The Family Holiday Gathering Problem or Fair and Periodic Scheduling of Independent Sets

Amihood Amir
Bar Ilan University, Israel
Johns Hopkins University, USA
amir@cs.biu.ac.il

Oren Kapah
Bar Ilan University, Israel
orenkapah.ac@gmail.com

Tsvi Kopelowitz
University of Michigan, USA
kopelot@gmail.com

Moni Naor[*]
Weizmann Institute of Science, Israel
moni.naor@weizmann.ac.il

Ely Porat
Bar Ilan University, Israel
porat@cs.biu.ac.il

ABSTRACT

We introduce the *Holiday Gathering Problem* which models the difficulty in scheduling non-interfering transmissions in (wireless) networks. Our goal is to schedule transmission rounds so that the antennas that transmit in a given round will not interfere with each other, i.e. all of the other antennas that can interfere will not transmit in that round, while minimizing the number of consecutive rounds in which antennas do not transmit.

Following a long tradition in Computer Science, we introduce the problem by an intuitive story. Assume we live in a perfect world where families enjoy being together. Consequently, parents, whose children are in a monogamous relation, would like to have *all* their children at home for the holiday meal (i.e. there is a special pleasure gained by hosting all the children simultaneously and they wish to have this event occur as frequently as possible). However, the conflict is that the in-laws would also be happiest if all their children come to them. Our goal can be described as scheduling an infinite sequence of "guest lists" in a distributed setting so that each child knows where it will spend the holiday. The holiday gathering problem is closely related to several classical problems in computer science, such as the *dining philosophers problem* on a general graph and periodic scheduling.

The process of the scheduling should be done with no further communication after initialization, by using a small amount of local data. The result should minimize the number of consecutive holidays where the family is not together. In a good sequence this number depends on local properties of the parents (e.g., their number of children). Furthermore, solutions that are periodic, i.e. a gathering occurs every fixed number of rounds, are useful for maintaining a small amount of information at each node and reducing the amount of ongoing communication and computation.

Our algorithmic techniques show interesting connections between periodic scheduling, coloring, and universal prefix free encodings. We develop a coloring-based construction where the period of each node colored with the c is at most $2^{1+\log^* c} \cdot \prod_{i=0}^{\log^* c} \log^{(i)} c$ (where $\log^{(i)}$ means iterating the log function i times). This is achieved via a connection with *prefix-free encodings.* We prove that this is the best possible for coloring-based solutions. We also show a construction with period at most $2d$ for a node of degree d.

[*]Incumbent of the Judith Kleeman Professorial Chair.

SPAA '16, July 11 - 13, 2016, Pacific Grove, CA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4210-0/16/07...$15.00

DOI: http://dx.doi.org/10.1145/2935764.2935788

1. INTRODUCTION

We examine the problem of scheduling an infinite sequence of independent sets in a given graph G in a *distributed* manner, with the following three objectives:

- *Periodicity.* For every node v, if v appears in the ith independent set then the next independent set containing v is the $(i+\pi_v)$th independent set, where π_v is a fixed positive integer. We say that π_v is the *period* of v. Periodicity is useful since it allows each node v to maintain a small amount of information in order to determine the subsequence of independent sets in which it participates. In particular, a periodic schedule guarantees that once the period is set the amount of communication and computation is minimized, saving both bandwidth and energy.

- *Locality.* π_v depends only on local properties of v (such as its degree), and not global properties of the graph (such as the maximum degree or the diameter). Locality provides a sense of fairness since for every node the length of its period only depends on properties of that node.

- *Minimize Starvation.* The periods should collectively be as small as possible.

This problem has direct applications in the realm of common resource scheduling. Suppose that in a world with many agents, each agent requires some shared resources in order to perform some job. For example, it would be beneficial if wireless radios could guarantee that when they broadcast on a particular frequency, none of the other radios interfere. In this application the shared resource is the air which is within transmission radius of more than one radio. We can model this as radios being vertices and two radios which share some air are modelled as an edge.

In the Computer Science tradition of defining systems problems as "social" problems, e.g. the dining philosophers problem, we view the wireless radios model as a "holiday gathering" problem. One of the anxiety-causing problems before the holidays is where to go for the holiday dinner? Parents, whose children are in a monogamous relationship, would obviously (?!) like to have *all* their children at home for the holiday meal (i.e. there is a special pleasure gained by the festive experience of hosting all the children simultaneously and intuitively the goal is to have this event occur as frequently as possible). We say that such parents wish to be *happy* during the holiday. However, the conflict is that the in-laws would also be happy if all their children come to them.

This problem is closely related to the classic *dining philosophers problem*, which we discuss later. Furthermore, scheduling problems are part of our mainstay, and these problems are indeed scheduling problems. To this end, we focus on algorithms whose goal is to schedule which parents are happy during any given holiday in a *distributed* manner with the objective of minimizing the number of consecutive holidays in which a parent is not happy[1].

Connection to coloring.
As in many problems in computer science, for some special inputs the holiday gathering problem is simple. For example, if G is bipartite with two sets of nodes A and B, then there is a very good sequence: during odd rounds the independent set will be all of A and during the even rounds the independent set will be all of B. The reason this example works out so well is because it is based on covering all of the nodes using as few independent sets as possible - a coloring problem. In the bipartite case, two colors suffice. In the setting of wireless networks the topology of the network may also lead to a low chromatic number.

In a general graph where Δ is the largest degree, it is immediate that one can color the graph in $\Delta + 1$ colors. Such a coloring can be obtained in a distributed manner by applying, for example, the recent randomized algorithm of Barenboim, Elkin, Pettie, and Schneider [5] (denoted by the BEPS algorithm for short) running in $O(\log \Delta + 2^{O(\sqrt{\log \log n})})$ rounds. The coloring leads to a scheduling of independent sets where the period of each node is exactly $\Delta + 1$. This scheduling is obtained by partitioning the sequence into phases of $\Delta + 1$ independent sets, and in each phase the ith round corresponds to all of the nodes colored i. However, such a sequence is not local as the length of any period depends on a global graph parameter (the maximum degree), and so this approach requires some refinement.

[1]One may consider the problem of maximizing happiness for a given year, but this problem is $\mathbb{NP}$-hard. One may also consider an objective function relating to the satisfaction of parents. Details for both can be seen in the Appendix.

What is the fair share of parents? Defining fairness is the subject of much debate in philosophy, game theory and theology. Much of cooperative game theory deals with fair allocation of resources. In our case the problem seems hard: given the tight relationship with coloring and maximum independent set, we cannot even determine efficiently the 'value' of the full coalition (see Appendix B.2). On the other hand, consider the following simple 'chaotic' process called "first come first grab": parents wake up at a random time and grab their available (those who have not been grabbed) children. The probability that a node p manages to grab *all* its children is $1/(deg(p) + 1)$, where $deg(p)$ is the degree of p. So the expected time until hosting all the children is $deg(p)$ and this is the landmark we will try to obtain, i.e. we would like every parent to host a holiday with all their children every $O(deg(p))$ years. It is also clear that in general we cannot hope to get a better than $deg(p) + 1$ result, if the conflict graph is a clique. Nevertheless, when the input graph has a small chromatic number, as discussed above, we can do better.

1.1 Our Results and Techniques

The main contributions of this paper are, thus:

1. A **Color-bound** solution for the holiday gathering problem which is based on any given coloring of the graph. The length of a period for a node v with color c is at most $\pi_v \leq 2^{1+\log^* c} \cdot \prod_{i=0}^{\log^* c} \log^{(i)} c$ where $\log^{(i)} c$ is the iterative log function of c taken i times. This is achieved by introducing an interesting new technique that utilizes a connection with *prefix-free encoding.* We also show that, for color-based techniques, our algorithm is close to optimal. This is done by proving a lower bound on the length of a period of $\pi_v \geq \prod_{i=0}^{\log^* c} \log^{(i)} c$ for any scheduling algorithm based on graph coloring. This lower bound makes use of the Cauchy condensation test.

2. A **Degree-bound** solution for the holiday gathering problem where π_v for a node v with degree d children is exactly $2^{\lceil \log d \rceil} \leq 2d$.

We emphasize that it is always possible to color the graph where the color of a node of degree d is at most $d + 1$ (for example using the BEPS algorithm, see [6] for a more recent version). If, in a given coloring, all of the colors are much smaller than the respective degrees (like a 2-coloring of a bipartite graph) then clearly the first algorithm is preferable. However, in terms of worst-case dependence on the degree of a node, the second algorithm is preferable.

We are mainly interested in periodic scheduling, but we also discuss some non-periodic solutions in Appendix A that serve as a sanity check and help us understand what is the best one could hope for without any constraints.

1.2 Related Work

As mentioned, the holiday gathering problem is related to several lines of investigation in computer science (not to mention other scholastic activities). Issues related to calendrical calculations have attracted the best minds since antiquity (see Dershowitz and Reingold [11]). No lesser than al-Khwarizmi (after whom the term 'algorithm' is named) wrote a treatise on the Hebrew Calendar ("Risala fi istikhraj ta'arikh al-yahud"), see Knuth [21]. Some calendars are fixed in the sense the that it is known in advance when each holiday will occur, as is the case with the Western (Gregorian) calendar and the Hebrew calendar, while others, like the Muslim Calendar or the Old Hebrew Calendar are determined on-the-fly, e.g. based on lunar observations. This is reminiscent of some of the issues that arise in our algorithms (the one in Appendix A vs. those of Sections 3 and 4).

The dining philosophers problem is the famous resource allocation problem introduced by Dijkstra [12], see Lynch [25]. In this problem, there is a given conflict graph where each node represents a processor and each edge represents a resource (a "fork" in the story where the processors are philosophers who would like to eat) which is shared by the two endpoint processors. At any time, a fork can be "owned" by at most one of the processors that share it. Each processor can be in one of three states: resting, hungry, or eating. A resting processor can become hungry at any time. In order to eat, a processor must obtain all the forks on its adjacent edges. A processor eats for at most a bounded time, after which it returns to the resting state. The problem, and its many variations, have played a major role in concurrent programming and distributed computation. For this problem there are solutions that minimize the wait chain, based on coloring of the edge or the nodes (see [24, 33, 10, 29, 26]). The main difference between the focus of this work and most work on the dining philosophers problem is that we assume that the philosophers want to eat *all* the time (i.e. they become hungry right after they finish eating)[2] and that the meal takes a fixed amount of time and the main issue is how can we provide an efficient and high throughput solution while guaranteeing some reasonably fair allocation.

Problems related to ours have appeared frequently in the scheduling literature, starting perhaps from the Chairman Assignment Problem of Tijdeman [34]. The assumption is that there is one resource and all users are interested in using it. Each user or task has a weight and the goal is to schedule the users so that they obtain the resource proportionally to their weight. Sometime there are multiple identical resources, but each task can be assigned to one resource concurrently [7, 23]. This is similar to our setting when the graph is a clique (or composed of components that are cliques) and all the weights are the same. In a 'perfectly periodic scheduling' the goal is to schedule the users in a *periodic way* (every user i gets the resource every τ_i rounds) where each user gets its weight (see [3]). The algorithms of Sections 3 and 4 are perfectly periodic. In the chromatic sum problem (see [2]) one tries to find a coloring that minimizes the the total sum of the colors (where the assumption is that the colors are in $\mathbb{N}$). The motivation is, again, from scheduling with the goal of minimizing *average* waiting time, rather than the maximum waiting time as a function of the degree, as is our goal.

Fairness in online scheduling has received some attention as well, for instance the carpool problem [14, 1, 28], that can also be viewed as a generalization of the Chairman Assignment Problem of Tijdeman [34].

The $\mathcal{LOCAL}$ model of computation in distributed computing was first considered by Linial [22] and much developed since then, see Peleg [31]. The problems of interest are especially those of coloring and maximal independent set. For both of these problem good randomized algorithms are known, see the monograph by Barenboim and Elkin for a survey of recent results [4]. Coming up with deterministic polylog in n algorithms for these problems is a major open problem in the area.

[2]So one may call the problem "The Fressing Philosophers Problem", as suggested by Cynthia Dwork.

Minimizing interference in wireless networks.

There has been a lot of research focusing on scheduling devices in wireless networks so as to minimize interference. This work has focused on determining which nodes may transmit on which time-frequency slot, while optimizing fairness and throughput [17]. This line of work leverages the specific topology of wireless networks. For example, some research has looked at conflict graphs that represent geometric intersection graphs such as the unit disk graph [15]. However, many believe that such graphs do not do a good job of representing the reality of wireless graphs (see [27]). Thus, a lot of recent work has focused on the SINR model where signals decays as it travels. In this model the success of a transition depends on its strength at the receiver relative to the strength of other interfering transmissions [18, 16].

In light of the above, we emphasize that our color-based schemes allow to implicitly account for special topologies that have a conflict graph with a low chromatic number, as one may expect from a wireless network. Moreover, the previous line of work has focused on scheduling transmissions in several channels in order to minimize interference, while our focus is on one channel that does not allow any interference.

2. PRELIMINARIES

Our universe is a graph $G = (V, E)$ where $|V| = n$. Let $deg(v)$ be the degree of a node $v \in V$. Most of this work assumes that the G is fixed, but we also consider the dynamic case and briefly discuss it in Section 5.

DEFINITION 2.1. *For a given independent set h we say that node $v \in V$ is* happy *in h if v participates in h.*

DEFINITION 2.2. *Let $H = h_i$, $i = 1, \ldots, \infty$ be a sequence of independent sets and denote the subsequence $h_i, ..., h_j$ by $h[i:j]$. If $v \in V$ is not happy at any $h \in h[i:j]$ then we call the interval $h[i:j]$ an* unhappiness interval *for v and call the longest such interval a* maximum unhappiness interval *of v and denote its length by $mui_H(v)$ (or $mui(v)$ when H is clear from the context). If all (locally) maximal unhappiness intervals of v are of the same length π_v (except for possibly the first interval which may be shorter) then we say that H is periodic for v and π_v is the length of the period of v. We say that H is* periodic-for-all *if it is periodic for all of the nodes in V.*

The combinatorial definition of the Holiday Gathering Problem is: given a graph G, find an infinite sequence H of independent-sets of G. The objective function is to minimize, for every node v, the maximal gap between two appearances of v. Intuitively, our goal is to devise a sequence H of independent sets that is periodic-for-all, while minimizing π_v for each $v \in V$.

Algorithm Scheme

```
1. Color $G$.
2. At holiday $i$, if $decode(i) = col(p)$ then make $p$ happy.
end Algorithm
```

Figure 1: Generic coloring based scheme for the Holiday Gathering problem.

3. A PERIODIC COLOR-BOUND ALGORITHM

We are now interested in providing a mechanism whereby the decision of when a node is happy is dependent on *local* properties of that node. We present a general scheme for such a mechanism that depends on the color nodes receive during an initial coloring algorithm. To obtain such a coloring we begin by distributively coloring the graph (for example using the BEPS algorithm). However, we do not make any assumptions on the coloring algorithm, and so this algorithm works for any graph coloring, including the (possibly difficult to obtain) optimal one.

We consider a mapping of the independent set numbers to colors. The mapping needs to satisfy two conditions: (1) Every independent set number is mapped to at most one color, and (2) The resulting sequence guarantees that nodes do not have very large periods.

The basic idea is captured by the following algorithm (see Figure 1). We first color the G with some coloring algorithm, and then we use a decoding scheme so that at round i if the decoding of i is the color of v then v is part of the ith independent set.

Examples:

Trivial: Consider the trivial example where the nodes are colored sequentially from 1 to n. At independent set i, make $v = i \bmod n$ happy. No two adjacent nodes are encoded to the same number, but $\forall v \in V$ we have $\pi_v = n$, which means that it depends on global properties of G.

Prefix Free Color Code: Apply on G the BEPS distributed graph coloring algorithm that colors each graph node v by a color not exceeding $deg(v) + 1$. Now encode the colors using some prefix-free binary code. On independent set i, consider the binary representation of i from right to left (with an infinite sequence of 0's padded to it). Any node v is made happy if the prefix-free encoding of $col(v)$ is a prefix of i. The solution is appropriate, since no two adjacent nodes will be made happy concurrently at any given independent set i: they will be assigned different colors and the two different colors are encoded in a prefix-free code and hence the binary representation of i cannot encode both of them. For every $v \in V$, we have that π_v of this procedure is dependent on the length of the prefix-free encoding of v's color.

We will indeed use prefix-free binary codes in our algorithm. Notice that if a node is given a color c which uses x_c bits in its prefix-free code, then the schedule of when that node is happy is periodic with period 2^{x_c}. In other words, every 2^{x_c} independent sets the node will be happy

3.1 Upper Bound - Elias Code

We recursively define the function $\phi : \mathbb{N} \to \mathbb{R}$. For $i \leq 1$ we have $\phi(i) = 1$. Otherwise, $\phi(i) = i \cdot \phi(\log i)$. Explicitly, $\phi(i) = i \cdot \log i \cdot \log\log i \cdot \log\log\log i \cdots 1$, or $\phi(i) = \prod_{i=0}^{\log^* c} \log^{(i)} c$, where $\log^{(i)}$ means iterating the log function i times.

Elias omega code Algorithm

```
1. Color $G$.
2. At holiday $i$, if $LSB(B(i)) = \omega(p)^R$ then make $p$ happy.
end Algorithm
```

Figure 2: Scheduling for the Holiday Gathering problem based on the Elias omega code.

In this section we present an algorithm that will guarantee that no two nodes with different colors are made happy during the same independent set, and that $\pi_v = \phi(c)2^{\log^* c+1}$, where c is the color of v. This algorithm is based on the *Elias omega code.* The Elias omega code is a universal code for the natural numbers developed by Peter Elias [13]. It is one of the prefix-free codes that represents the integers by a number of bits relative to their size. While the omega code is not the most practical code, it is theoretically the most efficient Elias code. Our algorithm in Figure 2 is correct for all Elias codes, but we chose the omega code for its almost optimal complexity.

Details of the Elias omega code are given in Appendix C. The important properties that we need from the code are that it is a prefix free code, and that the length of the coding of i is given by:

$$\begin{aligned}\rho(i) =& 1 + \lceil \log(i) \rceil + \lceil \log(\lceil \log(i) \rceil - 1) \rceil \\ &+ \lceil \log(\lceil \log(\lceil \log(i) \rceil - 1) \rceil - 1) \rceil + \cdots .\end{aligned}$$

Our algorithm will use the Elias omega code of the colors in reverse. Denote by S^R the string S reversed. For example, $(abcdef)^R = fedcba$. Denote by $LSB(S, k)$ the suffix of S of length k, or the k least significant bits of S. Denote by $B(x)$ the binary representation of x, i.e. $B(x)$ is a string over alphabet $\{0, 1\}$ which is the representation of n in base 2 with no leading zeros.

THEOREM 3.1. *The Elias omega code algorithm guarantees happiness for node $v \in V$ in every cycle of length bounded above by $\phi(c)2^{\log^* c+1}$, where c is the color of v.*

PROOF. The time between happy independent sets for v is $2^{\rho(c)}$. Notice that $\rho(c) \leq 1+\sum_{i=1}^{\log^* c} \lceil \log^{(i)} c \rceil \leq 1+\log^* c+\sum_{i=1}^{\log^* c} \log^{(i)} c$. The $\log^* c$ term is a result of the rounding up of the log at every recursion of $\rho(n)$, since the number of bits cannot be a fraction. For some colors this term may be less than $\log^* c$, or even a constant, but it will never exceed $\log^* c$. Now, the time between happy independent sets is $2^{\rho(c)} \leq 2^{1+\log^* c} \cdot \prod_{i=0}^{\log^* c-1} \log^{(i)} c = 2^{1+\log^* c}\phi(c)$. □

3.2 Lower Bounds

The coloring-based algorithm scheme of Theorem 3.1 starts by assigning colors to the nodes in G. We would like every node to use its color in order to compute a period π such that every π years that node is happy. In this section we compute lower bounds on that period as a function of the color. This defines the best period one can hope to achieve by the coloring-based scheme. Our lower bound almost matches the algorithm of Theorem 3.1.

THEOREM 3.2. *Let G be legally colored and every node $v \in V$ has a color c_v. Let sequence $H = h_i,\ i = 1, ..., \infty$ of independent sets be such that there is no independent set h_{i_0} in which the nodes of two different colors, $c_1 \neq c_2$ are happy. If there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for every $v \in V$ we have $\pi_v = f(c_v)$, then $f(c) \in \Omega(\phi(c))$.*

PROOF. Consider a subsequence $h[i_0 : j_0]$ of length m. We require that at any independent set in the sequence, there is at most one color c which is happy. The conclusion is that there are at least $\lceil \frac{m}{f(c)} \rceil$ occurrences in the sequence where nodes of color c are happy. This is true for all colors c that get mapped to the subsequence. Let $C = \{c | c \text{ is a color mapped to } h[i_0 : j_0]\}$. Then we have $\sum_{c \in C} \lceil \frac{m}{f(c)} \rceil \leq m \Rightarrow \sum_{c \in C} \frac{1}{f(c)} \leq 1$. Taking into account all possible sets C as the size of G goes to infinity, we have $\sum_{c=1}^{\infty} \frac{1}{f(c)} \leq 1$. Clearly this will not hold if we have $f(c) = c$. It holds when $f(c) = 2^c$ and even when $f(c) = c^{1+\epsilon}$ for any constant $\epsilon > 0$. According to Cauchy's condensation test [9] the smallest function for which this inequality holds is $f(c) = \phi(c)$, i.e. $f(c) = \prod_{i=0}^{\log^* c} \log^{(i)} c$. □

4. A PERIODIC DEGREE-BOUND ALGORITHM

While many graphs have a low chromatic number that may be obtainable[3], other graphs do not. Furthermore, while some classes of graphs may have a low chromatic number, it is not clear how to obtain algorithms that achieve $o(\Delta)$ colors for some of these classes. In such cases a color-bound algorithm may not suffice considering the inherent lower-bound. We could use the bound of $c \leq d+1$ for a node v with color c and degree d, to obtain a guaranteed degree-bound of $\pi_v \leq \phi(d+1)2^{\log^*(d+1)+1}$ using Theorem 3.1, but we can do better.

To this end, we present a second local algorithm which is degree-bound, i.e. π_v is bounded by a function of $d = deg(v)$. In particular, our algorithm guarantees that this bound is at most $2d$, which is very close to the bound $d+1$ obtained by the non-periodic algorithm of Appendix A. We first describe a sequential version of our algorithm, and later show how to convert it to the distributed setting.

A Sequential Algorithm.

The periodic scheduling is obtained using a greedy algorithm, where nodes are arranged in a *non-increasing* order of their degrees and each node v in its turn chooses a non-negative integer. Thus, when v with degree d has to chose an integer, none of its smaller than degree d neighbors has chosen an integer. Let $j = \lceil \log(d+1) \rceil$. When it is v's turn to pick an integer, there must be at least one integer x in the range $[0, 2^j - 1]$ such that no neighbor of v has chosen a number x' where $x = x' \bmod 2^j$. So v picks x to be its integer.

The independent set i now is determined by checking whether $i \equiv x \mod 2^j$. If 'yes' then v is happy in independent set i, otherwise it is not. We can immediately see that v is happy once every $2^j = 2^{\lceil \log(d+1) \rceil} \leq 2d$ independent sets. We will now show that there are no conflicts.

[3]For instance, for triangle-free graphs Pettie and Su [32] very recently gave an $O(\Delta/\log \Delta)$ distributed coloring.

Algorithm – Distributed degree-bound algorithm

```
1. For $i = \lceil \log(\Delta+1) \rceil$ to 0
    (a) Let $P_i = \{p \in P$ such that
            $\lceil \log(\deg(p)+1) \rceil = i\}$
    (b) For each $p \in P_i$ restrict the palette of $p$ to integers that are not equal modulo $2^i$ with any integer already picked by a neighbor of $p$.
    (c) Run the BEPS algorithm on $P_i$ with the restricted palettes.
```

end Algorithm

Figure 3: A degree based scheme for the Holiday Gathering problem.

LEMMA 4.1. *Let v_1 and v_2 be two adjacent nodes in G sharing an edge with degrees d_1 and d_2 respectively. Let $j_1 = \lceil \log(d_1+1) \rceil$ and $j_2 = \lceil \log(d_2+1) \rceil$, and let x_1 and x_2 be the integers picked by v_1 and v_2 respectively using the above algorithm. Then v_1 and v_2 will never be in the same independent set.*

PROOF. Without loss of generality let $d_1 \leq d_2$. Assume, for the sake of contradiction, that at independent set i both v_1 and v_2 try to host. This implies that $i = x_1 + a_1 2^{j_1} = x_2 + a_2 2^{j_2}$. But this also means that $x_1 \equiv x_2 \mod 2^{j_1}$, and so when it was the turn of v_1 to pick its integer the algorithm could not pick x_1 for v_1, giving a contradiction. □

The Distributed Algorithm.

For the distributed setting we run $\lceil \log(\Delta+1) \rceil$ phases of the BEPS algorithm with the following modification. Starting with $i = \lceil \log(\Delta+1) \rceil$ and going to $i = 0$, during phase i all of the nodes with degree d such that $\lceil \log(d+1) \rceil = i$ will participate in the coloring algorithm for that phase. However, we must restrict the palette of colors for each node v to be comprised only of integers that do not collide (under modulo 2^i) with integers of neighbors of v that already participated in early phases. We emphasize that Barenboim et.al. [6] show that the analysis of the BEPS algorithm does not change with this restriction. The algorithm appears in detail in Figure 3.

Each phase takes $O(\log \Delta + 2^{O(\sqrt{\log \log n})})$ rounds, and we have $O(\log \Delta)$ phases, for a total of $O(\log^2 \Delta + \log \Delta \cdot 2^{O(\sqrt{\log \log n})})$ rounds. In a manner similar to Lemma 4.1 we can show that a conflict can never arise.

THEOREM 4.2. *The distributed degree-bound algorithm guarantees happiness for node $v \in V$ in every cycle of length bounded above by $2d$, where d is the degree of v.*

5. THE DYNAMIC SETTING AND OPEN PROBLEMS

So far we have assumed that the conflict graph is fixed and does not change throughout the lifetime of the system. However, as we know, relationships are not fixed and new connections may be created or old connections may dissolve. How do our algorithms fare under such conditions? Clearly two nodes that become adjacent and were scheduled to host the same upcoming independent set need to recolor themselves.

In the algorithm of Section 3 all we needed is a valid coloring. So if two nodes v and u that have the same color become connected, one of them, say v, needs to find a new color. But this is relatively simple since v's palette should grow by one more color, since d=$deg(v)$ was increased by 1. So given the new color, the new periodic schedule for v is derived from the prefix-free encoding of the new color. This means that after at most $\phi(d)2^{\log^* d+1}$ rounds after quiescence v will get to host. Note that in this algorithm, if there are w events of adding new neighbors, then the time a node hosts a independent set may be postponed up to $w \cdot \phi(d)2^{\log^* d+1}$ rounds. In the event of deletion of edges, presumably there is nothing to be done. However, if this happens too frequently, then the rate of hosting becomes disproportional to the current degree and we will need to recolor the node (again simple even give the smaller palette).

Our Algorithm of Section 4 does not fare so well in a dynamic graphs. It is very important in that algorithm to let the higher degree nodes color themselves before the lower degree ones (since the latter's frequency is much higher and they will occupy available slots if colored before the higher degree).

So a main open problem this work presents is whether it is possible to have a degree bound algorithm that works in a dynamic graph. Another issue is whether it is possible to get to the bound of Appendix A of frequency $d+1$ in a periodic or at least succinctly defined manner.

A lower bound conjecture for periodic scheduling.

In Section 4 we showed an algorithm that achieves a $2d$ upper bound on the period of a node with degree d. This is in contrast to the non periodic scheduling obtained in Appendix A where we can obtain a $d+1$ guarantee. We conjecture that there is a separation between what is obtainable when one requires periodicity versus the general case, and we leave as an open problem to prove that if one requires a periodic schedule then the best guarantee obtainable is $d+\omega(1)$.

6. ACKNOWLEDGMENTS

This work is partially supported by ISF grants 571/14 and 4/11, BSF grants 2014028 and 2012348, NSF grants CCF-1217338, CNS-1318294, and CCF-1514383, and the I-CORE Program of the Planning and Budgeting Committee.

We thank Amotz Bar-Noy for helpful comments.

7. REFERENCES

[1] Miklós Ajtai, James Aspnes, Moni Naor, Yuval Rabani, Leonard J. Schulman, and Orli Waarts. Fairness in scheduling. *J. Algorithms*, 29(2):306–357, 1998.

[2] Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.

[3] Amotz Bar-Noy, Aviv Nisgav, and Boaz Patt-Shamir. Nearly optimal perfectly periodic schedules. *Distributed Computing*, 15(4):207–220, 2002.

[4] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring*. `http://www.cs.bgu.ac.il/~elkinm/BarenboimElkin-monograph.pdf, 2013.`

[5] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *53rd Annual IEEE Symposium on Foundations of Computer Science, (FOCS)*, pages 321–330, 2012.

[6] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *CoRR*, abs/1202.1983, 2015.

[7] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 345–354, New York, NY, USA, 1993. ACM.

[8] P. Berman and T. Fujito. On approximation properties of the independent set problem for degree 3 graphs. In *Workshop on Algorithms and Data Structures (WADS)*, volume 955 of *LNCS*, pages 449–460. Springer, 1995.

[9] D. D. Bonar, Jr. M. J. Khoury, and M. Khoury. *Real Infinite Series*. The Mathematical Association of America, 2006.

[10] Manhoi Choy and Ambuj K. Singh. Efficient fault-tolerant algorithms for distributed resource allocation. *ACM Trans. Program. Lang. Syst.*, 17(3):535–559, 1995.

[11] Nachum Dershowitz and Edward M. Reingold. *Calendrical Calculations*. Cambridge University Press, New York, NY, USA, 3rd edition, 2007.

[12] Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138, 1971.

[13] P. Elias. Universal codeword sets and representations of the integers. *Information Theory, IEEE Transactions on*, 21(2):194–203, Mar 1975.

[14] Ronald Fagin and John H. Williams. A fair carpool scheduling algorithm. *IBM Journal of Research and Development*, 27(2):133–139, 1983.

[15] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[16] Magnús M. Halldórsson, Stephan Holzer, Pradipta Mitra, and Roger Wattenhofer. The power of non-uniform wireless power. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1595–1606, 2013.

[17] Magnús M. Halldórsson and Pradipta Mitra. Nearly optimal bounds for distributed wireless scheduling in the SINR model. In *Automata, Languages and Programming - 38th International Colloquium, ICALP, Part II*, pages 625–636, 2011.

[18] Magnús M. Halldórsson and Tigran Tonoyan. How well can graphs represent wireless interference? In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pages 635–644, 2015.

[19] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.

[20] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Computing*, 2(4):225–231, 1973.

[21] Donald E. Knuth. Algorithms in modern mathematics and computer science. In Andrei P. Ershov and Donald E. Knuth, editors, *Algorithms in Modern Mathematics and Computer Science*, volume 122 of *Lecture Notes in Computer Science*, pages 82–99. Springer, 1979.

[22] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[23] Ami Litman and Shiri Moran-Schein. On centralized smooth scheduling. *Algorithmica*, 60(2):464–480, 2011.

[24] Nancy A. Lynch. Upper bounds for static resource allocation in a distributed system. *J. Comput. Syst. Sci.*, 23(2):254–278, 1981.

[25] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers,, San Mateo, CA, USA, 1996.

[26] Alain J. Mayer, Moni Naor, and Larry J. Stockmeyer. Local computations on static and dynamic graphs. In *ISTCS*, pages 268–278, 1995.

[27] Thomas Moscibroda, Roger Wattenhofer, and Aaron Zollinger. Topology control meets SINR: the scheduling complexity of arbitrary topologies. In *Proceedings of the 7th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing, (MobiHoc).*

[28] Moni Naor. On fairness in the carpool problem. *Journal of Algorithms*, 55(1):93 – 98, 2005.

[29] Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

[30] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* MIT Press, Cambridge, MA, USA, 1994.

[31] David Peleg. *Distributed computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[32] Seth Pettie and Hsin-Hao Su. Fast distributed coloring algorithms for triangle-free graphs. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 681–693. Springer, 2013.

[33] Eugene Styer and Gary L. Peterson. Improved algorithms for distributed resource allocation. In Danny Dolev, editor, *PODC*, pages 105–116. ACM, 1988.

[34] R. Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32(3):323 – 330, 1980.

APPENDIX

Appendix

A. THE NON-PERIODIC DEGREE-BOUND ALGORITHM

Our goal here is to guarantee *happiness* to every node within a reasonable number of independent sets, but not necessarily in a periodic sequence. We present[4] an algorithm

[4]We remark here that while this algorithm is less interesting from a technical perspective, it is useful for set up, and gives us a benchmark for comparison when attempting to understand the strengths of the lightweight algorithms.

```
Algorithm – Phased Greedy Coloring
  1. Initialization: Assign every node $p$ a color.
  2. For $i = 1$ to $\infty$
     (a) For every node $p \in P$: if $col(p) = i$ then make $p$
         happy and recolor $p$:
         i. Let $p_1, ..., p_\ell$ be the nodes adjacent to $p$ in $P$.
            A. Let $s = min\{t | i < t \leq i + \ell + 1,$
                       $t \notin \{col(p_1), ..., c(p_\ell)\}\}$.
            B. $col(p) \leftarrow s$.
end Algorithm
```

Figure 4: The phased greedy coloring algorithm.

guaranteeing that a node of degree d has to wait at most $d+1$ steps till it is happy. However, the node does not know in advance all the times in which it will host the independent set, just the next time it will do so.

The algorithm we consider starts with a *distributed graph coloring algorithm.* As mentioned in Section 1, there is a simple mechanism using a $\Delta+1$ coloring to obtain a sequence H such that for every node $v \in V$ we have $mui_H(p) = \Delta+1$. Such a coloring can be obtained in a distributed manner by applying, for example, the recent randomized algorithm of Barenboim, Elkin, Pettie, and Schneider [5] (denoted by the BEPS algorithm for short) running in $O(\log \Delta + 2^{O(\sqrt{\log \log n})})$ rounds. The BEPS algorithm also has the property that the color c picked for a node with degree d will always be bound by $c \leq d + 1$ (see [6] for details). So at first the smaller degree nodes will be happy pretty quickly. However, for their next turn they will have to wait time proportional to Δ. As mentioned, we seek an efficient mechanism for constructing a sequence H where $mui_H(p)$ depends on local properties of v.

To solve this problem we will use a phased algorithm where colors are reassigned every phase (independent set), providing a sequence of independent sets. The initial coloring is the one obtained by the BEPS algorithm.

At independent set i, greedily, recolor the nodes whose current color is i: color each such node v with the smallest number $j > i$ such that none of v's neighbors has color j. At independent set i the nodes colored i are happy.

The algorithm appears in detail in Figure 4. We denote by $col(p) \in \mathbb{N}$ the current color of node v.

For every phase i we perform $O(1)$ rounds of communication, This is true since every node needs to only communicate with its neighbors and choose the smallest number that is both greater than i and different from the color of all its neighbors.

THEOREM A.1. *There is a independent set scheduling algorithm, whose initialization takes $O(\log \Delta + 2^{O(\sqrt{\log \log n})})$ rounds, and executing each independent set takes another $O(1)$ rounds, which guarantees that for all $v \in V$ we have $muis_S(v) \leq deg(v) + 1$. In words, for every node v, within every sequence of $deg(v) + 1$ independent sets v is happy at least once.*

PROOF. Apply the Phased Greedy Coloring algorithm. The number of rounds is $O(\log \Delta + 2^{O(\sqrt{\log \log n})})$ for the initialization and another $O(1)$ for every phase. For every node $v \in V$, if it is made happy at phase i then in phase $i + 1$ it is re-colored. The number it chooses is the smallest number that exceeds i and is not equal to the number of any of its neighbors. However, since it has $deg(v)$ neighbors then the color it gets can not exceed $i + deg(v) + 1$. $\square$

Recall that if we think of the "first come first grab" where nodes wake up at random at grab all there available neighbors the probability of happiness of a node is $1/(d+1)$, so the expected time till happiness is $d+1$. This algorithm can be seen as providing a guarantee for the waiting time.

Notice that this algorithm generally does *not* give a periodic schedule. Also, it requires communication to take place at every phase, which implies the need to invest a lot of energy if we are considering an application such as cellular communication. An alternative solution would be to have each node know and remember *all* of the conflict graph locally (and then simulate the algorithm locally to obtain determine when it should host). But such an approach requires a lot of local memory, may be unfeasible, and may cause privacy issues.

B. THE COMPLEXITY OF HAPPINESS AND SATISFACTION

In this section we consider the problems of maximizing happiness (all children are hosted) or satisfaction (at least one child is hosted) at a given round with no thought of the other rounds.

B.1 Achieving Maximum Happiness in Hard

Maximizing happiness at a given year means finding the largest set of nodes that can each host all their children. It is hard to maximize happiness and this follows from the tight relationship with the maximum independent set.

OBSERVATION B.1. *Maximizing Happiness is $\mathcal{MAXSNP}$-hard.*

PROOF. Consider the conflict graph G as defined in Section 2, It is easy to see that maximizing happiness means finding the maximum independent set which is $\mathcal{MAXSNP}$-hard even for degree 3 graphs [8]. $\square$

B.2 On the Hardness of Being Fair

The observation on the hardness of maximizing happiness also implies that any sort of fairness based on maximum happiness will be hard to compute (and hence hard to achieve). For instance, consider the coalitional game defined by the conflict graph $G = (V, E)$ where for each subset $S \subseteq V$ of nodes the value $v(S)$ is the size of the maximum independent set (MIS) of the subgraph induced by S (which represents the maximum happiness the parents in S can collectively obtain even if all the other nodes give up). Then clearly it is hard to compute $v(S)$. Moreover, we claim that solution concepts such as the *Shapley Value*[5] of this game are hard to compute: take an arbitrary order of the nodes and consider the total (i.e. sum of) marginal contributions of the nodes according to this order. It is always equal to the size of the MIS of the graph, since the number of times $MIS(S)$ can grow as S goes from the empty set to the full set is exactly the size of the MIS on the full set of nodes. Therefore any system that approximates the shares according to this definition can also be used to approximate the size of the MIS on the full graph: take the total happiness over a long enough period of time and it should approximate the MIS size. The inapproximability of the MIS problem to a factor of $n^{1-\varepsilon}$ (see [19]) implies a large difference between the average rate of hosting for a random node (which should be close to the fair share) and the size of the MIS divided by n. Thus, we left without a more sophisticated choice than competing with the 'chaotic' "first-come-first-grab" scheme described in the Introduction.

[5]The Shapley Value of a player is based on the expected *marginal contribution* of a player to the value of the game when the players are ordered at random; see more details in Osborne and Rubinstein [30] and [28] for its application in the carpool problem.

B.3 Maximum Satisfaction is Linear-Time computable

We say that parents are satisfied if at least one of their children comes home for the holiday.

In contrast to maximizing happiness, the problem of maximizing satisfaction is computationally easy.

THEOREM B.2. *Maximum satisfaction can be achieved in linear time.*

PROOF. Maximizing satisfaction means finding the maximum bipartite matching of the bipartite graph $G' = (V + C, E')$ where the set of nodes corresponds to the vertices V and edges $C = E$ from G and there is an edge between nodes corresponding to a node from V and the nodes corresponding to its edges in E. This can be done in time $O(\sqrt{n}|E'|)$ by the Hopcroft-Karp algorithm [20].

However, a general algorithm for maximum matching in bipartite graphs is an overkill for this problem given that every node in C has exactly two edges: it is possible to find a maximum matching by starting from the nodes in V parents that have just neighbor in G'; each such node is matched with its child (if two such nodes are share the same edge in G, then the satisfied one is decided arbitrarily). Nodes who became nodes with degree one since their other neighbors in G' have been matched are treated similarly. This continues until there are no degree one nodes in V. Then all the remaining nodes can be satisfied: pick an arbitrary neighbor and match it to a the node. At any point there can be at most one node with degree one. Match it to its only neighbor. $\square$

Note that this solution cannot be found in a distributed manner quickly, given that maximum matching requires $\Omega(n)$ distance communication in some graphs, such as the cycle of length n.

While achieving maximum satisfaction is fast, the solution is not socially acceptable, since the same nodes will be happy every round while others will never be happy. Note that if we want a scheme whereby all nodes are guaranteed to be happy within some cycle of time, then we can guarantee that a node will not be unsatisfied for more than one round: each edge simply alternates between its vertices.

C. DETAILS FOR THE ELIAS OMEGA CODE

DEFINITION C.1. *Let $i \in \mathbb{N}$. Denote by $B(i)$ the binary representation of i, i.e. $B(i)$ is a string over $\{0,1\}$ which is the representation of i in base 2 with no leading zeros. Denote the number of bits in $B(i)$ (the highest power of 2, b such that $2^b \leq i$) by $|B(i)|$. Let S be a binary number.*

Denote by $LSB(S,k)$ the suffix of S of length k, or the k least significant bits of S.

Let i be a positive integer. Denote the empty string by λ. Given two strings u and v, denote by $u \circ v$ the concatenation of u and v. Recursively define a binary string $re(i)$ as follows:

$$re(i) = \begin{cases} \lambda & \text{if } i\text{=}1, \\ re(|B(i)|-1) \circ B(i) & \text{if } i > 1. \end{cases}$$

The Elias omega encoding *of i, is $re(i) \circ 0$ and is denoted by $\omega(i)$.*

Example: Consider the Elias omega code of the following numbers:

1. $i = 1$: $re(1) = \lambda$. *Elias omega code*: 0.
2. $i = 9$: $B(9) = 1001$. $|B(9)| - 1 = 3$. $B(3) = 11$. $|B(3)| - 1 = 1$. Therefore $re(9) = re(1)B(3)B(9) = \lambda\ 11\ 1001$
   Elias omega code: 11 1001 0.
3. The Elias omega codes of the numbers 1 to 15 are: 0, 10 0, 11 0, 10 100 0, 10 101 0, 10 110 0, 10 111 0, 11 1000 0, 11 1001 0, 11 1010 0, 11 1011 0, 11 1100 0, 11 1101 0, 11 1110 0, 11 1111 0.

Properties 1.

1. *The Elias omega code is a prefix-free code.*
2. *The Elias omega code uses $\rho(i) = 1 + rb(i)$ bits to represent the number i, where $rb(i)$ is recursively defined as follows:*

$$rb(i) = \begin{cases} 0 & \text{if } i\text{=}1, \\ \lceil \log(i) \rceil + rb(\lceil \log(i) \rceil - 1) & \text{if } i > 1. \end{cases}.$$

   Explicitly, $\rho(i) = 1 + \lceil \log(i) \rceil + \lceil \log(\lceil \log(i) \rceil - 1) \rceil + \lceil \log(\lceil \log(\lceil \log(i) \rceil - 1) \rceil - 1) \rceil + \cdots$.