Deterministic Local Algorithms, Unique Identifiers, and Fractional Graph Colouring

Henning Hasemann[1], Juho Hirvonen[2], Joel Rybicki[2], and Jukka Suomela[2]

[1] Institute of Operating Systems and Computer Networks, TU Braunschweig, Germany
[2] Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Finland

Abstract. We show that for any $\alpha > 1$ there exists a deterministic distributed algorithm that finds a fractional graph colouring of length at most $\alpha(\Delta + 1)$ in any graph in one synchronous communication round; here Δ is the maximum degree of the graph. The result is near-tight, as there are graphs in which the optimal solution has length $\Delta + 1$.

The result is, of course, too good to be true. The usual definitions of scheduling problems (fractional graph colouring, fractional domatic partition, etc.) in a distributed setting leave a loophole that can be exploited in the design of distributed algorithms: the size of the local output is not bounded. Our algorithm produces an output that seems to be perfectly good by the usual standards but it is impractical, as the schedule of each node consists of a very large number of short periods of activity.

More generally, the algorithm shows that when we study distributed algorithms for scheduling problems, we can choose virtually any trade-off between the following three parameters: T, the running time of the algorithm, ℓ, the length of the schedule, and κ, the maximum number of periods of activity for a any single node. Here ℓ is the objective function of the optimisation problem, while κ captures the "subjective" quality of the solution. If we study, for example, bounded-degree graphs, we can trivially keep T and κ constant, at the cost of a large ℓ, or we can keep κ and ℓ constant, at the cost of a large T. Our algorithm shows that yet another trade-off is possible: we can keep T and ℓ constant at the cost of a large κ.

1 Introduction

In the study of *deterministic distributed algorithms*, it is commonly assumed that there are *unique numerical identifiers* available in the network: in an n-node network, each node is labelled with a unique $O(\log n)$-bit number.

In the general case, numerical identifiers are, of course, very helpful—many fast distributed algorithms crucially depend on the existence of numerical identifiers, so that they can use the Cole–Vishkin technique [2] and similar tricks. However, when we move towards the fastest possible distributed algorithms, the landscape looks very different.

1.1 Local Algorithms and Numerical Identifiers

We focus on *local algorithms* [8,11], i.e., distributed algorithms that run in constant time (a constant number of communication rounds), independently of the size of the network. In this context, it is no longer obvious if unique identifiers are of any use:

1. In their seminal work, Naor and Stockmeyer [8] prove that there is a class of problems—so-called LCL problems—that do not benefit from unique numerical identifiers: if an LCL problem can be solved with a local algorithm, it can also be solved with an *order-invariant* local algorithm. Order-invariant algorithms do not exploit the numerical value of the identifier; they merely compare the identifiers with each other and use the relative order of the identifiers.
2. More recently, Göös et al. [3] have shown that for a large class of optimisation problems—so-called PO-checkable problems—local algorithms do not benefit from any kind of identifiers: if a PO-checkable optimisation problem can be approximated with a local algorithm, the same approximation factor can be achieved in anonymous networks if we are provided with a port-numbering and an orientation.

While the precise definitions of LCL problems and PO-checkable problems are not important here, they both share the following seemingly technical requirement: it is assumed that the *size of a local output is bounded by a constant*. That is, for each node in the network, there is only a constant number of possible local outputs, independently of the size of the network. However, previously it has not been known whether this is a necessary condition or merely a proof artefact—while contrived counter-examples exist, natural counter-examples have been lacking.

1.2 Contributions

In this work we provide the missing piece of the puzzle: we show that the condition is necessary, even if we focus on natural graph problems and natural encodings of local outputs. More precisely, we show that there is a classical graph problem—namely, *fractional graph colouring* (see Sect. 2)—with the following properties:

1. In a natural problem formulation, the local outputs can be arbitrarily large.
2. The problem can be solved with a deterministic local algorithm; the algorithm exploits both numerical identifiers and unbounded local outputs.
3. The problem cannot be solved with a deterministic local algorithm without numerical identifiers.
4. The problem cannot be solved with a deterministic local algorithm if we require that the local outputs are of a constant size.

Moreover, this is not an isolated example. The same holds for many other scheduling problems—for example, *fractional domatic partitions* have similar properties. It is up to the reader's personal taste whether this work should be interpreted as a novel technique for the design of local algorithms, or as a cautionary example of a loophole that needs to be closed.

1.3 Comparison with Other Graph Problems

In the study of local algorithms, we often have to focus on bounded-degree graphs [4,5,6,7]. If we have a constant maximum degree Δ, then a constant-size local output is a very natural property that is shared by a wide range of combinatorial graph problems—at least if we use a natural encoding of the solution:

1. Independent set, vertex cover, dominating set, connected dominating sets, etc.: The output is a subset $X \subseteq V$ of nodes. Each node outputs 1 or 0, indicating whether it is part of X.
2. Matching, edge cover, edge dominating set, spanning subgraphs, etc.: The output is a subset $Y \subseteq E$ of edges. A node of degree d outputs a binary vector of length d, with one bit for each incident edge.
3. Vertex colouring, domatic partition, minimum cut, maximum cut, etc.: The output is a partitioning of nodes, $X_1 \cup X_2 \cup \cdots \cup X_k = V$. Each node outputs an integer $i \in \{1, 2, \ldots, k\}$, indicating that it belongs to subset X_i. In most cases, there is a natural constant upper bound on k: for example, a vertex colouring does not need more than $\Delta + 1$ colours, a domatic partition cannot contain more than $\Delta + 1$ disjoint dominating sets, and a cut by definition has $k = 2$.
4. Graph properties: Each node outputs 1 or 0. For a yes-instance, all nodes have to output 1, and for a no-instance, at least one node has to output 0.

Now if we consider the linear programming (LP) relaxations of problems such as independent sets, vertex covers, or dominating sets, we arrive at a graph problem in which local outputs could be potentially arbitrarily large: each node outputs a rational number, and there is no a priori reason to require that the size of the output (i.e., the length of the binary encoding of the rational number) is bounded. However, it seems that for these problems the size of the output cannot be exploited by a local algorithm—for example, in the case of packing and covering LPs, an exact solution cannot be found by any local algorithm, and the local approximation schemes [6,7] do not need to exploit unbounded local outputs. Indeed, if we had an algorithm that produces arbitrarily large outputs, we could apply a simple rounding scheme without losing too much in the approximation ratio.

However, fractional graph colouring—the LP relaxation of the vertex colouring problem—is a different story. There we not only have unbounded local outputs, but we show that we can exploit this property in the design of local algorithms.

2 Fractional Graph Colouring

In the fractional graph colouring problem, the task is to coordinate the activities of the nodes in a conflict-free manner. Each node has to perform at least one unit of work, and whenever a node is active all of its neighbours have to be inactive. The objective is to minimise the total length of the schedule, i.e., complete the

activities as quickly as possible. The applications include the coordination of radio transmissions in a wireless network: each node must transmit one unit of data, and the transmissions of adjacent nodes interfere with each other.

Definitions. Let $G = (V, E)$ be a simple, undirected graph that represents a distributed system: each node $v \in V$ is a computational entity, and each edge $\{u, v\} \in E$ represents a communication link between a pair of nodes. Let

$$\mathcal{I} = \{I \subseteq V : \text{if } u, v \in I \text{ then } \{u, v\} \notin E\}$$

consist of all *independent sets* of G. A *fractional graph colouring* associates a value $x(I) \geq 0$ to each $I \in \mathcal{I}$ such that

$$\sum_{I \in \mathcal{I}:\, v \in I} x(I) \geq 1 \quad \text{for all} \quad v \in V.$$

The *length* of a colouring x is

$$\ell(x) = \sum_{I \in \mathcal{I}} x(I),$$

and an optimal fractional graph colouring minimises $\ell(x)$. See Fig. 1a for an illustration.

The connection between a colouring x and a conflict-free schedule is straightforward: we simply allocate a time slot of length $x(I)$ to I. For example, if we are given a colouring x, we can choose an arbitrary ordering $\mathcal{I} = \{I_1, I_2, \dots\}$ on $\mathcal{I}$, and schedule the activities of the nodes as follows: first all nodes in I_1 are active for $x(I_1)$ time units, then all nodes in I_2 are active for $x(I_2)$ time units, etc.; after $\ell(x)$ time units each nodes have been active for at least one time unit. Conversely, if we can coordinate the activities, we can construct a graph colouring x, as at each point in time the set of active nodes is in $\mathcal{I}$.

Schedules of Nodes. When we study fractional graph colouring in a distributed setting, we assume that each node produces its own part of the solution. That is, each node must know when it is supposed to be active. Formally, the *schedule of a node* $v \in V$ is a union of disjoint intervals

$$s(v) = (a_1, b_1] \cup (a_2, b_2] \cup \cdots \cup (a_k, b_k].$$

Here $0 \leq a_1 < b_1 < a_2 < b_2 < \cdots < a_k < b_k$ are rational numbers. We require that the total length of the time intervals is at least 1, that is, $\sum_i (b_i - a_i) \geq 1$. The *local output* of node v is a binary encoding of the sequence $a_1, b_1, a_2, b_2, \dots, a_k, b_k$.

We say that node v is active at time t if $t \in s(v)$; let $A(t, s) = \{v \in V : t \in s(v)\}$ consist of the nodes that are active at time t. It is straightforward to see that a schedule s defines a fractional graph colouring x of length at most L if

$$A(t, s) = \emptyset \text{ for all } t > L, \quad \text{and} \quad A(t, s) \in \mathcal{I} \text{ for all } t \leq L.$$

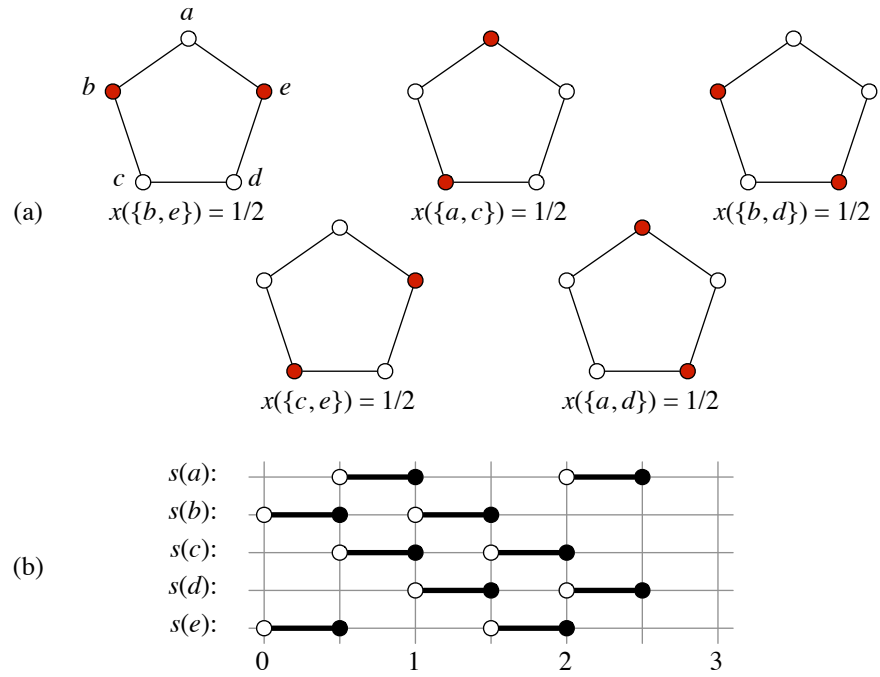

Fig. 1. (a) A fractional graph colouring x of length $\ell(x) = 5/2$ for the 5-cycle. (b) The schedules of the nodes; each node is active for 1 time unit in total, and no node is active after time 5/2.

Equivalently, we have the locally checkable conditions

$$\max s(v) \leq L \text{ for each } v \in V, \quad \text{and} \quad s(u) \cap s(v) = \emptyset \text{ for each } \{u, v\} \in E.$$

See Fig. 1b for an illustration.

3 Model of Distributed Computing

All of our results hold in the *LOCAL model* [9]. In this model, we assume that each node $v \in V$ has a unique identifier $f(v) \in \{1, 2, \ldots, \mathsf{poly}(|V|)\}$. Initially, each node knows its own identifier. Computation proceeds in synchronous communication rounds. In each round, each node in parallel (1) sends a message to each of its neighbours, (2) receives a message from each of its neighbours, (3) updates its own state. After each round, a node can stop and announce its local output. All state transitions are deterministic; there is no source of randomness available. The *running time* is the number of communication rounds until all nodes have stopped. The size of a message is unbounded, and we do not restrict local computation.

To keep our positive result as general as possible, we will not use the assumption that we have globally unique identifiers. We only assume that we have

some labelling $f\colon V \to \mathbb{N}$ such that $f(u) \neq f(v)$ for each edge $\{u, v\} \in E$. Put otherwise, we only assume that we are given some proper vertex colouring f of G—this is not to be confused with the fractional graph colouring x that we are going to output.

4 Main Results

Now we are ready to give the main result of this work.

Theorem 1. *For any $\alpha > 1$ there exists a deterministic local algorithm $\mathcal{A}$ such that in any graph G algorithm $\mathcal{A}$ finds a fractional graph colouring x for G in one communication round. Moreover, the length of x is at most $\alpha(\Delta + 1)$, where Δ is the maximum degree of G.*

We emphasise that algorithm $\mathcal{A}$ does not need to know the number of nodes in G, the maximum degree of G, or any other properties of G. Moreover, the running time is 1, independently of G. However, the theorem heavily abuses the fact that the size of the output is unbounded—the size of a local output depends on graph G and its labelling f.

The result is near-tight in the sense that there are graphs that do not have a fractional graph colouring of length shorter than $\Delta + 1$. A simple example is the complete graph on $\Delta + 1$ nodes: an optimal fractional graph colouring has length $\Delta + 1$.

From the perspective of the approximability of minimum-length fractional graph colouring, we cannot do much better, either; the following lower bound leaves only a logarithmic gap. Note that the lower bound holds even in the case of d-regular graphs, and even if the running time of the algorithm is allowed to depend on d.

Theorem 2. *Let $\mathcal{F}_d$ be the family of d-regular graphs, and let $\mathcal{A}_d$ be a deterministic algorithm that finds a fractional graph colouring for any $G \in \mathcal{F}_d$ in T_d communication rounds. Then for each d there is a graph $G_d \in \mathcal{F}_d$ such that G_d admits a fractional graph colouring of length 2, but $\mathcal{A}_d$ outputs a fractional graph colouring of length $\Omega(d/\log d)$.*

Incidentally, in the case of triangle-free graphs, the gap could be closed—we could improve the upper bound by borrowing ideas from Shearer's algorithm [10]. Closing the gap for the case of general graphs is left for future work.

5 Proof of Theorem 1

Informally, our algorithm builds on the following idea: We take an appropriate *randomised* algorithm $\mathcal{A}'$ that produces independent sets. The running time of the randomised algorithm is 1, and it does not require that the random numbers are independent for nodes that are not adjacent. Then we build a deterministic schedule that, essentially, goes through a (very large) number of

"random" numbers, and feeds these numbers to $\mathcal{A}'$. Then we simply put together all "random" independent sets that are produced by $\mathcal{A}'$. The approach is general, in the sense that we could plug in any randomised algorithm $\mathcal{A}'$ that satisfies certain technical properties. However, to keep the presentation readable, we hard-code a specific concrete choice of $\mathcal{A}'$.

5.1 Preliminaries

Choose $\varepsilon > 0$ and $\beta > 0$ such that

$$\frac{1+\beta}{1-\varepsilon} \leq \alpha.$$

Define $R(x) = \lceil (x+1)/\varepsilon \rceil$. We use the notation $N(v) = \{u \in V : \{u, v\} \in E\}$ for the set of neighbours of $v \in V$, and we write $\deg(v) = |N(v)|$ for the degree of v. Let $N^+(v) = \{v\} \cup N(v)$. The case of an isolated node is trivial; hence we assume that $\deg(v) \geq 1$ for every node v.

5.2 Communication

Recall the definitions of Sect. 3; we assume that we are given a function f that is a proper vertex colouring of graph $G = (V, E)$. The communication part of the algorithm is nearly trivial: *each node v sends its colour $f(v)$ and its degree $\deg(v)$ to each of its neighbours.*

This information turns out to be sufficient to find a fractional graph colouring. The rest of this section explains the local computations that are done by each node; they do not involve any communication at all.

5.3 Scheduling Colours

Let $g\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. We say that g is a *scheduling colour function* if

$$\begin{aligned} &g(i,j) \geq j && \text{for all } i \text{ and } j, \\ &g(i,j) \neq g(i',j') && \text{for all } i,\ i',\ j, \text{ and } j' \text{ such that } i \neq i'. \end{aligned}$$

In the algorithm, we will need a scheduling colour function g. For the sake of concreteness, we give an example of such a function:

$$g(i,j) = B(i+j-1) + i - 1, \quad \text{where} \quad B(k) = 2^{\lceil \log_2 k \rceil}.$$

Other choices of g are equally good for our purposes; the choice of g only affects the size of the local outputs.

We define that the *scheduling colour* of a node v is

$$c(v) = g\big(f(v), R(\deg(v))\big).$$

We make the following observations:

1. Function $c\colon V \to \mathbb{N}$ is a proper colouring of G, as f was a proper colouring of G.
2. We have $c(v) \geq R(\deg(v))$ for each node v.
3. Each node v knows $c(u)$ for all $u \in N^+(v)$.

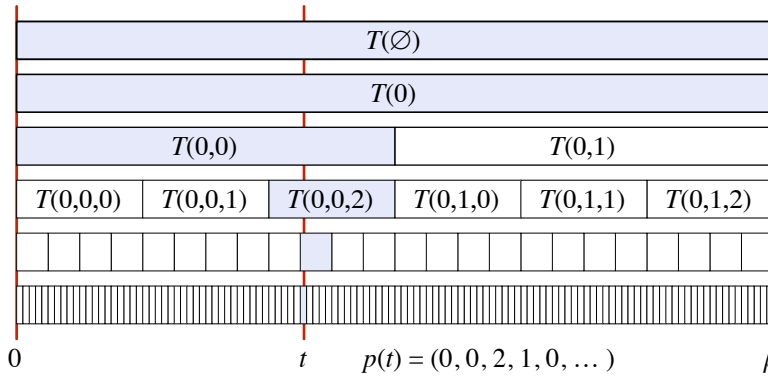

Fig. 2. Recursive partitioning $T(p)$ of the interval $(0, \beta]$.

5.4 Coordinates

A *coordinate* is a sequence $p = (p_1, p_2, \ldots, p_\ell)$ where $p_i \in \{0, 1, \ldots, i-1\}$. Here ℓ is the *dimension* of the coordinate; we write $\emptyset$ for the coordinate of dimension $\ell = 0$.

Define $\beta_i = \beta/(i!)$ for each $i \geq 0$. With each coordinate p of dimension ℓ, we associate a time interval $T(p)$ of length β_ℓ as follows (see Fig. 2 for an illustration):

1. For the 0-dimensional coordinate, set $T(\emptyset) = (0, \beta_0]$.
2. Assume that p is a coordinate of dimension $i-1$ with $T(p) = (a, a + \beta_{i-1}]$. For each $j = 0, 1, \ldots, i-1$, we define

$$T(p, j) = \big(a + j\beta_i,\ a + (j+1)\beta_i\big].$$

5.5 First Fragment of the Schedule

Now we are ready to define the schedule within time interval $T(\emptyset)$. To this end, consider a point in time $t \in T(\emptyset)$. Time t defines a unique infinite sequence

$$p(t) = \big(p(1,t), p(2,t), \ldots\big)$$

such that for any i we have

$$t \in T\big(p(1,t), p(2,t), \ldots, p(i,t)\big).$$

Define the *weight* of the colour class $k \in \mathbb{N}$ at time t as follows:

$$W(k,t) = \frac{p(k,t)}{k}.$$

We define the weight of a node v at time t as the weight of its scheduling colour:

$$w(v,t) = W(c(v),t).$$

Finally, we define that v is active at time t if it is strictly heavier than any neighbour, that is

$$w(v,t) > w(u,t) \quad \text{for all } u \in N(v). \tag{1}$$

Note that each node v knows $c(u)$ for each $u \in N^+(v)$. Hence each node knows when it is active. Moreover, the schedule can be efficiently computed and it is of finite length. To see this, let

$$c'(v) = \max_{u \in N^+(v)} c(u).$$

Let p be a coordinate of length $c'(v)$. Now the weights $w(u,t)$ for $u \in N^+(v)$ are constant during $t \in T(p)$; hence v is either active or inactive during the entire time period $T(c'(v))$. Hence it is sufficient to consider a finite number of time periods.

We will now argue that the schedule for $T(\emptyset)$ is feasible and, moreover, each node is active for a substantial fraction of $T(\emptyset)$. To this end, define

$$h(v) = \frac{1-\varepsilon}{\deg(v)+1}.$$

Lemma 1. *If $\{u,v\} \in E$, nodes u and v are never active simultaneously during $T(\emptyset)$.*

Proof. This is trivial, as we had a strict inequality in (1).

Lemma 2. *Each node $v \in V$ is active for at least $\beta h(v)$ time units within time interval $T(\emptyset)$.*

Proof. Assume that we choose a point in time $t \in T(\emptyset)$ uniformly at random. Then the random variables $p(i,t) \in \{0,1,\ldots,i-1\}$ for $i = 1,2,\ldots$ are independent and uniformly distributed; it follows that the random variables $W(i,t)$ are also independent and uniformly distributed. For any i and any $0 \le x \le 1$ we have

$$\Pr\big[W(i,t) < x\big] \ge x.$$

Let $v \in V$, and let $C = \{c(u) : u \in N(v)\}$ be the set of scheduling colours in the neighbourhood of v; note that $c(v) \notin C$. Let $n = |C|$ and $k = c(v)$. Summing over all possible values of $W(k,t)$, we have

$$\begin{aligned}
&\Pr\big[\text{node } v \text{ is active at time } t\big] \\
&= \Pr\big[w(v,t) > w(u,t) \text{ for all } u \in N(v)\big] \\
&= \Pr\big[W(k,t) > W(i,t) \text{ for all } i \in C\big]
\end{aligned}$$

$$= \sum_{j=0}^{k-1} \Pr\Big[W(k,t) = \frac{j}{k}\Big] \cdot \Pr\Big[\frac{j}{k} > W(i,t) \text{ for all } i \in C\Big]$$

$$\geq \sum_{j=0}^{k-1} \frac{1}{k}\Big(\frac{j}{k}\Big)^n = \frac{1}{k^{n+1}}\Big(\sum_{j=1}^{k} j^n\Big) - \frac{1}{k}$$

$$\geq \frac{1}{k^{n+1}} \int_0^k x^n \, dx - \frac{1}{k} = \frac{1}{n+1} - \frac{1}{k}.$$

Moreover, $n \leq \deg(v)$ and $k \geq R(\deg(v)) \geq (\deg(v)+1)/\varepsilon$. Therefore node v is active at time t with probability at least

$$\frac{1}{n+1} - \frac{1}{k} \geq \frac{1-\varepsilon}{\deg(v)+1} = h(v).$$

5.6 Complete Schedule

In Sect. 5.5 we defined the schedule for time interval $T(\emptyset)$. As such, this does not yet constitute a valid fractional graph colouring—indeed, it cannot be the case, as $T(\emptyset)$ is far too short.

However, we can now easily construct a valid solution by repeating the solution that we defined for $T(\emptyset)$. Define

$$H(v) = \left\lceil \frac{1}{\beta h(v)} \right\rceil. \tag{2}$$

Now the schedule $s(v)$ of node v is defined as follows: repeat the schedule defined for $T(\emptyset)$ for $H(v)$ times.

More formally, let $t > 0$. If $t \leq \beta$, we have defined in Sect. 5.5 whether v is active at time t. Otherwise $t = i\beta + t'$, where $t' \in T(\emptyset)$ and $i \in \mathbb{N}$. If $i \geq H(v)$, node v is inactive. Otherwise node v is active at time t iff it is active at time t'.

Lemma 3. *Each node $v \in V$ is active for at least 1 time unit within time interval $(0, \beta H(v))$.*

Proof. Follows from Lemma 2 and (2).

Lemma 4. *If the maximum degree of G is Δ, then the length of the schedule is at most $\alpha(\Delta+1)$.*

Proof. Let $v \in V$. We have

$$\beta H(v) \leq \frac{1}{h(v)} + \beta = \frac{\deg(v)+1}{1-\varepsilon} + \beta \leq \frac{\Delta+1}{1-\varepsilon} + \beta \leq \frac{1+\beta}{1-\varepsilon}(\Delta+1) \leq \alpha(\Delta+1).$$

That is, after time $\alpha(\Delta+1)$, node v is no longer active.

This concludes the proof of Theorem 1—we have designed an algorithm that only needs one communication round, yet it yields a fractional graph colouring of length at most $\alpha(\Delta+1)$.

6 Proof of Theorem 2

The theorem holds even if f assigns unique identifier from the set $\{1, 2, \ldots, n\}$, where n is the number of nodes in G_d. The proof uses the following lemma.

Lemma 5 (Bollobás [1]). *For any given integers $d \geq 3$ and $g \geq 3$, there exists a d-regular graph G with n nodes and girth at least g such that any independent set has size at most $O(n \log(d)/d)$.*

Let $\mathcal{F}$ be the family of d-regular graphs. Let $\mathcal{A}$ be a deterministic algorithm, with running time T, that finds a fractional graph colouring for any graph in $\mathcal{F}$. Now let $G = (V, E)$ be a d-regular graph with girth $g \geq 2T + 1$ obtained from Lemma 5; we have $G \in \mathcal{F}$. Each independent set I of G has size at most $c|V| \log(d)/d$, for some constant c. Thus any fractional graph colouring of G has length at least $d/(c \log d)$. Choose a bijection $f \colon V \to \{1, 2, \ldots, |V|\}$.

If we run algorithm $\mathcal{A}$ on G with identifiers given by f, the output is a fractional graph colouring x of length at least $d/(c \log d)$. In particular there must be a node $v^* \in V$ that is active at time $t \geq d/(c \log d)$. Moreover, the radius-T neighbourhood of v^* is a d-regular tree, as G was a high-girth graph.

Now let $G' = (V', E')$ be the bipartite double cover of G. That is, for each node v of G we have two nodes v_1 and v_2 in G', and for each edge $\{u, v\}$ of G we have two edges $\{u_1, v_2\}$ and $\{u_2, v_1\}$ in G'. There is a covering map $\phi \colon V' \to V$ that maps $v_1 \mapsto v$ and $v_2 \mapsto v$; let $\{v_1^*, v_2^*\} = \phi^{-1}(v^*)$. Graph G' has the following properties.

1. Graph G' is bipartite; therefore there is a fractional graph colouring x' in G' with $\ell(x') = 2$.
2. Graph G' is d-regular; that is, $G' \in \mathcal{F}$.
3. The radius-T neighbourhood of $v_1^* \in V'$ is a d-regular tree.
4. The number of nodes is $|V'| = 2|V|$.

To prove the theorem, it is sufficient to show that we can choose the identifiers for $G' \in \mathcal{F}$ so that $\mathcal{A}$ outputs a fractional graph colouring of length $\Omega(d/\log d)$. To this end, observe that we can choose a bijection $f' \colon V' \to \{1, 2, \ldots, |V'|\}$ so that the radius-T neighbourhood of v_1^* in (G', f') is isomorphic to the radius-T neighbourhood of v^* in (G, f). Now apply $\mathcal{A}$ to (G', f'). By construction, the local output of v_1^* in (G', f') equals the local output of v^* in (G, f); in particular, the length of the schedule x' constructed by $\mathcal{A}'$ is $\Omega(d/\log d)$.

7 Discussion

We have shown that the fractional graph colouring problem can be solved very quickly in a distributed setting—if and only if we do not impose artificial restrictions on the size of the local outputs.

More generally, we can approach scheduling problems from the following perspective. We have three parameters:

1. T, the running time of the distributed algorithm,
2. ℓ, the length of the schedule (objective function),
3. κ, the maximum number of disjoint time intervals in the schedule of a node.

Now for the sake of concreteness, let us focus on the case of bounded-degree graphs, i.e., $\Delta = O(1)$. Our work shows that we can keep any two of T, ℓ, and κ constant, but not all three of them:

1. $T = O(1)$ and $\kappa = O(1)$: trivial, set $s(v) = (f(v), f(v) + 1]$.
2. $\kappa = O(1)$ and $\ell = O(1)$: easy, find an $O(1)$-colouring c and set $s(v) = (c(v), c(v) + 1]$.
3. $T = O(1)$ and $\ell = O(1)$: possible, using Theorem 1.
4. $T = O(1)$, $\ell = O(1)$, and $\kappa = O(1)$: impossible. Now we have an LCL-problem. It is easy to see that the problem cannot be solved with an order-invariant local algorithm (consider a cycle), and hence the result by Naor and Stockmeyer [8] implies that the problem cannot be solved with any local algorithm.

Acknowledgements. We thank the anonymous reviewers for their helpful comments. This work has been partially supported by the European Union under contract number ICT-2009-258885 (SPITFIRE), the Academy of Finland, Grants 132380 and 252018, the Research Funds of the University of Helsinki, and the Finnish Cultural Foundation.

References

1. Bollobás, B.: The independence ratio of regular graphs. Proceedings of the American Mathematical Society 83(2), 433–436 (1981)
2. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Information and Control 70(1), 32–53 (1986)
3. Göös, M., Hirvonen, J., Suomela, J.: Lower bounds for local approximation. In: Proc. 31st Symposium on Principles of Distributed Computing (PODC 2012).
4. Hirvonen, J., Suomela, J.: Distributed maximal matching: greedy is optimal. In: Proc. 31st Symposium on Principles of Distributed Computing (PODC 2012).
5. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proc. 23rd Symposium on Principles of Distributed Computing (PODC 2004). pp. 300–309. ACM Press, New York (2004)
6. Kuhn, F., Moscibroda, T., Wattenhofer, R.: The price of being near-sighted. In: Proc. 17th Symposium on Discrete Algorithms (SODA 2006). pp. 980–989. ACM Press, New York (2006)
7. Kuhn, F., Moscibroda, T., Wattenhofer, R.: Local computation: Lower and upper bounds (2010), manuscript, arXiv:1011.5470 [cs.DC]
8. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM Journal on Computing 24(6), 1259–1277 (1995)
9. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia (2000)
10. Shearer, J.B.: A note on the independence number of triangle-free graphs. Discrete Mathematics 46(1), 83–87 (1983)
11. Suomela, J.: Survey of local algorithms. ACM Computing Surveys (2011), `http://www.cs.helsinki.fi/local-survey/`, to appear.