LCL problems on grids

Sebastian Brandt · sebastian.brandt@tik.ee.ethz.ch · ETH Zürich

Juho Hirvonen · juho.hirvonen@aalto.fi · IRIF, CNRS and University Paris Diderot

Janne H. Korhonen · janne.h.korhonen@aalto.fi · Aalto University

Tuomo Lempiäinen · tuomo.lempiainen@aalto.fi · Aalto University

Patric R. J. Östergård · patric.ostergard@aalto.fi · Aalto University

Christopher Purcell · christopher.purcell@aalto.fi · Aalto University

Joel Rybicki · joel.rybicki@helsinki.fi · University of Helsinki

Jukka Suomela · jukka.suomela@aalto.fi · Aalto University

Przemysław Uznański · przemyslaw.uznanski@inf.ethz.ch · ETH Zürich

Abstract. LCLs or locally checkable labelling problems (e.g. maximal independent set, maximal matching, and vertex colouring) in the LOCAL model of computation are very well-understood in cycles (toroidal 1-dimensional grids): every problem has a complexity of $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$, and the design of optimal algorithms can be fully automated.

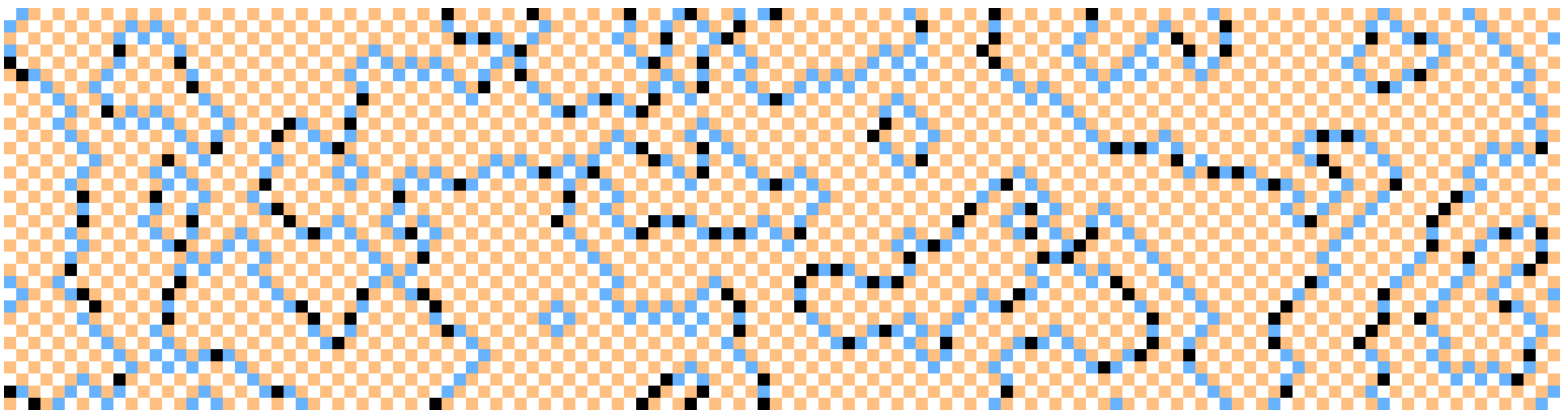
This work develops the complexity theory of LCL problems for toroidal 2-dimensional grids. The complexity classes are the same as in the 1-dimensional case: $O(1)$, $\Theta(\log^* n)$, and $\Theta(n)$. However, given an LCL problem it is undecidable whether its complexity is $\Theta(\log^* n)$ or $\Theta(n)$ in 2-dimensional grids.

Nevertheless, if we correctly guess that the complexity of a problem is $\Theta(\log^* n)$, we can completely automate the design of optimal algorithms. For any problem we can find an algorithm that is of a normal form $A' \circ S_k$, where A' is a finite function, S_k is an algorithm for finding a maximal independent set in kth power of the grid, and k is a constant.

With the help of this technique, we study several concrete LCL problems, also in more general settings. For example, for all $d \geq 2$, we prove that:

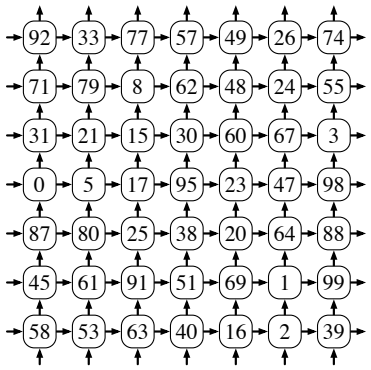
- d-dimensional grids can be k-vertex coloured in time $O(\log^* n)$ iff $k \geq 4$,
- d-dimensional grids can be k-edge coloured in time $O(\log^* n)$ iff $k \geq 2d + 1$.

The proof that 3-colouring of 2-dimensional grids requires $\Theta(n)$ time introduces a new topological proof technique, which can also be applied to e.g. orientation problems.



1 Introduction

1.1 Problem setting: LCL problems on grids



Grids. In this work, we study distributed algorithms in a setting where the underlying input graph is a grid. Specifically, we consider the complexity of locally checkable labelling problems, or LCL problems, in the standard LOCAL model of distributed complexity, and consider graphs that are toroidal two-dimensional $n \times n$ grids with a consistent orientation; we focus on the two-dimensional case for concreteness, but most of our results generalise to d-dimensional grids of arbitrary dimensions.

This setting occupies a middle ground between the well-understood directed n-cycles [9, 30], where all solvable LCL problems are known to have deterministic time complexity either $O(1)$, $\Theta(\log^* n)$ or $O(n)$, and the more complicated setting of general n-vertex graphs, where intermediate problems with time complexities such as $\Theta(\log n)$ are known to exist, even for bounded-degree graphs. Grid-like systems with local dynamics also occur frequently in the study of real-world phenomena. However, grids have so far not been systematically studied from a distributed computing perspective.

LOCAL model and LCL problems. In the LOCAL model of distributed computing, nodes are labelled with unique numerical identifiers with $O(\log n)$ bits. A time-t algorithm in this model is simply a mapping from radius-t neighbourhoods to local outputs; equivalently, it can be interpreted as a message-passing algorithm in which the nodes exchange messages for t synchronous rounds and then announce their local outputs.

LCL problems are graph problems for which the feasibility of a solution can be *verified* by checking the solution for each $O(1)$-radius neighbourhood; if all local neighbourhoods look valid, the solution is also globally valid. Examples of such problems include vertex colouring, edge colouring, maximal independent sets, and maximal matchings. We refer to Section 3 for precise definitions.

Example: colouring the grid. To illustrate the type of questions we are interested in this work, consider k-colouring on $n \times n$ grids. For $k = 2$, the problem is inherently global with complexity $\Theta(n)$, while colouring any graph of maximum degree $\Delta = 4$ with $\Delta + 1 = 5$ colours can be done in $O(\log^* n)$ rounds. But what about $k = 3$ and $k = 4$? In particular, does either of these have an intermediate (polylogarithmic) complexity, as is known to happen with Δ-colouring on general bounded-degree graphs [9, 32]? We show that neither 3-colouring nor 4-colouring is intermediate on grids: 3-colouring requires $\Theta(n)$ rounds, while 4-colouring can be solved in $O(\log^* n)$ rounds.

1.2 Results: classification and synthesis

Classification. Our first contribution is a complete *complexity classification* for LCL problems on grids in the case of deterministic algorithms. That is, we show that any LCL problem on $n \times n$ grids has one of the following time complexities, similarly to the case of cycles:

(a) $O(1)$ ("trivial" problem)
(b) $\Theta(\log^* n)$ ("local" problem)
(c) $\Theta(n)$ ("global" problem)

In particular, there are no problems of an intermediate complexity, such as $\Theta(\log n)$.

The separation between $O(1)$ and $\Omega(\log^* n)$ follows directly from the work of Naor and Stockmeyer [30], and obviously all problems can be solved in $O(n)$ rounds on $n \times n$ grids (assuming they can be solved at all). The interesting part is the separation between (b) and (c); here we extend the recent speed-up lemma of Chang et al. [9] to grids.

Undecidability of classification. It is known that the classification for LCL problems on cycles is decidable, that is, there is an algorithm that decides to which complexity class a given LCL problem belongs (see Section 4). We show that two-dimensional grids are fundamentally different from cycles in this regard: even if we have a promise that a given LCL problem has complexity of either $\Theta(\log^* n)$ or $\Theta(n)$, distinguishing between these cases is undecidable.

Algorithm synthesis for $\Theta(\log^* n)$ problems. The undecidability result would seem to suggest that automating the design of distributed algorithms on $n \times n$ grids is essentially impossible. Surprisingly, this is not the case: we develop a *synthesis algorithm* that, given a specification of an LCL problem P with complexity $O(\log^* n)$, produces an asymptotically optimal algorithm for P on grids. The caveat is that if the input problem P is a global problem with complexity $\Theta(n)$, this algorithm cannot detect it and will never stop.

From a theory perspective, this means that for each LCL problem P we will only need 1 bit of advice—whether P is $O(\log^* n)$ or $\Theta(n)$—and then we can find an optimal algorithm for solving P: for $O(\log^* n)$ problems, we apply the synthesis algorithm, and for $\Theta(n)$ problems, brute force is optimal. From a practical perspective, we can use the synthesis algorithm as a one-sided oracle for understanding the complexity of LCL problems on grids: if the synthesis produces an output, we have an optimal algorithm, and if it does not, we can conjecture that the problem in question might be inherently global.

Normal form for $\Theta(\log^* n)$ problems. The algorithm synthesis is based on a result showing that every LCL problem P with complexity $\Theta(\log^* n)$ on $n \times n$ grids has an algorithm of a specific *normal form*; see Figure 1. That is, there is an algorithm A for P that has the form $A = A' \circ S_k$ for some constant k, where

- S_k is a problem-independent algorithm that finds a maximal independent set I_k in the kth power of the $n \times n$ grid (we call these nodes "anchors"), and
- A' is a problem-dependent algorithm with running time $O(k)$ that takes as an input only the set of anchors I_k and the global orientation of the grid.

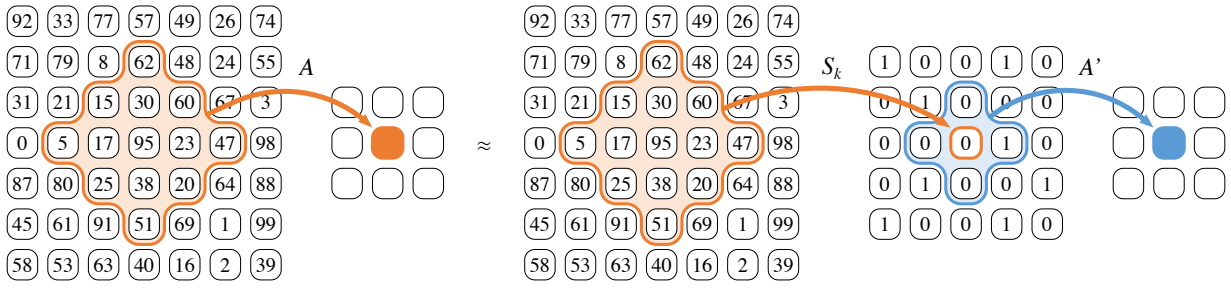


Figure 1: Any sublinear-time algorithm A can be normalised as $A' \circ S_k$, where S_k is a problem-independent $O(\log^* n)$-time symmetry-breaking component and A' is a problem-specific constant-time component.

Note that here only the part of finding the set of anchors takes $\Theta(\log^* n)$ time, and all of the remaining parts can be done in $O(1)$ time. In particular, the only problem-dependent part besides the constant k is the finite function defining the algorithm A'; thus, the algorithm synthesis becomes a matter of searching through the finite-size space of possible functions.

1.3 Results: upper and lower bounds for concrete LCL problems

Next, we turn our attention to concrete LCL problems. In particular, we are interested in problem families defined for a range of parameters, so that it makes sense to ask where exactly is the border between local and global problems.

- *Vertex colouring.* For k-colouring, we show that the problem is solvable in $O(\log^* n)$ rounds for $k \geq 4$ and global for $k \leq 3$. In the lower bound proof for $k = 3$ we introduce a new proof technique that has a topological flavour.
- *Edge colouring.* For k-edge colouring, we show that the problem is solvable in $O(\log^* n)$ rounds for $k \geq 5$ and global for $k \leq 4$.
- *Edge orientations.* For set $X \subseteq \{0, 1, 2, 3, 4\}$, an X-orientation is an orientation of the edges such that for each node $v \in V$ we have in-deg$(v) \in X$. The problem is trivial if $2 \in X$. We show that if $\{0, 1, 3\} \subseteq X$ or $\{1, 3, 4\} \subseteq X$, the problem is solvable in $O(\log^* n)$ rounds, and otherwise it is global. Here the lower bound proof relies on the same idea as the lower bound proof for 3-colouring.

The results on colourings can be generalised to d-dimensional grids. We show that a 4-colouring can be found in time $\Theta(\log^* n)$, while 3-colouring is global for any $d \geq 2$. In the case of edge colouring, we show that a $(2d + 1)$-colouring can be found in time $\Theta(\log^* n)$, for all dimensions $d \geq 1$, while $2d$-colouring is global. Both of the upper bounds hold even without any orientation or dimensional information, while both of the lower bounds hold even with full information.

2 Related work

LCL problems on cycles. As we noted before, two-dimensional grids can be seen as a generalisation of the widely studied setting of cycles; indeed, LCL problems were first studied on cycles in the distributed setting. Cole and Vishkin [12] showed that cycles can be 3-coloured in time $O(\log^* n)$, and Linial [28] showed that this is asymptotically optimal. This implies, via simple reductions, that many classical LCL problems, such as maximal independent set and maximal matching, also have a complexity of $\Theta(\log^* n)$ on cycles.

LCL problems on graphs of bounded maximum degree. Naor and Stockmeyer [30] showed that there exists a non-trivial LCL problem that can be solved in constant time: weak 2-colouring on graphs of odd degree. Many LCL problems are known to either have complexity $\Theta(\log^* n)$ [3, 5, 16, 31] or be global on graphs of bounded maximum degree. Until recently, no problems of an intermediate complexity were known. While Kuhn et al. [26] gave a lower bound of $\min\{\log \Delta / \log\log \Delta, \sqrt{\log n / \log\log n}\}$ for, among others, maximal independent set, this proof does not give an infinite family of graphs with a fixed maximum degree Δ. Brandt et al. [8] showed that sinkless orientation and Δ-colouring have randomised complexity $\Omega(\log\log n)$, and Chang et al. [9] proved that this implies a deterministic lower bound of $\Omega(\log n)$. These lower bounds provide the first examples of LCL problems with provably intermediate time complexity. Ghaffari and Su [20]

proved a matching upper bound for sinkless orientation; no tight bounds are known for Δ-colouring, but there is a polylogarithmic upper bound due to Panconesi and Srinivasan [32].

Complexity theory of LCL problems. LCL problems were formally introduced by Naor and Stockmeyer [30]. They showed that if there exists a constant-time algorithm for solving an LCL problem P, then there exists an order-invariant constant-time algorithm for P, such that the algorithm only uses the relative order of unique identifiers given to the nodes. Their argument works for any time $t = o(\log^* n)$: a time-t distributed algorithm implies a constant-time order-invariant algorithm; hence there are no LCL problems with complexities strictly between $\omega(1)$ and $o(\log^* n)$.

Recently Chang et al. [9] showed that there are further gaps in the time complexities of LCL problems. They gave a speed-up lemma for simulating any deterministic $o(\log n)$-time algorithm in time $O(\log^* n)$ by computing new small and locally unique identifiers for the input graph. This implies that there are no LCL problems with deterministic complexity $\omega(\log^* n)$ and $o(\log n)$. They also show that the deterministic complexity of an LCL on instances of size n is at most the randomised complexity on instances of size 2^{n^2}. This implies a similar gap for randomised complexities between $\omega(\log^* n)$ and $o(\log \log n)$.

LCL problems in restricted graph families. It appears that the complexity of LCL problems specifically on grids has not been studied beyond the case of cycles. LCL problems have been, however, studied on other restricted graph classes, such as graphs of bounded independence [4, 19, 25, 37], bounded growth [38] and bounded diversity [6].

Existence of algorithms and algorithm synthesis. The notion of automatic synthesis of algorithms has been around for a long time; for example, already in the 1950s Church proposed the idea of synthesising circuits [10, 41]. Since then synthesis of distributed and parallel protocols has become a well-established research area in the formal methods community [1, 7, 11, 15, 27, 29, 35]. However, synthesis has received considerably less attention in the distributed computing community, even though they have been used to discover e.g. novel synchronisation algorithms [2, 7, 13] and local graph algorithms [22, 36].

The synthesis of optimal distributed algorithms in general is often computationally hard and even undecidable. In the context of the LOCAL model and LCL problems, Naor and Stockmeyer [30] show that simply deciding whether a given problem can be solved in constant time is undecidable; hence we cannot expect to completely automate the synthesis of asymptotically optimal distributed algorithms for LCL problems in general graphs. This result holds even if we study non-toroidal two-dimensional grids, but it does not hold in toroidal grids. In essence, in toroidal grids only trivial problems are solvable in constant time, and as we will see, the interesting case is the time complexity of $O(\log^* n)$.

Other grid-like models. While grids have not been studied from a distributed computing perspective, grid-like models with local dynamics have appeared in many different contexts:

- *Cellular automata* [18, 42, 45] have been studied both as a primitive computational model, and as a model for various complex systems and emergent phenomena, e.g. in ecology, sociology and physics [17, 23, 39].
- Various *tiling models* [21] have connections to computability questions, such as the abstract *Wang tilings* [43] and the variants of the *abstract Tile Assembly Model (aTAM)* [14, 33, 44, 46] for DNA self-assembly.

However, the prior work of this flavour is usually interested in understanding the dynamics of a specific fixed process, or what kind of global behaviours can arise from fixed number of local states—in particular, whether the system is computationally universal. Our distributed complexity perspective to grid-like systems seems mostly novel, and we expect it to have implications in other fields. Applying an existing result of distributed computing to tiling models has been previously demonstrated by Sterling [40], who makes use of a weak-colouring lower bound by Naor and Stockmeyer [30].

3 Preliminaries

LOCAL model. In the LOCAL model of distributed computing [28, 34], we have a computer network that is represented as a graph $G = (V, E)$; each node is a computer and each edge is a bidirectional communication link. The computers collaborate in order to solve a graph problem related to the structure of the graph G; note that here the same graph is both the topology of the computer network and the input graph.

Each node $v \in V$ is labelled with a unique identifier from the set $\{1, 2, \ldots, \mathrm{poly}(|V|)\}$. Each node has to produce its own part of the output: for example, if we are solving a graph colouring problem, each node has to output its own colour, and if our task is to find a maximal independent set I, each node has to output a binary label that indicates whether it is in set I. This can be extended in a straightforward manner to edge labellings.

All nodes run the same deterministic algorithm. Computation proceeds in synchronous rounds. In each round, all nodes in parallel send messages (of an arbitrary size) to their neighbours, then the messages are propagated along the edges to the recipients, then all nodes in parallel receive messages from each of their neighbours, and finally all nodes update their local state. The running time of an algorithm is the number of communication rounds until all nodes have stopped and announced their local outputs.

Note that in a time-t algorithm, each node can gather its radius-t neighbourhood and choose its local output based on this information. In essence, a time-t algorithm in the LOCAL model is simply a mapping from radius-t neighbourhoods to local outputs. Note that the neighbourhood contains not only the topology of the network but also the unique identifiers.

LCL problems. In distributed algorithms, the class of LCL problems [30] plays a role somewhat analogous to the class NP in centralised computing. Informally, problems in the class LCL are precisely those problems that can be solved in *constant time* with a *nondeterministic* algorithm in the LOCAL model: in LCL problems all nodes can nondeterministically guess the solution and then verify it by checking that the solution looks consistent in all local neighbourhoods. The key question is which of the LCL problems can be solved efficiently (e.g., in constant or near-constant time) with *deterministic* algorithms. (Cf.: which problems in NP are also in P.)

More precisely, let P be a graph problem that associates with each unlabelled input graph $G = (V, E)$ a set of feasible node labellings $P(G)$; here each $f \in P(G)$ is a mapping $f \colon V \to X$ for some set of output labels X. We say that P is an LCL problem if

(1) the set of local outputs X is a finite set of size $|X| = O(1)$,
(2) there is a constant $r = O(1)$ such that for any candidate labelling $f \colon V \to X$ we have $f \in P(G)$ if and only if each radius-r neighbourhood is compatible with some $g \in P(G)$.

Informally, for an LCL problem, a solution is feasible if it looks like a feasible solution in all local neighbourhoods. Examples of such problems include k-vertex colouring for $k = O(1)$, maximal

independent sets, and minimal dominating sets. Again, we can extend the definitions in a natural manner to edge labellings; hence also k-edge colourings for $k = O(1)$, maximal matchings, and minimal edge dominating sets can be interpreted as LCL problems.

Radius-1 LCL problems. Above, parameter r is called the *checkability radius* or simply *radius* of problem P. In bounded-degree graphs we can always define another LCL problem P' with radius $r' = 1$ that is equivalent to P in the following sense: P' can be solved in time t in the LOCAL model if and only if P can be solved in time $t \pm O(1)$. In essence, the output labels in P' are radius-r neighbourhoods in P; given an algorithm for P we can spend additional r rounds to solve P', and given an algorithm for P', we can also directly solve P. Therefore we will often tacitly assume $r = 1$, with the understanding that this will only influence additive constants in the running time.

Grid graphs. Unless otherwise stated, we will study graphs that are 2-dimensional toroidal $n \times n$ square grids. Define $G_n = (V_n, E_n)$, where $V_n = \{(x, y) : 0 \le x, y < n\}$. We will use the shorthand $u = (x_u, y_u)$ for the coordinates of each node $u \in V_n$. The nodes do not have access to these coordinates. Two nodes u and v are connected by an edge if and only if $|x_u - x_v| + |y_u - y_v| = 1$, where all coordinates are modulo n. All edges are oriented in a consistent manner towards the larger coordinate, and labelled so that each node knows which edge points "north" (increasing y coordinate), "east" (increasing x coordinate), "south", and "west". By definition the grid wraps around in both dimensions, forming a torus. We will use the shorthands $V = V(G)$ and $E = E(G)$ for the node and edge sets, respectively, of G. We will assume that all nodes are given the value of n as input.

On unsolvable problems. Many LCL problems are unsolvable for some values of n. For example, there does not exist a 2-colouring if n is odd, and many problems are ill-defined for e.g. $n = 1$. Throughout this text we will usually assume that n is sufficiently large so that the problem that we consider is meaningful. Problems for which there are infinitely many values of n for which solution does not exist (e.g. 2-colouring) are regarded as global problems. Indeed, often the fact that there solutions do not exist at all for some values of n is a simple way of proving a lower bound of $\Omega(n)$, and such a bound holds even if we had a promise that n is chosen so that a solution exists.

Notation. From now on, we write $G^{(k)}$ for the kth power of a graph G. That is, $V(G^{(k)}) = V(G)$ and $E(G^{(k)}) = \{\{u, v\} : \mathrm{dist}_G(u, v) \le k\}$. We denote the set $\{0, 1, \ldots, k-1\}$ by $[k]$.

4 Warm-up: directed cycles

As a gentle introduction to our research questions, we will first have a look at LCL problems in directed cycles (i.e., 1-dimensional grids). This case is completely understood by prior work, but we will present it from a new perspective: in the 1-dimensional case, any LCL problem P can be conveniently represented as a directed graph H. By studying elementary properties of graph H, we can directly deduce the computational complexity of problem P, and derive an asymptotically optimal algorithm for solving P—everything is decidable and algorithm synthesis is computationally tractable (see Figure 2).

We construct an *output neighbourhood graph* $H = (V, E)$ as follows. Problem P can be interpreted as a set of feasible radius-r local neighbourhoods $u_1 u_2 \ldots u_{2r+1} \in P$. For every such neighbourhood, we will have an edge $(u_1 u_2 \ldots u_{2r}, u_2 u_3 \ldots u_{2r+1}) \in E$ in graph H. For example, in the 3-colouring problem, the sequence "132" is a feasible neighbourhood, and hence we will have an edge $(13, 32)$ in

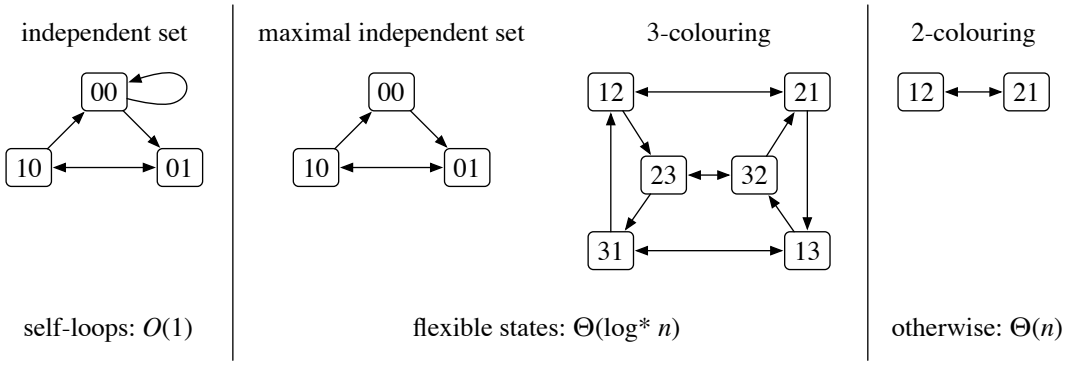


Figure 2: LCL problems for cycles can be represented as directed graphs. Here we have four LCL problems with radius $r = 1$; each node corresponds to a sequence of $2r = 2$ output labels, and each edge corresponds to a sequence of $2r + 1 = 3$ output labels. We can read the time complexity of the LCL problem directly from the properties of the graph. For example, in the maximal independent set problem, state 00 is flexible, as we can find walks from 00 back to itself of lengths 3 and 5, and hence also of any length larger than 7.

the graph; here e.g. "13" corresponds to a node with output 3 that has a predecessor with output 1 (see Figure 2). The key observation is that walks in graph H correspond to feasible output labellings in problem P.

Now we say that a node $u \in V$ is *flexible* if there exists an integer k such that for all $k' \geq k$, there exists a walk in G of length k' that starts and ends at u. We call the smallest such k the *flexibility* of u. It is clear that u is flexible if and only if there are circuits C, C' containing u whose respective lengths are coprime.

Claim 1. *The complexity of P is $O(1)$ if some node of H has a self-loop; otherwise $\Theta(\log^* n)$ if some node of H is flexible; and otherwise $\Theta(n)$.*

Proof. The case of $O(1)$ time is straightforward. Recall the result of Naor and Stockmeyer [30] that shows that unique identifiers do not help with $o(\log^* n)$-time algorithms; hence we have only trivial problems for which a constant labelling is a feasible.

Also the case of $\Theta(n)$ time is straightforward. There are only constantly many neighbourhoods, and hence some neighbourhood $u \in V$ has to be used $\Omega(n)$ times in the output. However, u is not flexible, and hence the spacing between u-neighbourhoods requires global coordination. (For example, in 2-colouring the distance between any two occurrences of neighbourhood "12" has to be a multiple of 2.)

It remains to be shown that if u is a flexible node with some minimum flexibility k, we can solve P in time $O(\log^* n)$. Let G be a cycle graph and let $G^{(k)}$ be the kth power of G. We can find a maximal independent set I in $G^{(k)}$ in time $O(\log^* n)$. Let v be a node in I, and let v' be the next node in I by the ordering of the nodes of G. Let the distance from v to v' in G be i; we have $k + 1 \leq i \leq 2k + 1$. We label v and v' using the neighbourhood u, and fill in the gap between v and v' by following some circuit C_i of length i from u back to u in H. □

It would be tempting to try to generalise this result to 2-dimensional grids. Unfortunately, this is not possible; as we will see in Section 6, there does not exist an algorithm for finding the time complexity of a given LCL problem in 2-dimensional grids. Nevertheless, we can still prove that any LCL problem has a complexity of $O(1)$, $\Theta(\log^* n)$, or $\Theta(n)$ also in 2-dimensional grids. We will next prove the key ingredient: any $o(n)$-time algorithm can be turned into an $O(\log^* n)$-time algorithm that has a convenient structure.

5 Speed-up and normal form

In this section, we give the speed-up result underlying both the complexity classification of LCL problems and the synthesis. The following result is essentially a refined version of the speed-up lemma of Chang et al. [9] for two-dimensional oriented grids; the proof immediately yields the normal form algorithm for any LCL problem as discussed in Section 1.

Theorem 2. *Given any* LCL P *with an algorithm* A *that solves* P *in time* $T(n) = o(n)$*, there exists an algorithm* B *that solves* P *and has running time* $O(\log^* n)$*.*

Proof. Recall that w.l.o.g., we can assume that problem P has checkability radius $r = 1$. Algorithm B solves problem P in an $n \times n$ grid G as follows:

(1) Pick the smallest even $k \geq 4$ such that $T(k) < k/4 - 4$. Such a k exists by assumption, and it is a constant that only depends on T.
(2) Find a maximal independent set I in $G^{(k/2)}$. This can be done in time $O(\log^* n)$; the nodes of I are called *anchors*.
(3) Simulate A with *locally unique identifiers* from $[k^2]$ around each anchor in I.

Step (3) of the simulation proceeds as follows. First, G is divided into a Voronoi tiling with respect to I, breaking ties arbitrarily—that is, we associate with each node $v \in I$ a Voronoi tile $T(v) = \{u \in V : v \text{ is the closest anchor to } u\}$; each node can compute which tile they belong to in constant time. Then, each node v is assigned a *local coordinate* $c(v) = (x_v - x_{a(v)}, y_v - y_{a(v)})$, where $a(v)$ is the anchor of v's tile. The local coordinates will be interpreted as locally unique identifiers.

There are no repeating identifiers within distance $k/2$ of any node: If two nodes u and v have the same coordinate, they are in different Voronoi tiles. Since the anchors are at distance at least $k/2$, and u and v are by assumption in the same relative positions with respect of their anchors, also u and v are at distance at least $k/2$.

Each Voronoi tile $T(v)$ holds nodes at distance at most $k/2 + 1$, since any node at distance $k/2 + 2$ from v must have another anchor within distance $k/2$. We can calculate that the size of each tile is at most $|T(v)| \leq k^2$; hence we only need k^2 distinct locally unique identifiers.

Next we simulate A on G, with a bit of cheating: we tell A that we are actually solving P for an instance of size $k \times k$; for each local neighbourhood of G, we feed it locally unique identifiers from $[k^2]$. Now A has a running time $T(k) < k/4$, and hence it does not ever see repeating identifiers; it has to solve problem P correctly in each local neighbourhood as this might be a legitimate instance of size $k \times k$. More precisely, if the local outputs of A violated the constraints of the LCL problem P for some local neighbourhoods, we could also construct a genuine instance H of size $k \times k$ with globally unique identifiers, and A would fail to solve P in H. Hence the local outputs of A have to constitute a globally feasible solution for P also in G. □

6 Undecidability of classification

In this section we show that in general deciding whether the running time of a given LCL problem is $\Theta(\log^* n)$ or $\Theta(n)$ is undecidable. We achieve this by defining, for each Turing machine M, an LCL problem L_M such that L_M can be solved in time $\Theta(\log^* n)$ if and only if M halts, and in time $\Theta(n)$ otherwise.

Theorem 3. *The problem of deciding whether a given* LCL *can be solved in time* $\Theta(\log^* n)$ *or* $\Theta(n)$ *on grids is undecidable.*

It is good to compare this with the result of Naor and Stockmeyer [30]. They study grids with boundaries (non-toroidal grids; there are nodes of degrees 3 and 2). In such grids, deciding if an LCL can be solved in time $O(1)$ is already undecidable. In essence, for any Turing machine M we can construct an LCL that specifies that in the lower-left corner of the grid we have to write out the complete execution history of M, and everything else can be padding. Now if and only if M halts in some finite time t, then LCL can be solved in time $O(t) = O(1)$, as it suffices to check if we are within distance $\Theta(t)$ from the corner and otherwise we can just output padding.

In our case of toroidal grids, this no longer holds. It is trivial to decide if a given LCL can be solved in $O(1)$ time; only trivial problems in which a constant output is a feasible solution admit an $O(1)$-time solution in toroidal grids. For example, the problem constructed by Naor and Stockmeyer [30] is now trivial, as there are no corners, and we can always output padding.

We develop a different LCL problem L_M that forces any efficient algorithm to *create corners*. The problem is defined so that the grid can be partitioned in "tiles" of arbitrary dimensions, but there are additional requirements:

(1) inside each tile we have to solve an inherently global problem, and
(2) in the "corner" of each tile we must output the complete execution table of M.

Now property (1) prevents efficient algorithms from producing an output that says that the entire grid consists of one tile. But as soon as the algorithm creates some tile boundaries, property (2) kicks in and makes sure that we can have finite tiles if and only if M halts in finite time. Additional care is needed to make sure that the problem is solvable but global if M does not halt, as we will discuss next.

LCL problem L_M in detail. For each Turing machine M, we define L_M as the disjoint union of two locally checkable labellings P_1 and P_2. The problem P_1 is defined to 3-colouring in order to make sure that L_M can always be solved in time $O(n)$, independent of M. On the other hand, 3-colouring requires time $\Omega(n)$ by Theorem 9. The problem P_2 is a problem that involves labelling the grid with the execution table of M, started on an empty tape. This problem is formulated so that it can be solved in time $O(\log^* n)$ if and only if M halts on the empty tape.

Each node is labelled with the Turing machine M and a *type*: each node is either an *anchor* or belongs either into one of the four *quadrants* NW, NE, SE, and SW, or one of the four *borders* N, S, E, and W. We overload the notation and define incidence operators as follows. For an arbitrary node $v = (x, y)$, define

$$\begin{aligned} \mathsf{NW}(v) &= (x-1, y+1), & \mathsf{NE}(v) &= (x+1, y+1), \\ \mathsf{SE}(v) &= (x+1, y-1), & \mathsf{SW}(v) &= (x-1, y-1), \\ \mathsf{N}(v) &= (x, y+1), & \mathsf{S}(v) &= (x, y-1), \\ \mathsf{E}(v) &= (x+1, y), \text{ and} & \mathsf{W}(v) &= (x-1, y). \end{aligned}$$

We will use the types of the nodes to refer to the corresponding incidence operators. The idea of the type labels is that they can be followed to find an anchor.

Let $Q(u) \in \{\mathsf{NE}, \mathsf{SE}, \mathsf{SW}, \mathsf{NW}, \mathsf{N}, \mathsf{E}, \mathsf{S}, \mathsf{W}, \mathsf{A}\}$, denote the type of a node, and $x(u) \in \{0, 1\}$ a colouring. Define the diagonal neighbour $\mathrm{diag}(u)$ of node u as the node reached by taking a step in direction $Q(u)$. For example, if $Q(u) = \mathsf{NW}$, then $\mathrm{diag}(u) = \mathsf{NW}(u)$. For completeness, define the diagonal of an anchor is the node itself.

We have the following local rules.

(1) If $Q(u) = \mathsf{NE}$, then $Q(\mathrm{diag}(u)) \in \{\mathsf{NE}, \mathsf{N}, \mathsf{E}, \mathsf{A}\}$,

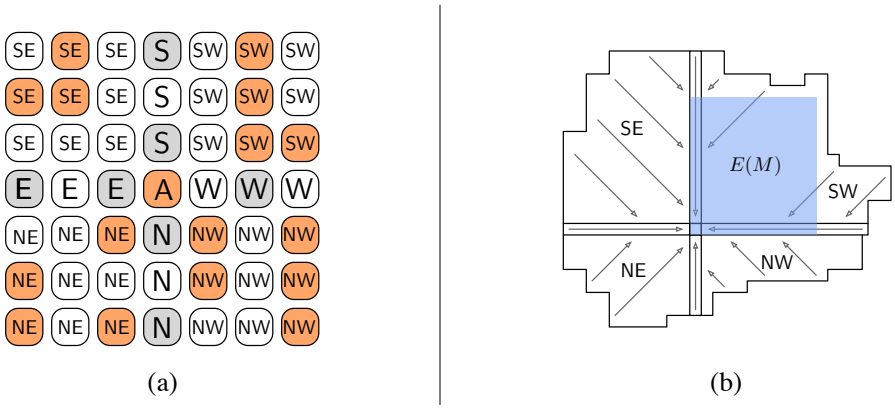


Figure 3: (a) The local rules for the labelling problem P_2: all nodes must have a type, indicated by the label, and their diagonal neighbours must have a type that is compatible. Every diagonal must be 2-coloured, that is, for every pair u, v such that $u = \mathrm{diag}(v)$ and $Q(u) = Q(v)$ we must have that $x(u) \neq x(v)$. The anchor must be surrounded by the other labels. (b) The general structure of $O(\log^* n)$ time solution to P_2 if M halts. The Voronoi tile of an anchor is split into four quadrants and four borders. Every diagonal can be followed to reach the anchor. An encoding of the execution table $E(M)$ starts from the anchor and is contained inside the Voronoi tile.

(2) if $Q(u) = \mathsf{SE}$, then $Q(\mathrm{diag}(u)) \in \{\mathsf{SE}, \mathsf{S}, \mathsf{E}, \mathsf{A}\}$,
(3) if $Q(u) = \mathsf{SW}$, then $Q(\mathrm{diag}(u)) \in \{\mathsf{SW}, \mathsf{S}, \mathsf{W}, \mathsf{A}\}$, and
(4) if $Q(u) = \mathsf{NW}$, then $Q(\mathrm{diag}(u)) \in \{\mathsf{NW}, \mathsf{N}, \mathsf{W}, \mathsf{A}\}$.

On the borders, we must have that $Q(u) = Q(\mathrm{diag}(u))$, or that $Q(\mathrm{diag}(u)) = \mathsf{A}$. In addition, we require that the borders are surrounded with different labels. In particular we must have that

(1) If $Q(u) = \mathsf{N}$, then $Q(\mathsf{W}(u)) = \mathsf{NE}$ and $Q(\mathsf{E}(u)) = \mathsf{NW}$,
(2) If $Q(u) = \mathsf{S}$, then $Q(\mathsf{W}(u)) = \mathsf{SE}$ and $Q(\mathsf{E}(u)) = \mathsf{SW}$,
(3) If $Q(u) = \mathsf{E}$, then $Q(\mathsf{N}(u)) = \mathsf{SE}$ and $Q(\mathsf{S}(u)) = \mathsf{NE}$,
(4) If $Q(u) = \mathsf{W}$, then $Q(\mathsf{N}(u)) = \mathsf{SW}$ and $Q(\mathsf{S}(u)) = \mathsf{NW}$.

Finally for any anchor v, we must have that $Q(\mathsf{N}(v)) = \mathsf{S}$, $Q(\mathsf{NW}(v)) = \mathsf{SW}$, $Q(\mathsf{E}(v)) = \mathsf{W}$, $Q(\mathsf{SE}(v)) = \mathsf{NW}$, $Q(\mathsf{S}(v)) = \mathsf{N}$, $Q(\mathsf{SW}(v)) = \mathsf{NE}$, $Q(\mathsf{W}(v)) = \mathsf{E}$, and $Q(\mathsf{NW}(v)) = \mathsf{SE}$.

In addition, the diagonals must be 2-coloured, that is, we require that if $Q(u) = Q(\mathrm{diag}(u))$, then $x(u) \neq x(\mathrm{diag}(u))$. This condition ensures that any fast solution cannot have large (e.g. linear-sized) contiguous fragments of nodes with the same type, and that anchor nodes must appear in the solution.

Finally, we require that starting from each anchor, the grid is labelled with the encoding of the execution table of M when started on an empty tape. This encoding is detailed in the following paragraph.

Either P_1 or P_2 must be used, but not both.

Encoding the execution table of a Turing machine M. Consider an anchor node v. We will translate the coordinate system of G so that $v = (0, 0)$.

Assume that M runs for s steps on the empty tape. The encoding of the execution table $E(M)$ of M consists of an $(s + 1) \times r$ rectangular subgrid, where $r \leq s + 1$, with the bottom left corner at the anchor v. Each row j of $E(M)$ encodes the contents of the tape at the beginning of step j. Each column i corresponds to a cell of the tape. Thus, each node (i, j) is labelled with the contents

of the cell i before step j. In addition, one node on each row holds the machine head and the state of M. Each node u on the left boundary of $E(M)$ must have $Q(u) = \mathsf{S}$, and each node w on the bottom boundary must have $Q(w) = \mathsf{W}$.

On the first row each cell is empty and the anchor v holds the machine head. Every 2×2 subgrid of $E(M)$ must be consistent with the transition rules of M. On the last row one of the nodes holds the machine head in a halting state. Each node may hold the encoding of at most one machine. Only nodes with labels $Q \in \{\mathsf{S}, \mathsf{W}, \mathsf{SW}\}$ may be labelled with an encoding of the Turing machine.

Note that since the labels contain no references to s or the position of any node on $E(M)$, the encoding can be done using a constant number of labels.

Local checkability of the encoding. Since the nodes can detect if both P_1 and P_2 are used, we can look at the two cases separately. Clearly a 2-colouring is locally checkable. Now assume that the labelling P_2 is used.

The local rules related to the labelling ℓ are clearly locally checkable. The nodes can check that they agree on the identity of the Turing machine M. The Turing machine encoding is also locally checkable: every anchor and the nodes on the W border can check that the tape is initially empty. Between the rows, nodes can check that the encoding respects the transition rules of M. On the top and the right border of the tape nodes can check that the final state is a stopping state and that the encoding is complete.

Solving L_M in time $O(\log^* n)$ if M halts. Assume that M halts in s steps. L_M can be solved in time $O(\log^* n)$ as follows.

(1) If G has size $n < 2(s+1)$, solve P_1 by brute force.

(2) Else, find a maximal independent set I in $G^{(4(s+1))}$. This is the set of anchors.

(3) Construct the Voronoi tiling T of the anchor set I, breaking ties in an arbitrary but consistent manner. The size of each tile is bounded by a constant. Inside each tile $T(v)$ with anchor $v = (x, v)$, label nodes according their position with respect to the anchor:

$$Q(u) = \begin{cases} \mathsf{NW}, \text{ if } x_u > x, y_u < y, \\ \mathsf{NE}, \text{ if } x_u < x, y_u < y, \\ \mathsf{SW}, \text{ if } x_u > x, y_u > y, \text{ and} \\ \mathsf{SE}, \text{ if } x_u < x, y_u > y. \end{cases} \tag{1}$$

Similarly, the borders are labelled as follows.

$$Q(u) = \begin{cases} \mathsf{N}, \text{ if } x_u = x, y_u < y, \\ \mathsf{S}, \text{ if } x_u = x, y_u > y, \\ \mathsf{E}, \text{ if } x_u < x, y_u = y, \text{ and} \\ \mathsf{W}, \text{ if } x_u > x, y_u = y. \end{cases} \tag{2}$$

From each anchor, start a labelling with the execution table of M as described above. The distance of at least $4(s+1)$ between anchors guarantees that each Voronoi tile can fit the execution table encoding inside it.

Everything is constant time, except finding the maximal independent set, which can be done in time $O(\log^* n)$.

Solving L_M requires time $\Omega(n)$ if M does not halt. Now assume that M does not halt on the empty tape. Solving P_1 naturally requires $\Omega(n)$ time. There are two possibilities for the labelling P_2: either the labelling contains an anchor, or not.

First, assume that the labelling contains an anchor $v = (x, y)$. This means that around the anchor, the grid must be labelled with the execution table of M, starting with an empty tape. The nodes $(x, y + j)$, with $j > 0$, must be labelled with S and the contents of the first cell of M's tape before time steps j. The nodes $(x + i, y)$, with $i > 0$, must be labelled with W and the initial, empty contents of M's tape. Since M does not halt on the empty tape, either some node detects and illegal transition in the encoding of the execution table, or the table wraps around the grid. Then there must be a node labelled with N or NW, and contents of M's tape, a contradiction to the correctness of the output.

Now assume that there are no anchors. If there are no borders, all nodes must be labelled with the same quadrant $Q \in \{\mathsf{NW}, \mathsf{NE}, \mathsf{SE}, \mathsf{SW}\}$, as otherwise there would a node with the wrong type of diagonal neighbour. Then we can find diagonals of length $\Omega(n)$ that must be 2-coloured, requiring time $\Omega(n)$. Now assume that there exists a node labelled with a border. Since there are no anchors, this node must have a diagonal labelled with the same border, until the border wraps around. The border has length $\Omega(n)$ and must again be 2-coloured, leading to a running time of $\Omega(n)$.

Solving L_M requires time $\Omega(\log^* n)$ if M halts. Finally, we note that solving L_M requires time $\Omega(\log^* n)$, as it requires breaking symmetry between nodes.

We have shown that L_M has an $O(\log^* n)$ time algorithm if and only if M halts on an empty tape. This is known to be an undecidable problem, and therefore the problem of deciding whether an $O(\log^* n)$ time algorithm exists is in general also undecidable.

7 Synthesis

At first, the undecidability result of Section 6 seems to suggest that there is little hope in automating algorithm design for LCL problems in grids. Indeed, given an LCL problem P, we cannot even tell if it can be solved in $O(\log^* n)$ time or if it is inherently global.

However, in a sense this is the *only* obstacle for automatic synthesis of optimal algorithms! Let us assume that we are given 1 bit of advice indicating whether P is local (solvable in time $O(\log^* n)$) or global. We will now argue that this information is enough to automatically synthesise an asymptotically optimal algorithm for P.

If P is global, then there is a trivial brute-force algorithm of time $O(n)$ that merely gathers the entire output at a single node and solves the problem globally.

If P is local, we can first check whether it is trivial. If there is a constant label that can be used to fill the entire grid, then (and only then) the problem is solvable in time $O(1)$.

The remaining case is a local problem that cannot be solved in time $O(1)$. Now Theorem 2 and the classical result of Naor and Stockmeyer [30] imply that the only possibility is the complexity of $\Theta(\log^* n)$. Moreover, the proof of Theorem 2 suggests a convenient *normal form*: problem P can be solved with an algorithm of the form $A' \circ S_k$ for some constant k, where

- S_k finds a set of anchors I that forms a maximal independent set in $G^{(k)}$,
- A' is an algorithm with running time bounded by $O(k)$ that takes as an input only the set of anchors I and the global orientation of the grid.

In the proof of Theorem 2, algorithm A' first constructs Voronoi tiles, then assigns locally unique identifiers, and then simulates some $O(k)$-time algorithm A. But we do not need to worry about

such details here; we can see this entire process as a black box A' that simply takes the placement of anchors in the radius-$O(k)$ neighbourhood as input, and using only this information produces the final local output. In particular, A' does not depend on the assignment of unique identifiers or on the value of n.

It follows that A' is a *finite function*, mapping radius-$O(k)$ neighbourhoods in a $\{0,1\}$-labelled grid to local outputs. There are only finitely many ways to assign $\{0,1\}$ labels in a constant-sized fragment of the grid, and hence A' can be conveniently represented as a finite lookup table.

The only missing piece is finding the value of k, and to do that, we can simply start with $k = 1$ and increment it until synthesis succeeds. (Note that if we were dealing with a global problem instead of a local problem this loop will never terminate.)

For each value of k, we proceed as follows. We pick sufficiently large values $r_1, r_2 = \Theta(k)$. Then we enumerate all possible ways in which the anchors may appear within a $r_1 \times r_2$ fragment of the grid; these are called *tiles*. We describe in Appendix A.1 a practical algorithm for such an enumeration. For example, for $k = 1$ we have the following 3×2 tiles; if we consider a maximal independent set in the grid, and pick a 3×2 window, we will see one of these configurations:

$$\boxed{\begin{matrix}00\\00\\10\end{matrix}}\ \boxed{\begin{matrix}00\\00\\01\end{matrix}}\ \boxed{\begin{matrix}00\\10\\00\end{matrix}}\ \boxed{\begin{matrix}00\\10\\01\end{matrix}}\ \boxed{\begin{matrix}00\\01\\00\end{matrix}}\ \boxed{\begin{matrix}00\\01\\10\end{matrix}}\ \boxed{\begin{matrix}10\\00\\00\end{matrix}}\ \boxed{\begin{matrix}10\\00\\10\end{matrix}}\ \boxed{\begin{matrix}10\\00\\01\end{matrix}}\ \boxed{\begin{matrix}10\\01\\00\end{matrix}}\ \boxed{\begin{matrix}10\\01\\10\end{matrix}}\ \boxed{\begin{matrix}01\\00\\00\end{matrix}}\ \boxed{\begin{matrix}01\\00\\10\end{matrix}}\ \boxed{\begin{matrix}01\\00\\01\end{matrix}}\ \boxed{\begin{matrix}01\\10\\00\end{matrix}}\ \boxed{\begin{matrix}01\\10\\01\end{matrix}}$$

We will then construct a *neighbourhood graph* $H = (V_H, E_H)$, in which each node $u \in V_H$ corresponds to a $r_1 \times r_2$ tile, and each edge corresponds to a tile of dimensions $(r_1 + 1) \times r_2$ or $r_1 \times (r_2 + 1)$. For example, there is a 3×3 tile

$$\boxed{\begin{matrix}000\\010\\100\end{matrix}}$$

and hence in the neighbourhood graph of 3×2 tiles there is a directed *horizontal edge*

$$\left(\boxed{\begin{matrix}00\\01\\10\end{matrix}}, \boxed{\begin{matrix}00\\10\\00\end{matrix}}\right).$$

Similarly, we can identify directed vertical edges. Now A' is simply a mapping from V_H to local outputs; $A'(u)$ is what we output for a node whose local neighbourhood with respect to I is equal to u. Furthermore, the constraints of the LCL problem P (once sufficiently normalised) can be encoded as constraints related to horizontal and vertical edges. For example, in the 4-colouring problem, the constraint is simply that adjacent tiles have different labels.

Hence the task of synthesising algorithm A' reduces to a combinatorial constraint satisfaction problem in which our task is to find a labelling of the nodes of graph H that satisfies all constraints on the edges of the graph; if such an assignment does not exist, we simply repeat the process with a larger value of k and larger tile dimensions.

We have successfully used this approach with many concrete LCL problems discussed in this work, and it works well in practice. As a concrete nontrivial example, consider the problem of 4-colouring 2-dimensional grids. Here it can be shown that no solution exists for $k = 1$ or $k = 2$, but synthesis succeeds with $k = 3$ for e.g. 7×5 tiles. While a priori it might seem that the number of tiles is impractical for such parameter values ($2^{7 \cdot 5}$ candidate tiles?), the key observation is that 1's are fairly sparse in any maximal independent set of $G^{(k)}$, and it turns out that we only need to consider 2079 tiles. Finding a proper 4-colouring of the neighbourhood graph can be done with modern SAT solvers in a matter of seconds.

8 Vertex colouring d-dimensional grids with 4 colours

Theorem 4. *For every fixed $d \geq 2$, the complexity of 4-colouring d-dimensional grids is $\Theta(\log^* n)$.*

Here we consider d-dimensional (toroidal) grids, for some fixed dimension d. Particularly, each of the n^d vertices v has d coordinates: $v = (v_1, v_2, \ldots, v_d)$, where $v_i \in \{0, 1, \ldots, n-1\} = [n]$. For the sake of simplifying notation, we will not distinguish between a vertex and the vector of its coordinates, and treat all arithmetic operations on coordinates as happening in a $(\mathrm{mod}\, n)$ regime, thus for any $u, v \in [n]^d$ we have $uv, u - v \in [n]^d$. For $x \in [n]$ we define $\|x\| = \min\{x, n - x\}$, and for $v \in [n]^d$ the L_1 norm as $\|v\| = \sum_{1 \le i \le d} \|v_i\|$ and the L_∞ norm as $\|v\|_\infty = \max_{1 \le i \le d} \|v_i\|$. The L_1 and L_∞ distance definition follows from the corresponding norm definition. Observe that L_1 distance corresponds to the distance using grid edges.

Definition 5. We define the *radius-r ball* of u as

$$B_\infty(u, r) = \{v : \|u - v\|_\infty \le r\}.$$

Moreover, we denote by $G^{[k]}$ the kth power of G according to the L_∞ norm, i.e., $V(G^{[k]}) = V(G)$ and

$$E(G^{[k]}) = \big\{\{u, v\} : \|u - v\|_\infty \le k\big\}.$$

We also need the notion of *conflict colouring*, as given by Fraigniaud et al. [16].

Definition 6. Given graph G, lists of available colours for each vertex and lists of forbidden colour pairs for each edge, we say that a problem of assigning colours to vertices so that: (i) each vertex is assigned one of colours from this list and (ii) no edge observes on its endpoints a pair of colours from forbidden pair; is a (ℓ, d)-conflict colouring problem, if:

(1) each available colour list is of length at least ℓ,

(2) for each edge, for each colour on one endpoint, there are at most d forbidden colours on the other endpoint.

Fraigniaud et al. [16] show that if $\ell/d > \Delta$, then there is a distributed algorithm solving (ℓ, d)-conflict colouring in $\widetilde{O}(\sqrt{\Delta}) + \log^* n$ rounds. However, we observe that a greedy approach gives a good enough running time for our purposes: (i) colour vertices of graphs using Δ^2 colours (classical vertex-colouring problem) (ii) in Δ^2 rounds, iterate through colours, in round i vertices of colour i take any colour in a greedy fashion.

Proof of Theorem 4. Let us name a parameter ℓ, of even value to be fixed later. We use a set of anchors M being the maximal independent set of vertices of $G^{[\ell]}$. Since the degree of a vertex in $G^{[\ell]}$ is at most $(2\ell + 1)^d$, M can be found in $O((2\ell + 1)^{2d} + \log^* n)$ rounds on $G^{[\ell]}$. Since $\|\cdot\|_1 \le d\|\cdot\|_\infty$, any algorithm can be simulated on G with $\ell \cdot d$ multiplicative slowdown, giving in total $O(\ell \cdot d \cdot (2\ell + 1)^{2d} + \ell \cdot d \cdot \log^* n))$ rounds.

Our aim is to assign to every vertex of $v \in M$ a radius $r(v) \in \mathbb{Z}^+$, such that:

(1) $\{B_\infty(v, r(v) - 1) : v \in M\}$ covers all V,

(2) for any $u, v \in M$ such that if $B_\infty(u, r(u) + 1) \cap B_\infty(v, r(v) + 1) \neq \emptyset$ then the bounding hyperplanes for those L_∞ balls are separated, that is:

$$\forall_{1 \le i \le d} \min_{\varepsilon_1, \varepsilon_2 \in \{-1, 1\}} \|(u_i + \varepsilon_1 \cdot r(u)) - (v_i + \varepsilon_2 \cdot r(v))\| \ge 2$$

Consider the family of L_∞ balls of radius ℓ centred in every vertex of M: $\{B_\infty(v, \ell) : v \in M\}$. By the properties of MIS, this family covers every vertex of V, as otherwise we could add one more vertex to M, thus to satisfy (1) it is enough to have $r(v) > \ell$.

Next we show that for large enough ℓ we can find an appropriate assignment of radii fast, by reduction to local conflict colouring, with colours $\ell < r(v) < 2\ell$.

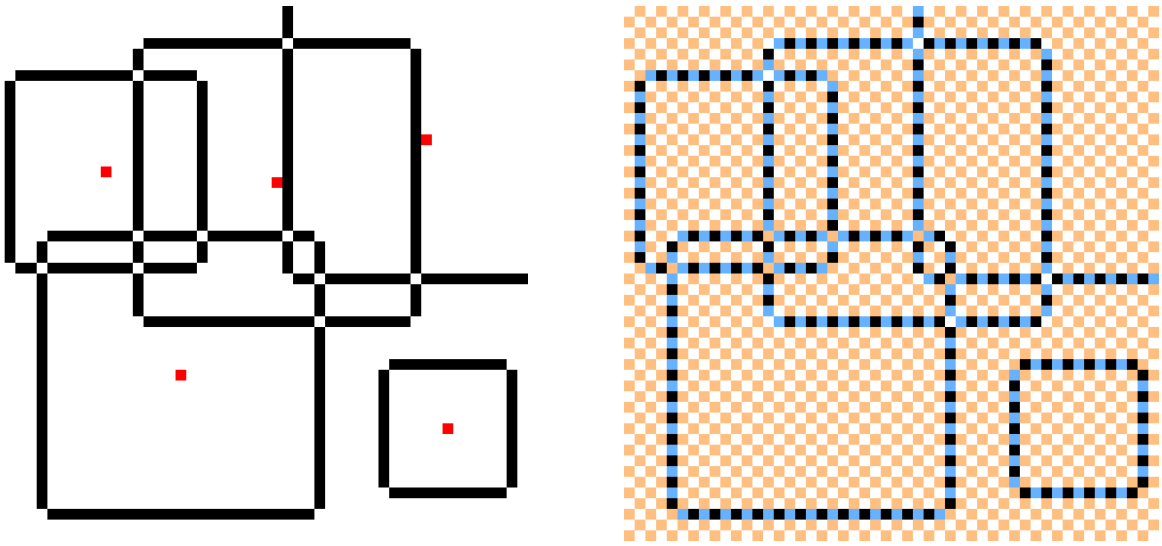


Figure 4: On the left, partition resulting from a choice of ball centres and ball radii. Induced vertex 4-colouring on the right.

Lemma 7. *Consider family of L_∞ balls of radius $c\ell$ centred in every vertex of M. Then every such ball intersects with at most $(8c)^d$ other balls.*

Proof. Consider $u, v \in M$, $u \neq v$. By the triangle inequality $\|u - v\|_\infty > \ell$, thus $B_\infty(v, \ell/2) \cap B_\infty(v, \ell/2) = \emptyset$. Also, $B_\infty(v, c\ell)$ intersects $B_\infty(u, c\ell)$ iff $v \in B_\infty(u, 2c\ell)$, but that implies that $|B_\infty(v, \ell/2) \cap B_\infty(u, 2c\ell)| \geq (\ell/2 + 1)^d$. Using bounds on first, the fact that all $B_\infty(v, \ell/2), v \in M$ balls are disjoint, and then that if the points are centres of balls of radius $c\ell$ that are intersecting with ball centred in u, then they intersect on large volume, we can bound the total number of intersecting balls as:

$$\frac{|B_\infty(u, 2c\ell)|}{(\ell/2+1)^d} = \left(\frac{4c\ell + 1}{\ell/2 + 1}\right)^d \leq (8c)^d. \qquad \square$$

Instead of considering conflicts over intersection of balls $B_\infty(v, r(v))$, we will guarantee no conflicts over intersections of $B_\infty(v, 2\ell)$. Now consider the graph H, over vertex set M, with edges connecting every pair of u, v such that $B_\infty(u, r(u) + 1) \cap B_\infty(v, r(v) + 1) \neq \emptyset$. By Lemma 7, the maximum degree in H can be upper-bounded as $\Delta_H \leq 16^d$.

While vertices do not know their coordinates, that is v does not have information on the values of $(v_1, \ldots, v_n)$, a pair of vertices u, v such that $(u, v) \in H$ is able to determine $u_i - v_i$ for all i. To satisfy condition (2) it is enough to exclude at most 12 possible values of $r(u)$, per each dimension and each value of $r(v)$. That is, we need to ensure that:

$$\forall_{(u,v)\in H} \forall_{1 \leq i \leq d} \forall_{\varepsilon_1, \varepsilon_2 \in \{-1,1\}} \forall_{x \in \{-1,0,1\}} \quad \varepsilon_1 \cdot r(u) \neq x + \varepsilon_2 \cdot r(v) + (v_i - u_i). \tag{3}$$

Thus our problem is an $(\ell, 12d)$-conflict colouring, and can be solved locally if $(\ell - 1)/(12d) > \Delta_H$, so it is enough to set $\ell = 1 + 12d \cdot 16^d$ for our purposes. Running time is upper-bounded by $O(\text{poly}(\Delta_H) + \log^* n)$ rounds in H, which can be simulated with multiplicative overhead of ℓ, giving total time of this part $O(\ell \cdot 16^{O(d)} + \ell \cdot \log^*(n))$ rounds.

What remains to show, is that given M and all $r(v)$, we can compute locally a $(2, 2d\ell)$ weak diameter network decomposition of G: a decomposition $V = V_1 \cup V_2$ into disjoint sets, such that each connected component of V_i is of diameter $2d\ell$. Given such a network decomposition, a 4 colouring of G can be found in $O(d\ell)$ rounds trivially.

We say that $u \in V$ is on the ith border of $v \in M$, if $u_i \in \{v_i - r(v), v_i + r(v)\}$ and $\|u - v\| - r(v)$. By the property (2) of radii, we know that every vertex v is on the ith dimension border of at most one vertex $u \in M$, and it can be decided locally during the computation of the radii. We define

$$\text{count}(v) = \Big|\{(i, u) : v \text{ is on the } i\text{-th dimension border of } u\}\Big|.$$

We assign v to V_1 iff count(v) is odd and to V_2 iff count(v) is even.

Lemma 8. *If $u, u' \in V$ are neighbouring in G, and there is v such that: $\|u - v\|_\infty = r(v) - 1$ and $\|u' - v\|_\infty = r(v)$, then count$(u) + 1 =$ count(u').*

Proof. Let j be the dimension such that $u_j \neq u'_j$. Then $\|u'_j - v_j\| = r(v)$, $\|u_j - v_j\| = r(v) - 1$ and $\forall_{i \neq j} \|u'_i - v_i\| = \|u_i - v_i\| \leq r(v) - 1$, that is j was the only dimension on which it was on the border of v. Moreover, u cannot be on the jth dimension border for any other vertex v', as then v and v' would violate property (2) of radii over the jth coordinate.

Now we observe that, while u' might be on the ith dimension border for some w, $i \neq j$, $w \neq v$, those remain the same for u. Namely, if we assume otherwise, that is that $\|u' - w\|_\infty = r(w)$ and $\|u - w\|_\infty \neq r(w)$, then by simple observation that only the jth coordinate changes in those difference vectors, we would have that u' is on jth dimension border for w, a contradiction. By analogous reasoning, we have that for any ith dimension border that u is on, it remains the same for u'.

All in all, we have that u is on one less dimension border than u'. □

Now we proceed to show that every connected component of V_1 or V_2 is a subset fully contained in $B_\infty(v, r(v) - 1)$ for some $v \in M$. Let us assume that this is not the case. Take any connected component X and $u \in X$, and let v be such that $u \in B_\infty(v, r(v) - 1)$ (by property (1) there is always one). If $X \not\subseteq B_\infty(v, r(v) - 1)$, then there are neighbouring $u', u'' \in X$, such that $\|u' - v\|_\infty = r(v) - 1$ and $\|u'' - v\|_\infty = r(v)$. However, by Lemma 8 they cannot be on the same side of the partition, a contradiction. □

9 Lower bound for 3-colouring 2-dimensional grids

Theorem 9. *The complexity of 3-colouring on 2-dimensional grids is $\Omega(n)$.*

The rough outline of the proof is as follows:

- We first show that a certain artificial coordination problem requires $\Omega(n)$ rounds on directed cycles.
- We then reduce this problem to 3-colouring two-dimensional grids. Essentially, we show that any 3-colouring algorithm for grids solves an instance of the aforementioned coordination problem for each row of the grid.

The q-sum coordination problem. Let $q \colon \mathbb{N} \to \mathbb{Z}$ be a function. In the *q-sum coordination problem*, we assume that the input graph is a directed cycle with unique identifiers, and each node v has to output $\ell(v) \in \{-1, 0, 1\}$ such that $\sum_{v \in V} \ell(v) = q(n)$, where $n = |V|$. That is, this is a family of problems, one for each possible function q. We now show that this problem is global for even fairly simple choices of q.

Theorem 10. *Let $q \colon \mathbb{N} \to \mathbb{Z}$ be a function such that*

(1) $q(n)$ is odd when n is odd, and
(2) $|q(n)| \leq n/2$ for all n.

Then q-sum coordination requires $\Omega(n)$ rounds.

Proof. Assume that we have an algorithm A that solves the problem in $T(n) = o(n)$ rounds. Fix a sufficiently large odd n such that $T(n) < n/200$. We show that we can construct an identifier assignment for a directed cycle of length n for which the sum of the outputs of A is greater than $n/2$, giving a contradiction.

We say that an *input fragment* F is a sequence of unique identifiers. We may interpret an input fragment F as a connected subgraph of a possible input graph of size n; we denote the length of sequence F by $|F|$, and say that fragments F_1 and F_2 are disjoint if the corresponding identifier sets are disjoint. Given at least two disjoint input fragments $F_1, F_2, \ldots, F_k$ and $|F_i| \geq n/100$, we define $A(F_1F_2 \cdots F_k)$ as the sum of output labels A gives to vertices from the midpoint of F_1 (inclusive) to the midpoint of F_k (exclusive) in the subgraph corresponding to the concatenated sequence $F_1F_2 \cdots F_k$. Note that since $T(n) < n/100$, this value only depends on $F_1, F_2, \ldots, F_k$. Moreover, denote by $P(F_1F_2 \cdots F_k)$ the parity of $A(F_1F_2 \cdots F_k)$. It follows immediately from the definition that $P(F_1 \cdots F_jF_{j+1} \cdots F_k) = P(F_1 \cdots F_j) + P(F_{j+1} \cdots F_k)$.

Lemma 11. *There are disjoint input fragments F_1 and F_2 with $|F_1| = |F_2| = \lceil n/100 \rceil$ such that for some input fragments X_1, X_2 disjoint from F_1 and F_2 with $|X_1|, |X_2| \in [2n/100, 96n/100]$ we have $P(F_1X_1F_2) \neq P(F_1X_2F_2)$.*

Proof. Assume that the claim does not hold. Then, for any disjoint input fragments F_1 and F_2 with $|F_1| = |F_2| = \lceil n/100 \rceil$ there is a value $P(F_1{*}F_2)$ such that $P(F_1XF_2) = P(F_1{*}F_2)$ for all X with $|X| \in [2n/100, 96n/100]$. By considering a cycle of form $F_1X_1F_2X_2$, where all fragments are disjoint, $|X_1|, |X_2| \in [2n/100, 96n/100]$ and $|F_1| + |X_1| + |F_2| + |X_2| = n$, we observe that for any F_1 and F_2 we have that $P(F_1{*}F_2)$ and $P(F_2{*}F_1)$ have fixed, different values, since

$$P(F_1{*}F_2) + P(F_2{*}F_1) = P(F_1X_1F_2) + P(F_2X_2F_1) = q(n)\,,$$

which is odd. Moreover, fixing disjoint F_1, F_2 and F_3 such that $P(F_1{*}F_2) = 0$ and considering a length-n cycle of form $F_1X_1F_2X_2F_3X_3$, we observe by a similar argument that either $P(F_2{*}F_3) = 0$ or $P(F_3{*}F_1) = 0$; by relabelling F_1, F_2 and F_3 if necessary, we can assume that $P(F_1{*}F_2) = P(F_2{*}F_3) = 0$. Considering disjoint fragments Y_1 and Y_2 with $|Y_1| = |Y_2| = \lceil 2n/100 \rceil$, we finally observe that

$$P(F_1{*}F_3) = P(F_1Y_1F_2Y_2F_3) = P(F_1Y_1F_2) + P(F_2Y_2F_3) = P(F_1{*}F_2) + P(F_2{*}F_3) = 0\,.$$

Now assume that F_1, F_2 and F_3 are disjoint fragments with $|F_1| = |F_2| = |F_3| = \lceil n/100 \rceil$ as above, and let X be a fragment disjoint from F_1, F_2 and F_3 with $|X| = \lceil 2n/100 \rceil$. We now have the following:

$$P(F_1XF_2) = 0 \quad \Rightarrow \quad P(F_1X) = P(XF_2)\,, \tag{4}$$
$$P(F_2XF_3) = 0 \quad \Rightarrow \quad P(F_2X) = P(XF_3)\,, \tag{5}$$
$$P(F_1XF_3) = 0 \quad \Rightarrow \quad P(F_1X) = P(XF_3)\,. \tag{6}$$

Thus, we have

$$P(F_2X) \overset{(5)}{=} P(XF_3) \overset{(6)}{=} P(F_1X) \overset{(4)}{=} P(XF_2)\,. \tag{7}$$

Furthermore, we have that

$$P(F_3XF_2) = 1 \quad \Rightarrow \quad P(F_3X) = P(XF_2) + 1\,, \tag{8}$$
$$P(F_2XF_1) = 1 \quad \Rightarrow \quad P(XF_1) = P(F_2X) + 1\,. \tag{9}$$

Thus,

$$1 = P(F_3XF_1) = P(F_3X) + P(XF_1) \overset{(8,9)}{=} P(XF_2) + 1 + P(F_2X) + 1 \overset{(7)}{=} 0\,,$$

which is a contradiction. □

Now let F_1, F_2, X_1 and X_2 be as in Lemma 11. If $|X_1| = |X_2|$, we are done, since for any fragment Y disjoint from the other fragments such that $|F_1| + |F_2| + |X_1| + |Y| = n$, the cycles $F_1X_1F_2Y$ and $F_1X_2F_2Y$ are valid instances of q-sum coordination with different outputs. Otherwise, we can assume that $|X_1| + 1 = |X_2|$ without loss of generality; in fact, we can assume that X_2 is obtained from X_1 by adding an unique identifier to the start of X_1, since by the above observation $P(F_1XF_2)$ is defined by the length of X.

Now let $d = A(F_1X_2F_2) - A(F_1X_2F_2)$, and observe that $|d| \geq 1$. Let X_3 be obtained by adding another unique identifier to the start of X_2. Consider removing the last identifier from X_1, X_2 and X_3 to obtain X_1^-, X_2^- and X_3^-, respectively. First, consider X_2^-; clearly $A(F_1X_2^-F_2) = A(F_1X_1F_2)$ since $|X_2^-| = |X_1|$. Thus, removing the last identifier v from the end of X_2 reduces the sum of the outputs in the $T(n)$-neighbourhood of v by d, and since X_2 is sufficiently long, this does not effect the first vertices of X_2. However, since the local changes look the same within a $T(n)$-radius neighbourhood, this implies that we also have $A(F_1X_3F_2) - A(F_1X_3^-F_2) = d$ and $A(F_1X_1F_2) - A(F_1X_1^-F_2) = d$. That is, adding an identifier to the front of X_2 increases the score by d, and removing an identifier from the end of X_1 decreases the score by d.

By repeatedly applying this argument to both directions, we can construct a sequence of fragments $Y_1, Y_2, \ldots, Y_{\lceil 9n/10 \rceil}$ such that $|Y_1| = \lceil 2n/100 \rceil$, $|Y_{k+1}| = |Y_k| + 1$ and $A(F_1Y_{k+1}F_2) = A(F_1Y_1F_2) + kd$. By definition of the problem, $|A(F_1Y_1F_2)| \leq 4n/100$, so $|A(F_1Y_{\lceil 9n/10 \rceil}F_2)| \geq 8n/10$ and $|Y_{\lceil 9n/10 \rceil}| \geq 92n/100$. But this means that the sum of outputs of A on any input containing the fragment $F_1Y_{\lceil 9n/10 \rceil}F_2$ has absolute value more than $n/2$, which is a contradiction. □

Reduction to 3-colouring. Fix an algorithm A for 3-colouring grids, and assume A runs in $T(n) = o(n)$ rounds. By adding a one-round preprocessing step, we may assume that A produces a greedy colouring, that is, if node v has colour 3, then it has neighbours of colour 1 and 2. We now show that algorithm A can be used to solve q-sum coordination in $T(n)$ round for some q satisfying the conditions in Section 10.

Fix the input size n and an input grid G, and consider the colouring $c\colon V(G) \to \{1, 2, 3\}$ produced by A. We will now define an auxiliary directed graph H with node set $V(H) = \{v \in V(G)\colon c(v) = 3\}$

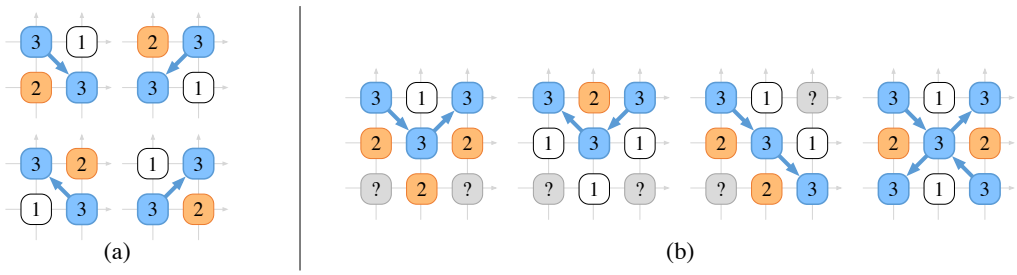


Figure 5: (a) Edge directions in H. (b) Possible neighbourhoods in H up to rotation.

as follows. We add a directed edge to $E(H)$ between two nodes u, v with $c(v) = c(u) = 3$ if they share two neighbours w, w' such that $c(w) = 1$ and $c(w') = 2$, and we direct this edge so that the common neighbour with colour 1 is to the "left" of the edge (Figure 5(a) shows all possibilities). There are four possible neighbourhoods for a node in H, up to rotation (Figure 5(b)):

(1) If v has exactly one neighbour of colour 1 (say to the north), then there is an in-edge from the node to the north-west and an out-edge to the node to the north-east.

(2) If v has exactly one neighbour of colour 2 (again to the north), then there is an in-edge from the node to the north-east and an out-edge to the node to the north-west.

(3) If v has two neighbours of colour 1 (say to the north and east), then there is an in-edge from the node to the north-west and an out-edge to the node to the south-east.

(4) If v has two neighbours of colour 1 (say to the north and south), then there are in-edges from the nodes to the north-west and to the south-east, as well out-edges to the nodes in the north-east and in to the south-west.

In particular, each node has either in-degree 1 and out-degree 1, or in-degree 2 and out-degree 2 in H. Thus, we can partition $E(H)$ into a collection $\mathcal{C}$ of edge-disjoint directed cycles.

Consider a cycle $C \in \mathcal{C}$ and a row r of G, and let u, v, w be nodes on C such that $(u, v) \in C$ and $(v, w) \in C$. We say that v is a *northbound intersection* if u is on the row south of v and w is on the row north of v. Similarly, we say that v is a *southbound intersection* if u is on the row north of v and w is on the row south of v. Let $\mathrm{north}_r(C)$ be the number of northbound intersections on C and $\mathrm{south}_r(C)$ the number of southbound intersections on C and define $i_r(C) = \mathrm{north}_r(C) - \mathrm{south}_r(C)$.

Lemma 12. *For all rows r_1 and r_2, we have that $i_{r_1}(C) = i_{r_2}(C)$.*

Proof. It is enough to show that this is the case for two adjacent rows, so let $r_1 = r$ and $r_2 = r + 1$, that is, r_2 is the row immediately to the north of r_1. In the case that C does not intersect either row the claim holds. Otherwise, the set $I = \{v \in V(C) \colon v \text{ is an intersection on } r_1 \text{ or } r_2\}$ is non-empty. For $u, v \in I$, we say u *follows* v if u is the next element of I we reach when following the cycle from v in the direction of the edges; likewise, we say that v *precedes* u.

The set I may contains four types of intersections: northbound on r_1 (denoted by N_{r_1}), southbound on r (S_{r_1}), northbound on r_2 (N_{r_2}) and southbound on r_2 (S_{r_2}). We now observe that following hold:

(1) N_{r_1} is followed by N_{r_2} or S_{r_1}.
(2) S_{r_2} is followed by S_{r_1} or N_{r_2}.
(3) N_{r_2} is preceded by N_{r_1} or S_{r_2}.
(4) S_{r_1} is preceded by S_{r_2} or N_{r_1}.

We prove (1); the other cases follow by a similar argument. Consider an N_{r_1} intersection v. Following the cycle C forward from v, we observe that every other node is on row $r_2 = r + 1$ and every other node is on row $r_1 = r$, until we either have a node on row $r + 2$ or on row $r - 1$. That is, the next intersection we encounter is either a northbound intersection on r_2 or a southbound intersection on r_1.

For each intersection v of type N_{r_1} or S_{r_2}, we define the *pair* $p(v)$ of v to be the following intersection on C, and for each intersection of type N_{r_2} or S_{r_1} we define the pair as the preceding intersection on C. By the above case analysis, this partitions I to disjoint pairs $\{v, p(v)\}$.

Now we observe that there are four possible types of pairs, each of which contributes the same amount to $i_{r_1}(C)$ and $i_{r_2}(C)$:

(1) N_{r_1} and N_{r_2}: contributes 1 to $i_{r_1}(C)$ and $i_{r_2}(C)$.
(2) S_{r_1} and S_{r_2}: contributes -1 to $i_{r_1}(C)$ and $i_{r_2}(C)$.
(3) N_{r_1} and S_{r_1}: contributions to $i_{r_1}(C)$ cancel out, contributes nothing to $i_{r_2}(C)$.
(4) N_{r_2} and S_{r_2}: contributions to $i_{r_2}(C)$ cancel out, contributes nothing to $i_{r_1}(C)$.

Summing over all pairs, we have that $i_{r_1}(C) = i_{r_2}(C)$.

□

As a corollary we have that

$$\sum_{C \in \mathcal{C}} i_{r_1}(C) = \sum_{C \in \mathcal{C}} i_{r_2}(C)$$

for all rows r_1 and r_2 in G. Writing $s(G)$ for this sum, we make the following claim.

Lemma 13. *We have $s(G_1) = s(G_2)$ for any $n \times n$ grids G_1 and G_2 when $T(n) < n/4$.*

Proof. Construct an $n \times n$ grid H_1 from G_1 by replacing the unique identifiers on rows 1 to $\lceil n/2 \rceil$ by identifiers that do not appear in either G_1 and G_2. Since $T(n) < n/4$, the output on row $\lceil 3n/4 \rceil$ is the same on G_1 and H_1, so by previous results we have $s(G_1) = s(H_1)$. Constructing a graph H_2 from G_2 by replacing the identifiers on rows 1 to $\lceil n/2 \rceil$ with the same ones that appear in H_1, we have the same argument that $s(G_2) = s(H_2)$ and, using row $\lceil n/4 \rceil$, that $s(H_1) = s(H_2)$. □

Since $T(n) = o(n)$, there is a constant n_0 such that $T(n) < n/4$ for all $n \geq n_0$. Let us define a function $s(n)$ so that if $n < n_0$, then $s(n) = 1$ if n is odd and $s(n) = 0$ if n is even; if $n \geq n_0$, then $s(n) = s(G)$ for any $n \times n$ grid G.

Lemma 14. *If n is odd, then $s(n)$ is odd. Moreover, for all n, we have $|s(n)| \leq n/2$.*

Proof. The claim is trivially true when $n < n_0$, so assume $n \geq n_0$. Fix an arbitrary $n \times n$ grid G, a row r of G and a 3-colouring $c\colon V(G) \to \{1, 2, 3\}$ given by algorithm A. Moreover, let H be the auxiliary graph on colour 3 nodes as before. Assign a label $\ell(v) \in \{-1, 0, 1\}$ to each node v on row r:

(1) If $c(v) = 3$ and $\text{in-deg}_H(v) = \text{out-deg}_H(v) = 1$, let u, w be the unique colour 3 nodes such that $(u, v) \in E(H)$ and $(v, w) \in E(H)$. We define the label $\ell(v)$ based on the positions of u and w as follows:
    (1) $\ell(v) = 1$ if u is on row $r - 1$ and w is on row $r + 1$,
    (2) $\ell(v) = -1$ if u is on row $r + 1$ and w is on row $r - 1$, and
    (3) $\ell(v) = 0$ if u and w are on the same row.
(2) We define $\ell(v) = 0$ in all other cases.

Informally, this means that $\ell(v) = 1$ if v is a northbound intersection on some cycle, $\ell(v) = -1$ if v is a southbound intersection, and $\ell(v) = 0$ if v is both or neither. Directly by definitions, we have $s(n) = \sum_v \ell(v)$. Since any row in a colouring can have at most $\lfloor n/2 \rfloor$ nodes of colour 3 and only nodes of colour 3 have non-zero $\ell(v)$, we have that $|s(n)| \leq n/2$.

It remains to show that $s(n)$ is odd if n is odd. Assume that n is odd; since the colouring c is greedy, there are two adjacent nodes on row r that have colours 1 and 2. By shifting the identifiers, we may assume that these are nodes $v_0 = (r, 0)$ and $v_{n-1} = (r, n - 1)$. For any node v on row r with colour 1 or 2, define the *parity* of $v = (r, y_v)$ as $p(v) = y_v + c(v) \bmod 2$. We now make the following observations:

– If two nodes u, v are adjacent on row r with colours 1 and 2, and are not v_0 and v_{n-1}, they have the same parity.

– If two nodes u, v on row r with colours 1 and 2 are separated by a single node w with $c(w) = 3$, then u and v have a different parity if and only if $\ell(w) \in \{-1, 1\}$; this follows by a simple case analysis (compare with Figure 5(b)).

Finally, we observe that $p(v_0) \neq p(v_{n-1})$. Thus, following row r from v_0 to v_{n-1}, we must have an odd number of colour 3 nodes v with $\ell(v) \in \{-1, 1\}$, which implies that $s(n)$ is odd. □

We can now solve s-sum coordination on directed cycles in time $T(n)$ as follows. If $n < n_0$, we gather full information about the input cycle in n_0 rounds; all nodes output 0 except the one with smallest identifier, which outputs $s(n)$. If $n \geq n_0$, we simulate A on a $T(n)$-wide strip; each node looks at the middle row of the strip and outputs $\ell(v)$ as in the proof of Lemma 14. Since $T(n) = o(n)$, this gives a contradiction with Theorem 10.

10 Edge colouring d-dimensional grids with $2d + 1$ colours

Theorem 15. *For every fixed d, the complexity of edge $(2d + 1)$-colouring d-dimensional grids is $\Theta(\log^* n)$.*

Again, the lower bound follows from the result of Linial [28], so it remains to show the upper bound. Moreover, we will show that this is tight in the sense that it is not possible to edge-colour the d-dimensional grid using $2d$ colours when n is odd.

High-level idea. The general idea of the colouring is to have two exclusive colours for each dimension and to use the last remaining colour c in order to colour a set of edges that cuts each row in each dimension into pieces of constant length which can then be coloured alternately by the two colours for the edges in the respective dimension. In order to find such a set S of (pairwise non-adjacent) edges, we first find a set of nodes that is able to locally choose the edges from S such that the required conditions are met.

Consider an arbitrary dimension. For each row in this dimension, find a maximal independent set of large distance and denote the union of these maximal independent sets by M. Now move the nodes in M on their respective rows until each node is the centre of a radius-r ball (according to the L_∞ norm, i.e., the ball is essentially a hypercube) that intersects no radius-r ball from another node from M. By making sure that the initial maximal independent sets are of sufficiently large distance, arbitrarily large radii r can be achieved, since each node from M has sufficiently large space on its row compared to the number of nodes from M in its vicinity. Repeat the whole process for each remaining dimension.

Now each node from each of the obtained M colours a nearby edge (i.e., one in its radius-r ball) in the row, the node was initially chosen from, with colour c. By making the radii r sufficiently large (depending on d) in the beginning, the nodes can ensure that none of these coloured edges are adjacent, since the number of radius-r balls (with nodes from some of the aforementioned maximal independent sets as the centres) that intersect the nearby part of the row, from which a node chooses the to-be-coloured edge, can be bounded by a function that depends only on d. Essentially, even if a node chooses the edge it wants to colour last of all choosing nodes, it always has an edge available that is not adjacent to an already coloured edge. Moreover, the distances in the initially chosen maximal independent sets must be sufficiently large (as a function of d) to ensure sufficiently large radii r, but since d is fixed they can still be chosen to be constant, and likewise the distances the

nodes are moved can be bounded by a function in d. Hence, the pieces obtained in each row by removing the edges of colour c are of constant size and can be coloured with two colours, following our initial scheme.

Preliminaries. We use the same setting and notations as for the vertex colouring in Section 8. We will call a row in dimension q a *q-directional* row. As before, we call a set of nodes of G a *maximal independent set of distance k* if it is a maximal independent set in $G^{(k)}$, the kth power of G. Furthermore, we need the following generalisation of a vertex colouring:

Definition 16. A vertex colouring of a d-dimensional grid G is a *colouring of L_∞ distance k* if no two adjacent nodes of $G^{[k]}$ have the same colour.

Note that a vertex colouring is a colouring of L_∞ distance $2k$ if and only if for any node u, $B_\infty(u, k)$ contains no two nodes of the same colour. The following lemma establishes a bound on the time it takes to find a specific colouring of a certain distance that we will need later.

Lemma 17. *For every fixed d, there is a distributed algorithm that finds a vertex $(2k+1)^d$-colouring of L_∞ distance k of G in time $O(k(\log^* n + k^d))$.*

Proof. The nodes in G can simulate any distributed algorithm on $G^{[k]}$ with a multiplicative overhead of kd. Observe that a proper vertex colouring of $G^{[k]}$ induces a colouring of L_∞ distance k of G. Now, since $G^{[k]}$ has a maximum degree of $(2k+1)^d - 1$, finding a $(2k+1)^d$-colouring of $G^{[k]}$ can be done in time $O(\log^* n + (2k+1)^d)$ using the algorithm of Barenboim et al. [5]. Counting the simulation, total running time is $O(kd(\log^* n + (2k+1)^d))$; noting that d is constant yields the desired bound. □

In the high-level overview of the main algorithm, we mentioned nodes that will locally choose the edges that will be coloured with the special colour that is not assigned to some specific dimension. These nodes have to have two properties: On the one hand they should not be too far from each other in order to be able to choose a nearby edge each, such that each row in each dimension is thereby cut in sufficiently small pieces (for the later 2-colouring of the pieces); on the other hand they should be far enough from each other such that each node has enough space to choose an edge that is not adjacent to any other chosen edge. We formalize these considerations in the following definition:

Definition 18. A *j, k-independent set w.r.t. dimension q* is a set M of nodes of the grid with the following properties:

(1) For any node $w \notin M$ there is a node $u \in M$ in the same q-directional row with $\text{dist}(u, w) \leq j$.

(2) For any two nodes $u, v \in M$ we have that $B_\infty(u, k) \cap B_\infty(v, k) = \emptyset$.

Finding a j, k-independent set. In the following we describe a distributed algorithm that finds a j, k-independent set w.r.t. dimension q, where $j = 3(4k+1)^d$ and $k \geq 1$.

W.l.o.g. let $q = 1$ and denote the directions belonging to dimension 1 by *west* and *east* where the coordinate of dimension 1 increases stepwise in eastern direction. We simply use the term *row* when referring to a 1-directional row.

For each row r, choose a maximal independent set M_r of distance $2(4k+1)^d$ in r, i.e., in the graph induced by the nodes in row r. Moreover choose a (vertex) $(8k+1)^d$-colouring c of L_∞ distance $4k$ of the whole grid where the colours are chosen from $\{1, \ldots, (8k+1)^d\}$.

Let M be the union of the M_r taken over all rows r in the grid. We will now transform M into a j,k-independent set by deleting and adding nodes. More specifically, we repeatedly delete nodes from M and replace them by the respective next node in eastern direction. When we perform such a replacement of a node u by its eastern neighbour, we say that *u moves to the east*. For simplicity, we denote the new node in M again by u and assign it the same colour u had before.

The replacements take place in phases, starting with Phase 1. In Phase p the following steps are performed: Each node that does not have colour p does nothing. Each node u of colour p checks whether it is contained in M and whether $B_\infty(u, 2k)$ contains a second node from M (i.e., a node different from u itself). If both is the case, then u moves to the east and continues moving to the east until $B_\infty(u, 2k)$ does not contain a node from M any more, apart from u itself. Any node of colour p that stops moving to the east (or did not start moving in the first step of the phase) does not start moving again, even if a node from M moves into its radius-$2k$ ball. (Recall that the radius is taken according to the L_∞ norm.) Each phase ends after $(4k+1)^d - (4k+1)$ steps upon which the next phase starts. This concludes the description of the algorithm.

We note that any node moves to the east in at most one phase and therefore it moves at most $(4k+1)^d - (4k+1)$ steps to the east. Since this is less than the distance between any two points from the same M_r, no node moves "over" another node from M. Hence, the colours of the nodes that are passed by a node moving eastwards are irrelevant since only nodes from M have an active role in the algorithm. Thus, the presented algorithm is well-defined despite those colours not being specified. Note further that we can assume that all nodes start at the same time with the different phases (and that the nodes move with the same "speed"), by using standard synchronisation arguments.

In order to be able to show the correctness of the algorithm, we need the following lemma:

Lemma 19. *When a node u stops moving to the east, then $B_\infty(u, 2k)$ contains no node from M, apart from u itself.*

Proof. There are two reasons a node might stop moving to the east, namely that $B_\infty(u, 2k)$ contains no node from M, apart from u itself, or that the respective phase ends. Since each node starts moving at the beginning of the phase (if it moves in that phase at all), it is enough to show that each node u would stop moving to the east after at most $(4k+1)^d - (4k+1)$ steps even if the phase still continued. To this end, we assume for the remainder of the proof that there is no cutoff of a phase after $(4k+1)^d - (4k+1)$ steps, but that instead the phase ends after the last moving node stops moving.

For any node u, let $Z(u)$ be the set of nodes such that $v \in Z(u)$ if and only if v is in the same row as u and $v_1 - u_1 = z(4k+1)$ for some integer $z \in \{0, \dots, (4k+1)^{d-1} - 1\}$. Consider the $(4k+1)^{d-1}$ radius-$2k$ balls of the nodes in $Z(u)$. By the definition of $Z(u)$ these balls are lined up one after the other in eastern direction, but no pair of them intersects. Moreover, the union B of these balls intersects exactly $(4k+1)^{d-1}$ rows of the grid and each of these rows has exactly $(4k+1)^d$ consecutive nodes in B.

Order the nodes from M by the time they stop moving, breaking ties arbitrarily. Note that during the whole algorithm, a node moves either once (but then possibly a number of consecutive steps) or not at all. In the latter case we set the point in time at which the node stops moving to 0. In the case that a node moves on infinitely, we set the stopping time at ∞. Denote the ordered nodes by $u(0), u(1), \dots$ where the stopping time increases (or stays the same) with increasing argument.

We show now by induction (on the argument of the node) that no node from M moves further to the east than $(4k+1)^d - (4k+1)$ steps, i.e., no further than the centre of the furthest of the balls defined above.

Consider M at the point in time when $u(0)$ stops moving, or, if $u(0)$ moves on infinitely (a possibility that we cannot exclude yet), at an arbitrary point in time in the phase corresponding

to the colour of $u(0)$. Observe that at no point in time, a node that is still moving contains another moving node in its radius-$2k$ ball. (The reason for this is that all (moving) nodes move synchronously and in the beginning of each phase p, no node of colour p contains another node of colour p in its radius-$2k$ ball, by the definition of our colouring c.) Hence, the current non-moving nodes in M (at their current places in the grid), denoted by M_{static}, are the only nodes that can have caused $u(0)$ to move at all. Moreover, since $u(0)$ stops moving first, the nodes in M_{static} are at the exact same places as they were in the beginning of phase 1. In the beginning of phase 1, any two nodes in $M_{\mathsf{static}} \cup \{u(0)\}$ in the same row have a distance of at least $2(4k+1)^d$, by the definition of the M_r. Since $(4k+1)^d < 2(4k+1)^d$, each of the $(4k+1)^{d-1} - 1$ rows intersecting B contains at most one node from $M_{\mathsf{static}} \cap B$, by our above observation about B. Furthermore, the row containing $u(0)$ contains no node from $(M_{\mathsf{static}} \setminus \{u(0)\}) \cap B$. Hence, B contains at most $(4k+1)^{d-1} - 1$ nodes from $M_{\mathsf{static}} \setminus \{u(0)\}$. By the pigeonhole principle, one of the $(4k+1)^{d-1}$ radius-$2k$ balls of the nodes in $Z(u(0))$ does not contain a node from $M_{\mathsf{static}} \setminus \{u(0)\}$. Thus, $u(0)$ stops moving when it arrives at the centre of this ball, at the latest, which yields a maximum of $((4k+1)^{d-1} - 1)(4k+1) = (4k+1)^d - (4k+1)$ steps taken. This concludes the base case of the induction.

For the induction step, consider an arbitrary node $u(a) \in M, a \geq 1$ at the point in time it stops moving, or, if $u(a)$ moves on infinitely, at a point in time when $u(a)$ has already started to move and $u(a-1)$ has stopped moving (the induction hypothesis ensures that such a point in time actually exists). Define M_{static} analogously to the definition in the base case. By the induction hypothesis, we can assume that each node from M_{static} has moved at most $(4k+1)^d - (4k+1)$ steps to the east from its initial position. Thus, by the definition of the M_r, any two nodes in M_{static} in the same row have a distance of at least $2(4k+1)^d - ((4k+1)^d - (4k+1)) > (4k+1)^d$ from each other (and also from the initial location of $u(a)$). Now the proof of the induction step follows analogously to the proof of the base case. □

Using Lemma 19, we prove the correctness of the above algorithm and give an upper bound for its time complexity.

Lemma 20. *Let $j = 3(4k+1)^d$, $k \geq 1$ and $1 \leq q \leq d$. The algorithm described above finds a j,k-independent set w.r.t. dimension q in time $O(k^d \log^* n + k^{2d+1})$.*

Proof. As above, w.l.o.g. let $q = 1$, and let M denote the final set obtained by our algorithm. We start by showing that M is indeed a j,k-independent set w.r.t. dimension 1, by checking the properties given in Definition 18.

Regarding Property (1), we use similar observations to the ones made in the proof of Lemma 19: In the beginning of Phase 1, any node has distance at most $2(4k+1)^d$ to some node from M (as it was in the beginning of Phase 1) in the same row, by the definition of the M_r. Since any node from M moves at most $(4k+1)^d - (4k+1)$ steps to the east, any node in the grid has distance at most $2(4k+1)^d + (4k+1)^d - (4k+1) < 3(4k+1)^d$ to some node from M in the same row, which proves Property (1). Property (2) follows from Lemma 19. Note that if a node $u \in M$ stops moving, then no other node $v \in M$ will stop moving in $B_\infty(u, 2k)$ since $v \in B_\infty(u, 2k)$ is equivalent to $u \in B_\infty(v, 2k)$.

Now we examine the time complexity of our algorithm. Finding the M_r can be done in time $O(k^d \log^* n)$, in each row in parallel, by finding a maximal independent set in the $(2(4k+1)^d)$th power of each row. Finding the $(8k+1)^d$-colouring c can be done in time $O(k(\log^* n + k^d))$, by Lemma 17. Each phase contains $O(k^d)$ steps per node and checking the radius-$2k$ ball of a node can be done in time $O(k)$, resulting in a total time of $O(k^{2d+1})$ for the $(8k+1)^d$ phases (in a simple implementation).

We obtain a total runtime of $O(k^d \log^* n + k^{2d+1})$. □

Upper bound for edge colouring. We now proceed to describe an algorithm that finds an edge colouring with $2d+1$ colours in time $O(\log^* n)$. The algorithm starts by finding, for each $1 \leq q \leq d$, a j,k-independent set I_q w.r.t. dimension q, where $j = 3(4k+1)^d$ and $k = 2d$. By Lemma 20, this is possible in time $O(\log^* n)$ since d is fixed and k only depends on d. Now, we again proceed in phases, starting with Phase 1 and ending with Phase d. In Phase p, each node $u \in I_p$ marks an edge in $B_\infty(u,k)$ (i.e., an edge for which both of its endpoints are contained in $B_\infty(u,k)$) that is not adjacent to a previously marked edge, runs in direction of dimension p and is in the same p-directional row as u (cf. Figure 6 for an illustration in the 2-dimensional case).

In order to show that there is always such a non-adjacent edge in $B_\infty(u,k)$ available, consider the number of already marked edges that have at least one endpoint in $B_\infty(u,k) \cap R_p(u)$, where $R_p(u)$ denotes the set of nodes in the same p-directional row as u: Observe that for any arbitrary set $\mathcal{B}$ of pairwise disjoint radius-k balls of dimension d in our grid, at most two of the balls in $\mathcal{B}$ can intersect $B_\infty(u,k) \cap R_p(u)$ since any such intersecting ball from $\mathcal{B}$ must contain at least one of the two "endpoints" of the path $B_\infty(u,k) \cap R_p(u)$. Moreover, for any $1 \leq q \leq d$, the radius-k balls of the nodes in I_q are pairwise disjoint, by the definition of the I_q and Property (2) of Definition 18. Hence, $B_\infty(u,k) \cap R_p(u)$ intersects at most $2(d-1)$ radius-k balls of some node in some I_q (apart from the ball $B_\infty(u,k)$ itself). Now, since an edge with at least one endpoint in $B_\infty(u,k) \cap R_p(u)$ can only be marked by a node whose radius-k ball intersects $B_\infty(u,k) \cap R_p(u)$ and each of these nodes marks only one edge, there can be at most $2(d-1)$ marked edges with one endpoint in $B_\infty(u,k) \cap R_p(u)$, before u marks an edge. Each such marked edge prevents at most two of the edges in $B_\infty(u,k)$ in the same p-directional row as u to be marked by u because of the adjacency condition. But since the p-directional row containing u has $2k > 4(d-1)$ edges inside $B_\infty(u,k)$, there must be an edge left that u can mark without violating any of the required conditions. Marking the edges as described above can be done in time $O(dk) = O(1)$.

Now, the idea to colour the edges of the grid is simple (cf. Figure 7 for an illustration in the 2-dimensional case): Each marked edge gets colour $2d+1$. Each remaining edge that runs in the direction of dimension q gets colour $2q-1$ or $2q$. For that, each edge of colour $2d+1$ (or, more precisely, the endpoints of that edge) negotiates with the next edge of colour $2d+1$ in the same row in the same dimension the colouring of the in-between edges, such that the two available colours alternate. Observe that for any node from I_q there is another node from I_q in the same q-directional row (in both directions) with distance at most $2j+1$, by Property (1) of Definition 18. Moreover, since each node marks an edge that is in distance at most k, any edge of colour $2d+1$ has a distance of at most $2k+2j+1$ to the next edge of colour $2d+1$ in the same row in the same dimension. Hence, the colouring of the edges can be completed (in parallel) in time $O(1)$. The construction of the colouring ensures that no two adjacent edges have the same colour. This proves the upper bound claimed in Theorem 15.

Edge colouring with $2d$ colours. As mentioned in the beginning of this section, the bound on the number of colours given in Theorem 15 is tight:

Theorem 21. *Let d be fixed. Any d-dimensional grid G_n with n odd admits no edge $2d$-colouring.*

Proof. Let n be odd and assume for a contradiction that there exists an edge $2d$-colouring of G_n. Let c be one of the $2d$ colours. Since each node of the grid has degree $2d$, each node must have exactly one incident edge of colour c. Summing up the number of incident edges of colour c over all nodes, we obtain n^d. Since this sum counts each edge of colour c exactly twice, the total number of edges of colour c must be $n^d/2$. Since n is odd, $n^d/2$ is not an integer, yielding a contradiction. □

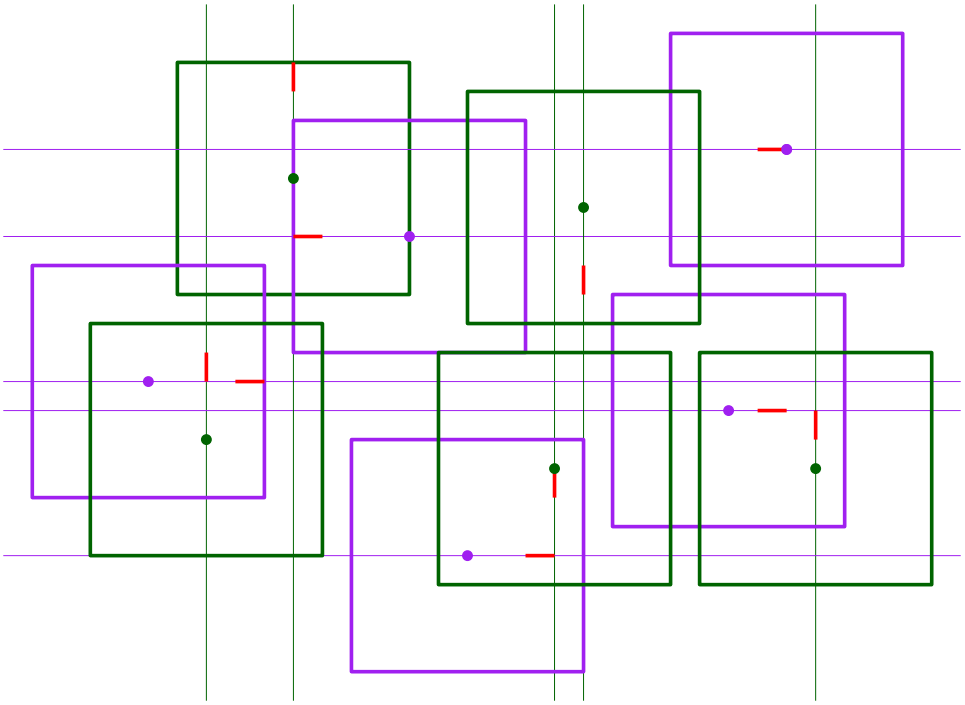


Figure 6: Nodes from two j, k-independent sets in the 2-dimensional grid, together with their radius-k balls and the rows (resp. columns) they are on. From the latter each node chooses an edge in its radius-k ball that guarantees non-adjacency of the chosen edges.

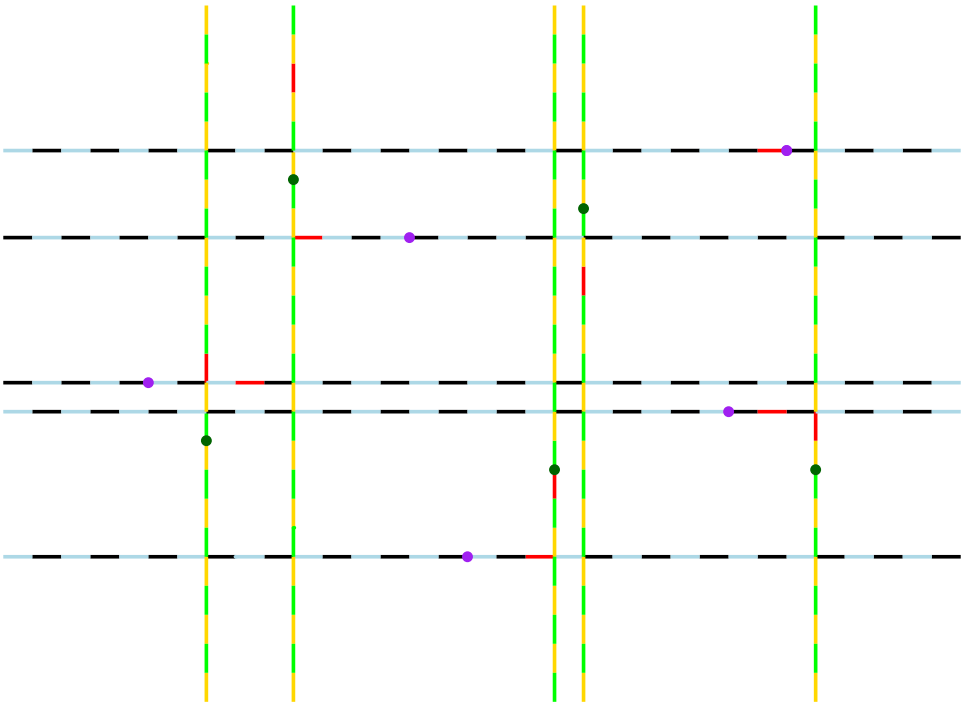


Figure 7: The edge colouring resulting from the edge choices made in Figure 6. The non-depicted rows and columns are coloured alternately in the considered part of the grid.

11 Edge orientations

Recall that for a set $X \subseteq \{0, 1, 2, 3, 4\}$, an X-orientation is an orientation of the edges such that for each node $v \in V$ we have in-deg$(v) \in X$. In this section, we present an exhaustive classification for X-orientation problem:

Theorem 22. *X-orientation problem for 2-dimensional grids has the following complexity:*

- $\Theta(1)$ *if* $2 \in X$.
- $\Theta(\log^* n)$ *if* $\{1, 3, 4\} \subseteq X$ *or* $\{0, 1, 3\} \subseteq X$.
- *Otherwise no solution exists for infinitely many n.*

We first make the following simple observations:

- If $2 \in X$, the existing input orientation of the grid is a valid solution.
- $\{1, 3, 4\}$-orientation and $\{0, 1, 3\}$-orientation have the same complexity, as one can be obtained from the other by flipping edge directions.

The following lemmas cover the remaining cases.

Lemma 23. *$\{1, 3, 4\}$-orientation has complexity $\Theta(\log^* n)$.*

Proof. For the lower bound of $\Omega(\log^* n)$, notice that a constant output is not feasible solution and hence there is no constant-time solution. For the upper bound we resort to computational techniques; we can synthesise an $O(\log^* n)$-time algorithm, using techniques outlined in Section 7 with $k = 1$. □

Lemma 24. *There is no $\{1, 3\}$-orientation for grids with odd n.*

Proof. Consider grid with odd n and any $\{1, 3\}$-orientation of it. Since the sum of all in-degrees is equal to number of edges, being $2n^2$, it is even. Thus number of vertices with in-degree 1 matches with parity to number of vertices with in-degree 3, meaning that total number of vertices is even, a contradiction. □

Theorem 25. *$\{0, 3, 4\}$-orientation problem is global on 2-dimensional grids.*

Proof. Our proof follows the steps of proof of the lower bound for vertex 3-colouring of grids. Assume that we have an algorithm A that solves the problem in $T(n) = o(n)$ rounds. We will show that such algorithm can be used to solving q-sum coordination problem, which by Theorem 10 leads to contradiction.

We label nodes with values of their in-degrees, that is 0, 3 or 4. Observe that no two 0 can be neighbours, similarly no two 4 can be neighbours. When speaking of nodes labelled 3, we will also refer to a direction of its only outgoing edge as *pointing* to.

Consider two consecutive rows of vertices of the grid, i and $i + 1$. A row of vertical edges connecting them will be referred to as i-th vertical row of edges.

Consider labelling of edges from i-th vertical row of edges with values from $\{-1, 0, +1\}$ in a following manner. Let u^+ and u^- be the vertices 0 in rows i or $i + 1$, in the columns closest to the left and closest to the right from considered edge.

- If there is vertex 0 on one of the endpoint of considered edge, assign label 0.
- If vertices u^+ and u^- are at odd L_1 distance and edge is oriented "up", assign label $+1$.
- If vertices u^+ and u^- are at odd L_1 distance and edge is oriented "down", assign label -1.

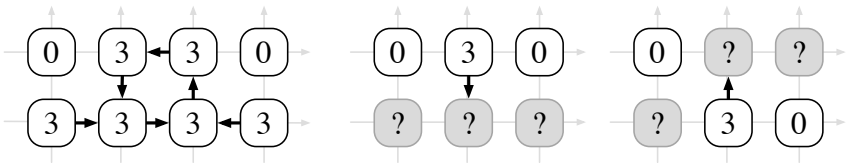


Figure 8: Possible relations between vertical edges and nodes with 0. Only in the last case the edge is labelled with non-zero value.

– If vertices u^+ and u^- are at even L_1 distance, assign label 0.

Consider two vertex rows, i and $i+1$. The gaps between nodes 0 are at most 2 columns wide, and the only way to have exactly 2 columns gap is to put nodes labelled 3 into 2×2 square with outgoing edges forming cycle. (See Figure 8.)

Since the gaps are bounded in length, we immediately conclude that vertical edges compute their labels in at most 2 additional rounds. Let $r(i)$ be the sum of labels on the i-th vertical row of edges.

Consider all non-0 vertices and edges between them. Every vertex has out-degree at most 1, so they form 1-forest, with connected components being 1-trees or trees. Observe that two vertices from separate branches of trees cannot be neighbouring, as it is not possible to orient properly edge connecting them. Fix one of the components as D. It is composed of (possibly empty) set of vertices and edges forming a directed cycle, denoted C, and tree-like attachments. A border of D is composed of pairwise nonadjacent 0. However, if we consider diagonal adjacency of 0 vertices, border is either 1 or 2 diagonally connected components: $\partial D = B_1$ or $\partial D = B_1 \cup B_2$, and each component B_i has the same parity of its vertices. We observe that only vertical edges from C can contribute non-zero to any $r(i)$, as any other edge is bordered by vertices from the same B_i. Thus, if B_1 and B_2 have different parity, C contributes $+1$ every time it crosses horizontal line "up" and -1 every time it crosses horizontal line "down", which is identical for all the rows (and counts the invariant of how many "wraps around" the cycle does on the grid).

This proves that $r(i)$ is constant for a single grid G, denoted $r(G)$. The proof that for two grids G_1 and G_2 of the same size, $r(G_1) = r(G_2)$, follows from adapting Lemma 13.

Moreover, we observe that along any horizontal line, any two u^+ and u^- contribute to the sum of labels iff they are at odd L_1 distance. Thus doing the full traversal and returning to the starting vertex, the parity of sum is the parity of n. Additionally, we observe that edges contributing non-zero to sum cannot be denser than every second edge, thus $|r(i)| \leq n/2$.

The claim now follows by a straightforward application of Theorem 10 to $r(G)$. □

12 Discussion and open questions

Randomised complexity. Chang et al. [9] showed that the randomised complexity of any LCL on instances of size n is at least its deterministic complexity on instances of size $\sqrt{\log n}$. This, combined with our Theorem 2, implies that there are no LCL problems with randomised complexity between $\omega(\log^* n)$ and $o(\sqrt{\log n})$ on the grid. Whether problems with randomised complexity $O(\sqrt{\log n})$ exist is left as an open question.

High-dimensional grids. As mentioned in the beginning of the introduction, we can also consider the setting of d-dimensional (hypertoroidal oriented) grids with n^d nodes. The complexity results extend to this setting: the classification theorem and the undecidability of classification hold for

d-dimensional grids, and as noted before, the vertex and edge colouring results generalise. The techniques used in the synthesis algorithm also generalise to d-dimensional grids. However, we have not yet implemented the synthesis beyond $d = 2$, and we expect that the increased size of the search space may make the synthesis less feasible.

Bounded growth graphs. The proof of Theorem 2 intrinsically exploits the fact that the size of a neighbourhood $N_r(v)$ grows quadratically in r, and thus any algorithm with running time $T(n) = o(n)$ cannot see all n^2 nodes of the graph for large n. We show that this is not a phenomenon restricted to grids: for any class of graphs with limited neighbourhood growth rate, we get a large complexity gap. See Appendix A.2 for the precise statement and the proof.

Sublinear problems on general graphs. Finally, we use techniques inspired by grid graphs to expand our understanding of the complexity landscape of LCL problems on general bounded-degree graphs. Recall that in general we know that the lower end of the complexity landscape is sparse: for deterministic algorithms, there is nothing between the classes $O(1)$, $\Theta(\log^* n)$, and $\Theta(\log n)$. There are also obviously problems of complexity $\Theta(n)$, but the gap between $\Theta(\log n)$ and $\Theta(n)$ is largely unexplored. In Appendix A.3 we show how to engineer an LCL problem with a complexity of precisely $\Theta(\sqrt{n})$ in general bounded-degree graphs.

Acknowledgements. We would like to thank Orr Fischer for many discussions related to these research questions. This work was supported in part by the Academy of Finland, Grants 285721 and 289002.

References

[1] Paul C. Attie and E. Allen Emerson. Synthesis of concurrent programs for an atomic read/write model of computation. *ACM Trans. Program. Lang. Syst.*, 23(2):187–242, 2001. doi:10.1145/383043.383044.

[2] Yoah Bar-David and Gadi Taubenfeld. Automatic discovery of mutual exclusion algorithms. In *Proc. 17th International Conference on Distributed Computing (DISC 2003)*, volume 2848 of *Lecture Notes in Computer Science*, pages 136–150. Springer, 2003. doi:10.1007/978-3-540-39989-6_10.

[3] Leonid Barenboim. Deterministic $(\Delta + 1)$-coloring in sublinear (in Δ) time in static, dynamic and faulty networks. In *Proc. 34th ACM Symposium on the Principles of Distributed Computing (PODC 2016)*, pages 345–354. ACM, 2015. doi:10.1145/2767386.2767410.

[4] Leonid Barenboim and Michael Elkin. Distributed deterministic edge coloring using bounded neighborhood independence. *Distributed Computing*, 26(5):273–287, 2013. doi:10.1007/s00446-012-0167-7.

[5] Leonid Barenboim, Michael Elkin, and Fabian Kuhn. Distributed $(\Delta + 1)$-coloring in linear (in Δ) time. *SIAM Journal on Computing*, 43(1):72–95, 2014. doi:10.1137/12088848X.

[6] Leonid Barenboim, Michael Elkin, and Tzalik Maimon. Deterministic distributed $(\delta + o(\delta))$-edge-coloring and vertex-coloring of graphs with bounded diversity, 2016. arXiv:1610.06759.

[7] Roderick Bloem, Nicolas Braud-Santoni, and Swen Jacobs. Synthesis of self-stabilising and Byzantine-resilient distributed systems. In *Proc. 28th International Conference on Computer*

Aided Verification (CAV 2016), volume 9779 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2016. doi:10.1007/978-3-319-41528-4_9.

[8] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A Lower Bound for the Distributed Lovász Local Lemma. In *Proc. 48th Annual Symposium on the Theory of Computing (STOC 2016)*, pages 479–488. ACM, 2016. doi:10.1145/2897518.2897570. arXiv:1511.00900.

[9] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An Exponential Separation Between Randomized and Deterministic Complexity in the LOCAL Model. In *Proc. 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*, pages 615–624. IEEE, 2016. arXiv:1602.08166.

[10] Alonzo Church. Application of recursive arithmetic to the problem of circuit synthesis. In *Summaries of talks presented at the Summer Institute of Symbolic Logic*, volume 1, pages 3–50, 1957.

[11] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. 3rd Workshop on Logic of Programs (LOP 1981)*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1982. doi:10.1007/BFb0025774.

[12] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.

[13] Danny Dolev, Keijo Heljanko, Matti Järvisalo, Janne H. Korhonen, Christoph Lenzen, Joel Rybicki, Jukka Suomela, and Siert Wieringa. Synchronous counting and computational algorithm design. *Journal of Computer and System Sciences*, 82(2):310–332, 2016. doi:10.1016/j.jcss.2015.09.002.

[14] David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, 2012. doi:10.1145/2380656.2380675.

[15] Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 321–330. IEEE, 2005. doi:10.1109/LICS.2005.53.

[16] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local Conflict Coloring. In *Proc. 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*, pages 625–634. IEEE, 2016. doi:10.1109/FOCS.2016.73. arXiv:1511.01287.

[17] Niloy Ganguly, Biplab K. Sikdar, Andreas Deutsch, Geoffrey Canright, and P. Pal Chaudhuri. A survey on cellular automata. Technical report, Centre for High Performance Computing, Dresden University of Technology, 2003.

[18] Martin Gardner. The fantastic combinations of John Conway's new solitaire game 'life'. *Scientific American*, 223(4):120–123, 1970.

[19] Beat Gfeller and Elias Vicari. A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs. In *Proc. 26th Annual ACM Symposium on Principles of Distributed Computing (PODC 2007)*, pages 53–60, New York, NY, USA, 2007. ACM. doi:10.1145/1281100.1281111.

[20] Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 2505–2523. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.166.

[21] Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman and Company, New York, NY, USA, 1987.

[22] Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs, February 2014. arXiv:1402.2543.

[23] Jarkko Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1): 3–33, 2005. doi:10.1016/j.tcs.2004.11.021.

[24] Donald E. Knuth. Dancing links. In Jim Davies, Bill Roscoe, and Jim Woodcock, editors, *Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford–Microsoft Symposium in Honour of Sir Tony Hoare*, Cornerstones of Computing, pages 187–214. Palgrave Macmillan, 2000. arXiv:cs/0011047.

[25] Fabian Kuhn, Thomas Moscibroda, Tim Nieberg, and Roger Wattenhofer. Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In *Proc. 19th International Symposium on Distributed Computing (DISC 2005)*, volume 3724 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2005. doi:10.1007/11561927_21.

[26] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local Computation: Lower and Upper Bounds. *Journal of the ACM*, 63(2):17:1–17:44, 2016. doi:10.1145/2742012. arXiv:1011.5470.

[27] Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *Proc. of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS 2001)*, pages 389–398, Washington, DC, USA, 2001. IEEE Computer Society.

[28] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1): 193–201, 1992. doi:10.1137/0221015.

[29] Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, 1984.

[30] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

[31] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001. doi:10.1007/PL00008932.

[32] Alessandro Panconesi and Aravind Srinivasan. The local nature of Δ-colouring and its algorithmic applications. *Combinatorica*, 15(2):255–280, 1995. doi:10.1007/BF01200759.

[33] Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014. doi:10.1007/s11047-013-9379-4.

[34] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2000.

[35] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proc. 31st Annual Symposium on Foundations of Computer Science (FOCS 1990)*, volume 2, pages 746–757, 1990. doi:10.1109/FSCS.1990.89597.

[36] Joel Rybicki and Jukka Suomela. Exact bounds for distributed graph colouring. In *Proc. 22nd International Symposium on Structural Information and Communication Complexity (SIROCCO 2015)*, volume 9439 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2015. doi:10.1007/978-3-319-25258-2. arXiv:1502.04963.

[37] Johannes Schneider and Roger Wattenhofer. An optimal maximal independent set algorithm for bounded-independence graphs. *Distributed Computing*, 22(5):349–361, 2010. doi:10.1007/s00446-010-0097-1.

[38] Johannes Schneider, Michael Elkin, and Roger Wattenhofer. Symmetry breaking depending on the chromatic number or the neighborhood growth. *Theoretical Computer Science*, 509:40–50, 2013. doi:10.1016/j.tcs.2012.09.004.

[39] Alvy Ray Smith, III. Introduction to and survey of cellular automata or polyautomata theory. In *Automata, Language, Development*, pages 405–422. North-Holland Publishing Co., 1976.

[40] Aaron D. Sterling. A limit to the power of multiple nucleation in self-assembly. In *Proc. 22nd International Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2008. doi:10.1007/978-3-540-87779-0_31. arXiv:0902.2422.

[41] Moshe Y. Vardi. From Church and prior to PSL. In *25 years of model checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 150–171. Springer, 2008. doi:10.1007/978-3-540-69850-0_10.

[42] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[43] Hao Wang. Proving theorems by pattern recognition – II. *Bell System Technical Journal*, 40 (1):1–41, 1961.

[44] Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998. URL `http://resolver.caltech.edu/CaltechETD:etd-05192003-110022`.

[45] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

[46] Damien Woods. Intrinsic universality and the computational power of self-assembly. *Philosophical Transactions of the Royal Society A*, 373(2046), 2015. doi:10.1098/rsta.2014.0214.

A Appendix

A.1 Generating tiles

Consider a graph G with a maximal independent set I. A *tile* of (G, I) is a pair (G', I'), where G' is an induced subgraph of G and $I' = V(G') \cap I$. Observe that the property of being a tile is *hereditary*: If (G', I') is a tile, G'' is an induced subgraph of G', and $I'' = V(G'') \cap I'$, then (G'', I'') is a tile of the original graph G. Consequently, one may construct tiles of a graph through a sequence of induced subgraphs. To present the algorithm for one step in such a sequence we need to define the concept of closed neighbourhood.

The *closed neighbourhood* of a vertex $v \in V(G)$ consists of v and the vertices adjacent to v and is denoted by $N_G[v]$. For a set of vertices $V' \subseteq V(G)$, we further define $N_G[V'] := \bigcup_{v \in V'} N_G[v]$. In the sequel, unless otherwise mentioned, we assume that we are dealing with the graph G so that G' is an induced subgraph of G and G'' is an induced subgraph of G' (as well as G, obviously).

We now want to extend tiles (G'', I'') to tiles (G', I') in all possible ways (the unknown is I'). Let $V_d = (V(G') \setminus V(G'')) \setminus N_G[I'']$. For each independent set I_d of V_d, $(G', I'' \cup I_d)$ is a candidate to be tile and has to be checked. This can be done as follows. Let $V_u = V(G') \setminus N_G[I'' \cup I_d]$. If $V_u = \emptyset$, then we have a tile since any independent set can be extended to a maximal independent set and none of the additional vertices could come from $V(G')$.

If $V_u \neq \emptyset$, then we have a tile if and only if there is an independent set I_n in $(V(G) \setminus V(G')) \setminus N_G[I'' \cup I_d])$ such that $V_u \subseteq N_G[I_n]$. We now form sets $S_v = (N_G[v] \setminus V(G')) \setminus N_G[I'' \cup I_d])$ for each $v \in V_u$ and the computational problem is to find an independent set that intersects each of S_v. This resembles variants of the set cover (hitting set) problem, and corresponding instances may be solved using a SAT solver or by implementing a tailored backtrack search, e.g., along the lines of Knuth [24].

For (powers of) toroidal grid graphs we consider "rectangular" tiles with $a \times b$ vertices and sequences such as $0 \times b \to 1 \times b \to 2 \times b \to \cdots \to a \times b$.

In the construction of tiles, one could further make use of symmetries (automorphism groups) to achieve some speed-up. However, for tiles of the types mentioned above, the orders of the groups are only 4 (rectangular case) and 8 (square case), and the basic algorithm is already fast enough to handle the instances considered in this work.

A.2 Speed-up on graphs of bounded growth

Let $f\colon \mathbb{N} \to \mathbb{N}$ be a strictly increasing function, and let $\mathcal{G}$ be a class of graphs. Recall that $\mathcal{G}$ is *f-growth-bounded* if for every $G \in \mathcal{G}$ and $v \in V(G)$, we have that $|N_r(v)| \leq f(r)$. Moreover, we say that $\mathcal{G}$ is *neighbourhood-hereditary* if there is a constant C such that for any graph $G \in \mathcal{G}$, vertex $v \in V(G)$ and constant r, for any $k \geq C|N_r(v)|$ there is a graph $G' \in \mathcal{G}$ such that $|V(G')| = k$ and $G[N_r(v)]$ is isomorphic to a induced subgraph of G'.

Lemma 26. *Let $\mathcal{G}$ be a neighbourhood-hereditary f-growth-bounded family of graphs with constant maximum degree Δ, where $f(n) = \omega(n)$. If LCL problem P can be solved deterministically on $\mathcal{G}$ in $T(n) = o\big(f^{-1}(n)\big)$ rounds, then P can be solved deterministically on $\mathcal{G}$ in $O(\log^* n)$ rounds.*

Proof. Denote the radius parameter of the LCL problem P with r, and assume that there exists an algorithm A for P with complexity $T(n) = o\big(f^{-1}(n)\big)$. We show that there exists an algorithm A' for P that runs in $O(\log^* n)$ rounds on $\mathcal{G}$.

First, we fix a constant k such that $f(2T(k) + 3) < k/C$, where C is as in the definition of the neighbourhood-hereditary graph class; such k exists since we have $T(n) = o\big(f^{-1}(n)\big)$. The algorithm A' now functions as follows on an input graph $G \in \mathcal{G}$:

(1) Find a distance-$(2T(k)+3)$ colouring with $f(2T(k)+3)+1 \leq k$ colours. This can be done in $O(\log^* n)$ rounds by simulating a $(\Delta+1)$-colouring algorithm in the power graph $G^{(2T(k)+3)}$, since the maximum degree in $G^{(2T(k)+3)}$ is at most k.
(2) Simulate A on G with implicit assumption that the instance size is k, using the colours given by (1) as unique identifiers; as the simulation runs in $T(k)$ rounds, nodes will not see any duplicate colours.

Now consider any node $v \in V(G)$; we want to show that the labelling given to $N(v)$ by A' is valid. Since $\mathcal{G}$ is neighbourhood-hereditary class, there is a graph $G' \in \mathcal{G}$ with $|V(G')| = k$ such that $N_{2T(k)+3}(v)$ is isomorphic to an induced subgraph of G'. Moreover, since no colour given by (1) occurs twice in $N_{2T(k)+3}(v)$, this colouring can be extended to a valid assignment of unique identifiers on G'. Since A produces a valid output on G', the output of A' on $N(v)$ is also valid. □

A.3 Sublinear complexity problems on general graphs

In this paper we have considered $n \times n$ grids. In this setting a global problem has $\Omega(\sqrt{N})$ complexity where $N = n^2$ is the size of the input. We define an LCL problem with complexity $\Theta(\sqrt{n})$ where the input is a graph G on n vertices without any restrictions. The basic idea is that if G is a grid, we force the corners to coordinate, and otherwise we allow the nodes to have any output. Before we define the problem, we introduce the following terms to describe nodes with different local neighbourhoods. Any node whose $O(1)$ radius neighbourhood is not isomorphic to the neighbourhood of some node in a grid is said to be a *broken* node. Any other node is a *corner* node if it has degree 2 and an *internal* node otherwise.

The corner coordination problem:

- If there are no corner nodes, then nodes can output anything
- Otherwise, nodes must direct some (or possibly none) of their incident edges according to the following rules

  (1) The set of directed edges forms a set of directed pseudotrees: each node must have at most one outgoing edge in each tree
  (2) The pseudotrees have a consistent orientation: a path in one of the pseudotrees can cross each row and column at most once
  (3) Only corner nodes can be roots or leaves of the pseudotrees
  (4) Pseudotrees can only meet at corners or broken nodes
  (5) Each corner must be the root or leaf of at least one pseudotree

In order to make this a locally checkable labelling problem, the output of a node v should be a (possibly empty) set of labels for its incident edges, which must include an indication of the forbidden rows and columns for any pseudotree containing the labelled edge. This can be achieved in the following manner. Suppose v needs to direct the edge $e = vw$ toward w. Then v can include in the label the identifier of one of its neighbours v' that is a forbidden neighbour of a successor of w in the pseudotree. In other words, if w wants to direct the edge $e' = wx$ towards x, then x cannot be adjacent to u. Furthermore, if w does direct e' toward x, it will include a the identifier of w' in the label and this must be consistent with the choice of v': w' should be adjacent to v' or it should be adjacent to v. This is certainly locally checkable, but we need the set of labels to be of

constant size. We can derive a port numbering from the unique identifiers of the neighbourhood of v. Instead of the identifier, v can include the port number of v' in the label of e. The degree of v is at most 4 and so the set of labels is of constant size.

Theorem 27. *The corner coordination problem has complexity* $\Theta(\sqrt{n})$

Proof. First we prove a lower bound. Suppose there exists an algorithm A for the corner coordination problem that runs in time $T(n) = o(\sqrt{n})$. Let G be a 2-dimensional grid on $N = m^2$ vertices. We refer to the nodes of G by coordinates i, j in the obvious way so that $v_{0,0}$ has degree 2. This is a convention to aid our discussion; the nodes are not aware of these coordinates. Consider the set of pseudotrees in the output of $A(G)$. Since every corner node must be the root or leaf of at least one pseudotree, there must be a pseudotree T that consists of a path along one side of the grid. Without loss of generality T is the directed path $(v_{0,0}, v_{0,1}, \ldots, v_{0,m})$. We obtain a new input graph G' from G by taking the ball $B_\infty(v_{0,\frac{m}{2}}, \epsilon m)$ for some sufficiently small constant ϵ and rotating it about $v_{0,\frac{m}{2}}$. Now consider the output of $A(G')$. Since the $T(n)$-radius neighbourhood of $v_{0,0}$ in G is isomorphic to the $T(n)$-radius neighbourhood of $v_{0,0}$ in G', there is a tree $T_{0,0}$ in the output of $A(G')$ whose root is $v_{0,0}$. Similarly, there is a tree $T_{0,m}$ whose leaf is $v_{0,m}$. There is also a pseudotree T' with a path going through $v_{0,\frac{m}{2}}$ but in G' this path goes the "wrong" way in the sense that it points *towards* $v_{0,0}$. This forces $T_{0,0}$ and $T_{0,m}$ to be different trees, since $T_{0,0}$ must eventually include a vertex $v_{1,j}$ and therefore its leaf can not be $T_{0,m}$. Furthermore, since the first edge of $T_{0,0}$ is $v_{0,0}v_{0,1}$, the leaf of $T_{0,0}$ cannot be $v_{m,0}$. So the leaf of $T_{0,0}$ must be $T_{m,m}$ and by a similar argument, the root of $T_{0,m}$ is $T_{m,0}$. But this is a contradiction as the pseudotrees cannot cross.

Now we show that the problem can be solved in $2\sqrt{n}$ rounds. It is enough to show that in $2\sqrt{n}$ rounds, a corner node v sees a corner node or a broken node. Suppose that v has not seen a corner or broken node after r rounds.

Proposition 28. *The r-radius neighbourhood of a corner node that has not seen a corner or broken node contains* $\binom{r+2}{2}$ *nodes.*

The number of nodes at distance exactly k from the corner is at most the number of ordered pairs of non negative integers that sum to k, which is $k + 1$. It is well known that $\sum_{k=0}^{r} k + 1 = \binom{r+2}{2}$. Now when $r = 2\sqrt{n}$, the number of vertices that v has seen is greater than n. So in $2\sqrt{n}$ rounds, v must see a corner node or a broken node. This completes the proof. □