

# Speedup of Distributed Algorithms for Power Graphs in the CONGEST Model

Leonid Barenboim 

Open University of Israel, Ra'anana, Israel

Uri Goldenberg 

Ben-Gurion University of the Negev, Beersheba, Israel

---

## Abstract

We obtain improved distributed algorithms in the CONGEST message-passing setting for problems on power graphs of an input graph  $G$ . This includes Coloring, Maximal Independent Set, and related problems. For  $R = f(\Delta^k, n)$ , we develop a general deterministic technique that transforms  $R$ -round LOCAL model algorithms for  $G^k$  with certain properties into  $O(R \cdot \Delta^{k/2-1})$ -round CONGEST algorithms for  $G^k$ . This improves the previously-known running time for such transformation, which was  $O(R \cdot \Delta^{k-1})$ . Consequently, for problems that can be solved by algorithms with the required properties and within polylogarithmic number of rounds, we obtain *quadratic* improvement for  $G^k$  and *exponential* improvement for  $G^2$ . We also obtain significant improvements for problems with larger number of rounds in  $G$ . Notable implications of our technique are the following deterministic distributed algorithms:

- We devise a distributed algorithm for  $O(\Delta^4)$ -coloring of  $G^2$  whose number of rounds is  $O(\log \Delta + \log^* n)$ . This improves exponentially (in terms of  $\Delta$ ) the best previously-known deterministic result of Halldorsson, Kuhn and Maus. [25] that required  $O(\Delta + \log^* n)$  rounds, and the standard simulation of Linial [30] algorithm in  $G^k$  that required  $O(\Delta \cdot \log^* n)$  rounds.
- We devise an algorithm for  $O(\Delta^2)$ -coloring of  $G^2$  with  $O(\Delta \cdot \log \Delta + \log^* n)$  rounds, and  $(\Delta^2 + 1)$ -coloring with  $O(\Delta^{1.5} \cdot \log \Delta + \log^* n)$  rounds. This improves quadratically, and by a power of  $4/3$ , respectively, the best previously-known results of Halldorsson, Kuhn and Maus. [25].
- For  $k > 2$ , our running time for  $O(\Delta^{2k})$ -coloring of  $G^k$  is  $O(k \cdot \Delta^{k/2-1} \cdot \log \Delta \cdot \log^* n)$ . Our running time for  $O(\Delta^k)$ -coloring of  $G^k$  is  $\tilde{O}(k \cdot \Delta^{k-1} \cdot \log^* n)$ . This improves best previously-known results quadratically, and by a power of  $3/2$ , respectively.
- For constant  $k > 2$ , our upper bound for  $O(\Delta^{2k})$ -coloring of  $G^k$  nearly matches the lower bound of Fraigniaud, Halldorsson and Nolin. [16] for *checking* the correctness of a coloring in  $G^k$ .

**2012 ACM Subject Classification** Theory of computation → Distributed computing models; Mathematics of computing → Graph coloring

**Keywords and phrases** Distributed Algorithms, Graph Coloring, Power Graph, CONGEST

**Digital Object Identifier** 10.4230/LIPIcs.DISC.2024.6

**Related Version** *Full Version:* <https://arxiv.org/abs/2305.04358> [10]

**Funding** This research was supported by The Open University of Israel's Research Fund.

## 1 Introduction

### 1.1 Model and Results

In the distributed message-passing model a communication network is represented by an unweighted  $n$ -vertex graph  $G = (V, E)$  with maximum degree  $\Delta$ . Each vertex has a unique ID, represented by  $O(\log n)$  bits. Computations proceed in synchronous discrete rounds, each of which consists of message exchange between neighbours, and local computations of vertices. The input for an algorithm is the network graph  $G$ , where initially each vertex



© Leonid Barenboim and Uri Goldenberg;

licensed under Creative Commons License CC-BY 4.0

38th International Symposium on Distributed Computing (DISC 2024).

Editor: Dan Alistarh; Article No. 6; pp. 6:1–6:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

knows only its own ID and the IDs of its neighbours<sup>1</sup>, and the values  $n$  and  $\Delta$ . During an execution, within  $k > 0$  rounds, vertices may obtain information about other vertices in their  $k$ -hop-neighborhood. The output of an algorithm consists of the final answers returned by all vertices. The running time of an algorithm is the number of rounds until the algorithm terminates in all vertices. In the current paper we focus on the *CONGEST* model. In this model the number of bits that can be sent over each edge in each round is bounded by  $O(\log n)$ . Consequently, within  $k$  rounds each vertex can learn only a small portion of the information that resides in its  $k$ -hop-neighborhood. Such a neighborhood may contain up to  $\Delta^k + 1$  vertices, whose information is much larger than what can be received in  $k$  rounds, namely up to  $O(k \cdot \Delta \cdot \log n)$  bits. This is in contrast to the distributed *LOCAL* model, where message size is unbounded, and each vertex can learn the entire  $k$ -hop-neighborhood within  $k$  rounds.

Among the most studied problems in this setting are coloring and maximal independent set (henceforth, MIS). These problems are very well motivated by real-life network tasks, such as resource allocation, scheduling, channel assignment, etc. Often, in order to perform such a task, a coloring or MIS has to be computed on a *power graph* of the network, rather than the original graph that represents the network. For example, in job scheduling, where each vertex can perform one job at a time, and can send one neighbor a job to perform, a coloring is used for job scheduling. In this scheduling all vertices of a certain color are executed at the same time. Then these vertices and neighbours selected by them execute jobs. However, an ordinary proper coloring will not suffice, since a vertex may have several neighbours with the same color, who send it jobs to perform. Since it can handle only one job, the other jobs that are sent simultaneously are lost. To prevent this, 2-distance coloring is used, where each pair of vertices at distance at most 2 one from another in  $G$  obtain distinct colors. (This is equivalent to an ordinary proper coloring of the power graph  $G^2$ .) Now each vertex has at most one neighbor with a certain color, and no more than one job arrives to the vertex at a time.

Because of their importance, problems on power graphs for computing coloring, MIS, and related tasks have been very intensively studied in the distributed setting. A plethora of significant results have been obtained in recent years in the CONGEST setting [4, 13, 16, 25, 26, 33]. In particular, Bar-Yehuda, Censor-Hillel, Maus, Pai and Pemmaraju [4] devised approximate Minimum Dominating Set and Minimum Vertex cover (MVC) algorithms for  $G^2$ . Halldorsson, Khun, Maus and Nolin. [26] devised a logarithmic-time randomized algorithm for distance-2 coloring using  $\Delta^2 + 1$  colors. (For a positive integer  $q$ , distance-2 coloring with  $q$  colors is equivalent to ordinary proper coloring of  $G^2$  with  $q$  colors.) Halldorsson, Khun and Maus [25] devised deterministic algorithms for distance-2 coloring using  $O(\Delta^2)$  colors with time  $\text{polylog}(n)$ , and using  $(\Delta^2 + 1)$  colors with time  $O(\Delta^2 + \log^* n)$ . Fraigniaud, Halldorsson and Nolin [16] showed that for  $k > 2$ , testing whether a given proper coloring is correct requires  $\Omega(\Delta^{\lfloor (k-1)/2 \rfloor})$  rounds. Also, a general well-known scheme for simulating LOCAL  $R$ -round algorithms for  $G^k$  in the CONGEST setting provides algorithms with time  $O(R \cdot \Delta^{k-1})$  in the CONGEST setting.

In the last decades, much attention of researchers was devoted to understanding the complexity of these problems on  $G$  and  $G^k$  as a function of  $\Delta$ , modulo the unavoidable factor of  $\log^* n$ . A major question that remained open for many years is whether  $o(\Delta) + \log^* n$  solutions are possible for  $O(\Delta)$ -coloring, MIS and related problems on  $G$ . In recent breakthroughs it was shown that while MIS and Maximal Matching require  $\Omega(\Delta)$  time [3], the problem

---

<sup>1</sup> A node can learn the ids of its neighbours within a single round in the CONGEST model

of  $O(\Delta)$ -coloring can be solved in  $\tilde{O}(\sqrt{\Delta} + \log^* n)$  time [5]. However, this question for coloring power graphs in sublinear-in- $\Delta^k$  time still remained open, since the best results for  $O(\Delta^k)$ -coloring of  $G^k$  were at least  $O(\Delta^k)$ . Specifically, for  $k = 2$  it is  $O(\Delta^2 + \log^* n)$  [25], and for  $k > 2$  it is  $\tilde{O}(\Delta^{k+k/2-1} + \Delta^{k-1} \log^* n)$  (by applying a standard simulation for  $G^k$  to [9]).

In this paper we answer this question in the affirmative, by providing deterministic CONGEST algorithms for  $O(\Delta^2)$ -coloring of  $G^2$  with  $\tilde{O}(\Delta + \log^* n)$  rounds, and  $O(\Delta^k)$ -coloring of  $G^k$  with  $\tilde{O}(\Delta^{k-1} + \log^* n)$  rounds<sup>2</sup>. More generally, we provide a speedup technique for various problems, including coloring and MIS, that improves quadratically each phase of an algorithm for  $G^k$  that adheres to certain requirements. In particular, such algorithms that perform the standard simulation are improved from running time  $O(R \cdot \Delta^{k-1})$  to  $O(R \cdot \Delta^{k/2-1})$ . In the case of  $G^2$  our speedup of a phase is *exponential* (in terms of  $\Delta$ ), resulting in running time  $O(R \cdot \text{polylog}(\Delta))$ . For example, we compute  $O(\Delta^4)$ -coloring of  $G^2$  within  $O(\log \Delta + \log^* n)$  time, improving the best previously-known result of  $O(\Delta + \log^* n)$ .

Our results also give rise to a quadratic improvement in the *memory complexity* per vertex. Specifically, when using *aggregation functions*, the size of the result computed by  $w$  for each vertex in its  $k/2$ -hop-neighborhood is reduced from  $\Delta^{k/2}$  to a much smaller value, ideally,  $O(1)$ . Using such technique the required memory per vertex is also reduced from  $\tilde{O}(\Delta^k)$  to  $\tilde{O}(\Delta^{k/2})$ .

An interesting implication of our results is that  $O(\Delta^{2k})$ -coloring of  $G^k$  can be computed in the CONGEST model in  $O(\Delta^{k/2-1} \log \Delta \log^* n)$  rounds. This nearly matches the lower bound of Fraigniaud, Halldorsson and Nolin [16] for testing a proper coloring that requires  $\Omega(\Delta^{\lfloor (k-1)/2 \rfloor})$  rounds.

## 1.2 Our Techniques

The previously-known technique for simulating CONGEST algorithms from  $G$  on  $G^k$  proceeds as follows. For each round of the simulated algorithm, each vertex  $v \in V$  has to obtain its  $k$ -hop-neighborhood information. Since the number of vertices in the  $k$ -hop-neighborhood is bounded by  $O(\Delta^k)$  and the number of edges is bounded by  $O(\Delta^k \cdot \Delta) = O(\Delta^{k+1})$ , this neighborhood structure (consisting of vertices and edges between them) can be delivered to  $v$  within  $O(\Delta^k)$  rounds. This information is sufficient for  $v$  to simulate its local computation in that round for the algorithm in  $G^k$ . Then,  $v$  broadcasts a message to all vertices in its  $k$ -hop-neighborhood. Such a broadcast is performed by all vertices of  $G$  in parallel, and requires  $O(\Delta^{k-1})$  rounds. We note that often it is sufficient to employ  $O(\Delta^{k-1})$  rounds also in the stage of obtaining the  $k$ -hop-neighborhood information. This is the case when only information about vertices is needed, rather than how they are connected. Consequently, various CONGEST algorithms that require  $f(\Delta, n)$  rounds in  $G$  can be transformed into  $O(f(\Delta^k, n) \cdot \Delta^{k-1})$ -round algorithms for  $G^k$ .

Our new method improves this idea, by performing stages of information collection and information broadcast only half the way, to distance  $k/2$  rather than  $k$ . Indeed, for each pair of vertices  $u, v \in V$  at distance  $k$  one from another, there is a vertex  $w$  in the middle of a path between them that can obtain their information by collecting its  $k/2$ -hop-neighborhood. Then  $w$  can perform a computation on  $u$  and  $v$  (as well as all other vertices in its  $k/2$ -hop-neighborhood) and return the results to them. However, if this is done in a trivial way, then  $w$

<sup>2</sup> For simplicity of presentation, we assume that  $k$  is even. Our results extend directly to any positive integer  $k \geq 2$ , by replacing  $k/2$  with  $\lceil k/2 \rceil$ .

should return information of size  $\Delta^{k/2}$  to each vertex in its  $k/2$ -hop-neighborhood. In general, this is done not only by  $w$ , but by all vertices in the graph in parallel. Consequently, in order to collect  $\Delta^{k/2}$  pieces of information from each of  $\Delta^{k/2}$  vertices in a  $k/2$ -hop-neighborhood, again  $O(\Delta^k)$  rounds are needed. Thus, a straightforward approach for going half the way does not provide an improvement. But we obtain an improvement using the following more sophisticated method.

Various algorithms can be decomposed into basic steps that perform such operations as checking whether a variable of a vertex appears in its  $k$ -hop-neighborhood, summing the number of certain value, computing maximum or minimum, etc. In such cases, rather than collecting the entire  $k$ -hop-neighborhood information and then computing a function locally, aggregation functions can be applied iteratively and distributively. For example, consider a function for computing the maximum of variables in the  $k$ -hop-neighborhood. In this case,  $w$  who is in the middle of a  $k$ -length path between  $u$  and  $v$  applies the function iteratively and locally on all vertices in the  $k/2$ -hop-neighborhood of  $w$ . The vertex  $w$  sends the result to all its immediate neighbours, including the one that is at distance  $k/2 - 1$  from  $v$ . Then this vertex computes the maximum of the maxima it received from its neighbours, and sends it to its own neighbours, one of which is at distance  $k/2 - 2$  from  $v$ . After  $k/2$  such rounds,  $v$  receive the maximum value in its  $k$ -hop-neighborhood.

The example of the maximum function is a simple case, when the maximum in the  $k/2$ -hop-neighborhood of  $w$  is the same for all its vertices. However, in general, different answers may be needed for different vertices. Another example is a function that checks whether variables of vertices are equal. It may be the case that  $u$  has a vertex in the  $k/2$ -hop-neighborhood of  $w$  with a variable that equals to that of  $u$ , but  $v$  does not have such a variable. Then,  $w$  must store for each vertex in its  $k/2$ -hop-neighborhood its own answer. This is done by  $w$ , by applying aggregation functions locally and iteratively, for each such vertex. The result of size  $\Delta^{k/2}$  is sent to all neighbours of  $w$  in parallel. Then, each neighbor applies the aggregation functions for all vertices in its  $(k/2 - 1)$ -hop-neighborhood. It does so starting from the results it received from  $w$ . This way, the outcome now is regarding distance  $k/2 + 1$ . This continues for  $k/2$  phases, where in each phase less data has to be sent (it is reduced by a factor of  $\Delta$  in each phase), but the radius of the computation grows by 1. After  $k/2$  such phases, each vertex holds the result of function applications in its  $k$ -hop-neighborhood.

In order for this technique to work, it employs functions that are (1) *commutative* and (2) *idempotent*. That is, (1) the order of function application must not affect the result, and (2) applying the same function several times must not affect the result, no matter how many times it is applied. For example, this is the case in the function  $\max_t(x) = \max(x, t)$ . A series of applications  $\max_{t_1}(\max_{t_2}(\dots(\max_{t_q}(x)\dots))$  can be applied in any order, and each  $\max_{t_i}$  can appear any positive number of times, without changing the outcome. This is important to the success of our method, since pairs of vertices  $u, v$  may belong to many  $k/2$ -hop-neighborhoods that perform computations for them, and the order of computations is not predefined. As mentioned above, we show that various algorithms for complicated tasks, including coloring and MIS, can be constructed from steps consisting of such operations.

We extend our technique also to functions that are not idempotent, in order to generalize it further. In particular, the counting operation is a very useful building block in various algorithms. For example, it makes it possible to compute how many times a certain value appears in the variables of a  $k$ -hop-neighborhood of a vertex. (The function increases the result by one each time it is applied on a variable with that value. But when the function is applied on a variable with a different value, the result does not change.) However, this

operation is not idempotent, since the outcome depends on the number of times the function is applied. Consequently, if we apply our technique as described above, the result may be larger than the actual number of variables with that value. This is because a certain function invocation may be repeated several times by different vertices in the  $k$ -hop-neighborhood.

We propose two solutions for this challenge. In the first solution we analyze how much the result over-counts the correct answer. In certain cases this can be bounded, so an algorithm still works, even with over-counting. In more complicated cases, when precise computation is required, we use BFS trees of radius  $k/2$  that are constructed from all vertices in parallel. These trees are broadcasted to distance  $k/2$ . Then unique paths can be produced between pairs of vertices at distance  $k$  one from another. These paths are used to make sure that each function application is executed exactly once, for an (ordered) pair of vertices in a  $k$ -hop-neighborhood. This incurs an additional running time of  $O(\Delta^{k-1})$ , but only once during execution. Consequently, the running time of a transformation of an  $f(\Delta, n)$ -round algorithm for  $G$  becomes  $O(\Delta^{k-1} + f(\Delta^k, n) \cdot \Delta^{k/2-1})$  in  $G^k$ . This is again a significant improvement over the previously known time of  $O(f(\Delta^k, n) \cdot \Delta^{k-1})$ .

Another tool we introduce for shrinking message size, which may be of independent interest, is *binary search in neighborhoods*. A common building block of distributed algorithms is performing computations on lists that vertices hold, as follows. A vertex has to compute a certain function on its own list and the lists of its neighbours. In some functions the outcome can be determined by a single element of a list. (For example, finding an element that does not appear in the neighbours lists.) A naive computation for lists of size  $t$  requires  $t$  rounds, since the lists have to be delivered to neighbours that apply the function on them. To speedup this process we perform a binary search with assistance of neighbours, so that lists shrink by a certain factor in each round, but each of them still contains an element from which the function outcome can be deduced. Finally, each list contains just a single element, who provides the desired result. This tool is a main ingredient in our algorithm for  $O(\Delta^4)$ -coloring of  $G^2$  in  $O(\log \Delta + \log^* n)$  time, and  $O(\Delta^2)$ -coloring of  $G^2$  in  $O(\Delta \cdot \log \Delta + \log^* n)$  time.

### 1.3 Related Work

Among the first works on deterministic distributed symmetry breaking are algorithms for (1-hop) coloring and MIS of paths and trees. An  $O(\log^* n)$ -round algorithm for 3-coloring paths was devised by Cole and Vishkin in 1986 [12]. This was extended to oriented trees by Goldberg, Plotkin, and Shannon in 1987 [24]. A (1-hop) coloring deterministic algorithm for general graphs that employs  $O(\Delta^2)$ -colors and has running time  $O(\log^* n)$  was obtained by Linial in 1987 [30]. This algorithm gives rise to  $(\Delta + 1)$ -coloring in  $O(\Delta^2 + \log^* n)$  time. The running time for  $(\Delta + 1)$ -coloring was improved to  $O(\Delta \cdot \log \Delta + \log^* n)$  by Szegedy and Vishwanathan in 1993 [36], and by Kuhn and Wattenhofer in 2006 [29], by a more explicit construction. This was further improved to  $O(\Delta + \log^*)$  by Barenboim, Elkin and Kuhn in 2009 [7]. The latter result also implies an algorithm for MIS in  $O(\Delta + \log^* n)$  rounds. As proven in [3], this result for MIS is tight. On the other hand, sublinear-in- $\Delta$  algorithms for  $(\Delta + 1)$ -coloring are possible, as shown by Barenboim [5], who devised such an algorithm with running time  $\tilde{O}(\Delta^{3/4} + \log^* n)$ . This was further improved in several works [9, 17, 18, 32], to the current state of the art, which is  $\tilde{O}(\sqrt{\Delta} + \log^* n)$ .

In addition to the thread of research on algorithms with dependency on  $\Delta$  in the running time, there has been also progress with deterministic algorithms that depend on  $n$  (in a stronger way than just  $\log^* n$ ). Several results were obtained using network decompositions [2, 34, 21], and the recent breakthrough of Rozhon and Ghaffari [21] makes it possible to compute coloring and MIS deterministically within  $\text{poly}(\log n)$  time. In addition, it

is even possible to obtain  $(\Delta + 1)$ -coloring and MIS without network decompositions in  $O(\log n \log^2 \Delta)$  time [23, 14] and  $(\log^2 n)$  for MIS [20]. Both threads of research are very important and attracted much attention of researchers. In the case where the dependency on  $n$  is larger than  $\log^* n$ , a main goal is improving this dependency, as well as the dependency on  $\Delta$ . In the case that the dependency on  $n$  is  $O(\log^* n)$ , which is unavoidable, the main goal is improving the dependency on  $\Delta$ .

Randomized symmetry breaking algorithms have been very extensively studied as well. The first algorithms for  $(\Delta + 1)$ -coloring and MIS, due to Luby [31] and Alon, Babai and Itai [1] required  $O(\log n)$ -time. This was improved by Kothapalli, Scheideler, Onus and Schindelhauer in 2006 [28] who obtained  $O(\Delta)$ -coloring in  $\tilde{O}(\sqrt{\log n})$  rounds. In several major advances, this was improved even further to  $O(\log \Delta + \text{poly}(\log \log n))$  for MIS [19, 35],  $O(\sqrt{\log \Delta} + \text{poly}(\log \log n))$  for coloring [27], and  $O(\text{poly}(\log \log n))$  for coloring [11].

Since currently-known randomized algorithms have better dependency on  $\Delta$ , while deterministic algorithms have better dependency on  $n$ , the improvement of either randomized or deterministic solutions is valuable. In particular, improving the dependency on  $\Delta$  in deterministic algorithms is very important, since the gap between the current deterministic and randomized solutions is quite large.

Recently, distance- $k$  problems and computations on power graphs attracted much attention in the research of distributed algorithms. Emek, Pfister, Seidel and Wattenhofer [13] proved that every problem that can be solved (and verified) by a randomized anonymous algorithm can also be solved by a deterministic anonymous algorithm provided that the latter is equipped with a distance-2 coloring of the input graph. Computing distance- $k$  coloring is a key component in the derandomization of LOCAL distributed algorithms, due to Ghaffari, Harris and Kuhn [22] from 2019. Upper- and lower-bounds for approximate Minimum Dominating Set on power graphs were devised by Bar-Yehuda, Censor-Hillel, Maus, Pai and Pemmaraju [4] in 2020. Deterministic and randomized distance-2 coloring algorithms were obtained by Halldorsson, Kuhn and Maus [25]. Improved randomized results for distance-2 coloring were obtained by Halldorsson, Kuhn, Maus and Nolin [26]. Very recently, in 2023, Maus, Peltonen, and Uitto [33] devised deterministic algorithms for  $k$ -ruling sets on  $G^k$  with time  $\tilde{O}(k^2 \log^4 n \log \Delta)$ . They also devised randomized algorithm for this problem, as well as for MIS, with logarithm dependency on  $\Delta$  and poly-log-log dependency on  $n$ . The most recent result for 2-distance coloring is a randomized algorithm by Flin, Halldorson and Nolin [15], whose running time is  $O(\log^6 \log n)$ .

## **2 Distance-2 coloring $G$ with $O(\Delta^4)$ colors in $O(\log \Delta + \log^* n)$ rounds**

In this section we devise an algorithm for distance-2 coloring of  $G$  using  $O(\Delta^4)$  colors, which is a distance-1 coloring of  $G^2$ . Our algorithm significantly speeds-up the previously-known algorithms for distance-2 coloring with this number of colors. The previous algorithms [25] are based on simulating Linial's [30] algorithm in  $G^2$ . (See also [8], for more details about the original algorithm of Linial [30].) The algorithm of [25] for  $G^2$  requires spending  $\Delta$  rounds to simulate certain rounds of Linial's algorithm. On the other hand, our new method improves this, so that only  $O(\log \Delta)$  rounds are required to simulate a round of Linial's algorithm. Consequently, our algorithm produces a proper  $O(\Delta^4)$  coloring of  $G^2$  within  $O(\log \Delta + \log^* n)$  rounds in the CONGEST model. Moreover, most rounds of Linial's algorithm can be simulated within  $O(1)$  rounds.

The principle of our method is binary search. Consider the  $i$ -th round,  $i = 1, 2, \dots, O(\log^* n)$ , of Linial's original algorithm. Each vertex  $v$  generates a subset of possible colors  $S(\phi(v)) = \{s_1, s_2, \dots, s_j\}$  from the palette of  $\{1, 2, \dots, (\Delta \cdot \log^{(i)} n)^2\}$ , such that there exists a color  $s_j \in S(\phi(v))$ , where  $s_j \notin S(\phi(u))$ , for all neighbours  $u$  of  $v$ . Moreover, Linial's algorithm makes it possible to construct a set system, such that for any pair of neighbours  $u, v$ , it holds that

$$\frac{|S(\phi(u))|}{|S(\phi(u)) \cap S(\phi(v))|} > \Delta \text{ and } \frac{|S(\phi(v))|}{|S(\phi(u)) \cap S(\phi(v))|} > \Delta.$$

Each vertex  $v$  selects such a color  $s_j$ , which is from its own set  $S(\phi(v))$ , but does not belong to any of its neighbours sets, from which the neighbours select colors. Consequently, the coloring is proper. Since in each round the subsets are taken from smaller sets, the number of colors is reduced in each round. We show that the element  $s_j$  can be found using binary search, without knowing the neighbours sets  $S(\phi(u))$ , but only knowing the number of intersections with neighbours' sets in a specific range. When solving 2-distance-coloring using this idea in a straightforward way, each node needs to receive all the subsets from 2-distant neighbours. This causes all nodes to send messages with size of at least  $O(\Delta)$  for each node in order to receive messages with all of its distance-2 neighbours colors. (Each neighbor of a given vertex sends it the colors of all its own neighbours.) Using this information a vertex can compute the available color to choose. However, this approach exchanges much more information than needed and can be optimized for restricted bandwidth models. We provide this optimization in Section 2.1.

## 2.1 High level description

Our technique does not use an ordinary set, but an ordered set, thus we can perform a binary search. The goal of the binary search, for a vertex  $v \in V$ , is finding an element in  $S(v)$ , that does not belong to any set  $S(u)$ , for  $u$  in the 2-hop-neighborhood of  $v$ . To this end, for each vertex we define a range, that initially contains all elements of  $S(v)$ . Then we reduce the range size by a factor of at least 2 in each stage of the binary search. Eventually, each  $v \in V$  reduces its range to contain a single element that does not belong to any range in its 2-hop-neighborhood. However, there is a cost in running time because each binary search requires  $O(\log k)$  rounds where  $k$  is the size of the colors palette of the current stage.

The technique in high level is that each vertex knows its 1-distance neighbours subsets that are based on the coloring  $\phi(u)$  and marked  $S(\phi(u))$  and computes the number of intersections with these subsets. Each node holds two indices that constitute the beginning and end of the relevant range. The neighbours are aware of those indices and refer to the beginning index by *left* and the ending index by *right*. On every iteration the number of intersecting values of the subsets is reduced by a factor of at least two, simply by counting the elements in each half of a range and choosing the half-range with less intersections. On each round each vertex receives the number of intersecting values for both their left half and right half from its neighbours and decides whether in the next round it will use the left half of the range or the right half, and update its *left*, *right* indices accordingly. The selected half of the range is the one with fewer intersecting values.

Next, we provide the pseudocode of our algorithm (see Algorithm 1 below), called *2-Distance-Linial*, and analyze its correctness and running time.

■ **Algorithm 1** 2-Distance-Linial's algorithm phase.

---

```

1: Let  $t$  be the size of each set in the current phase /* All sets have the same size in the
   same phase */
2: Let  $S(v) = S(\phi(v)) = \{s_1(v), s_2(v), \dots, s_t(v)\}$  be the ordered set produced by the al-
   gorithm of Linial for the vertex  $v$ . /*  $S(\phi(v))$  is computed locally by Linial's algorithm
   as a function of  $\phi(v)$ . */
3: Let  $S(u_i) = S(\phi(u_i)) = \{s_1(u_i), s_2(u_i), \dots, s_t(u_i)\}$ 
4: From Linial's proof we know that every  $S$  contains an element  $s_i$ , such that  $s_i$  belongs
   to  $S$  and does not belong to any other  $S(\phi(u_j))$  with  $\phi(u_j) \neq \phi(u_i)$ 
5: The vertex  $v$  performs a binary search in the set  $S$ , with assistance of its neighbours, as
   follows:
6:  $left = 1, right = t$ 
7: while  $left \neq right$  do
8:   for  $u \in \Gamma(v)$  in parallel do
9:      $Int_l(v, u) =$  Number of intersections of  $S(v)$  and  $\{s_i(u) \mid i \in [left, \dots, \frac{right}{2} - 1]\}$ 
10:     $Int_r(v, u) =$  Number of intersections of  $S(v)$  and  $\{s_i(u) \mid i \in [\frac{right}{2}, right]\}$ 
11:   end for
12:   All neighbours  $u$  of  $v$  notify  $v$ , in parallel, about  $Int_l(u, v)$  and  $Int_r(u, v)$ .
13:   All nodes in parallel send  $Int_l(u_i, v)$  and  $Int_r(u_i, v)$  to 1-hop-neighbours  $u_i$ 
14:   A node  $v$  receives  $Int_l(u, v)$  and  $Int_r(u, v)$  from all of its neighbours
15:   The vertex  $v$  computes the sums  $sum_l = \sum_{u \in \Gamma(v)} Int_l(u, v)$  and
      $sum_r = \sum_{u \in \Gamma(v)} Int_r(u, v)$ .
16:   if  $sum_r \geq sum_l$  then
17:      $right = \frac{right}{2} - 1$ 
18:     Send to all neighbours "left chosen"
19:   else
20:      $left = \frac{right}{2}$ 
21:     Send to all neighbours "right chosen"
22:   end if
23:   The new set  $S(v)$  for the next phase of the binary search is
      $S(v) = \{s_{left}(v), \dots, s_{right}(v)\}$ 
24:   Receive all neighbours [left \ right] choices and compute  $S(u)$  for all  $u \in \Gamma(v)$ 
25: end while
26: return the color  $s_{left}(v)$  /* now  $s_{left}(v) = s_{right}(v)$  */

```

---

## 2.2 Proof and run time analysis

► **Lemma 2.1.** *After each invocation of 2-Distance-Linial's algorithm phase the coloring  $\phi$  remains proper.*

► **Lemma 2.2.** *The running time of 2-distance-Linial in the CONGEST model is  $O(\log^* n \cdot \log \Delta + \log \log n)$ .*

Details and proofs of the lemmas in section 2.2 appear in the full version of this paper [10].

Next, we provide an improvement, which removes the  $O(\log \log n)$  factor from Lemma 2.2. To this end, we perform each binary search for  $O(\log \Delta)$  phases, rather than  $O(\log t)$ . Moreover, in each phase we send just one bit to indicate whether the left half is chosen or the right one, rather than sending indices of ranges. This information is sufficient to compute the new range from the previous one. After  $O(\log \Delta)$  phases, we obtain consecutive range of



size  $O(t/\Delta) = O(\log n)$ . Recall that the previous range size is  $t = O(\Delta \cdot \log n)$ , which is a square root of the number of colors. The new range defines a bit-string of size  $O(\log n)$  that represents whether there is a conflict for each element in  $\{s_{left}(v), \dots, s_{right}(v)\}$ . This string is then sent directly to  $v$  by each of its 1-hop-neighbours. This is done using  $O(\log n)$ -bits messages, within one round in parallel by all neighbours. Then  $v$  finds an index  $i \in O(\log n)$  of a bit with a 0 value, which exists since there is an index without conflicts in the range  $\{s_{left}, \dots, s_{right}\}$ . The resulting color with no conflicts is  $s_{left+i}(v)$ . We summarize this in the next theorem.

► **Theorem 2.3.** *A proper distance-2 coloring with  $O(\Delta^4)$ -colors can be computed in  $O(\log \Delta \cdot \log^* n)$  rounds in the CONGEST model.*

Theorem 2.3 demonstrates that for each of the  $\log^* n$  iterations of Linial's algorithm,  $O(\log \Delta)$  rounds are performed to compute the color for the next iteration. We now argue that it is sufficient to perform just two iterations with  $O(\log \Delta)$  rounds, while the remaining  $O(\log^* n)$  iterations require  $O(1)$  rounds each. The idea is similar to an improvement from  $O(\Delta \log^* n)$  to  $O(\Delta + \log^* n)$  of [25]. Specifically, after 2 iterations the number of colors becomes  $O(\Delta^4 \log^2 \log n)$ . If  $\Delta^4 > \log^2 \log n$ . Then this is an  $O(\Delta^8)$ -coloring. It can be converted into an  $O(\Delta^4)$ -coloring within a single iteration, using a field of size  $\Theta(\Delta^2)$ , and polynomials of degree 4. Indeed, each of the current  $O(\Delta^8)$  colors can be assigned a unique polynomial, and each such polynomial has at least one non-intersecting point with any  $\Delta^2$  others. These are used for computing new colors in a range of size  $O(\Delta^4)$ . The other possibility is that  $\Delta^4 \leq \log^2 \log n$ . Then, instead of performing a binary search, one can directly send  $\Delta$  messages, each of which consists of a color in a range of  $O(\Delta^4 \log^2 \log n) = O(\log^4 \log n)$ . Since  $\Delta \leq \sqrt{\log \log n}$ , the number of bits in the concatenation of these  $\Delta$  messages is  $\text{poly}(\log \log n)$ , so it can be sent over an edge within  $O(1)$  rounds of the CONGEST model.

► **Corollary 2.4.** *A proper distance-2 coloring with  $O(\Delta^4)$ -colors can be computed in  $O(\log \Delta + \log^* n)$  rounds in the CONGEST model.*

### 3 Distance-2 coloring with $O(\Delta^2)$ colors in $O(\Delta \cdot \log \Delta + \log^* n)$ rounds

#### 3.1 High level description

In this section we provide an improved algorithm for distance-2 coloring of  $G$  using  $O(\Delta^2)$  colors, which is distance-1 coloring of  $G^2$ . The improvement is from  $O(\Delta^2 + \log^* n)$  rounds to  $O(\Delta \cdot \log \Delta + \log^* n)$  rounds in the CONGEST model. The result is achieved by applying our technique to the algorithm of [9] that provides an  $O(\Delta)$  coloring of an input graph  $G$  in  $O(\sqrt{\Delta} \log \Delta + \log^* n)$  rounds. The algorithm of [9] is based on the following notions.

A *p-defective coloring* is a vertex coloring such that each vertex may have up to  $p$  neighbours with its color.

An *p-arbdefective coloring* is a vertex coloring, such that each subgraph induced by a color class of the coloring has arboricity bounded by  $p$ . The *arboricity* is the minimum number of forests into which the edge set of a graph can be decomposed.

The algorithm of [9] for distance-1 coloring consists of three stages:

1. Computing  $O(\sqrt{\Delta})$ -defective  $O(\Delta)$ -coloring of  $G$  in  $O(\log^* n)$  time.
2. Computing  $O(\sqrt{\Delta})$ -arbdefective  $O(\sqrt{\Delta})$ -coloring of  $G$  in  $O(\sqrt{\Delta})$  time.
3. Iterating over the  $O(\sqrt{\Delta})$  color classes of step 2, and computing a proper coloring of  $G$  iteratively. Each iteration of Stage 3 requires a constant number of rounds.

As a first step of our extension of this scheme to work in  $G^2$ , we introduce the following *proxy* communication method. The goal is establishing a single path between any pair of vertices that need to communicate, and are at distance at most 2 one from another. (In section A.2, we generalize this to vertices of distance  $k$  one from another.) This way, the desired information is passed only once, which improves communication costs, and avoids miscalculations caused by duplicated data. To this end, for each vertex, a BFS tree of radius two that is rooted at the vertex is computed. This computation starts with sending the list of neighbours of each vertex to all its neighbours. This requires  $O(\Delta)$  rounds, because in each round each vertex sends the information to a neighbor that has not been sent yet. Then, each vertex is aware of the neighbours of its neighbours. For any distance-2 neighbor, it knows the immediate neighbours that connect to it, and selects exactly one of these neighbours. The selected neighbor is referred to as *proxy*. Note that a vertex  $v$  with its proxy nodes and their neighbours form a BFS of radius two that contains the two-hop neighborhood of  $v$ .

In Sections 3.2.1 - 3.2.3, we describe the generalizations and modifications in the above-mentioned stages (1) - (3), for coloring  $G^2$  in  $O(\Delta \cdot \log \Delta + \log^* n)$  rounds in the CONGEST setting. For these computations we need the above-mentioned notion of proxy nodes.

The running time analysis assumes that the proxy nodes have already been computed. Otherwise, an  $O(\Delta)$  term should be added. However, this does not affect the overall running time of the entire algorithm of Section 3, which is  $O(\Delta \log \Delta + \log^* n)$ .

## 3.2 Detailed description of the algorithm

### 3.2.1 Our variant for distance-2 defective coloring

The computation of  $O(\Delta)$ -defective  $O(\Delta^2)$ -coloring of  $G^2$  proceeds as follows.

The procedure starts by computing a proper  $O(\Delta^4)$ -coloring of  $G^2$ , using Algorithm 1. Next, find a prime  $q = \Theta(\Delta)$ , such that the number of colors is bounded by  $q^4$ . Note that each color is represented by a tuple  $\langle a, b \rangle$ ,  $a, b \in q$  and  $Z_q$  is a field modulo  $q$ . Assign each color a unique polynomial  $p(x) = a + b \cdot x + c \cdot x^2 + d \cdot x^3$ , such that  $0 \leq a, b, c, d \leq q - 1$ . Assign each vertex a polynomial according to its color. We say that two polynomials  $p(x), p'(x)$  *intersect* at the value  $t, 0 \leq t \leq q - 1$ , if  $p(t) = p'(t)$ . Next, each vertex  $v \in V$  finds a value  $t, 0 \leq t \leq q - 1$ , such that  $p(t)$  intersects with the minimum number of polynomials of vertices of distance at most 2 from  $v$ . This is done as follows, by a binary search. The vertex  $v$  sends its polynomial to its 1-hop neighbours. Each of these neighbours  $u \in \Gamma(v)$  computes the number of intersections of  $v$ 's polynomial with polynomials of neighbours  $w$  of  $u$ , such that  $u$  is the proxy for  $\{v, w\}$ . In addition, each  $u \in \Gamma(v)$  computes the number of intersections of its polynomial with that of  $v$ . The number of intersections is computed for each half of the range  $\{0, 1, \dots, q - 1\}$ , i.e.,  $\{0, 1, \dots, \lceil q/2 \rceil\}$  and  $\{\lceil q/2 \rceil + 1, \lceil q/2 \rceil + 2, \dots, q - 1\}$ . This information is returned to  $v$  by all its 1-hop neighbours. Then  $v$  knows how many intersections with its 2-hop neighbours its polynomial has in  $\{0, 1, \dots, \lceil q/2 \rceil\}$  and  $\{\lceil q/2 \rceil + 1, \lceil q/2 \rceil + 2, \dots, q - 1\}$ . The half-range with fewer intersections is selected for the next iteration of the binary search. This is repeated for  $\log q$  iterations, until the range contains a single element  $t \in \{0, 1, \dots, q - 1\}$ . The color of  $v$  returned by the procedure is  $\langle t, p(t) \rangle$ . This completes the description of the procedure. Its correctness and running time are analyzed below.

► **Lemma 3.1.** *The procedure computes an  $O(\Delta)$ -defective  $O(\Delta^2)$ -coloring of  $G^2$ .*

Proof's details appear in the full version of this paper [10].

► **Lemma 3.2.** *The running time of the procedure is  $O(\log \Delta + \log^* n)$ .*

**Proof.** First, in order to compute  $O(\Delta^4)$  coloring we employ Algorithm 1 that has running time  $O(\log \Delta + \log^* n)$ , by Corollary 2.4. The remaining part of the procedure is a binary search on a range of size  $O(\Delta^4)$ , and thus requires  $O(\log(\Delta^4)) = O(\log \Delta)$  phases, each of which consists of a constant number of rounds. The overall running time is  $O(\log \Delta + \log^* n)$ . ◀

### 3.2.2 Algorithm for Distance-2 Arbdefective Coloring

For a graph  $G$ , given an  $O(\Delta)$ -defective  $O(\Delta^2)$ -coloring of  $G^2$ , we would like to produce an  $O(\Delta)$ -arbdefective  $O(\Delta)$ -coloring of  $G^2$  within  $O(\Delta)$  rounds. The algorithm is as follows:

This algorithm extends the ideas of [9] to work in  $G^2$  in the CONGEST model. In that paper the authors devised an  $O(\sqrt{\Delta})$ -arbdefective  $O(\sqrt{\Delta})$ -coloring algorithm for  $G$  with  $O(\sqrt{\Delta} + \log^* n)$  rounds. The main idea of the algorithm is as follows. In each round, each vertex counts how many of its neighbours have the same color as its own. The number of such neighbours is the *number of conflicts*. If a vertex has too many conflicts, it selects a new color, using a certain function. Otherwise, the vertex finalizes its color. The original algorithm [9] proceeds for  $O(\sqrt{\Delta})$  rounds, and selects the round with the smallest number of conflicts. By the pigeonhole principle, there must be a round in which the number of color conflicts is at most  $O(\sqrt{\Delta})$ . However, computing the number of conflicts with all 2-hop-neighbours is expensive, since each original round can take up to  $O(\Delta)$  rounds, when applied in a straightforward way to  $G^2$ . To improve this, each vertex collects information about conflicts from its 2-hop-neighborhood in a bit-efficient manner. Specifically, a vertex receives from each of its 1-hop neighbours the number of conflicts it has with 2-hop-neighbours, instead of lists of their colors. During the execution of the algorithm, in each iteration the conflicts are counted, and if the total number of 2-distance conflicts is below a predefined  $t$ , the vertex finalizes the current color. For any  $t \in [1, \Delta^2]$ , this stage requires  $O(\frac{\Delta^2}{t})$  time and it results in  $O(t)$ -arbdefective  $O(\frac{\Delta^2}{t})$ -coloring of  $G^2$ . Setting  $t = O(\Delta)$  results in  $O(\Delta)$ -arbdefective  $O(\Delta)$ -coloring in time  $O(\Delta)$ . See pseudocode of Algorithms 2 - 3 below. (Each color in an initial  $O(\Delta^2)$ -coloring is represented by an ordered pair  $\langle a, b \rangle$ , where  $a, b \in O(\Delta)$ . When Algorithm 3 terminates, the resulting color resides in the  $b$ -coordinate, and it is in the range  $[0, 1, \dots, O(\Delta)]$ .) Next, we analyze Algorithm 3.

■ **Algorithm 2** Procedure number-of-conflicts ( $\langle a_0, b_0 \rangle, \langle a_1, b_1 \rangle \dots \langle a_n, b_n \rangle$ ).

---

```

1: /* This procedure is performed internally by a vertex, within 0 rounds */
2: numberOfConflicts = 0
3: for  $i = 1, 2, \dots, n$  do
4:   if  $b_0 = b_i$  then
5:     numberOfConflicts = numberOfConflicts + 1
6:   end if
7: end for
8: return numberOfConflicts

```

---

■ **Algorithm 3** 2-Distance AG Arbdefective Coloring( $maxDefect = t$ ).

---

```

1: We are given a  $p$ -coloring  $\phi$ . Denote  $q$  as the smallest prime number such that  $q \geq \sqrt{p}$ .
   The parameter  $maxDefect$  is the maximal arb-defect allowed for coloring.
2: Denote  $\phi(v) = \langle a, b \rangle$ , where  $a, b \leq q$ 
3: while  $\phi(v) \neq \langle 0, b \rangle$  do
4:   Denote by  $conflicts(v, u) \leftarrow \text{number-of-conflicts}(\phi(v) \cup_{u_i \in \Gamma(u)} \phi(u_i))$ 
5:   for  $i = 1, 2, \dots, deg(v)$  in parallel do
6:     send to the  $i$ th neighbor of  $v$ , which is  $u_i$ , the message  $conflicts(u_i, v)$ 
7:   end for
8:   Receive all  $conflicts(v, u_i)$  messages from neighbours
9:   if  $\sum_{(v, u_i) \in E} conflicts(v, u_i) \leq maxDefect$  then
10:     $\phi(v) = \langle 0, b \rangle$ 
11:    Send “Done” to all neighbours
12:   else
13:     $\phi(v) = \langle a, a + b \bmod q \rangle$ 
14:    Send “Not done” to all neighbours
15:   end if
16:   Receive all “Done”, “Not Done” messages from neighbours, and compute  $\phi(u_i)$  for
      $i = 1, 2, \dots, deg(v)$ 
17: end while

```

---

► **Lemma 3.3.** *After running 2-Distance AG Arbdefective Coloring for  $\lceil 2\Delta^2 / maxDefect \rceil$  rounds, all vertices have colors of the form  $\langle 0, b \rangle$ , and each color class has arboricity at most  $maxDefect$  in  $G^2$ .*

Proof’s details appear in the full version of this paper [10].

### 3.2.3 Iterative Algorithm for Distance-2 Proper Coloring

In this subsection we describe an algorithm that produces an  $O(\Delta^2)$ -proper-coloring of  $G^2$  within  $O(\Delta \cdot \log \Delta + \log^* n)$  rounds. This algorithm is based on a technique devised in [9], but our algorithm extends this technique to work for  $G^2$ . For more details regarding this technique, see section 3 in [5]. Our new algorithm starts with computing an  $O(\Delta)$ -arbdefective  $O(\Delta)$ -coloring  $\varphi$  for  $G^2$ . The coloring  $\varphi$  constitutes a partition of the graph into  $O(\Delta)$  color sets  $V_1, V_2, \dots, V_d$ ,  $d \in O(\Delta)$ . Each color class is  $O(\Delta)$ -arbdefective. This means that the arboricity of a subgraph of  $G^2$  induced by  $V_j$ ,  $j \in O(\Delta)$ , is bounded by  $O(\Delta)$ . Moreover, each pair of vertices  $u, v$  at distance at most 2 one from another have a parent-child relation in a certain forest. Specifically, when an arbdefective coloring is computed with Algorithm 3, if  $u, v$  terminate (arrive to step 10 of the algorithm) in distinct rounds, then the parent is the vertex that terminated earlier. Otherwise, the parent is the vertex with lower ID. Vertices do not have to know their parents explicitly.

The algorithm iterates over  $i = 1, 2, \dots, d$ . In each iteration  $i$  the algorithm computes a new color  $\varphi'$  for all of the nodes with color  $\phi(v) = i$ , using at most  $O(\Delta^2)$  colors. To this end, each vertex constructs a set of polynomials, and finds a polynomial  $P$  in this set, such that:

- (1) The number of intersections of the polynomial  $P$  with colors  $\varphi'$  of vertices in its 2-hop neighborhood that already selected such colors in previous rounds is as small as possible.
- (2) The number of intersections of  $P$  with polynomials of its parents in its 2-hop-neighborhood that are active in the same round  $i$  is as small as possible.

The construction of the polynomial set of a vertex  $u \in V_i$  is performed as follows. Let  $q = O(\Delta)$  be a prime, such that  $q > c \cdot \Delta$ , for a sufficiently large constant  $c \geq 1$ . We represent the color  $\varphi(u)$  by  $\langle a, b \rangle$ , where  $0 \leq a, b < q$ . The set of polynomials of  $u$  is  $\{a \cdot x^2 + b \cdot x + j \mid j = 0, 1, \dots, q - 1\}$ . The number of polynomials in the set is  $q = O(\Delta)$ .

According to (1), our goal is finding a polynomial  $P$  in the set of  $u$ , such that the number of vertices at distance at most 2 from  $u$  with the following property is minimized.

(\*) For a vertex  $w$  that already has a color  $\varphi'(w) = \langle a'_w, b'_w \rangle$ , there exists  $t \in 0, 1, \dots, q - 1$ , such that  $\langle a'_w, b'_w \rangle = \langle t, P(t) \rangle$ .

According to the Pigeonhole principle, there must be a polynomial in the set of size  $q > c \cdot \Delta$ , for which at most  $q/2$  vertices at distance at most 2 satisfy this property. This is because each vertex satisfies this property for at most one polynomial in this set (the set consists of non-intersecting polynomials), and the number of vertices at distance at most 2 is at most  $q^2/2$ . Our goal is finding such a polynomial. The challenge is that when running a naive version of this algorithm in the CONGEST model, every vertex needs to know its 2-hop neighbours' colors. Sending this information requires  $O(\Delta)$  rounds. Next we describe an optimization that requires only to compute how many intersections there are in sets of polynomials. This speeds up the running time from  $O(\Delta)$  to  $O(\log \Delta)$ .

Next, we describe how each vertex selects the desired polynomial  $a \cdot x^2 + b \cdot x + j$  from its set within  $O(\log \Delta)$  rounds. This is done using a binary search on  $j$ . To this end, each vertex has to inform its neighbours about its set of polynomials. Even though there are  $q$  polynomials in the set, this is done just within one round, as follows. Given a set of polynomials  $\{a \cdot x^2 + b \cdot x + j \mid j = 0, 1, \dots, q - 1\}$  of a vertex  $u$ , only the coefficients  $a, b$  are sent to the neighbours of  $u$ . (Each coefficient requires  $O(\log \Delta)$  bits.) Then the neighbours can reconstruct the set of polynomials from  $a, b$ , since they know that  $j$  runs from 0 to  $q - 1$ . Next, every vertex initialize  $start = 1$  and  $end = q$  and defines two ranges. The ranges are  $low = [start, \lceil \frac{end-start}{2} \rceil]$  and  $high = [\lceil \frac{end-start}{2} \rceil + 1, end]$ . At the first step each vertex sends to each of its neighbours  $w$  the number of intersections of colors  $\varphi'$  with polynomials that have  $j$  in range  $low$ , as well as the number of intersections for  $j$  in range  $high$ . In the next step each vertex receives from its neighbours the number of such intersections in ranges  $low$  and  $high$  in its 2-hop-neighborhood. Then each vertex decides for its new  $start$  and  $end$  according to the half range in which there are fewer intersection with its polynomials. Consequently, after halving  $O(\log \Delta)$  times, the range contains just a single value  $\hat{j}$ . It defines a single polynomial from the set, which is  $a \cdot x^2 + b \cdot x + \hat{j}$ .

The next lemma provides a helpful property of the polynomials, which will assist us to compute the coloring of a set  $V_i$ , given colorings of  $V_1, V_2, \dots, V_{i-1}$ .

► **Lemma 3.4.** *Suppose that we are given a graph with  $O(\Delta)$ -arbdefective  $O(\Delta)$ -coloring  $\varphi$  of  $G^2$ , that partitions the input graph into subsets  $V_1, V_2, \dots, V_{O(\Delta)}$ , according to color classes of  $\varphi$ . Moreover for an integer  $i \geq 0$ , suppose that we already have a proper 2-distance coloring  $\varphi'$  for  $V_1, V_2, \dots, V_{i-1}$ . Then we can find a polynomial  $P = a \cdot x^2 + b \cdot x + \hat{j}$  for each vertex  $u$  with  $\varphi(u) = i$ , such that at least half of the elements in the set  $\{\langle 0, P(0) \rangle, \langle 1, P(1) \rangle, \dots, \langle q - 1, P(q - 1) \rangle\}$  does not appear as  $\varphi'$  colors in the 2-hop neighborhood of  $u$ .*

Proof's details appear in the full version of this paper [10].

According to Lemma 3.4, it is possible to iterate over the color classes of the arbdefective coloring  $\varphi$ , for  $i = 1, 2, \dots, O(\Delta)$ . In each iteration  $i$ , each vertex in the color class  $i$  obtains a single polynomial with the properties stated in the lemma. Specifically, it has sufficiently many elements that still can be used for their  $\varphi'$  color. Specifically, the number of elements is larger (by a factor greater than 2) than the number of their parents in  $G^2$ . Consequently,

a variant of Linial's algorithm that considers only parents in the 2-hop-neighborhood can be executed. (For the case of distance-1 coloring, this is a well-known extension of Linial's algorithm, which is called arb-Linial [6].) In the case of distance-2 coloring it can be computed in  $O(\log \Delta)$  phases in the same way as in Corollary 2.4, but considering only 2-hop-parents, rather than entire 2-hop-neighborhood. Recall that a vertex can deduce the parent-child relationship of a pair of its neighbours, by inspecting their termination round in Algorithm 3 and their IDs. Hence, we obtain the following Corollary.

► **Corollary 3.5.** *It is possible to compute a proper distance-2 coloring with  $O(\Delta^2)$  colors within  $O(\Delta \cdot \log \Delta + \log^* n)$  rounds in the CONGEST model.*

### 3.2.4 Coloring $G^2$ using $(\Delta^2 + 1)$ colors in $O(\Delta^{\frac{3}{2}} \cdot \log \Delta + \log^* n)$ rounds

In this section we show how to reduce the number of colors to  $(\Delta^2 + 1)$ . To this end, we parameterize the steps of our scheme in a different way, as follows.

1. Compute an 0-defective  $O(\Delta^2)$ -coloring of  $G^2$ , i.e., a proper  $O(\Delta^2)$ -coloring.
2. Compute  $O(\sqrt{\Delta})$ -arbdefective  $O(\Delta^{3/2})$ -coloring of  $G^2$ .
3. Iterate over the  $O(\Delta^{3/2})$  color classes that were generated in step 2, and compute a proper coloring of  $G^2$  iteratively, using  $\Delta^2 + O(\Delta^{3/2})$  colors.
4. Apply a simple reduction to produce a  $(\Delta^2 + 1)$ -coloring of  $G^2$  from a  $\Delta^2 + O(\Delta^{3/2})$ -coloring.

The steps are performed as follows. Step 1 is obtained by applying Corollary 3.5. This step requires  $O(\Delta \cdot \log \Delta + \log^* n)$  rounds. Step 2 is obtained by applying Lemma 3.3 with  $\maxDefect = \sqrt{\Delta}$  and  $q = \Theta(\Delta^{3/2})$ . This step requires  $O(\Delta^{3/2})$  rounds. Step 3 is performed similarly to Section 3.2.3, but now we have  $O(\Delta^{3/2})$  color classes to iterate on, rather than  $O(\Delta)$ . On the other hand, the arboricity of each of them is significantly smaller, and consequently a proper coloring from a range of size  $\Delta^2 + O(\Delta^{3/2})$  can be computed. To this end, let  $q$  be a prime, such that  $q > \Delta + O(\sqrt{\Delta})$ ,  $q < 2 \cdot \Delta + O(\sqrt{\Delta})$ . Then each vertex  $v$  in  $G^2$  has at most  $\Delta^2$  neighbours which have finalized their colors in previous rounds, thus it cannot select their colors. By the pigeonhole principle, there exists a polynomial  $P$  of  $v$ , such that  $P$  intersects with at most  $\frac{\Delta^2}{q}$  neighboring colors. This polynomial is defined over a field of size  $q$ , and there is a value  $t \leq \Delta^2/q + O(\sqrt{\Delta})$ , such that  $\langle t, P(t) \rangle$  does not intersect with any neighboring polynomial, nor with any neighboring color. Thus,  $\langle t, P(t) \rangle$  is selected as the color of  $v$ , and it is from a range of size  $q \cdot (\frac{\Delta^2}{q} + O(\sqrt{\Delta})) + q = \Delta^2 + O(\Delta^{3/2})$ . This is the resulting number of colors of step 3, whose running time is  $O(\Delta^{3/2} \cdot \log \Delta)$  as described in Section 3.2.3.

Next we describe step 4, which reduces the number of colors from  $\Delta^2 + O(\Delta^{3/2})$  to  $\Delta^2 + 1$ . This step is performed using an adaptation of a simple color reduction for  $G$  to  $G^2$ . The simple color reduction for  $G$  works as follows. Each vertex whose color is greater than all its neighbours colors (and greater than  $\Delta(G) + 1$ ), selects a new color from  $\{1, 2, \dots, \Delta(G) + 1\}$  that is not used by any neighbor. By starting from a proper  $(\Delta(G) + k)$ -coloring, for a parameter  $k > 1$ , and repeating this for  $k - 1$  rounds, a proper  $(\Delta(G) + 1)$ -coloring is achieved. If message size is unbounded, this can be directly applied in  $G^2$ . However, each vertex needs to collect the colors of its 2-distance neighbourhood. This is in order to know which colors are available. Thus,  $O(\Delta)$  rounds are required to simulate each original round. To perform this in the CONGEST setting efficiently, we improve this by invoking a binary search to find an available color. This is done as in Section 2 and requires  $O(\log \Delta)$  rounds. Consequently, an  $O(\Delta^2 + O(\Delta^{3/2}))$ -coloring of  $G^2$  can be reduced into  $(\Delta^2 + 1)$ -coloring of  $G^2$  within  $O(\Delta^{3/2} \log \Delta)$  rounds. This completes the description of step 4. The result of invoking steps 1 to 4 is summarized in the next corollary.

► **Corollary 3.6.**  $(\Delta^2 + 1)$ -coloring of  $G^2$  can be computed within  $O(\Delta^{3/2} \log \Delta + \log^* n)$  rounds in the CONGEST setting.

## 4 Speedup technique for algorithms on $G^k$ in the CONGEST model

In this section we devise a method which speeds up the running time of a wide class of algorithms for problems on  $G^k$  in the CONGEST model. This includes algorithms for problems such as: MIS, maximal matching, edge coloring, vertex coloring, ruling set, cluster decomposition, etc. Our goal is reducing the amount of data passed in the network, by exchanging messages only half-the-way, compared to standard algorithm for  $G^k$ . In the general case, in a problem for  $k$ -distance, a node may have up to  $\Delta^k$  neighbours. Thus, previously-known solutions require  $\Delta^{k-1}$  rounds for collecting information about  $k$ -hop neighbours in each step of an algorithm. This way, an algorithm with  $f(\Delta, n)$  rounds for  $G$  is translated into an algorithm with  $f(\Delta^k, n) \cdot \Delta^{k-1}$  rounds for  $G^k$ . (In general, the running time may be even larger, but we focus on solutions in which each vertex makes a decision based on the current information of vertices in its  $k$ -hop-neighborhood, where each vertex holds  $O(\log n)$  bits.) On the other hand, our new technique makes it possible to collect aggregated data from distance only  $k/2$ . As a result, the size of the collected data becomes  $O(\Delta^{k/2})$ . This allows us to obtain a running time of  $f(\Delta^k, n) \cdot \Delta^{k/2-1}$  instead of  $f(\Delta^k, n) \cdot \Delta^{k-1}$ . Nowadays, various problems have algorithms with running time  $f(\Delta, n) = O(\text{polylog}(\Delta) + \log^* n)$  for  $G$ . In such cases our technique provides a quadratic improvement for  $G^k$ , i.e., for distance- $k$  computations on  $G$ . Our technique is based on idempotent functions, described below.

### 4.1 Idempotent functions

An *idempotent function* is a function  $f$  from a set  $A$  to itself, such that for every  $x \in A$ , it holds that  $f(f(x)) = f(x)$ . For example, the following boolean functions from  $\{0, 1\}$  to  $\{0, 1\}$  are idempotent.  $F_0(x) = x \text{ OR } 0$ ,  $F_1(x) = x \text{ OR } 1$ ,  $G_0(x) = x \text{ AND } 0$ ,  $G_1(x) = x \text{ AND } 1$ . Another example, are functions  $\hat{H}_t, \check{H}_t : N \rightarrow N$ , where  $t \in N$ , defined as follows.  $\hat{H}_t(x) = \max(x, t)$ ,  $\check{H}_t(x) = \min(x, t)$ .

A pair of functions  $f(), g()$  is *commutative* if  $f(g(x)) = g(f(x))$ . A set of functions is a *commutative set* if any pair of functions in the set is commutative.

We define an *idempotent composition* as follows. A set  $A$  with a commutative set of functions  $f_1, f_2, \dots, f_k$  from  $A$  to itself is an *idempotent composition*, if for any  $q$  (not necessarily distinct) indices  $j_1, j_2, \dots, j_q$ , in the range  $[k]$ , and  $p \leq q$  distinct indices  $i_1 \neq i_2, \dots, \neq i_p$ , such that  $\{i_1, i_2, \dots, i_p\} = \{j_1, j_2, \dots, j_q\}$ , it holds that

$$f_{j_1}(f_{j_2}(\dots f_{j_q}(x)\dots)) = f_{i_1}(f_{i_2}(\dots f_{i_p}(x)\dots)).$$

For example, the set  $\{0, 1\}$  with the functions  $F_0, F_1$  as defined above is an idempotent composition. Indeed,  $F_0(F_0(\dots F_0(x)\dots)) = F_0(x) = x$ ,  $F_1(F_1(\dots F_1(x)\dots)) = F_1(x) = 1$ , and any composition of functions  $F_0$  and  $F_1$  equals  $F_0(F_1(x)) = F_1(F_0(x)) = 1$ .

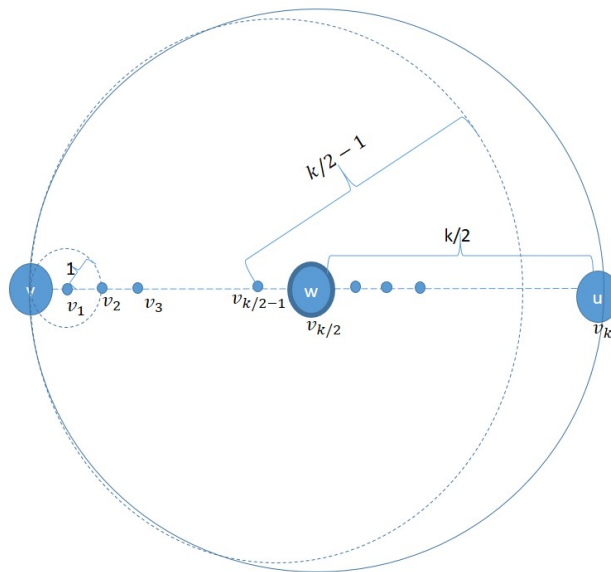
### 4.2 High level description of our technique for $G^k$

This section assumes  $k$  is even. In the case  $k$  is odd our technique cost another factor of  $O(\Delta)$  of communication rounds per algorithm round. Hence it behaves as if the distance required is  $k + 1$  in terms for CONGEST communication. Our method for distance- $k$  computations consists of two stages.

**6:16 Speedup of Distributed Algorithms for Power Graphs in the CONGEST Model**

In the first stage each vertex collects information from its  $k/2$ -hop-neighborhood. This is done by broadcasting in parallel from all vertices to distance  $k/2$ . Consequently, for any pair of vertices  $u, v$  at distance  $k$  one from another, there exists a vertex  $w$  in the middle of a path between  $u$  and  $w$  that received the information of  $u$  and  $v$ . Indeed, the distance between  $u$  and  $w$  is  $k/2$ , and between  $v$  and  $w$  it is  $k/2$ . Next,  $w$  computes internally, for each  $u$  in the  $k/2$ -hop-neighborhood of  $w$ , the available information for  $u$  regarding its  $k$ -hop-neighbours. That is, the neighbours at distance  $k$  from  $u$  who are also in the  $k/2$ -neighborhood of  $w$ , Which, sometimes contains all of the vertices in the  $k/2$ -neighborhood of  $w$  and sometimes only part of in case we would like to avoid double counting.

In the second stage, for all  $v \in V$ , all information computed for  $v$  in its  $k/2$ -hop-neighborhood should be delivered to  $v$ . Recall that the information computed by vertices in the  $k/2$ -hop-neighborhood of  $v$  is about the  $k$ -hop-neighborhood of  $v$ . But delivering the entire information in a straightforward way to  $v$  requires up to  $\Delta^k$  rounds. Indeed, vertices at distance  $k/2$  from  $v$  hold information of size up to  $\Delta^{k/2}$ , and the number of such vertices is up to  $\Delta^{k/2}$ . In order to reduce the amount of information that has to be passed, an aggregation function is used. Specifically, each vertex  $v$  collects information in a convergecast manner. That is, each vertex at distance  $k/2$  from  $v$  sends information of size  $\Delta^{k/2}$  to its neighbours. This requires  $O(\Delta^{k/2})$  rounds. Now vertices at distance  $k/2 - 1$  from  $v$  have received information from their neighbours at distance  $k/2$  from  $v$ . But instead of sending all this information to  $v$  they perform an aggregation in which the information size shrinks to  $\Delta^{k/2-1}$ . In the following phase, the information size shrinks to  $\Delta^{k/2-2}$ , etc. See Figure 1.



■ **Figure 1** A vertex  $w$  in the middle of a path between  $v$  and  $u$  collects information about its  $k/2$ -hop-neighborhood. Then a convergecast process is performed, in which balls of radius  $k/2 - i$ ,  $i = 1, 2, \dots, k/2$ , around  $v_{k/2-i}$  are formed within  $k/2$  stages. The balls contain aggregated information that after  $k/2$  stages is about the  $k$ -hop-neighborhood of  $v$ . After stage  $k/2$  this information resides in a ball of radius 0 of  $v$ , i.e., in  $v$  itself.

Further details of our technique are available in the appendix below.



## References

- 1 N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.
- 2 B. Awerbuch, A. Goldberg, V. Luby, and M. Plotkin. Network decomposition and locality in distributed computation. in *proc. of the 30th annual symposium on foundations of computer science (focs)*. pp., pages 364–369, 1989.
- 3 A. Balliu, S. Brandt, J. Hirvonen, D. Olivetti, M. Rabie, and J. Suomela. Lower bounds for maximal matchings and maximal independent sets. *proceeding of the 60th annual symposium on foundations of computer science (focs)*. pp., pages 481–497, 2019.
- 4 R. Bar-Yehuda, K. Censor-Hillel, Y. Maus, S. Pai, and S. V. Pemmaraju. Distributed approximation on power graphs. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 501–510, 2020.
- 5 L. Barenboim. Deterministic  $(\delta + 1)$ -coloring in sublinear (in  $\delta$ ) time in static, dynamic and faulty networks. *Journal of the ACM*, 63(5), 2016. doi:10.1145/2979675.
- 6 L. Barenboim and M. Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. In *Proc. of the 27th ACM Symp. on Principles of Distributed Computing*, pages 25–34, 2008.
- 7 L. Barenboim and M. Elkin. Distributed  $(\Delta + 1)$ - coloring in linear (in  $\Delta$ ) time. In *Proc. of the 41st ACM Symp. on Theory of Computing*, pages 111–120, 2009.
- 8 L. Barenboim and M. Elkin. *Distributed graph coloring: Fundamentals and recent developments*. Springer Nature, 2022.
- 9 L. Barenboim, M. Elkin, and U. Goldenberg. Locally-iterative distributed  $(\delta + 1)$ -coloring below szegedy-vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 437–446, 2018.
- 10 L. Barenboim and U. Goldenberg. Speedup of distributed algorithms for power graphs in the congest model. arXiv preprint, 2023. arXiv:2305.04358.
- 11 Y. Chang, W. Li, and S. Pettie. Distributed  $(\delta + 1)$ -coloring via ultrafast graph shattering. *SIAM Journal on Computing*, 49(3):497–539, 2020. doi:10.1137/19M1249527.
- 12 R. Cole and U. 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 Y. Emek, C. Pfister, J. Seidel, and R. Wattenhofer. Anonymous networks: randomization=2-hop coloring. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, 2014.
- 14 S. Faour, M. Ghaffari, C. Grunau, F. Kuhn, and V. Rozhoň. Local distributed rounding: Generalized to mis, matching, set cover, and beyond. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2023.
- 15 M. Flin, M. Halldórsson, and A. Nolin. Fast coloring despite congested relays. In *37th International Symposium on Distributed Computing (DISC)*, 2023.
- 16 P. Fraigniaud, M. M. Halldorsson, and A. Nolin. Distributed testing of distance-k colorings. *Proc.*, 27th Coll. on Structural Information and Communication Complexity (SIROCCO), 2020.
- 17 P. Fraigniaud, M. Heinrich, and A. Kosowski. Local conflict coloring. In *Proceeding of the 57th annual symposium on foundations of computer science (FOCS)*, pages 625–634, 2016.
- 18 M. Fuchs and F. Kuhn. List defective colorings: Distributed algorithms and applications. In *37th International Symposium on Distributed Computing (DISC)*, 2023.
- 19 M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the twenty-seventh annual ACM-SIAM Symposium on Discrete Algorithms. (SODA)*, pages 270–277, 2016.

- 20 M. Ghaffari and C. Grunau. Faster deterministic distributed mis and approximate matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*. 2023, 2023.
- 21 M. Ghaffari, C. Grunau, and V. Rozhoň. Improved deterministic network decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, pages 2904–2923, 2021.
- 22 M. Ghaffari, D. G. Harris, and F. Kuhn. On derandomizing local distributed algorithms. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2018.
- 23 M. Ghaffari and F. Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2022, pages 1009–1020, 2022.
- 24 A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988. doi:10.1137/0401044.
- 25 M. M. Halldorson, F. Kuhn, and Y. Maus. Distance-2 coloring in the congest model. in *proc. of the 39th acm symposium on principles of distributed computing (podc)*. pp., pages 233–242, 2020.
- 26 M. M. Halldorsson, F. Kuhn, Y. Maus, and A. Nolin. Coloring fast without learning your neighbours’ colors. *Proc. of the 34th International Symposium on Distributed Computing (DISC)*, 39(1-39):17, 2020.
- 27 D. Harris, J. Schneider, and H. Su. Distributed  $(\delta + 1)$ -coloring in sublogarithmic rounds. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing (STOC)*, pages 465–478, 2016.
- 28 K. Kothapalli, C. Scheideler, M. Onus, and C. Schindelhauer. Distributed coloring in  $\tilde{O}(\sqrt{\log n})$  bit rounds. in *proc. of the 20th International Parallel and Distributed Processing Symposium*, 2006.
- 29 F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. in *proc. 25th acm symp. Principles of Distributed Computing*, pp., pages 7–15, 2006.
- 30 N. Linial. Locality in distributed graph algorithms *siam j. comput.*, 21 (1992). pp., 193–201:275–290, 1992.
- 31 M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. on Computing*, 15:1036–1053, 1986. doi:10.1137/0215074.
- 32 Y. Maus. Distributed graph coloring made easy. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 362–372, 2021.
- 33 Y. Maus, S. Peltonen, and J. Uitto. Distributed symmetry breaking on power graphs via sparsification. In *Proceeding of the 42nd Symposium on Principles of Distributed Computing (PODC)*, pages 157–167, 2023.
- 34 A. Panconesi and A. Srinivasav. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996. doi:10.1006/JAGM.1996.0017.
- 35 V. Rozhoň and M. Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 350–363, 2020.
- 36 M. Szegedy and S. Vishwanathan. Locality based graph coloring. in *proc. In 25th ACM Symposium on Theory of Computing*, pages 201–207, 1993.

## A Appendix

A basic building block of ours for the first stage is a procedure for broadcasting to distance  $k/2$ , initiated by all vertices in parallel. In this procedure, each message contains three elements: Node id of the originator of broadcast, the distance from the originator, and additional information. The part that contains the id and the distance is called *header*. In addition, each vertex  $v$  maintains a local data structure  $D_v$  that initially contains only the element of  $v$ . In the beginning of the broadcast procedure, each vertex sends a message with

the contents of  $D_v$  (in the additional information field of the message) to all its neighbours. During the execution of the broadcast procedure, if a node receives several messages with the same id of the originator, the node considers only one of these messages (which was the first to arrive; ties are broken arbitrarily) and ignores the others. In each round of the broadcast, each vertex updates its data structure  $D_v$  with additional information it learnt, and sends it to its neighbours. Specifically the update of  $D_v$  considers each received message and each element in the message with ID that the vertex has not received yet. The vertex  $v$  adds each such element to  $D_v$ . Then the updated data structure  $D_v$  is sent to  $v$ 's neighbours. The broadcast procedure continues this way until the distance reaches  $k/2$ . Each message with distance greater than  $k/2$  from the origin is discarded. Thus the broadcast process stops after  $k/2$  rounds from initiation.

Once the broadcast stops, for any vertex  $w$  at distance at most  $k/2$  from  $v$ , the vertex  $w$  computes locally an appropriate value for  $v$ , using aggregation functions. These functions are applied iteratively for each vertex in the  $k/2$ -hop-neighborhood of  $w$ . Next, a convergecast process is performed for  $k/2$  phases. Its goal is that each  $v \in V$  learns the outcome of a series of function invocations that are applied to all vertices in the  $k$ -hop-neighborhood of  $v$ . (This is done with assistance of vertices at distance up to  $k/2$  from  $v$ .) To this end, aggregation functions  $f : A \rightarrow A, A \subseteq N_0$  are used. In certain cases  $A$  is of size 2, as in the example of deciding whether the value of  $v$  appears in its  $k$ -hop-neighborhood. But in other cases  $|A|$  may be larger. It has to satisfy  $\log(|A|) = o(\Delta^{k/2})$ , to provide a speed-up in our method. The number of aggregation functions is  $|A|$ , one per element of  $A$ . (This is because each element may affect uniquely the aggregated data, and an appropriate function for each element is needed.) In each phase  $i = 1, 2, \dots, k/2$  of the convergecast process each vertex  $v \in V$  receives from its neighbours the results of the aggregation functions applied to each vertex  $z$  in the  $(k/2 + 2 - i)$ -hop-neighborhood of  $v$ . (Note that these function applications hold results about distance  $(k/2 + i)$  from  $v$  at that stage.) For each such vertex  $z$ , there are up to  $\Delta$  results of functions application, one per neighbor of  $v$ . Then for each  $z'$  in the  $(k/2 + 1 - i)$ -hop-neighborhood of  $v$ , the vertex  $v$  applies the aggregation function iteratively, up to  $\Delta$  times, according to the number of results for  $z'$  that  $v$  received from its neighbours. This shrinks the amount of information by a factor of  $\Delta$  for the next phase, where the radius of function application becomes smaller by 1, and the radius of gained information becomes larger by 1. After  $k/2$  phases, each vertex  $v \in V$  holds a single result of the applications of aggregation functions on all vertices in the  $k$ -hop-neighborhood of  $v$ .

A key property of the above process, which constitutes its correctness, is provided in the next lemma.

► **Lemma A.1.** *After each phase  $i = 1, 2, \dots, k/2$ , for each pair of vertices  $v, u$  in  $G$  at distance  $k$  one from another, there exists a vertex  $z$  at distance  $k/2 - i$  from  $v$  whose result of applications of the aggregation functions until this stage includes an invocation for  $u$ .*

Proof's details appear in full version of the paper [10].

The above lemma shows that after  $i$  phases of the convergecast stage the  $(k/2 - i)$ -hop-neighborhood of each vertex  $v$  contains all needed information for  $v$  about its  $k$ -hop neighborhood. Indeed for any vertex  $u$  of distance up to  $k$  from  $v$  there is a vertex at distance  $(k/2 - i)$  from  $v$  who holds a result of a series of applications of the aggregation functions, one of which is an invocation for  $v$  on  $u$ . Thus after  $k/2$  phases each vertex  $v \in V$  holds the result of invocations of the aggregation functions for all vertices in its  $k$ -hop-neighborhood. The number of rounds of each phase  $i = 1, 2, \dots, k/2$  is  $O(\log |A| \cdot \Delta^{k/2+1-i})$ .

We illustrate this scheme by devising an algorithm for computing  $k$ -distance  $O(\Delta^k)$ -coloring. The algorithm generalizes 2-distance algorithms as follows and based on the additive group coloring (AG) algorithm. The aggregation functions are the OR function,

where  $A = \{0, 1\}$ . Specifically:  $F_0(x) = x \text{ OR } 0$ ;  $F_1(x) = x \text{ OR } 1$ . Each vertex  $w \in V$  collects the current colors of its  $k/2$ -hop-neighborhood. For each vertex  $v$  in the  $k/2$ -hop-neighborhood of  $w$ , the vertex  $w$  computes whether the  $b$  (We indicate a vertex color by the notation  $\langle a, b \rangle$  when we compute additive group coloring) element of the color of  $v$  equals to at least one of the  $b$  elements of the other vertices in the  $k/2$ -hop-neighborhood of  $w$ . To this end, for each vertex  $u$  in its  $k/2$ -hop-neighborhood,  $w$  applies a function  $F_i(x)$ , where  $i = 1$  if the  $b$  values of  $v$  equals to that of  $u$ , and  $i = 0$  otherwise. Each invocation of  $F_i(x)$  by  $w$ , except the first invocation of a phase, is applied on the result of another invocation by  $w$  in that phase. If this is the first invocation, then it is applied on  $x = 0$ . Once the vertices apply the function for their  $k/2$ -hop-neighbours, the resulting  $\Delta^{k/2}$  values are sent to immediate neighbours. Consequently, each vertex  $v \in V$  receives information of the  $k/2$ -hop-neighborhoods of its own neighbours. Note that vertices at distance up to  $k/2 - 1$  from  $v$  received information about vertices at distance up to  $k$  from  $v$ . Now the same is repeated, but for distance  $k/2 - 1$ , rather than  $k/2$ . This continues for  $k/2$  phases. Finally, as a result of the function invocations and exchange of messages, each vertex  $v \in V$  holds a single result, indicating whether there is another vertex in its  $k$ -hop-neighborhood with the same  $b$  value. This allows to compute the next color of  $v$ , as in the AG algorithm. Thus, within  $O(\Delta(G^k)) = O(\Delta^k)$  iteration of computing the next colors as described above, a proper distance- $k$  coloring with  $O(\Delta^k)$  colors is obtained, from an initial  $O(\Delta^{2k})$ -coloring to distance  $k$ . The latter coloring can be obtained by generalizing 2-distance-Linial's algorithm (Algorithm 1) to distance  $k$ . (We elaborate on this in Section A.1.) Each iteration requires  $O(\Delta^{k/2})$  rounds, and the overall running time is  $O(\Delta^{k+k/2} + k \cdot \Delta^{k/2} \cdot \log \Delta \cdot \log^* n)$ . We summarize this discussion with the next theorem.

► **Theorem A.2.** *Distance- $k$  coloring with  $O(\Delta^k)$  colors can be computed within  $O(\Delta^{k+k/2} + k \cdot \Delta^{k/2} \cdot \log \Delta \cdot \log^* n)$  rounds in the CONGEST model.*

In Section A.1 we show how the running time for distance- $k$  coloring can be improved.

We conclude the current section by improving the running time of simulating a round of an algorithm for  $G$  in  $G^k$ . The improvement is from  $O(\Delta^{k/2})$  to  $O(\Delta^{k/2-1})$ . Consider the last phase of broadcast, when each vertex needs to receive information about  $(k/2 - 1)$ -neighborhoods from each of its immediate neighbours. A  $(k/2 - 1)$ -neighborhood contains up to  $\Delta^{k/2-1}$  vertices, each of which holds  $O(\log n)$  bits of data for transmission. Thus,  $O(\Delta^{k/2-1})$  rounds are needed in the last phase of broadcast to deliver the required information to each vertex, which then employs it to compute locally the information regarding its  $k/2$ -hop-neighborhood. Then, in the first phase of convergecast, instead of sending the computation of the entire  $k/2$ -hop-neighborhood, each neighbor receives information only about its  $(k/2 - 1)$ -hop-neighborhood. Indeed, in the second phase of the convergecast  $(k/2 - 1)$ -hop-neighborhoods are considered, so this information from the first phase is sufficient. This requires  $O(\lceil \frac{\log |A|}{\log n} \rceil \cdot \Delta^{k/2-1})$  rounds. The other phases require a smaller number of rounds. Thus the overall running time becomes  $O(\lceil \frac{\log |A|}{\log n} \rceil \cdot \Delta^{k/2-1})$ . We summarize this in the next Corollary.

► **Corollary A.3.** *An  $R$  round algorithm for  $G$  in the CONGEST model that employs a set of commutative idempotent functions  $f_1, f_2, \dots, f_t : A \rightarrow A$  can be transformed into an algorithm for  $G^k$  in the CONGEST model, with running time  $O(R \cdot \lceil \frac{\log |A|}{\log n} \rceil \cdot \Delta^{k/2-1})$ .*

We also obtain an improvement in the memory complexity, as shown in the next corollary.

► **Corollary A.4.** *Denote by  $M$  the local memory of a vertex needed to complete an algorithm in a straightforward way for  $G^k$ ,  $k \geq 2$ , in the CONGEST model. The local memory required*

for an algorithm which uses our aggregation functions, such that  $\log |A| = O(\log n)$ , is bounded by  $O(\frac{M}{\Delta^{k/2}}) \sim O(\sqrt{M})$ .

Memory considerations as in Corollary A.4 are usually not analyzed in the CONGEST model. However, in real life applications that consider RAM limitations, our technique provides an improvement, which can make algorithm implementation much more efficient.

### A.1 Computations with non-idempotent functions

A function that is broadly used by us, and is not idempotent is the counting function. Specifically, given a value that a vertex holds, it needs to compute how many times this value appears in the  $k$ -hop-neighborhood of the vertex. It is applied on each vertex in the  $k$ -hop-neighborhood, and increments the result by one, each time it encounters the given value. (It does not change the result when it is applied on a different value.) Hence, the number of function applications may affect the result, and thus it is non-idempotent. Nevertheless, we are still interested in using such a function, since it appears in Linial's algorithm for computing number of polynomial intersections, and in arbdefective-coloring for computing the number of vertices of the  $k$ -hop-neighborhood with the same color.

There are two options to address this goal. The first option is analysing the solution with the possibility of over-counting. In this case the result is a correct upper bound on the desired solution, and the goal is analyzing how far it is from a solution that counts exactly. The second option is to obtain exact counting, using a more sophisticated construction, which we describe in the sequel.

In the first option we perform an adaption from  $G$  to  $G^k$  as in the case of idempotent functions, described in Section 4.2. Now we analyze the maximum number of function invocations in an iteration. Note that each invocation corresponds to a unique path of length at most  $k$  that starts from a vertex  $v$ , goes through a vertex  $w$  in the middle of the path, and ends in a vertex in the  $k/2$ -hop-neighborhood of  $w$ . The number of such paths is bounded by  $\Delta^k + \Delta^{k-1} + \Delta^{k-2} + \dots + \Delta = O(\Delta^k)$ . Denote this bound by  $d_p$ . Now we can use  $d_p$  instead of  $\Delta^k$  in the computations for  $k$ -distance-Linial algorithm or arbdefective colorings to distance  $k$ . (Note that  $\Delta$  is known to all vertices, so they can compute  $d_p = \Delta^k + \Delta^{k-1} + \Delta^{k-2} + \dots + \Delta$ .) These computations will provide correct results, according to the same pigeonhole principle, as in the proof of the distance-2 variants, but the results now depend on  $d_p$ , rather than  $\Delta^k$ .

We illustrate this with an adaptation of Linial's algorithm to provide distance- $k$  coloring with  $O(\Delta^{2k})$  colors, for a constant  $k$ . Let  $d_p = O(\Delta^k)$ , as described above. We compute a coloring with  $O(d_p^2) = O(\Delta^{2k})$  colors. This is done by  $O(\log^* n)$  stages, each of which perform a binary search for  $O(\log \Delta^k) = O(\log \Delta)$  phases. The binary search is done using the aggregation counting function, to count the number of conflicts in distance  $k$  of a polynomial  $P$ , for each half of its range. Even though this may cause over-counting, the aggregation functions are applied at most  $d_p$  times per polynomial per phase. Thus using a field of size  $q$ ,  $d_p < q < 2d_p$ , guarantees that there is going to be  $t \in \{0, 1, \dots, q-1\}$ , such that  $P(t)$  does not intersect with any polynomial in its  $k$ -hop-neighborhood. Such value  $t$  is going to be found by the binary search. This results in an  $O(d_t^2)$ -coloring in  $O(\Delta^{k/2-1} \cdot \log d_t \cdot \log^* n)$  time, i.e.,  $O(\Delta^{2k})$ -coloring in  $O(\Delta^{k/2-1} \cdot \log \Delta \cdot \log^* n)$  time. We summarize this in the next corollary.

► **Corollary A.5.** *For a constant  $k \geq 1$ , we compute  $O(\Delta^{2k})$ -coloring of  $G^k$  in  $O(\Delta^{k/2-1} \cdot \log \Delta \cdot \log^* n)$  rounds in the CONGEST model.*

While the above option is a simple solution, it provides an upper bound, which may be several times larger than the exact value. Also, for other functions, applying them more

than the required number of times may not work. Thus we propose a second option that allows executing the function exactly the desired number of times. This requires a one-time preprocessing stage with  $O(\Delta^{k-1})$  rounds. Still, for algorithms with running time  $R$  in  $G$  of at least  $\Delta^{1/2}$ , this does not affect significantly the overall running time of the transformation for  $G^k$ . (Recall that the best currently-known deterministic algorithm for  $O(\Delta)$ -coloring and MIS have running times  $\tilde{O}(\Delta^{1/2} + \log^* n)$  and  $O(\Delta + \log^* n)$ , respectively.)

In the preprocessing stage, each vertex  $v \in V$  computes a BFS tree of height  $k/2$  rooted at  $v$ . (Thus, the radius of the tree is also bounded by  $k/2$ .) This is the BFS tree of the  $k/2$ -hop-neighborhood of  $v$ . Then each vertex receives the BFS trees of its  $k/2$ -hop-neighborhood. Hence, information about  $k$ -hop-neighborhoods is obtained. The construction of BFS trees proceeds in  $k/2$  phases. In each phase  $i = 1, 2, \dots, k/2$ , BFS trees of height  $i$  are constructed from BFS trees of height  $i - 1$ , as follows. A vertex receives the BFS trees of height  $i - 1$  from its neighbours. It constructs locally a graph  $G_v$ , consisting of all vertices of all these trees, and all edges of the trees. (Note that such a composition of trees may cause cycles.) Now a local BFS algorithm is executed on  $G_v$ , starting from  $v$ . Note that in the graph  $G_v$  all edges containing  $v$  connect it to roots of trees of height  $i - 1$ . Consequently, the BFS of  $G_v$  results in a tree of height  $i$ . It is broadcasted to the neighbours of  $v$  in phase  $i$ . Since a BFS tree of height  $i$  may have up to  $O(\Delta^i)$  vertices and edges, the running time of phase  $i$  is  $O(\Delta^i)$ . The overall running time for computing BFS trees of height  $k/2$  is  $O(\Delta^{k/2-1})$ . This is because in the beginning of phase  $k/2$ , BFS trees of height  $k/2 - 1$  are received, which requires  $O(\Delta^{k/2-1})$  rounds. Then BFS trees of height  $k$  are constructed locally, which completes the computation of the BFS tree.

Next, for each vertex  $v \in V$  in parallel, the BFS tree that  $v$  computed is sent to all vertices in the  $k/2$ -hop-neighborhood of  $v$ . This is done by parallel broadcasting within  $k/2$  phases. Phase 1 requires  $O(\Delta^{k/2})$  rounds, phase 2 requires  $O(\Delta^{k/2+1})$  rounds, etc. Phase  $k/2$  requires  $O(\Delta^{k-1})$  rounds, which is also the overall time of the preprocessing stage.

Next, we explain how these BFS trees are used to make sure that each function invocation in the convergecast stage of our algorithm is performed exactly once, for each vertex in the  $k$ -hop-neighborhood of  $v$ . Let  $w \in V$  be a vertex that performs the convergecast stage, and  $v, u$  be vertices in the  $k/2$ -hop-neighborhood of  $w$ . The vertex  $w$  has to decide whether to invoke a function of  $v$  on  $u$ . Since  $w$  holds the BFS tree of height  $k/2$  of  $v$ , the vertex  $w$  knows who are all the other vertices  $w'$  that also considering at that stage whether to execute a function for  $v$ . Moreover,  $w$  knows the set  $W' = \{w' \mid w' \text{ is a vertex at distance } k/2 \text{ from } v, \text{ and at distance at most } k/2 \text{ from } u\}$ . This is because  $w$  holds all BFS trees of its  $k/2$ -hop-neighborhood. Now,  $w$  can find the vertex with the smallest ID in  $W'$ . If this is  $w$  itself it applies the function, and does not apply it otherwise. This way, out of all vertices  $w'$  that could have applied it, exactly one vertex does so.

It remains to address vertices  $u$  at distance less than  $k/2$  from  $v$ , for which there are no vertex  $w$  at distance  $k/2$  from  $v$ , such that  $u$  is in the  $k/2$ -hop-neighborhood of  $w$ . This is addressed directly by  $v$ , using the BFS tree of  $v$ . Since the BFS tree of  $w$  is also known to  $v$ , the vertex  $v$  can decide to apply the function on such  $u$  that are not in the BFS tree of  $w$ .

Now, our scheme that described in Section 4.2 for computations on  $G^k$  with idempotent functions can be extended to non-idempotent functions. The difference in the generalized scheme is that it starts with the preprocessing stage of computing BFS-trees of height  $k/2$ . In addition, during the main stages of the scheme, whenever a function has to be applied, a check is performed to ensure that it is applied exactly once for a pair  $v, u$ , as explained above. This gives rise to the following result.

► **Theorem A.6.** *Suppose we are given a set  $A$  of all possible inputs and outputs of aggregation functions, and an  $R$ -round algorithm for  $G$  in the CONGEST setting, where each round*

consists of a series of invocations of commutative aggregation functions  $f_1, f_2, \dots, f_t : A \rightarrow A$ . Then we can obtain an algorithm for the same problem on  $G^k$  whose running time in the CONGEST model is  $O(R \cdot \lceil \frac{\log |A|}{\log n} \rceil \cdot \Delta^{k/2-1} + \Delta^{k-1})$ .

## A.2 Applications of the speedup technique for $G^k$

In this section we provide several application of Theorem A.6. Consider the  $O(\Delta)$ -coloring algorithm for  $G$  on which the results of Section 3 are built. It consists of three parts:

- (1) Computing defective coloring. This part employs an aggregation function to compute number of intersections in Linial's algorithm.
- (2) Computing arbddefective coloring. This part employs an aggregation function to compute the number of neighbors with the same color.
- (3) Iterating over color classes and computing a proper coloring iteratively. This part employs an aggregation function that again computes the number of polynomial intersections, but this time according to the algorithm in Section 3.2.3.

In all these functions we can set  $A = \{1, 2, \dots, n\}$ , since the results are always bounded by the number of vertices in the graph. Since the algorithm for  $G$  composed of these steps requires  $\tilde{O}(\Delta^{1/2} + \log^* n)$  rounds [9], it gives rise to an  $\tilde{O}(k \cdot \Delta^{k-1} + k \cdot \Delta^{\frac{k}{2}} \log^* n)$ -round algorithm for  $G^k$ , by Theorem A.6. (The factor of  $k$  appears in the running time because the  $O(\log \Delta)$  running time for binary searches in  $G$  translates into  $O(\log \Delta^k) = k \log \Delta$  phases in  $G^k$ .) This is summarized in the next corollary.

► **Corollary A.7.** *For  $k > 1$ , an  $O(\Delta^k)$ -coloring of  $G^k$  can be computed in  $\tilde{O}(k \cdot \Delta^{k-1} + k \cdot \Delta^{\frac{k}{2}} \log^* n)$  rounds in the CONGEST model.*

Next, we obtain an MIS algorithm for  $G^k$ . It starts with computing  $O(\Delta^k)$ -coloring of  $G^k$ . Then, for each color class  $i = 1, 2, \dots$ , an iteration is performed, consisting of  $k$  rounds. Specifically, all vertices of color  $i$  perform broadcast to distance  $k$ . That is, each such vertex initializes a broadcast message with a counter that is initialized to 0. Each time such a message is received, the counter is incremented by 1. If a vertex receives several messages in parallel, it handles only one of them, and the others are discarded. If the counter of the message is smaller than  $k$  it is sent to the immediate neighbors. Otherwise, it is discarded. Therefore, an iteration completes within  $k$  rounds. Then vertices of color  $i$  that initiated the broadcast decide to join the MIS, and vertices that received the messages decide not to join. Vertices that made decisions become inactive. Then the next iteration starts on the remaining active vertices, etc. After  $O(\Delta^k)$  such iterations, the algorithm terminates.

► **Theorem A.8.** *MIS of  $G^k$  can be computed within  $\tilde{O}(k \cdot \Delta^k + k \cdot \Delta^{k-1} \cdot \log^* n)$  rounds in CONGEST model.*

Proofs and details of section A.2 appear in the full version of this paper [10].