 Weak coloring

We now describe a locally checkable labeling problem that can be solved locally in graphs containing only vertices of odd degree. A weak c-coloring of a graph is an assignment of numbers from $\{1, \ldots, c\}$ to the vertices of the graph such that for every non-isolated vertex v there is at least one neighbor w such that v and w receive different colors. Clearly weak c-coloring of a graph of degree at most d is an LCL problem of radius 1.

It is not hard to see that every graph has a weak 2-coloring: consider a breadth-first spanning tree of the graph. Assign one color to the even levels and a different color to the odd levels. However, this particular coloring cannot be computed locally. As we shall see, if all the vertices of the graph have odd degree, then it is possible to find a weak 2-coloring. As far as we know this is the first non-trivial LCL problem that has been shown to have a local algorithm. However, if the degree is even then it is impossible to compute such a coloring or any weak c-coloring for a fixed c locally.

6.1 Weakly coloring graphs of odd degree

We describe a way of finding a weak coloring in odd-degree graphs. We first show a two step method for weak $d(d+1)^{d+2}$-coloring and then show how to reduce the number of colors to two using additional steps.

Consider first the case of a d-regular graph where d is odd and $d \geq 3$. For a vertex v let $id(v)$ be the id number assigned to v. We denote the color of a vertex v by a vector $C_v = \langle C_v[0], C_v[1], \ldots, C_v[d+1] \rangle$ where each component is in $\{1, \ldots, d+1\}$. The following procedure is used at vertex v:

1. Get $id(w)$ for all neighbors w of v. Sort the set of id's of neighbors including $id(v)$. Let $r_v(w)$ denote the rank of $id(w)$ among the neighborhood of v (where the neighborhood of v includes v itself). For definiteness, say that the smallest id has rank 1, the second-smallest has rank 2, etc. Let $C_v[0]$ be $r_v(v)$.

2. Get $r_w(v)$ from each neighbor w, i.e., the rank of $id(v)$ among the neighborhood of w. Set $C_v[r_v(w)] = r_w(v)$.

Claim 6.1 *The coloring achieved by this algorithm is a legal weak coloring if d is odd.*

Proof. Consider a vertex v. If not for all neighbors w of v we have $r_w(w) = r_v(v)$, then we are done, since there will be a neighbor w of v such that the color of w differs from the color of v in the first component. Otherwise, there are two cases. In the first case, assume that $1 \leq r_v(v) \leq \frac{d+1}{2}$. This means that there are $d + 1 - r_v(v) \geq \frac{d+1}{2}$ neighbors w such that $id(w) > id(v)$. For each of them $r_w(v) < r_v(v)$, since $r_w(w) = r_v(v)$. Therefore, by the pigeonhole principle there are two neighbors w and x such that $r_w(v) = r_x(v) = j$. Hence

$$C_w[j] = C_w[r_w(v)] = r_v(w) \neq r_v(x) = C_x[r_x(v)] = C_x[j].$$

$C_w[j] \neq C_x[j]$ means that v has two neighbors with two different colors, one of which must be different than C_v. Similarly in the other case, if $\frac{d+1}{2} + 1 \leq r_v(v) \leq d+1$, then there are $r_v(v) - 1 \geq \frac{d+1}{2}$ neighbors w such that $id(w) < id(v)$. For each of them, $r_w(v) > r_v(v) \geq \frac{d+1}{2} + 1$, and a pigeonhole argument again shows that there must be two neighbors that are colored with two different colors. $\square$

If vertices can have different (odd) degrees, we can simply add another component to C_v which contains the degree of v. If v has a neighbor with a different degree, then it has a neighbor with a different color; otherwise, Claim 6.1 applies.

To go from $d(d+1)^{d+2}$ colors to two colors we employ two kinds of color reductions: one is a Cole-Vishkin [6] style that allows us to cut the number of colors logarithmically in every round, but seems to have its limit at four. The other method allows us to reduce the number of colors by one at a time.

The Cole-Vishkin style method is as follows. Suppose that we have a legal weak coloring with c colors and let c' be the smallest integer such that $\binom{c'}{\lfloor c'/2 \rfloor} \geq c$. Associate with every $i \in \{1, \ldots, c\}$ a different subset $S_i \subset \{1, \ldots, c'\}$ of size $\lfloor c'/2 \rfloor$. (Such an assignment is a Sperner system, i.e., no subset is contained in another.) We can reduce the number of colors from c to c' in one round. Every vertex v colored i finds a neighbor colored j such that $j \neq i$. There must be an element $x \in S_i$ such that $x \notin S_j$. x is v's next color. It is easy to see that this method preserves weak coloring and reduces the number of colors by almost a logarithmic factor per round. More precisely, the number c' of colors after a step of the reduction is related to the number c before the reduction by $c' = \log c + O(\log\log c)$ where logarithms are to the base 2. (This is (almost) a bit more efficient than the original Cole-Vishkin reduction, where the relation in our case is $c' = 2\lceil \log c \rceil$.) The method is applicable as long as $c > 4$. A simple calculation (cf. [8, pg. 437]) shows that a weak 4-coloring is found after $\log^* d + a$ rounds, for some constant a. (Another way to see this is to note that the base of the logarithm affects the expression $\log^* d + a$ only in the additive term a.)

When we are stuck (i.e., $c = 4$) we can recourse to the following reduction from a weak coloring with c colors $\{1, 2, ..., c\}$ (called the original coloring) to one with 2 colors $\{0, 1\}$ (called the recoloring). The recoloring is done in c rounds. At the ith round, every vertex with original color i recolors itself according to the following rules.

1. If v has original color i and all neighbors of v have original color $\geq i$, then v recolors itself 0.
2. Otherwise, v must have at least one neighbor with original color smaller than i, so it has at least one neighbor that has recolored itself at an earlier round. If all the recolored neighbors of v have color 1, then v recolors itself 0. Otherwise (v has at least one neighbor recolored 0), then v recolors itself 1.

It is easy to verify that this yields a weak 2-coloring. Every v that recolors itself using the second rule clearly has a neighbor recolored differently. Suppose that v recolors itself 0 using the first rule. Then it must have a neighbor w with original color $j > i$. Then w will recolor itself using the second rule during round j, and it will recolor itself 1 since it has a neighbor (namely, v) recolored 0 at an earlier round (namely, i). We therefore get:

Theorem 6.1 *Let $\mathcal{O}_d$ be the class of graphs of maximum degree d where the degree of every vertex is odd. There is a constant b such that, for every d, there is a local algorithm with time bound $\log^* d + b$ that solves the weak 2-coloring problem for $\mathcal{O}_d$.*

Remark. In Section 7 we will want to apply the weak coloring algorithm to graphs that may have vertices of even degree, and we will use the following additional property of the algorithm. Say that v is properly colored if it has at least one neighbor colored differently. Suppose that the weak 2-coloring algorithm is applied to an arbitrary (bounded degree) graph G. If v is not properly colored then (1) the degree d of v is even, (2) its rank $r_v(v)$ in its neighborhood is $d/2+1$, and (3) every neighbor w of v has degree d and rank $r_w(w) = d/2 + 1$ as well. To see that these properties hold, consider first the coloring produced by the initial two-step algorithm. Properties (1) and (2) follow since our proof of Claim 6.1 shows that v is properly colored if either $r_v(v) \leq \frac{d+1}{2}$ or $r_v(v) \geq \frac{d+1}{2} + 1$. The only other possibility is that d is even and $r_v(v) = d/2+1$. Since the color of a vertex u contains its degree and its rank $r_u(u)$, (3) is obvious. It is also easy to check that both of the color reduction methods preserve proper coloring, i.e., if v is properly colored before a reduction, then it is properly colored after the reduction.

To close this subsection we note that there is no one-step method for finding a weak c-coloring.

Theorem 6.2 *For any constants c and $d \geq 2$, there is no local algorithm with time bound 1 that solves weak c-coloring for the class of d-regular graphs.*

Proof. By Theorem 3.3, if there were such a local algorithm A there would be an order-invariant one A', also with time bound 1. Consider any d-regular graph that contains a vertex v such that $B(v, 2)$ (the neighborhood of radius 2 around v) is a tree of height 2 rooted at v. Number the vertices of $B(v, 2)$ with id's so that $r_v(v) = 2$ and $r_w(w) = 2$ for every neighbor w of v (it is easy to see that this can be done). Then A' assigns the same color to v and all its neighbors. □

6.2 Impossibility of weak coloring graphs of even degree

In this section we note that it is impossible in general to weakly color all graphs with even degree. In particular we show that for any c and k it is impossible to weakly c-color any class of graphs that contains the k-dimensional meshes. The vertex set of a k-dimensional mesh is $\{0, 1, \ldots, m\}^k$ for some m, and two vertices are connected by an edge if the L_1-distance between them is 1. A k-dimensional mesh has (some) vertices of even degree $d = 2k$.

Theorem 6.3 *For any c, k, and t, there is no local algorithm with time bound t that solves the weak c-coloring problem for the class of k-dimensional meshes.*

Proof. Theorem 3.3 says that if there exists a local algorithm for an LCL problem then there is one that uses only the relative order of the id's. For a vertex v of a mesh M, let $R_M(v, t)$ be the graph $B_M(v, t)$ (the neighborhood of radius t around v) where each vertex u is labeled with the rank of its id among the id's in $B_M(v, t)$. By Theorem 3.3 it is sufficient to come up with a way to assign id's to vertices such that for any t there will be a k-dimensional mesh M and a vertex v such that, for all neighbors u of v, $R_M(v, t)$ and $R_M(u, t)$ are the same. (I.e., v and its neighbors see the same relatively ordered t-neighborhood.) However, if M has diameter at least $2(t + 1)$ then it possible to achieve such an id assignment: consider the coordinates of a vertex and say that vertex u is larger than v if the lexicographical order of the coordinates of u is larger than that of v. It is clearly possible to assign id's such that $id(u) > id(v)$ iff u is larger than v. Hence any vertex that is of distance at least $t + 1$ from every boundary of the mesh has the property we are after. □

The same result holds for a class of $(2k)$-regular graphs, the k-dimensional analogue of torus graphs.

A consequence of this result is that if we extend the definition of weak coloring so that each vertex v must have at least 2 neighbors colored differently than v (call this *2-weak c-coloring*), then for every fixed d and c a coloring cannot be found in constant time for d-regular graphs even if d is odd. The reasoning is the following. Given any $(2k)$-regular graph G, form a $(2k + 1)$-regular graph G' by taking two copies of G with each pair of corresponding vertices connected by an edge. The id's in one copy are chosen all to be larger than the id's in the other copy, but so that the two copies appear identical with respect to the relative order within a copy. A 2-weak c-coloring of G' immediately gives a weak c-coloring in each copy of G. Given a local algorithm that finds a 2-weak c-coloring in graphs of odd degree $2k + 1$, we therefore obtain a local algorithm that finds a weak c-coloring in graphs of even degree $2k$.

7 A locally solvable resource allocation problem

We show how to solve the formal-dining philosophers problem mentioned in the Introduction. What we assume about the underlying graph is that the minimum degree is three. (If the minimum degree is two, then we cannot hope to solve it locally, as we argue below.)

We first start with a coloring with three colors $\{0, 1, *\}$ with the following property: all vertices colored $c \in \{0, 1\}$ have at least one neighbor colored $1 - c$. If v is colored with a $*$ or if any of the neighbors of v is colored with a $*$, then the degree of v is even and half the neighbors of v have an id smaller than $id(v)$. This coloring is a product of the method described in Section 6.1. Suppose that we run the algorithm described there. Since we do not assume here that every vertex has odd degree, the algorithm could *fail* at some vertices v, meaning that all the neighbors of v are colored the same as v. Suppose that if the algorithm fails at v, then v recolors itself with a $*$. By the remark following Theorem 6.1, the coloring fails at v only when the degree d of v is even, its rank $r_v(v)$ among its neighbors is $d/2 + 1$, every neighbor w of v has degree d, and $r_w(w) = d/2 + 1$ as well.

The algorithm for the formal-dining philosophers problem is a combination of two algorithms: one for the problem on graphs that are weakly 2-colored and the other for the case where half the neighbors of a vertex have a smaller id. The vertices colored $*$ essentially grab two of the adjacent cuff links permanently. More precisely, a vertex u colored $*$ picks two neighbors v and w such that $id(u) > id(v)$ and $id(u) > id(w)$, and assigns the cuff links on (u, v) and (u, w) to u permanently. After this we have that every vertex with color $c \in \{0, 1\}$ still has at least two non-assigned edges adjacent to it and at least one of its neighbors has color $1 - c$. This is true since if it is a neighbor of a $*$, then its degree is even (at least 4) and its rank among its neighbors is half the degree plus one. Hence at most half of its adjacent edges are grabbed permanently. Unlike the $*$ colored vertices, the $\{0, 1\}$ colored vertices must run a dynamic algorithm in order to get cuff links.

As for the $\{0, 1\}$ colored vertices, it is convenient for the exposition to first assume that we have a coloring of the graph with the property that every vertex has at least one neighbor colored 0 and at least one neighbor colored 1.[2] We will later remove this assumption. As a preliminary step, every vertex colored $c \in \{0, 1\}$ selects a particular neighbor colored c as its "first neighbor" and a particular neighbor colored $1 - c$ as its "second neighbor". When we say that a vertex p "requests" a cuff link, we mean that it tries to grab the cuff link; if the other vertex q sharing this cuff link currently has it, then p waits for q to release it. Now the protocol for a vertex colored $c \in \{0, 1\}$ is:

1. Request cuff link from the first neighbor (colored c).
2. Request cuff link from the second neighbor (colored $1 - c$).
3. Eat.
4. Release cuff links.

Claim 7.1 *The maximum length of a waiting chain in the above protocol is* 2.

[2] Though by the Lovasz Local Lemma such a coloring exists in regular graphs with sufficiently large degrees, it is impossible to find such a coloring locally even in odd-degree graphs.

Proof. If a vertex is waiting at Step 1, then the vertex it is waiting for must be at least at Step 2. If a vertex v is waiting for its second neighbor w at Step 2, then v and w are colored differently, which means that v is the second neighbor of w. So w must be at Step 3 or 4. □

Suppose now that all we can say is that a vertex colored $c \in \{0,1\}$ has at least one neighbor colored $1-c$, i.e., all its neighbors might be colored $1-c$. If at Steps 1 and 2, arbitrary neighbors colored $1-c$ are approached, then we are not guaranteed to be deadlock free anymore. The selection of second neighbors should be done in a way that does not induce long "neighborly" chains. Towards this end, we differentiate between the vertices colored 0 and 1. Each vertex colored 1 chooses a particular neighbor colored 0 as its second neighbor. These choices are announced to their neighbors. A vertex u colored 0 waits to hear whether it has been chosen as the second neighbor by any of its neighbors. If it has, then it tries to match their choices. I.e., if any of u's neighbors has designated it as a second neighbor, u picks it (or one of them in case there are several) as u's second neighbor. Otherwise, u chooses an arbitrary neighbor colored 1 as its second neighbor. Each vertex colored 0 or 1 then chooses an arbitrary neighbor, other than its second neighbor, to be its first neighbor. (Of course, u should not choose a neighbor w colored $*$ if w has permanently grabbed the cuff link on the edge (u,w). On the other hand, if w is colored $*$ and has not grabbed the cuff link then it never will, so u can choose this w, and u will never have to wait for w.)

Claim 7.2 *Given any assignment of first and second neighbors consistent with the above description, the maximum length of a waiting chain is at most* 4.

Proof. A configuration that the preliminary step as described above assures won't occur is as follows: three vertices w_1, w_2 and w_3 colored $1, 0$ and 1, respectively, such that w_2 is the first neighbor of w_3, w_2 is the second neighbor of w_1, and w_3 is the second neighbor of w_2. This cannot occur since w_2 was chosen to be a second neighbor of at least one vertex (namely, w_1), but w_2 is not the second neighbor of w_3. Hence w_2 would not choose w_3 as its second neighbor. Consider now a contradiction to the claim, i.e., six vertices $u_0, u_1, u_2, u_3, u_4, u_5$ such that each u_i is waiting for u_{i+1} for $0 \le i \le 4$. Let c be the color of u_1. Since u_1, u_2, u_3 and u_4 are waiting at Step 2, it must be the case that u_2 is colored $1-c$, u_3 is colored c, u_4 is colored $1-c$, and u_5 is colored c. Also for $1 \le i \le 3$ we have that u_i is the first neighbor of u_{i+1}, and for $1 \le i \le 4$ we have that u_{i+1} is the second neighbor of u_i. Therefore if $c = 0$ then the trio $\{u_2, u_3, u_4\}$ constitutes a forbidden configuration, and if $c = 1$ then $\{u_1, u_2, u_3\}$ constitutes a forbidden configuration.

(We remark that a waiting chain of length 4 can occur.) □

Since this argument remains valid if some of the u_i are the same, the argument shows that a deadlock (a waiting cycle) cannot occur, since a waiting cycle would produce, in effect, a waiting chain of arbitrary length.

Therefore the combined protocol is:

- Run the coloring algorithm of Section 6.1 resulting in a $\{0,1,*\}$ coloring.
- All the vertices colored $*$ permanently grab two cuff links as described above.
- All $\{0,1\}$ colored vertices find first and second neighbors as described above.

- When a vertex becomes hungry, then if it is colored $*$ it simply uses its permanently assigned cuff links. Otherwise it runs the protocol above.

Theorem 7.1 *The above protocol solves the formal-dining philosophers problem locally.*

A consequence of bounded-length waiting chains is that the failure locality of the protocol is constant. As defined by Choy and Singh [5], a protocol has *failure locality* l if every vertex v remains starvation free even if processors at distance larger than l from v fail. Another consequence is that the response time of the protocol is constant. This means that, for every upper bound ν on the message delivery time between adjacent vertices and every upper bound τ on the time that a vertex can remain in the eating state before entering the resting state, there is a $\rho = \rho(\nu, \tau)$ such that ρ is an upper bound on the time that a vertex can remain in the hungry state before entering the eating state. We now justify our requirement that the conflict graph have minimum degree three, by arguing that if the conflict graph is a ring then the formal-dining philosophers problem (which is the same as the usual dining philosophers problem in this case) cannot be solved locally, meaning in particular that the response time is constant. This follows from a more general result about the local unsolvability of certain dining philosophers problems where the condition under which a philosopher can eat is a threshold condition on the number of resources owned. For constants d and m with $d \geq 2$ and $m \leq d$, define the (d, m)-*dining philosophers problem* as follows: the conflict graph has minimum degree d; and in order to eat, a philosopher must own the resources on any m incident edges.

Theorem 7.2 *If* $m \geq \lceil d/2 \rceil + 1$, *there is no algorithm with constant response time for the* (d, m)-*dining philosophers problem.*

Proof. It suffices to prove this for every even d. Fix some $d = 2k$. We assume that the interconnection graph is the same as the conflict graph, a k-dimensional torus (i.e., a k-dimensional mesh with additional wrap-around edges to produce a d-regular graph). We first show that a solution to the (d, m)-dining philosophers problem with constant response time would give a local solution to the following LCL problem, for some constant p depending only on d:

1. Each vertex is labeled either 1 or 2;

2. Each k-dimensional $p \times p \times \cdots \times p$ submesh M contains some vertex labeled 1 and some vertex labeled 2.

Given an arbitrary id-numbered torus, run the assumed (d, m)-dining philosophers algorithm starting in the configuration where all vertices are initially hungry. Let the vertices operate in lock-step synchrony, i.e., each message delay is one time unit. When a vertex enters the eating state, let it remain in the eating state for one time unit, and remain in the resting state thereafter. So $\nu = \tau = 1$. The following rules are used to determine the label of vertex v. Let s be the step at which v enters the eating state. If v has not received the message "eating" from one of its neighbors at step s or earlier, then v chooses the label 1, and sends the message "eating" to all of its neighbors at step s (in addition to any messages that the (d, m)-dining philosophers algorithm sends at this step);

the "eating" message is received by v's neighbors at the next step $s+1$. Otherwise (v received an "eating" message at step s or earlier), then v chooses the label 2 and does not send an "eating" message. The labeling algorithm runs in constant time $\rho(1,1)$. It remains to show that condition 2 above holds for a large enough p. Assume that $p \geq 3$. If there is a $p\times\cdots\times p$ submesh M with all vertices labeled 2, then there would be some v such that v and all of its neighbors are labeled 2. But this is impossible since, if all the neighbors of v are labeled 2, they do not send "eating" to v, so v will be labeled 1 at the step when it eats. If there is a $p\times\cdots\times p$ submesh M with all vertices labeled 1, then all vertices of M enter the eating state at the same step s. This gives a contradiction as follows. The number of edges incident on vertices of M is at most $(d/2)p^k + O(p^{k-1})$. But in order for each vertex of M to own at least m of these edges, there must be at least mp^k of them. Since $m \geq d/2+1$ by assumption, we obtain a contradiction by choosing p large enough.

It is now easy to prove that this LCL cannot be solved locally for k-dimensional torus graphs. The proof is similar to that of Theorem 6.3. As before, it suffices to consider order-invariant algorithms. For the id-numbering described in the proof of Theorem 6.3, every sufficiently large torus will have a $p\times\cdots\times p$ submesh M such that every vertex of M receives the same label. But this contradicts the definition of the LCL. □

In particular, the (d,d)-dining philosophers problem is not solvable with constant response time for any $d \geq 2$.

Alain Mayer and the authors [12] have proved the converse of Theorem 7.2: If $m \leq \lceil d/2 \rceil$, then there is a protocol with constant response time for the (d,m)-dining philosophers problem.

8 Open questions

This is an early attempt to study what can and cannot be computed locally, and many questions remain open. A general direction for future work is to obtain more information about what sorts of labeling problems and resource allocation problems can be solved locally. In particular, the following specific questions are suggested by our work.

1. Consider the problem of assigning an orientation to some edges of the graph so that every vertex has either no edge directed in or two edges directed in, and the assignment is maximal with respect to the number of vertices that have two edges directed in. This problem (the *maximal in-degree 2 problem*) was suggested by the formal-dining philosophers problem. In the case that all philosophers are initially hungry, such an orientation corresponds to an assignment of cuff links that is maximal with respect to the number of philosophers who are eating. We can show that this problem cannot be solved in constant time for d-regular graphs where $d \leq 4$. Can it be solved in constant time for some $d \geq 5$?

2. We have shown that a weak 2-coloring can be found in time $O(\log^* d)$ in odd-degree graphs of maximum degree d. Is this the best possible time as a function of d? Or is there some fixed time t that is sufficient for all d?

3. We have shown that a weak c-coloring cannot be found in constant time for certain graphs having vertices of even degree (meshes). Does the same hold for trees where every non-leaf has

even degree? We conjecture that the result holds for any class of even-degree edge-transitive graphs. Is this conjecture true?

Some other questions are addressed in [12]:

1. The locality framework is extended to dynamic graphs, where edges can fail and later recover, and new nodes and edges can be added to the graph.

2. The amount of initial symmetry-breaking needed to solve certain problems locally is investigated.

Acknowledgments

We thank Ron Fagin for discussions regarding the relevance of locality in first-order logic, Shay Kutten for pointing out the relevance of locality in work on self-stabilization, and Seffi Naor for suggesting the formal-dining metaphor.

References

[1] L. Adleman, *Two theorems on random polynomial time*, in Proc. 19th IEEE Symposium on Foundations of Computer Science, 1978, pp. 75–83.

[2] Y. Afek, S. Kutten, and M. Yung, *Local detection for global self stabilization*, manuscript; preliminary version in Proc. 4th Workshop on Distributed Algorithms, Lecture Notes in Computer Science, Vol. 486, Springer-Verlag, New York, 1991, pp. 15–28.

[3] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin, *Network decomposition and locality in distributed computation*, in Proc. 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 364–369.

[4] K. M. Chandy and J. Misra, *The drinking philosophers problem*, ACM Trans. Programming Languages and Systems, 6 (1984), pp. 632–646.

[5] M. Choy and A. K. Singh, *Efficient fault tolerant algorithms for resource allocation in distributed systems*, in Proc. 24th ACM Symposium on Theory of Computing, 1992, pp. 593–602.

[6] R. Cole and U. Vishkin, *Deterministic coin tossing with applications to optimal parallel list ranking*, Inform. and Control, 70 (1986), pp. 32–53.

[7] G. N. Frederickson and N. A. Lynch, *Electing a leader in a synchronous ring*, J. Assoc. Comput. Mach., 34 (1987), pp. 98–115.

[8] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon, *Parallel symmetry-breaking in sparse graphs*, SIAM J. Disc. Math., 1 (1988), pp. 434–446.

[9] R. L. Graham, B. L. Rothschild and J. H. Spencer, *Ramsey Theory*, Wiley, New York, 1980.

[10] N. Linial, *Locality in distributed graph algorithms*, SIAM J. Comput., 21 (1992), pp. 193–201.

[11] N. A. Lynch, *Upper bounds for static resource allocation in a distributed system*, J. Comput. System Sci., 23 (1981), pp. 254–278.

[12] A. Mayer, M. Naor, and L. Stockmeyer, *Local computations on static and dynamic graphs*, in Proc. 3rd Israel Symposium on Theory of Computing and Systems, 1995, pp. 268–278.

[13] S. Moran, M. Snir, and U. Manber, *Applications of Ramsey's theorem to decision tree complexity*, J. Assoc. Comput. Mach., 32 (1985), pp. 938–949.

[14] M. Naor, *A lower bound on probabilistic algorithms for distributive ring coloring*, SIAM J. Disc. Math., 4 (1991), pp. 409–412.

[15] A. Panconesi and A. Srinivasan, *Improved distributed algorithms for coloring and network decomposition problems*, in Proc. 24th ACM Symposium on Theory of Computing, 1992, pp. 581–592.

[16] F. P. Ramsey, *On a problem of formal logic*, Proc. London Math. Soc., 2nd Ser., 30 (1930), pp. 264–286.

[17] E. Styer and G. L. Peterson, *Improved algorithms for distributed resource allocation*, in Proc. 7th ACM Symposium on Principles of Distributed Computing, 1988, pp. 105–116.

[18] A. C. Yao, *Should tables be sorted?*, J. Assoc. Comput. Mach., 28 (1981), pp. 615–628.