Guaranteeing Fair Service to Persistent Dependent Tasks

Amotz Bar-Noy* Alain Mayer† Baruch Schieber* Madhu Sudan*

August 17, 2001

Abstract

We introduce a new scheduling problem that is motivated by applications in the area of access and flow-control in high-speed and wireless networks. An instance of the problem consists of a set of *persistent tasks* that have to be scheduled repeatedly. Each task has a demand to be scheduled "as often as possible". There is no *explicit* limit on the number of tasks that can be scheduled concurrently. However, such limits are imposed implicitly because some tasks may be in conflict and cannot be scheduled simultaneously. These conflicts are presented in the form of a conflict graph. We define parameters which quantify the *fairness* and *regularity* of a given schedule. We then proceed to show lower bounds on these parameters, and present fair and efficient scheduling algorithms for the case where the conflict graph is an interval graph. Some of the results presented here extend to the case of perfect graphs and circular-arc graphs as well.

*IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598.
Email: {amotz,sbar,madhu}@watson.ibm.com.

†Dept. of Computer Science, Columbia University, New York, NY 10027. Email: mayer@cs.columbia.edu.
Part of this work was done while the author was at the IBM T. J. Watson Research Center. Partially supported by an IBM Graduate Fellowship, NSF grant CCR-93-16209, and CISE Institutional Infrastructure Grant CDA-90-24735

1 Introduction

In this paper we consider a new form of a scheduling problem that is characterized by two features:

Persistence: A task does not simply terminate once it is scheduled. Instead, each task must be scheduled infinitely many times. The goal is to schedule every task as frequently as possible.

Dependency: Some tasks conflict with each other and hence cannot be scheduled concurrently. These conflicts are represented by a conflict graph. This graph imposes constraints on the sets of tasks that may be scheduled concurrently. Note that these constraints are not based simply on the cardinality of the sets, but rather on the identity of the tasks within the sets.

We consider both the problems of *allocation*, i.e., how often should a task be scheduled and *regularity*, i.e., how evenly spaced are lengths of the intervals between successive scheduling of a specific task. We present a more formal description of this problem next and discuss our primary motivation immediately afterwards. While all our definitions are presented for general conflict graphs, our applications, bounds, and algorithms are for special subclasses – perfect graphs, interval graphs and circular arc-graphs.

Problem statement: An instance of the scheduling problem consists of a conflict graph G with n vertices. The vertices of G are the tasks to be scheduled and the edges of G define pairs of tasks that cannot be scheduled concurrently. The output of the scheduling algorithm is an infinite sequence of subsets of the vertices, $I_1, I_2, \ldots$, where I_t lists the tasks that are scheduled at time t. Note that I_t must be an independent set of G for all t.

In the form above, it is hard to analyze the running time of the scheduling algorithm. We consider instead a finite version of the above problem and use it to analyze the running time.

Input: *A conflict graph G and a time t.*

Output: *An independent set I_t denoting the set of tasks scheduled at time unit t.*

The objective of the scheduling algorithm is to achieve a *fair* allocation and a *regular* schedule. We next give some motivation and describe the context of our work. As we will see, none of the existing measures can appropriately capture the "goodness" of a schedule in our framework. Hence we proceed to introduce measures which allow a better presentation of our results.

1.1 Motivation

Session scheduling in high-speed local-area networks. The main motivation for this work arises from the session scheduling problem on a network called MetaRing. MetaRing

([CO93]) is a recent high-speed local-area ring-network that allows "spatial bandwidth reuse", i.e., in contrast to other ring networks which may only allow one source destination pair to communicate on the the ring at a time, the MetaRing can allow several such pairs to communicate with each other, provided the pairs do not use the same link for the communication. This concurrent access and transmission of user sessions is implemented using only minimal intermediate buffering of packets. A typical use of the MetaRing network involves several users trying to establish "sessions". A session is a simply a source destination pair, where the source wishes to send data over to the destination. The data is generated over time and needs to be shipped over at regular intervals, if not immediately. In the general setting, the set of sessions can change dynamically. We restrict our attention to the static case. This restriction is justified by the fact that sessions typically last for a long while. As a result of the minimal intermediate buffering, a session can send its data over only if it has exclusive use of all the links in its route. Consequently, sessions whose routes share a link are in conflict. These conflicts must be regulated by breaking the data sent in a session into units of quotas that are transmitted according to some schedule. This schedule has to be *efficient* and *fair*. Efficient means that the total number of quotas transmitted (throughput) is maximized. Fair means that the throughput of *each* session is maximized, and that the time between successive activation of a session is minimized, so that large buffers at the source nodes can be avoided. It has been recognized ([CCO93]) that the access and flow-control in such a network should depend on locality in the conflict graph. However, no firm theoretical basis for an algorithmic framework has been proposed up to now. To express this problem as our scheduling problem we create a circular-arc graph the vertices of which are the sessions, and in which vertices are adjacent if the corresponding paths associated with the sessions intersect. Since an independent set in this graph is a collection of mutually non-conflicting sessions, they can all communicate simultaneously. Thus a schedule is a sequence of independent sets in this graph. Our goal will be to produce such a schedule which is efficient and fair, in a sense to be made precise later.

Time sharing in wireless networks. A second naturally occurring scenario where our model can be applied is in the scheduling of base-station transmissions in a cellular network. We now describe this setting in detail. Most indoor designs of wireless networks are based on a *cellular* architecture with a very small cell size (see, e.g., [Goo90].) The cellular architecture comprises two levels – a stationary level and a mobile level. The stationary level consists of fixed base stations that are interconnected through a backbone network. The mobile level consists of mobile units that communicate with the base stations via wireless links. The geographic area within which mobile units can communicate with a particular base station is referred to as a cell. Neighboring cells overlap with each other, thus ensuring continuity of communications. The mobile units communicate among themselves, as well as with the fixed information networks, through the base stations and the backbone network. The continuity of communications is a crucial issue in such networks. A mobile user who crosses boundaries of cells should be able to continue its communication via the new base-station. To ensure this, base-stations periodically need to transmit their identity using the wireless communication. In some implementations the wireless links use infra-red waves. Therefore, two base-station

the cells of which overlap are in conflict and cannot transmit their identity simultaneously. These conflicts have to be regulated by a time-sharing scheme. This time sharing has to be *efficient* and *fair*. Efficient means that the scheme should accommodate the maximal number of base stations. Fair means that the time between two consecutive transmissions of the same base-station should be less then the time it takes a user to cross its corresponding cell. Once again this problem can be posed as our graph scheduling problem where the vertices of the graph are the base-stations and an edge indicates that the base stations belong to overlapping cells. Independent sets again represent lack of conflict and a schedule will thus be a sequence of independent sets.

1.2 Relationship to past work

Previous research on scheduling problems in our framework considered either persistence of the tasks or dependency among the tasks but not both.

An early work by Liu and Layland [LL73] considers persistent scheduling of tasks in a single processor environment. In their problem, tasks have deadlines specifying a time limit by when the ith scheduling of a given task must have occurred. They give an algorithm which achieves full processor utilization for this task. More recently, the problem of scheduling persistent tasks has been studied in the work of Baruah *et al.* [BCPV96]. Their setting is closer to ours and we describe their problem in detail: They considered the problem of scheduling a set of n tasks with given (arbitrary) frequencies on m machines. (The case $m = 1$ is equivalent to an instance of our problem in which the conflict graph is a clique.) To measure *regularity* of a schedule for their problem they introduced the notion of *P-fairness*. A schedule for this problem is P-fair (proportionate-fair) if at each time t for each task i the absolute value of the difference in the number of times i has been scheduled and $f_i t$ is strictly less than 1, where f_i is the frequency of task i. They provided an algorithm for computing a P-fair solution to their problem. Their problem fails to capture our situation due to two reasons. First, we would like to constrain the sets of tasks that can be scheduled concurrently according to the topology of the conflict graph and not according to their cardinality. Moreover, in their problem every feasible frequency requirement can be scheduled in a P-fair manner. For our scheduling problem, we show that such a P-fair schedule cannot always be achieved. To deal with feasible frequencies that cannot be scheduled in a P-fair manner, we define weaker versions of regularity.

The dependency property captures most of the work done based on the well-known "Dining Philosophers" paradigm (see for example [Dijk71], [Lyn80], [CM84], [AS90], [CS92], and [BP92]). In this setting, Lynch [Lyn80] was the first to explicitly consider the *response time* for each task. The goal of successive works was to make the response time of a node to depend only on its local neighborhood in the conflict graph (see, e.g., [BP92]). While response time in terms of a node's degree is adequate for "one-shot" tasks, it does not capture our requirement that a task should be scheduled in a regular and fair fashion over a period of time.

1.3 Notations and definitions

A *schedule* S is an infinite sequence of independent sets $I_1, I_2, \ldots, I_t, \ldots$ Let $S(i,t)$ denote the indicator variable that represents the schedule; it is 1 if task i is scheduled at time t and 0 otherwise. Let $f_i^{(t)}(S) = \sum_{\tau=1}^{t} S(i,\tau)/t$, we refer to $f_i^{(t)}(S)$ as the *prefix frequency* of task i at time t in schedule S. Let $f_i(S) = \liminf_{t\to\infty}\{f_i^{(t)}(S)\}$. We refer to $f_i(S)$ as the *frequency* of task i in schedule S. We say that a schedule S is *periodic* with *period* T, if $I_j = I_{T+j} = I_{2T+j} = \cdots$ for all $1 \le j \le T$. In periodic schedules we will refer to $f_i^{(T)}(S)$ as the frequency of task i. (In both $f_i^{(t)}(S)$ and $f_i(S)$ we drop the index S whenever the identity of the schedule S is clear from the context.)

Definition 1 *A vector of frequencies* $\hat{f} = (f_1, \ldots, f_n)$ *is* feasible *if there exists a schedule* S *such that the frequency of the* i-th *task in schedule* S *is at least* f_i.

Definition 2 *A schedule* S realizes *a vector of frequencies* $\hat{f}$ *if the frequency of the* i-th *task in schedule* S *is at least* f_i. *A schedule* S c-approximates *a vector of frequencies* $\hat{f}$ *if the frequency of the* i-th *task in schedule* S *is at least* f_i/c.

A measure of fairness. Fairness is determined via a partial order $\prec$ that we define on the set of frequency vectors.

Definition 3 *Given two frequency vectors* $\hat{f} = (f_1, \ldots, f_n)$ *and* $\hat{g} = (g_1, \ldots, g_n)$, $\hat{f} \prec \hat{g}$ *($\hat{f}$ is less fair than $\hat{g}$) if there exists an index i and a threshold f such that $f_i < f \le g_i$, and for all j such that $g_j \le f$, $f_j \le g_j$.*

Less formally, if $\hat{f} \prec \hat{g}$, then $\hat{g}$ "performs" better for some task i and all tasks with frequency smaller than the i-th task, i.e., those tasks that are least scheduled. It could be the case that $\hat{f}$ "performs" better for the other tasks, however our concern in fair allocation is with the less frequently scheduled tasks.

In Appendix A we prove that the relation $\prec$ is indeed a partial order.

Definition 4 *A vector of frequencies* $\hat{f}$ *is* max-min fair *if no feasible vector* $\hat{g}$ *satisfies* $\hat{f} \prec \hat{g}$.

Less formally, in a max-min fair frequency vector, one cannot increase the frequency of some task at the expense of less frequently scheduled tasks. This means that our goal is to let each task i have more of the resource as long as we have to take the resource away only from tasks which are better off, i.e., those that have more of the resource than task i.

Measures of regularity. We provide two measures by which one can evaluate a schedule for its *regularity*. We call these measures the *response time* and the *drift*.

Given a schedule S, the *response time* for task i, denoted r_i, is the largest interval of time for which task i waits between successive schedulings. More precisely,

$$r_i = \max\{t_2 - t_1 | 0 \le t_1 < t_2 \text{ s.t. } \forall_{t_1<t<t_2} S(i,t) = 0\}.$$

For any time t, the number of expected occurrences of task i can be expressed as $f_i t$. But note that if r_i is larger than $1/f_i$, it is possible that, for some period of time, a schedule allows a task to "drift away" from its expected number of occurrences. In order to capture this, we introduce a second measure for the regularity of a schedule. We denote by d_i the *drift* of a task i. It indicates how much a schedule allows task i to drift away from its expected number of scheduled units:

$$d_i = \max_t \{|f_i \cdot t - \sum_{r=1}^{t} S(i,r)|\}.$$

Note that if a schedule S achieves drift $d_i < 1$ for all i, then it is *P-fair* as defined in [BCPV96].

A schedule achieves its strongest form of regularity if each task i is scheduled every $1/f_i$ time-units (except for its first appearance). We say that a schedule is *rigid* if for each task i there exists a starting point s_i such that the task is scheduled on exactly the time units $s_i + j(1/f_i)$, for $j \geq 0$.

Graph subclasses. A graph is *perfect* if for all its induced subgraphs the size of the maximum clique is equal to the chromatic number (cf. [Gol80]). A graph is an interval graph (circular-arc graph) if its vertices correspond to intervals on a line (circle), and two vertices are adjacent if the corresponding intervals intersect (cf. [Tuc71]).

1.4 Results

In Section 2 we motivate our definition of max-min fairness and show several of its properties. First, we provide an equivalent (alternate) definition of feasibility which shows that deciding feasibility of a frequency vector is computable. Next, we prove that every graph has a *unique* max-min fair frequency vector. Then, we show that the task of even weakly-approximating the max-min fair frequencies on general graphs is NP-hard. As we mentioned above, many practical applications of this problem arise from simpler networks, such as buses and rings (i.e., interval conflict graphs and circular-arc conflict graphs). For the case of perfect-graphs (and hence for interval graphs), we describe an efficient algorithm for computing max-min fair frequencies. We prove that the period T of a schedule realizing such frequencies on a perfect graph satisfies $T = 2^{O(n)}$ and that there exist interval graphs such that $T = 2^{\Omega(n)}$.

The rest of our results deal with the problem of finding the most "regular" schedule that realizes any feasible frequency vector. In Section 3 we show the existence of interval graphs for which there is no P-fair schedule that realizes their max-min fair frequencies. In Section 4 we describe an algorithm for computing a schedule that realizes any given feasible frequencies on interval graphs. The schedule computed by the algorithm achieves response-time of $\lceil 4/f_i \rceil$ and drift of $O(\sqrt{\log T} n^{\epsilon})$. A slight modification of this algorithm yields a schedule that 2-approximates the given frequencies. The advantage of this schedule is that it achieves a bound of one on the drift and hence a bound of $\lceil 2/f_i \rceil$ on the response time. In Section 5 we present an algorithm for computing a schedule that 12-approximates any given feasible frequencies on interval graphs and has the advantage of being rigid. All algorithms run in polynomial

time. In Section 6 we show how to transform any algorithm for computing a schedule that c-approximates any given feasible frequencies on interval graphs into an algorithm for computing a schedule that $2c$-approximates any given feasible frequencies on circular-arc graphs. The response-time and drift of the resulting schedule are doubled as well.

Finally, in Section 7 we summarize our results, list a number of open problems and sketch what additional properties are required to obtain solutions for actual networks.

2 Max-min Fair Allocation

Our definition for max-min fair allocation is based on the definition used by Jaffe [Jaf81] and Bertsekas and Gallager [BG87], but differs in one key ingredient – namely our notion of feasibility. We study some elementary properties of our definition in this section. In particular, we show that our definition guarantees a unique max-min fair frequency vector for every conflict graph. We also show the hardness of computing the frequency vector for general graphs. However, for the special case of perfect graphs our notion turns out to be the same as that of [BG87].

The definition of [Jaf81] and [BG87] is considered as the traditional way of measuring throughput fairness in communication networks and is also based on the partial order $\prec$ as used in our definition. The primary difference between our definition and theirs is in the definition of *feasibility*. Bertsekas and Gallager [BG87] use a definition, which we call clique feasibility, that is defined as follows:

> A vector of frequencies $(f_1, \ldots, f_n)$ is *clique feasible* for a conflict graph G, if $\sum_{i \in C} f_i \leq 1$ for all cliques C in the graph G.

The notion of max-min fairness of Bertsekas and Gallager [BG87] is now exactly our notion, with feasibility replaced by clique feasibility.

The definition of [BG87] is useful for capturing the notion of fractional allocation of a resource such as bandwidth in a communication networks. However, in our application we need to capture a notion of integral allocation of resources and hence their definition does not suffice for our purposes. By definition, every frequency vector that is feasible in our sense is clique feasible. However, the converse is not true. Consider the case where the conflict graph is the five-cycle. For this graph the vector $(1/2, 1/2, 1/2, 1/2, 1/2)$ is clique feasible, but no schedule can realize this frequency vector.

2.1 An alternate definition of feasibility

Given a conflict graph G, let $\mathcal{I}$ denote the family of all independent sets in G. For $I \in \mathcal{I}$, let $\chi(I)$ denote the characteristic vector of I.

Theorem 5 *A vector of frequencies $\hat{f}$ is feasible if and only if $\hat{f}$ is a convex combination of the $\chi(I)$'s; that is, there exist weights $\{\alpha_I\}_{I \in \mathcal{I}}$, such that $\sum_{I \in \mathcal{I}} \alpha_I = 1$ and $\sum_{I \in \mathcal{I}} \alpha_I \chi(I) = \hat{f}$.*

Proof: Claim 6 proves the easier direction. Claim 7 proves the other direction only for the case when $\hat{f}$ can be expressed as a rational convex combination of the independent sets. The proof for the case when $\hat{f}$ is not a rational convex combination is given in Appendix B. □

Claim 6 *If a frequency vector $\hat{f}$ is feasible then there exists a sequence of weights $\{\alpha_I\}_{I\in\mathcal{I}}$ such that $\sum_I \alpha_I = 1$ and $\sum_I \alpha_I \chi(I) = \hat{f}$.*

Proof: To obtain a contradiction assume otherwise. By continuity there exists an $\epsilon > 0$, such that the vector $(1-\epsilon)\hat{f}$ cannot be expressed as a convex combination of the $\chi(I)$'s. Based on the definition of feasibility, there exists a schedule S which achieves a frequency of at least $\hat{f}$. In particular, there exists a time $T = T_\epsilon$ such that $f_i^{(T)} \geq f_i - \epsilon$ for all $1 \leq i \leq n$. Let α_I be the frequency of the independent set I in the first T time units in the schedule S. Then $\sum_I \alpha_I = 1$ and $\sum_I \alpha_I \chi(I) \geq (1-\epsilon)\hat{f}$, contradicting the choice of ϵ. □

Claim 7 *If a frequency vector $\hat{f}$ can be expressed as a rational convex combination of the independent sets, then $\hat{f}$ is feasible.*

Proof: Suppose that there exist rational weights $\{\alpha_I\}_{I\in\mathcal{I}}$, such that $\sum_{I\in\mathcal{I}} \alpha_I = 1$ and $\sum_{I\in\mathcal{I}} \alpha_I \chi(I) = \hat{f}$. Express α_I as p_I/q_I where p_I and q_I are integral and let $T = \mathrm{LCM}\{q_I\}$. Observe that $N_I \equiv \alpha_I T$ is integral. For each I we N_I times schedule the independent set I over a period of T intervals (in any arbitrary units of time). It is clear that there are enough slots for each independent set to be scheduled N_I times. □

The main impact of Theorem 5 is that it shows that the space of all feasible frequencies is well behaved (i.e., it is a closed, connected, compact space). In addition, it shows that determining whether a frequency vector is feasible is a computable task (a fact that may not have been easy to see from the earlier definition). We now use this definition to observe the following interesting connection:

Proposition 8 *Given a conflict graph G, the notions of feasibility and clique feasibility are equivalent if and only if G is perfect.*

Proof: The proof follows directly from well-known polyhedral properties of perfect graphs. (See [GLS87], [Knu94].) In the notation of Knuth [Knu94] the space of all feasible vectors is the polytope $\mathsf{STAB}(G)$ and the space of all clique-feasible vectors is the polytope $\mathsf{QSTAB}(G)$. The result follows from the theorem on page 38 in [Knu94] which says that a graph G is perfect if and only if $\mathsf{STAB}(G) = \mathsf{QSTAB}(G)$. □

2.2 Uniqueness and computability of max-min fair frequencies

In this subsection we prove that the max-min fair frequency vector is unique. We also show that finding this vector (or even approximating it) is computationally hard.

Theorem 9 *For any conflict graph there exists a unique max-min fair frequency vector.*

Proof: For a vector $\hat{f}$ let $\mathsf{sort}\hat{f}$ denote the vector obtained by permuting the vector $\hat{f}$ so that its coordinates appear in nondecreasing order. Let the relation $\prec_{\mathrm{lex}}$ on the feasible frequency

vectors be the lexicographic ordering on $\mathsf{sort}\hat{f}$. More precisely, $\hat{f} \prec_{\text{lex}} \hat{g}$ if either $\mathsf{sort}\hat{f} = \mathsf{sort}\hat{g}$ or there exists an index $i \geq 1$ such that $\mathsf{sort}\hat{f}_i < \mathsf{sort}\hat{g}_i$ and $\mathsf{sort}\hat{f}_j = \mathsf{sort}\hat{g}_j$ for all $1 \leq j < i$. (Note that $\prec_{\text{lex}}$ is not a partial order since it is not anti-symmetric.) It is easy to verify that if $\hat{f} \prec_{\text{lex}} \hat{g}$, and $\mathsf{sort}\hat{f} \neq \mathsf{sort}\hat{g}$ then $f \prec g$. Let the vector $\hat{f}$ be a max-min fair vector. (Such a vector exists in the space of feasible vectors, since this space is compact.) The vector $\hat{f}$ is also larger according to the ordering $\prec_{\text{lex}}$ than any vector $\hat{g}$ such that $\mathsf{sort}\hat{f} \neq \mathsf{sort}\hat{g}$. We now show that there is no vector $\hat{g}$ such that $\mathsf{sort}\hat{f} = \mathsf{sort}\hat{g}$. This implies that it is the unique max-min fair frequency vector.

To obtain a contradiction, suppose that there exists a vector $\hat{g}$ such that $\mathsf{sort}\hat{f} = \mathsf{sort}\hat{g}$. First observe that the vector $\hat{h} = (\hat{f} + \hat{g})/2$ is feasible. This is true because $\hat{f}$ and $\hat{g}$ can be expressed as a convex combination of the independent sets and $\hat{h}$ is a convex combination of $\hat{f}$ and $\hat{g}$. Thus $\hat{h}$ is a convex combination of the independent sets. Now assume without loss of generality that the indices of the vectors are arranged in increasing order of $f_i + g_i$. Let j be the smallest index such that $f_j \neq g_j$. Say, f_j is the smaller of the two. Then $((\hat{f} + \hat{g})/2)_j$ is greater than f_j and for all vertices i with smaller frequencies, $f_i = ((\hat{f} + \hat{g})/2)_i$. This implies that $\hat{f} \prec_{\text{lex}} (\hat{f} + \hat{g})/2$. A contradiction. □

We now turn to the issue of the computability of the max-min fair frequencies. While we do not know the exact complexity of computing max-min fair frequencies (in particular, we do not know if deciding whether a frequency vector is feasible is in NP ∪ coNP), it does seem to be very hard in general. Here, we consider the sub-problem of computing the smallest frequency assigned to any vertex by a max-min allocation and show the following:

Theorem 10 *There exists an $\epsilon > 0$, such that given a conflict graph on n vertices, approximating the smallest frequency assigned to any vertex in a max-min fair allocation to within a factor of n^{ϵ} is NP-hard.*

Proof: We relate the computation of max-min fair frequencies in a general graph to the computation of the fractional chromatic number of a graph. We then use the recent hardness result for approximating the (fractional) chromatic number, due to Lund and Yannakakis [LY93] to show that computing max-min fair frequencies in general graphs is very hard.

The fractional chromatic number problem (cf. [LY93]) is defined as follows:

> To each independent set I in the graph, assign a weight w_I, so as to minimize the quantity $\sum_I w_I$, subject to the constraint that for every vertex v in the graph, the quantity $\sum_{I \ni v} w_I$ is at least 1. The quantity $\sum_I w_I$ is called the *fractional chromatic number* of the graph.

Observe that if the w_I's are forced to be integral, then the fractional chromatic number is the chromatic number of the graph.

The following claim shows a relationship between the fractional chromatic number and the assignment of feasible max-min fair frequencies.

Claim 11 *Let $(f_1, f_2, \ldots, f_n)$ be a feasible assignment of frequencies to the vertices in a graph G. Then $1/(\min_i f_i)$ is an upper bound on the fractional chromatic number of the graph.*

Conversely, if k is the fractional chromatic number of a graph, then a schedule that sets the frequency of every vertex to be $1/k$ is feasible.

The proof of the above claim is straightforward given the definitions of fractional chromatic number and feasibility. We now show how to use the claim to prove the theorem.

The above claim, combined with the hardness of computing the fractional chromatic number [LY93], suffices to show the NP-hardness of deciding whether a given assignment of frequencies is feasible for a given graph. To show that the claim also implies the hardness of approximating the smallest frequency in the max-min fair frequency vector we inspect the Lund-Yannakakis construction a bit more closely. Their construction yields a graph in which every vertex participates in a clique of size k such that deciding if the (fractional) chromatic number is k or kn^ϵ is NP-hard. In the former case, the max-min fair frequency assignment to every vertex is at least $1/k$. In the latter case at least some vertex will have frequency smaller that $1/(kn^\epsilon)$. Thus this implies that approximating the smallest frequency in the max-min fair frequencies to within a factor of n^ϵ is NP-hard. □

2.3 Max-Min fair frequencies on perfect graphs

We now consider perfect graphs. We show how to compute in polynomial time max-min fair frequencies for this class of graphs and give bounds on the period of a schedule realizing such frequencies. As our main focus of the subsequent sections will be interval graphs, we will give our algorithms and bounds first in terms of this subclass and then show how to generalize the results to perfect graphs.

We start by describing an algorithm for computing max-min fair frequencies on interval graphs. As we know that clique-feasibility equals feasibility (by Proposition 8), we can use an adaptation of [BG87]:

Algorithm 1. Let $\mathcal{C}$ be the collection of maximal cliques in the interval graph. (Notice that $\mathcal{C}$ has at most n elements and can be computed in polynomial time.) For each clique $C \in \mathcal{C}$ the algorithm maintains a *residual capacity* which is initially 1. To each vertex the algorithm associates a label *assigned/unassigned*. All vertices are initially unassigned. Dividing the residual capacity of a clique by the number of unassigned vertices in this clique yields the *relative* residual capacity. Iteratively, we consider the clique with the smallest current relative residual capacity and assign to each of the clique's unassigned vertices this capacity as its frequency. For each such vertex in the clique we mark it assigned and subtract its frequency from the residual capacity of every clique that contains it. We repeat the process till every vertex has been assigned some frequency.

It is not hard to see that Algorithm 1 correctly computes max-min fair frequencies in polynomial-time. We now use its behavior to prove a tight bound on the period of a schedule for an interval graph. The following theorem establishes this bound:

Theorem 12 *Let $f_i = p_i/q_i$ be the frequencies in a max-min fair schedule for an interval graph*

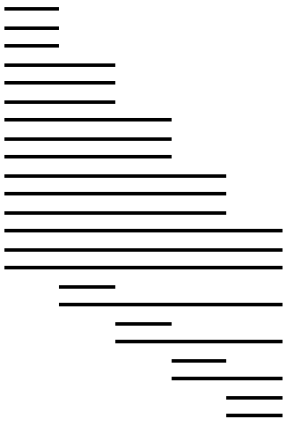

Figure 1: An interval graph with $n = 23$ intervals for which $T = \mathrm{LCM}_{i=1}^{n}\{q_i\} \geq 2^{\frac{n+2}{5}-1}$.

G, where p_i and q_i are relatively prime. Then, the period for the schedule $T = LCM_{i=1}^{n}\{q_i\}$ satisfies, $T = 2^{O(n)}$. Furthermore, there exist interval graphs for which $T = 2^{\Omega(n)}$.

We prove this theorem with the help of the following two lemmas:

Lemma 13 *$LCM_{i=1}^{n}\{q_i\} \leq 2^{n/2}$.*

Proof: Let n_j denotes the number of intervals that are assigned frequency $\frac{p_j}{q_j}$ in iteration j. That is, $\frac{p_j}{q_j}$ is the minimum relative residual capacity at iteration j. From the way the relative residual capacities are updated, it follows that q_i divides $\prod_{j=1}^{i} n_j$ for all $1 \leq i \leq n$. The lemma follows since assuming there were ℓ iterations, q_i divides $\prod_{j=1}^{\ell} n_j$, and $\prod_{j=1}^{\ell} n_j$ attains its maximum when $\ell = n/2$ and $n_i = 2$ for all $1 \leq i \leq \ell$. □

Lemma 14 *There exists an interval graph for which $LCM_{i=1}^{n}\{q_i\} \geq 2^{\frac{n+2}{5}-1}$.*

Proof: We show an interval graph in which $\max_{i=1}^{n}\{q_i\} \geq 2^{\frac{n+2}{5}-1}$, the lemma follows since trivially $\mathrm{LCM}_{i=1}^{n}\{q_i\} \geq \max_{i=1}^{n}\{q_i\}$. For simplicity we assume that 5 divides $n+2$. The reader may find it easier to follow the construction for $n = 23$ depicted in Figure 1. Let $x = \frac{3(n+2)}{5}$ and $y = \frac{n+2}{5} - 1$. In the construction x intervals start at 0 (the top 15 intervals in Figure 1), and two intervals start at i for all $1 \leq i \leq y$ (the bottom 8 intervals in Figure 1). Note that indeed $2y + x = n$. Out of the first x intervals, three intervals terminate at i for each $1 \leq i \leq y + 1$. Note that indeed $3(y + 1) = x$. For $1 \leq i \leq y$, out of the two intervals starting at i, one interval terminates at $i + 1$ and one continues until $y + 1$.

Now, since at 0 the size of the clique is x and at i the size of the clique is $x - 3i + (i - 1) + 2 = x - 2i + 1$, it follows that the algorithm for the frequency assignment handles the cliques from left to right and there are $y + 1$ different values for frequencies denoted by $w_0, \ldots, w_y$. We get

$$\begin{aligned} w_0 &= \frac{1}{x}\,, \\ w_1 &= \frac{3w_0}{2} = \frac{3}{2x}\,, \\ w_2 &= \frac{3w_0 + w_1}{2} = \frac{9}{4x}. \end{aligned}$$

In general, by induction we prove that $w_i = \frac{3}{x} \cdot \frac{2^i-1}{2^i}$ for $1 \leq i \leq y$.

$$w_{i+1} = \frac{3w_0 + w_i}{2} = \frac{3}{2x} + \frac{3}{x} \cdot \frac{2^i - 1}{2^{i+1}} = \frac{3}{x} \cdot \frac{1}{2} + \frac{2^i - 1}{2^{i+1}} = \frac{3}{x} \cdot \frac{2^{i+1} - 1}{2^{i+1}}.$$

Since the denominator of w_y is greater or equal to 2^y and neither 3 nor $2^y - 1$ has a common divisor with 2^y, it follows that $\max_{i=1}^{n}\{q_i\} \geq 2^{\frac{n+2}{5}-1}$. □

Algorithm 1 works for all graphs where clique feasibility determines feasibility; i.e., perfect graphs. However, the algorithm does not remain computationally efficient since it involves scanning all the cliques in the graph. Still, Theorem 12 can be directly extended to the class of perfect graphs. We now use this fact to describe a polynomial-time algorithm for assigning max-min fair frequencies to perfect graphs.

Algorithm 2. This algorithm maintains the labelling procedure *assigned/unassigned* of Algorithm 1. At each phase, the algorithm starts with a set of assigned frequencies and tries to find the largest f such that all unassigned vertices can be assigned the frequency f. To compute f in polynomial time, the algorithm uses the fact that deciding if a given set of frequencies is feasible is reducible to the task of computing the size of the largest weighted clique in a graph with weights on vertices. The latter task is well known to be computable in polynomial-time for perfect graphs. Using this decision procedure the algorithm performs a binary search to find the largest achievable f. (The binary search does not have to be too refined due to the upper bound on the denominators of the frequencies given in Theorem 12.) Having found the largest f, the algorithm finds a set of vertices which are saturated under f as follows: Let ϵ be some small number, with the property that the difference between any two distinct assigned frequencies is more than ϵ. By Theorem 12, $\epsilon = 2^{-n^2}$ is sufficient. Now the algorithm raises, one at a time, the frequency of each unassigned vertex to $f + \epsilon$, while maintaining the other unassigned frequencies at f. If the so obtained set of frequencies is not feasible, then it marks the vertex as assigned and its frequency is assigned to be f. The algorithm now repeats the phase until all vertices have been assigned some frequency.

3 Non-existence of P-fair allocations

We show that a P-fair scheduling under max-min fair frequencies need not exist for every interval graph.

Theorem 15 *There exist interval graphs G for which there is no P-fair schedule that realizes their max-min frequency assignment.*

Proof: In order to prove the theorem we produce a counter example as follows. We choose a parameter k and for every permutation π of the elements $\{1, \ldots, k\}$, we define an interval graph G_π. We show a necessary condition that π must satisfy if G_π has a P-fair schedule. Lastly, we show that there exists a permutation π of 12 elements which does not satisfy this condition.

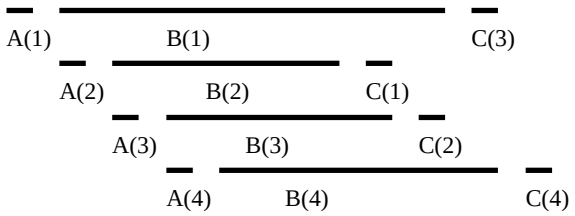

Figure 2: The graph G_π for $\pi = (3, 1, 2, 4)$

Given a permutation π on k elements, G_π consists of $3k$ intervals. For $1 \leq 1 \leq k$, define the intervals $A(i) = (i-1, i]$, $B(i) = (i, k + \pi(i) + 1]$ and $C(i) = (k+i+1, k+i+2]$. Observe that the max-min frequency assignment to G_π is the following: All the tasks $B(1), \ldots, B(k)$ have frequency $1/k$; task $A(i)$ has frequency $(k-i+1)/k$ for $1 \leq i \leq k$; and task $C(i)$ has frequency i/k for $1 \leq i \leq k$. (See Figure 2.)

We now observe the properties of a P-fair schedule for the tasks in G_π. (i) The time period is k. (ii) The schedule is entirely specified by the schedule for the tasks $B(i)$. (iii) This schedule is a permutation σ of k elements, where $\sigma(i)$ is the time unit for which $B(i)$ is scheduled. To see what kind of permutations σ constitute P-fair schedules of G_π we define the notion of when a permutation is *fair* for another permutation.

Definition 16 *A permutation σ_1 is* fair *for a permutation σ_2 if for all $1 \leq i, j \leq k$, σ_1 and σ_2 satisfy the conditions* cond_{ij} *defined as follows:*

$$\frac{ij}{k} - 1 < |\{\ell : \sigma_1(\ell) \leq j \text{ and } \sigma_2(\ell) \leq i\}| < \frac{ij}{k} + 1 \ .$$

Claim 17 *If a permutation σ is a P-fair schedule for G_π then σ is fair for the identity permutation and permutation π.*

Proof: For fixed i, we claim that the conditions cond_{ij} for $\sigma_1 = \sigma$ and σ_2 being the identity permutation, are exactly the conditions for a P-fair allocation of $A(i+1)$. Similarly, the conditions cond_{ij} for $\sigma_1 = \sigma$ and $\sigma_2 = \pi$ are the conditions for a P-fair allocation of $C(k-i)$. Thus, a permutation σ represents a P-fair schedule for G_π if and only if σ is fair for both π and the identity permutation.

We now show why the conditions cond_{ij} for $\sigma_1 = \sigma$ and σ_2 being the identity permutation, are exactly the conditions for a P-fair allocation of $A(i+1)$. The claim about the conditions cond_{ij} for $\sigma_1 = \sigma$ and $\sigma_2 = \pi$ is analogous. Recall that the frequency of task $A(i+1)$ is $(k-i)/k$ and that $A(i+1)$ can be scheduled only when tasks $B(\ell)$, for $1 \leq \ell \leq i$, are not scheduled. Consider the schedule up to time $j \leq k$. In order for the schedule to be P-fair, the number of occurrences of tasks $B(\ell)$, for $1 \leq \ell \leq \ell$, up to this time must be between $j - \frac{(k-i)j}{k} - 1 = \frac{ij}{k} - 1$ and $j - \frac{(k-i)j}{k} + 1 = \frac{ij}{k} + 1$. Note that the number of times these tasks are scheduled is the cardinality of the set $\{\ell : \sigma_1(\ell) \leq j \text{ and } \ell \leq i\}$, which translates to cond_{ij} for $\sigma_1 = \sigma$ and σ_2 being the identity permutation. □

Let $\pi = (1, 3, 4, 7, 8, 9, 11, 5, 12, 10, 2, 6)$ be a permutation on 12 elements. The following arguments show that no permutation σ is fair to both the identity permutation and the per-

mutation π.

1. We define a *block* as any contiguous set of elements in the range 1 to 12. We say that σ places an element i in the block $[j, \ell]$.

2. Without loss of generality assume that σ places the element 1 in the block $[1, 6]$.

3. Consider the elements in the following six pairs $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$, $\{7, 8\}$, $\{9, 10\}$, and $\{11, 12\}$. If the permutation σ is fair for the identity permutation, then it must place exactly one element of each pair in the block $[1, 6]$. To see this, note that if σ places both elements 1 and 2 in the block $[1, 6]$, then $|\{\ell : \sigma(\ell) \leq 6 \text{ and } \ell \leq 2\}| = 2$; violating $\text{cond}_{2,6}$. Thus only element 1 is in block $[1, 6]$. Inductively, it can be shown that if σ places both elements $2i - 1$ and $2i$, for $1 < i \leq 6$, in the block $[1, 6]$, then $\text{cond}_{2i,6}$ is violated.

4. A similar argument applied to π implies that σ must place exactly one element of each of the six pairs $\{1, 3\}$, $\{4, 7\}$, $\{8, 9\}$, $\{11, 5\}$, $\{12, 10\}$, and $\{2, 6\}$ in the block $[1, 6]$ if σ is fair for π.

5. Arguments 2, 3, and 4 imply that the first half of σ consists of the elements $\{1, 4, 8, 10, 11, 6\}$ and the second half consists of the elements $\{3, 7, 9, 12, 5, 2\}$.

6. Again, since σ is fair for the identity permutation, σ must place exactly one of the elements of each of the triplets $\{1, 2, 3\}$, $\{4, 5, 6\}$, and $\{7, 8, 9\}$ in each of the blocks $[1, 4]$, $[5, 8]$, and $[9, 12]$ in order not to violate conditions: $\text{cond}_{3,4}$, $\text{cond}_{3,8}$, and $\text{cond}_{3,12}$.

7. Similarly, since σ is fair for π, σ must place exactly one of the elements of the triplet $\{\pi(7), \pi(8), \pi(9)\} = \{11, 5, 12\}$ in each of the blocks $[1, 4]$, $[5, 8]$, and $[9, 12]$.

8. Since 1 appears in the block $[1, 6]$ and both 2 and 3 appear in the block $[7, 12]$, it follows from Argument 6 that exactly one of the elements 2 and 3 is placed in the block $[7, 8]$ by σ.

9. A similar argument applied to the triplet $\{7, 8, 9\}$ implies that exactly one of 7 and 9 is placed in the block $[7, 8]$ by σ.

10. Lastly, we examine the triplet $\{\pi(7), \pi(8), \pi(9)\} = \{11, 5, 12\}$. It follows from Argument 7 that one of 5 and 12 must appear in the block $[7, 8]$.

Since σ cannot place three elements in a block of size two, we obtain the contradiction.

The proof of Theorem 15 follows. □

4 Realizing frequencies exactly

In this section we first show how to construct a schedule which realizes any feasible set of frequencies (and hence in particular max-min frequencies) exactly on an interval graph. We prove its correctness and give a bound of $\lceil 4/f_i \rceil$ on the response time for each interval i. We

then proceed to introduce a potential function which can be used to yield a bound of $O(n^{\frac{1}{2}+\epsilon})$ on the drift for every interval. An easy consequence of our algorithm is for the special case in which the frequencies are of the form $1/2^i$, the drift can be bounded by 1 and thus the waiting time can be bounded by $\lceil 2/f_i \rceil$. This yields a 2-approximation algorithm with high regularity.

Input to the Algorithm: A unit of time t and a conflict graph G which is an interval graph. The graph G is represented by a set $I = \{I_1, \ldots, I_n\}$ of intervals on the unit interval $[0, 1]$ of the x-coordinate, where $I_i = [i.s, i.e]$ for $1 \leq i \leq n$. Every interval I_i has a frequency $f_i = p_i/q_i$ with the following constraint: $\sum_{I_i \ni x} f_i \leq 1$ for all $0 \leq x \leq 1$. For simplicity, we assume from now on that these constraints on the frequencies are met with equality and that $t \leq T = \text{LCM}\{q_i\}$.

Output of the Algorithm: An independent set I_t defining the set of tasks scheduled for time t such that the scheduled S, given by $\{I_t\}_{t=1}^{T}$ realizes frequencies f_i.

The algorithm is recursive. Let s_i denote the number of times a task i has to appear in T time units, i.e., $s_i = Tp_i/q_i$. The algorithm has $\log T$ levels of recursion. In the first level we decide on the occurrences of the tasks in each half of the period. That is, for each task we decide how many of its occurrences appear in the first half of the period and how many in the second half. This yields a problem of a recursive nature in the two halves. In order to find the schedule at time t, it suffices to solve the problem recursively in the half which contains t. (Note that in case T is odd one of the halves is longer than the other.) Clearly, if a task has an even number of occurrences in T it would appear the same number of times in each half in order to minimize the drift. The problem is with tasks that have an odd number of occurrences s_i. Clearly, each half should have at least $\lfloor s_i \rfloor$ of the occurrences. The additional occurrence has to be assigned to one of the halves in a way that both resulting sub-problems would still be feasible. This is the main difficulty of the assignment and is solved in the procedure *Sweep*.

Procedure Sweep: In this procedure we compute the assignment of the additional occurrence for all tasks that have an odd number of occurrences. The input to this procedure is a set of intervals $I_1, \ldots, I_m$ (those having odd s_i's) with the restriction that each clique in the resulting interval sub-graph is of even size. (Later, we show how to overcome this restriction.) The output is a partition of these intervals into two sets such that each clique is equally divided among the sets. This is done by a sweep along the x-coordinate of the intervals. During the sweep every interval will be assigned a variable which at the end is set to 0 or 1 (i.e., first half of the period or second half of the period). Suppose that we sweep point x. We say that an interval I_i is *active* while we sweep point x if $x \in I_i$. The assignment rules are as follows:

For each interval I_i that starts at x:

If the current number of active intervals is even:

A new variable is assigned to I_i (I_i is *unpaired*).

Otherwise; the current number of active intervals is odd:

I_i is paired to the currently unpaired interval I_j, and it is assigned the negation of I_j's variable.

Comment: No matter what value is later assigned to this variable, I_i and I_j will end up in opposite halves.

For each interval I_i that ends at x:

If the current number of active intervals is even:

Nothing is done.

Otherwise; the current number of active intervals is odd:

If I_i is paired with I_j:

I_j is now paired with the currently unpaired interval I_k. Also, I_j's variable is matched with the negation of I_k's variable.

Comment: This will ensure that I_j and I_k are put in opposite halves, or equivalently, I_i and I_k are put in the same halves.

If I_i is unpaired:

Assign arbitrarily 0 or 1 to I_i's variable.

It will be proven later that these rules ensure that whenever the number of active intervals is even, then exactly half of the intervals will be assigned 0 and half will be assigned 1. We note that since the conflict graph is an interval graph we are assured that when we apply the above rules pairing up arbitrary intervals will not result in a circular dependency of the variables (e.g., $x = y = \bar{x}$).

Recall that we assumed that the size of each clique is even. To overcome this restriction we need the following simple lemma. For $x \in [0, 1]$, denote by C_x the set of all the input intervals (with odd and even s_i's) that contain x; C_x will be referred to as a clique.

Lemma 18 *The period T is even if and only if $|\{i \ : \ I_i \in C \wedge s_i \text{ is odd}\}|$ is even for every clique C.*

Proof: Note that $\sum_{I_i \in C} f_i = 1 \Rightarrow \sum_{I_i \in C} s_i = T \Rightarrow \sum_{I_i \in C,\ s_i \text{ is odd}} s_i + \sum_{I_i \in C,\ s_i \text{ is even}} s_i = T$. Since the second summand is always even, T is even if and only if the first summand is also even. □

This lemma implies that if T is even then the size of each clique in the input to procedure *Sweep* is indeed even. If T is odd, then a dummy interval I_{n+1} which extends over all other intervals and which has exactly one occurrence is added to the set I before calling *Sweep*. Again, by Lemma 18, we are sure that in this modified set I the size of each clique is even. This would increase the period by one. The additional time unit will be allotted only to the dummy interval and thus can be ignored. We note that to produce the schedule at time t we

just have to follow the recursive calls that include t in their period.

Applying this algorithm to the max-min frequencies yields a polynomial in n algorithm. This is true because there are no more than $\log T$ such calls and because $T = 2^{O(n)}$ for max-min fair frequencies.

Lemma 19 *The algorithm produces a correct schedule for every feasible set of frequencies.*

Proof: We need to prove that feasibility is maintained with every recursive step. We show that the following invariant is maintained by *Sweep*:

> For every x for which the number of active intervals in C_x is even, exactly half of the intervals will be assigned 0 and half will be assigned 1.

This invariant is easily maintained when a new interval starts: If the current number of intervals is odd, then the new interval is paired up with the currently unpaired interval, and thus will be scheduled in the opposite half of its partner. The invariant holds also when an interval ends since by our rules whenever an interval ends any two unpaired interval are immediately paired up.

Now, if T is odd, then a dummy interval is added and hence *sweep* produces a feasible solution for $T + 1$. In this case the algorithm assigns the "smaller" half of T to the half to which *Sweep* assigned the dummy interval and feasibility is maintained. □

Lemma 20 *If the set of frequencies is of the form $1/2^i$ then the resulting schedule is P-fair. (i.e., the drift can be bounded by 1) and the response time is bounded by $\lceil 2/f_i \rceil$.*

Proof: Since our algorithm always divides even s_i into equal halves, the following invariant is maintained: In recursion level j, if $s_i > 1$ then s_i is even. Also note that $T = 2^k$, where $\min_i f_i = 1/2^k$ and thus each s_i is of the form $2^{\psi_i - k}$. Now, following the algorithm, it can be easily shown that there is at least one occurrence of task i in each time interval of size $2^{k-\psi_i}$. This implies that $\lfloor \frac{t}{2^{k-\psi_i}} \rfloor \leq \sum_{r=1}^{t} S(i,r) \leq \lceil \frac{t}{2^{k-\psi_i}} \rceil$ and thus P-fairness follows. Since the drift is bounded by one the response time is bounded by $\lceil 2/f_i \rceil$. □

Lemma 21 *The response time for every interval I_i is bounded by $\lceil 4/f_i \rceil$.*

Proof: Lemma 20 implies the case in which the frequencies are powers of two. Moreover, in case the frequencies are not powers of two, we can virtually partition each task into two tasks with frequencies a_i and b_i respectively, so that $f_i = a_i + b_i$, a_i is a power of two, and $b_i < a_i$. Then, the schedule of the task with frequency a_i has drift 1. This implies that its response time is at most $\lceil 2/a_i \rceil \leq \lceil 4/f_i \rceil$. □

Remark: It can be shown that the bound of the above lemma is tight for our algorithm.

We summarize the results in this section in the following theorem:

Theorem 22 *Given an arbitrary interval graph as a conflict graph, the algorithm exactly realizes any feasible frequency vector and guarantees that the response time is at most $\lceil 4/f_i \rceil$.*

4.1 Bounding the drift

Since the algorithm has $O(\log T)$ levels of recursion and each level may increase the drift by one, it follows that the maximum drift is bounded by $O(\log T)$. In this section we prove that we can decrease the maximum drift to be $O(\sqrt{\log T} n^{\epsilon})$, for any fixed ϵ, where n is the number of tasks. By Lemma 13 this implies that in the worst case the drift for max-min fair frequencies is bounded by $O(n^{\frac{1}{2}+\epsilon})$.

Our method to get a better drift is based on the following observation: At each recursive step of the algorithm two sets of tasks are produced such that each set has to be placed in a different half of the time-interval currently considered. However, we are free to choose which set goes to which half. We use this degree of freedom to decrease the bound on the drift. To make the presentation clearer we assume that T is a power of two and that the time units are $0, \ldots, T-1$. The arguments can be modified to hold in the general case.

Consider a sub-interval of size $T/2^j$ starting *after* time $t_\ell = i \cdot T/2^j - 1$ and ending at $t_r = (i+1) \cdot T/2^j - 1$, for $0 \leq i \leq 2^j - 1$. In the first j recursion levels we already fixed the number of occurrences of each task up to t_ℓ. Given this number, the drift d_ℓ at time t_ℓ is fixed. Similarly, the drift d_r at time t_r is also fixed. At the next recursion level we split the occurrences assigned to the interval $[t_\ell + 1, t_r]$, and thus fixing the drift d_m at time $t_m = (t_\ell + t_r)/2$. Optimally, we would like the drifts after the next recursion level at each time unit $t \in [t_\ell + 1, t_r]$ to be the weighted average of the drifts d_ℓ and d_r. In other words, let $\alpha = (t - t_\ell)/(t_r - t_\ell)$, then, we would like the drift at time t to be $\alpha d_r + (1-\alpha) d_\ell$. In particular, we would like the drift at t_m to be $(d_\ell + d_r)/2$. This drift can be achieved for t_m only if the occurrences in the interval $[t_\ell + 1, t_r]$ can be split equally. However, in case we have an odd number of occurrences to split, the drift at t_m is $(d_\ell + d_r)/2 \pm 1/2$, depending on our decision in which half interval to put the extra occurrence. Note that the weighted average of the drifts of all other points changes accordingly. That is, if the new d_m is $(d_\ell + d_r)/2 + x$, for $x \in \{\pm 1/2\}$, then the weighted average in $t \in [t_\ell + 1, (t_r + t_\ell)/2]$ is $\alpha d_r + (1-\alpha) d_\ell + 2\alpha x$, where $\alpha = (t - t_\ell)/(t_r - t_\ell) \leq 1/2$, and the weighted average in $t \in [(t_r + t_\ell)/2 + 1, t_r]$ is $\alpha d_r + (1-\alpha) d_\ell + 2(1-\alpha)x$, where $\alpha = (t - t_\ell)/(t_r - t_\ell) > 1/2$.

Consider now the two sets of tasks S_1 and S_2 that we have to assign to the two sub-intervals (of the same size) at level k of the recursion. For each of the possible two assignments, we compute a "potential" based on the resulting drifts at time t_m. For a given possibility let $D[t_m, i, k]$ denote the resulting drift of task i at t_m after k recursion levels. Define the potential of t_m after k levels as $POT(t_m, k) = \sum_{i=1}^{n} D(t_m, i, j)^{\phi}$, for some fixed even constant ϕ. We choose the possibility with the lowest potential. We now prove that using this policy the drift of any task after $\log T$ steps is bounded by $O(\sqrt{\log T} \cdot n^{\frac{1}{\phi}})$.

Consider a time t and a task i. The drift of task i at t is the outcome of at most $\log T$ recursion levels. Define the drift of task i at t after k levels, denoted $D(t, i, k)$, as the weighted average drift at t given the fixed drifts after k levels. It is easy to see that the initial drift is zero, and the final weighted average drift is the actual drift at t. Also, in each level the drift may either stay the same (in case we have to split an even number of occurrences of task i), or is changed by $\pm x$ where $0 \leq x \leq 1/2$. Note that x is positive if and only if the change in the

drift at the current median point closest to t is $+1/2$. We extend the definition of potentials to all time points t in the obvious way; that is, $POT(t,k) = \sum_{i=1}^{n} D(t,i,k)^{\phi}$. We show that the potential after $\log T$ levels is bounded by $O(T^{\phi/2} \cdot n)$. This implies the desired bound on the drift of each task at t since the potential is the sum of the drifts to the power of ϕ.

Lemma 23 *For all $0 \leq t \leq T-1$, and all $1 \leq k \leq \log T$,*

$$POT(t,k) \leq POT(t,k-1) + c \cdot \sum_{i=1}^{n} D(t,i,k-1)^{\phi-2}$$

for some constant c.

Proof: The increment of the potential at time t at the k-th level is bounded by the maximum over all disjoint sets $S_1, S_2 \subset \{1,\ldots,n\}$, such that $|S_1| = |S_2|$ of

$$\begin{aligned} \min_{S_1,S_2} \quad & \left\{ \sum_{i \in S_1} [D(t,i,k-1)+x]^{\phi} + \sum_{i \in S_2} [D(t,i,k-1)-x]^{\phi}, \right. \\ & \left. \sum_{i \in S_2} [D(t,i,k-1)+x]^{\phi} + \sum_{i \in S_1} [D(t,i,k-1)-x]^{\phi} \right\} \\ - \quad & \sum_{i \in S_1 \cup S_2} [D(t,i,k-1)]^{\phi} \end{aligned}$$

for some $0 \leq x \leq 1/2$. Since the minimum is always bounded by the average, the change is bounded by

$$\frac{1}{2} \left\{ \sum_{i \in S_1 \cup S_2} [D(t,i,k-1)+x]^{\phi} + [D(t,i,k-1)-x]^{\phi} - 2[D(t,i,k-1)]^{\phi} \right\}.$$

Finally, the maximum over all disjoint sets $S_1, S_2 \subset \{1,\ldots,n\}$, such that $|S_1| = |S_2|$ is achieved for $S_1 \cup S_2 = \{1,\ldots,n\}$, and it is $O(\sum_{i=1}^{n} [D(t,i,k-1)+x]^{\phi-2})$. □

Lemma 24 *For all $0 \leq t \leq T-1$, and all $0 \leq k \leq \log T$,*

$$\sum_{i=1}^{n} [D(t,i,k)]^{\phi-2} \leq (c \cdot \log T)^{\frac{\phi}{2}-1} \cdot n$$

where c is the constant of Lemma 23.

Proof: To obtain a contradiction assume that there exists $0 \leq t \leq T-1$ and $0 < k \leq \log T$ for which the bound does not hold. Consider the minimum such k. By Lemma 23 and the minimality of k, we get that By Lemma 23, we get that $POT(t,k) \leq POT(t,k-1) + c \cdot \sum_{i=1}^{n} D(t,i,k-1)^{\phi-2}$. Reapplying Lemma 23 and since the function $D(t,i,k)$ is increasing in k we get $POT(t,k) \leq kc \cdot \sum_{i=1}^{n} D(t,i,k-1)^{\phi-2}$. Finally by the minimality of k, $POT(t,k) \leq kc \cdot (c \cdot \log T)^{\frac{\phi}{2}-1} \cdot n = c^{\frac{\phi}{2}} \cdot (\log T)^{\frac{\phi}{2}-1} \cdot n \cdot k$. By our definition $POT(t,k) = \sum_{i=1}^{n} D(t,i,k)^{\phi}$. By Hölder inequality

$$\sum_{i=1}^{n} D(t,i,k)^{\phi} \geq n \left(\frac{\sum_{i=1}^{n} D(t,i,k)^{\phi-2}}{n} \right)^{\frac{\phi}{\phi-2}}.$$

However, by our assumption

$$n\left(\frac{\sum_{i=1}^{n} D(t,i,k)^{\phi-2}}{n}\right)^{\frac{\phi}{\phi-2}} > n\left(\frac{(c\cdot\log T)^{\frac{\phi}{2}-1}n}{n}\right)^{\frac{\phi}{\phi-2}} = (c\cdot\log T)^{\frac{\phi}{2}}\cdot n.$$

Combining the two inequalities we get

$$(\log T)^{\frac{\phi}{2}} \quad < \quad (\log T)^{\frac{\phi}{2}-1}\cdot k$$

But this inequality implies that $k > \log T$; a contradiction. □

Theorem 25 *The maximum drift is bounded by $O(\sqrt{\log T}\cdot n^{\epsilon})$, for any fixed ϵ.*

Proof: By Lemmas 23 and 24, the potential $POT(t, \log T)$, for all $1 \le t \le T$, is bounded by $\log T \cdot O((\log T)^{\frac{\phi}{2}-1}\cdot n) = O((\log T)^{\frac{\phi}{2}}\cdot n)$. This implies the bound on each drift, since the potential is the sum of the drifts to the power of ϕ. The constant ϵ is chosen to be $\frac{1}{\phi}$. □

5 Realizing frequencies rigidly

In this section we show how to construct a schedule that 12-approximates any feasible frequency vector in a *rigid* fashion on an interval graph. We reduce our Rigid Schedule problem to the Dynamic Storage Allocation problem. The Dynamic Storage Allocation problem is defined as follows. We are given objects to be stored in a computer memory. Each object has two parameters: (i) its size; that is, the number of cells needed to store it, and (ii) the time interval in which it should be stored. Each object must be stored in adjacent cells. The problem is to find the minimal size memory that can accommodate at any given time all of the objects that are needed to be stored at that time. The Dynamic Storage Allocation problem is a special case of the multi-coloring problem on interval graphs defined below.

A *multi-coloring* of a weighted graph G with the weight function $w : V \to \mathcal{N}$, is a function $F : V \to 2^{\mathcal{N}}$ such that (i) for all $v \in V$ the size of $F(v)$ is $w(v)$, and (ii) if $(v,u) \in E$ then $F(v) \cap F(u) = \emptyset$. The multi-coloring problem is to find a multi-coloring with minimal number of colors. This problem is known to be an NP-Hard problem [GJ79].

Two interesting special cases of the Multi-Coloring problem are when the colors of a vertex must be either contiguous or "spread well" among all colors. We call the first case the Cont-MC problem and the second case the Spread-MC problem. More formally, in a solution to Cont-MC if $F(u) = \{x_1 < \cdots < x_k\}$, then $x_{i+1} = x_i + 1$ for all $1 \le i < k$. Whereas in a solution to Spread-MC that uses T colors, if $F(u) = \{x_1 < \cdots < x_k\}$ then (i) k divides T, and (ii) $x_{i+1} = x_i + T/k$, for all $1 \le i < k$, and $x_k + T/k - T = x_1$.

It is not hard to verify that for interval graphs the Cont-MC problem is equivalent to the Dynamic Storage Allocation problem described above. Simply associate each object with a vertex in the graph and give it a weight equal to the number of cells it requires. Put an edge between two vertices if their time intervals intersect. The colors assigned to a vertex are interpreted as the cells in which the object is stored.

On the other hand, the Spread-MC problem corresponds to the Rigid Schedule problem as follows. First, we replace the frequency $f(v)$ by a weight $w(v)$. Let $k(v) = \lceil -\log_2 f(v) \rceil$, and let $k = \max_{v \in V}\{k(v)\}$, then $w(v) = 2^{k-k(v)}$. Clearly, $f(v)/2 \leq w(v)/2^k \leq f(v)$. Now, assume that the output for the Spread-MC problem uses T colors and let the colors of v be $\{x_1 < \cdots < x_k\}$ where $x_2 - x_1 = \Delta$. We interpret this as follows: v is scheduled in times $x_1 + i\Delta$ for all $i \geq 0$. It is not difficult to verify that the resulting schedule is rigid and it 2-approximates the given frequencies.

Although the Dynamic Storage Allocation problem is a special case of the multi-coloring problem it is still known to be an NP-Hard problem [GJ79]. Using similar arguments it can be shown that the Rigid Scheduling problem is also NP-Hard. Therefore, we are looking for an approximation algorithm. In what follows we present an approximation algorithm that produces a rigid scheduling that 12-approximates the given frequencies. For this we consider instances of the Cont-MC and Spread-MC problems in which the input weights are powers of two.

Definition 26 *A solution for an instance of* Cont-MC *is both* aligned *and* contiguous *if for all* $v \in V$, $F(v) = \{j \cdot w(v), \ldots, (j+1) \cdot w(v) - 1\}$ *for some* $j \geq 0$.

In [Kie91], Kierstead presents an algorithm for Cont-MC that has an approximation factor 3. A careful inspection of this algorithm shows that it produces solutions that are both aligned and contiguous for all instances in which the weights are power of two.

We show how to translate a solution for such an instance of the Cont-MC problem that is both aligned and contiguous into a solution for an instance of the Spread-MC problem with the same input weights.

For $0 \leq x < 2^k$, let $\pi(x)$ be the k-bit number the binary representation of which is the inverse of the binary representation of x.

Proposition 27 *For* $1 \leq i \leq k$ *and* $0 \leq j < 2^{k-i} = \Delta$, $\{\pi(j2^i), \ldots, \pi(j2^i + 2^i - 1)\} = \{\pi(j2^i), \pi(j2^i) + \Delta, \ldots, \pi(j2^i) + (2^i - 1)\Delta\}$.

This proposition says that an output of Cont-MC that uses c colors can be transformed into an output of Spread-MC that uses at most $2c$ colors.

Consider an instance of the Spread-MC problem in which all the input weights are powers of two. Apply the solution of Kierstead [Kie91] to solve the Cont-MC instance with the same input. This solution is both aligned and contiguous, and uses at most $3T'$ colors where T' is the number of colors needed by an optimal coloring. Let $T \geq 3T'$ be the smallest power of 2 that is greater than T'. It follows that $T \leq 6T'$. Applying the transformation of Proposition 27 on the output of the solution to Cont-MC yields a solution to Spread-MC with at most T colors. This in turn, yields an approximation factor of at most 12 for the Rigid Scheduling problem, since $w(v)/T \geq f(v)/2$.

Theorem 28 *The above algorithm computes a rigid schedule that 12-approximates any feasible frequency vector on an interval graph.*

6 Circular-Arc graphs

In this section we show how to transform any algorithm $\mathcal{A}$ for computing a schedule that c-approximates any given feasible frequency vector on interval graphs into an algorithm $\mathcal{A}'$ for computing a schedule that $2c$-approximates any given feasible frequencies on circular-arc graphs.

Let $\hat{f}$ be a feasible frequency vector on a circular-arc graph G.

Step 1: Find the maximum clique C in G.

Let $G' = G - C$. Note that G' is an interval graph. Let $\hat{g_1}$ and $\hat{g_2}$ be the frequency vectors resulting from restricting $\hat{f}$ to the vertices of G' and C, respectively. Note that $\hat{g_1}$ and $\hat{g_2}$ are feasible on G' and C, respectively.

Step 2: Using $\mathcal{A}$, find schedules S_1 and S_2 that c-approximate $\hat{g_1}$ and $\hat{g_2}$ on G' and C, respectively.

Step 3: Interleave S_1 and S_2.

Clearly, the resulting schedule $2c$-approximates $\hat{f}$ on the circular-arc graph G. Note also that all the three steps can be computed in polynomial time.

7 Conclusions and future research

In this paper we have introduced a new scheduling problem. It is characterized by the persistence and interdependency of the tasks involved. We have developed new measures that quantify the fairness and regularity of a schedule. We have shown that every conflict graph has a unique max-min fair frequency assignment and that, in general, this assignment is hard even to approximate. However, for perfect graphs, it turns out that max-min fair frequencies are easy to compute and we have given an algorithm for this purpose. The scheduling algorithms described in this paper exhibit a trade-off between the accuracy with which given frequencies are realized and their regularity. Furthermore, we have shown that a drift of one (i.e., P-fairness) is not achievable even for simple interval conflict graphs. This can be viewed as an indication that the problem in this paper is inherently more complex than the one considered in [BCPV96].

Many open problems remain. The exact complexity of computing a max-min fair frequency assignment in general graphs is not known and there is no characterization of when such an assignment is easy to compute. All the scheduling algorithms in the paper use the inherent linearity of interval or circular-arc graphs. It would be interesting to find scheduling algorithms for the wider class of perfect graphs. The algorithm for interval graphs that realizes frequencies exactly exhibits a considerable gap in its drift. It is not clear from which direction this gap can be closed.

Our algorithms assume a central scheduler that makes all the decisions. Both from theoretical and practical point of view it is important to design scheduling algorithms working in

more realistic environments such as high-speed local-area networks and wireless networks (as mentioned in Section 1.1). The distinguishing requirements in such an environment include a distributed implementation via a local signaling scheme, a conflict graph which may change with time, and restrictions on space per node and size of a signal. The performance measures and general setting, however, remain the same. A first step towards such algorithms has been recently carried out by Mayer, Ofek, and Yung in [MOY96].

Acknowledgment

We would like to thank Don Coppersmith and Moti Yung for many useful discussions.

References

[AS90] B. Awerbuch and M. Saks, A Dining Philosophers Algorithm with Polynomial Response Time. *Proc. 31st IEEE Symp. on Foundations of Computer Science* (1990), 65–75.

[BCPV96] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, Proportionate Progress: A Notion of Fairness in Resource Allocation. *Algorithmica*, 15(6):600–625 (1996).

[BG87] D. Bertsekas and R. Gallager, Data Networks. *Prentice Hall* (1987).

[BP92] J. Bar-Ilan and D. Peleg, Distributed Resource Allocation Algorithms. *Proc. 6th International Workshop on Distributed Algorithms* (1992), 277–291.

[CCO93] J. Chen, I. Cidon, and Y. Ofek, A Local Fairness Algorithm for Gigabit LANs/MANs with Spatial Reuse. *IEEE J. on Selected Areas in Communication*, 11(8):1183–1192 (1993).

[CM84] K. Chandy and J. Misra, The Drinking Philosophers Problem. *ACM Trans. on Programming Languages and Systems*, 6:632–646 (1984).

[CO93] I. Cidon and Y. Ofek, MetaRing – A Full-Duplex Ring with Fairness and Spatial Reuse. *IEEE Trans. on Communications*, 41(1):110–120 (1993).

[CS92] M. Choy and A. Singh, Efficient Fault Tolerant Algorithms for Resource Allocation in Distributed System. *Proc. 24th ACM Symp. on Theory of Computing* (1992), 593–602.

[Dijk71] E. W. Dijkstra, Hierarchical Ordering of Sequential Processes. *Acta Informatica*, 1:115–138 (1971).

[GJ79] M. Garey and D. Johnson, Computers and Intractability, a Guide to the Theory of NP-Completeness, *W. H. Freeman*, San Francisco, 1979.

[Gol80] M. Golumbic, Algorithmic Graph Theory and Perfect Graphs. *Academic Press*, New York, 1980.

[GLS87] M. Grötschel, L. Lóvasz and A.Schrijver, Geometric Algorithms and Combinatorial Optimization. *Springer-Verlag*, Berlin, 1987.

[Goo90] D. J. Goodman, Cellular Packet Communications. *IEEE Trans. on Communications*, 38:1272–1280 (1990).

[Jaf81] J. Jaffe, Bottleneck Flow Control. *IEEE Trans. on Communications*, 29(7):954–962 (1981).

[Kie91] H. A. Kierstead, A Polynomial Time Approximation Algorithm for Dynamic Storage Allocation. *Discrete Mathematics*, 88:231–237 (1991).

[Knu94] D. E. Knuth, The Sandwich Theorem, *The Electronic Journal of Combinatorics*, 1:1–48 (1994).

[LL73] C. L. Liu and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61 (1973).

[LY93] C. Lund and M. Yannakakis, On the Hardness of Approximating Minimization Problems. *Proc. 25th ACM Symp. on Theory of Computing* (1993), 286–293.

[Lyn80] N. Lynch, Fast Allocation of Nearby Resources in a Distributed System. *Proc. 12th ACM Symp. on Theory of Computing* (1980), 70–81.

[MOY96] A. Mayer, Y. Ofek, and M. Yung, Local Scheduling with Partial State Information for Approximate Max-min Fair Rates. *Proc. IEEE INFOCOM'96* (1996).

[Tuc71] A. Tucker, Matrix characterizations of circular-arc graphs. *Pacific Journal of Mathematics*, 39:535–545, (1971).

A The partial order $\prec$

In this appendix we prove that the relation $\prec$ is a partial order. We first observe that the definition can be restated as $\hat{f} \prec \hat{g}$ if there exists an index i and a threshold f such that $f_i < f \leq g_i$ (the *index* property), and for all $1 \leq j \leq n$, $g_j \geq \min\{f, f_j\}$ (the *threshold* property). The following two claims establish that $\prec$ is a partial order.

Claim 29 *The relation $\prec$ is anti-symmetric.*

Proof: To obtain a contradiction assume that there exist two vectors $\hat{f}$ and $\hat{g}$ such that $\hat{f} \prec \hat{g}$ and $\hat{g} \prec \hat{f}$. This implies that there exist two indices i and ℓ and two thresholds f and g such that:

1. $f_i < f \leq g_i$, and for all $1 \leq j \leq n$, $g_j \geq \min\{f, f_j\}$.
2. $g_\ell < g \leq f_\ell$, and for all $1 \leq j \leq n$, $f_j \geq \min\{g, g_j\}$.

Since $g_\ell \geq \min\{f, f_\ell\}$, $f_\ell > g_\ell$, and $g > g_\ell$, it follows that $g > f$. Similarly, since $f_i \geq \min\{g, g_i\}$, $g_i > f_i$, and $f > f_i$, it follows that $f > g$. We get the contradiction. □

Claim 30 *The relation $\prec$ is transitive.*

Proof: Suppose that $\hat{f} \prec \hat{g}$ and $\hat{g} \prec \hat{h}$. We show that $\hat{f} \prec \hat{h}$. Since $\hat{f} \prec \hat{g}$ and $\hat{g} \prec \hat{h}$ there exist two indices i and ℓ and two thresholds f and g such that:

1. $f_i < f \leq g_i$, and for all $1 \leq j \leq n$, $g_j \geq \min\{f, f_j\}$.
2. $g_\ell < g \leq h_\ell$, and for all $1 \leq j \leq n$, $h_j \geq \min\{g, g_j\}$.

We choose $h = \min\{f, g\}$ as the threshold for $\hat{f} \prec \hat{h}$. Now, for all $1 \leq j \leq n$,

$$h_j \geq \min\{g, g_j\} \geq \min\{g, \min\{f, f_j\}\} \geq \min\{\min\{f, g\}, f_j\} \geq \min\{h, f_j\} \ .$$

We still have to prove that there exists an index with the desired property. Assume first that $h = f \leq g$, then we choose i as the index and we need to show that $f_i < h \leq h_i$. Since $h = f$ it follows that $f_i < h$. Since $h_i \geq \min\{g_i, g\}$, $h \leq g$, and $g_i \geq f = h$, it follows that $h_i \geq h$. Now assume that $h = g < f$, then we choose ℓ as the index. Here we need to show that $f_\ell < h \leq h_\ell$. Since $g \leq h_\ell$ it follows that $h \leq h_\ell$. Since $g_\ell \geq \min\{f, f_\ell\}$, $g > g_\ell$, and $h = g < f$, it follows that $h > f_\ell$. □

B The complete proof of Theorem 7

We complete the proof of Theorem 5 for the case when $\hat{f}$ is not a rational convex combination.

Claim 31 *If a frequency vector $\hat{f}$ can be expressed as a convex combination of the independent sets, then $\hat{f}$ is feasible.*

Proof: Suppose that there exist weights $\{\alpha_I\}_{I\in\mathcal{I}}$, such that $\sum_{I\in\mathcal{I}}\alpha_I = 1$ and $\sum_{I\in\mathcal{I}}\alpha_I\chi(I) = \hat{f}$. We show how to obtain a schedule S that realizes the frequency vector $\hat{f}$. For every $k < \infty$, we pick $g_i^{(k)}$ to be a rational number between $f_i - 2^{-k}$ and f_i, and apply Claim 7 to construct a schedule A_k of finite length, denoted $T(A_k)$, that realizes the frequency vector $g^{\hat{(k)}}$.

We go on to construct schedules $S_1, S_2, \ldots, S_k$ with the following properties.

Property 1: Schedule S_k has finite length $T(S_k)$.

Property 2: For each task $1 \le i \le n$, schedule S_k achieves a frequency of at least $f_i - 2^{-(k-1)}$ for task i.

Property 3: Schedule S_{k-1} is a prefix of schedule S_k.

Property 4: In the infinite schedule $S_kS_kS_k\cdots$ (i.e., the schedule given by concatenating the schedule S_k infinitely many times), for any task $1 \le i \le n$ and time $t > T(S_k)$, $f_i^{(t)}(S_kS_kS_k\cdots) \ge f_i - 2^{-(k-2)}$. (Recall that $f_i^{(t)}(S)$ is the prefix frequency of task i at time t in schedule S.)

Property 5: For any task $1 \le i \le n$ and time $t > T(S_{k-1})$, $f_i^{(t)}(S_k) \ge f_i - 2^{-(k-3)}$.

We construct the S_k's inductively. The base case S_1 exists trivially (every non-empty schedule satisfies the required properties). Assume the schedules $S_1, \ldots, S_{k-1}$ exist. Schedule S_k is given by the concatenation of n_1 schedulings of S_{k-1} followed by n_2 schedulings of A_k. We now show that under an appropriate choice of n_1 and n_2, the schedule S_k satisfies the above properties. Let D_i be the maximum among the drift of task i in the schedule S_{k-1} and the drift of task i in the schedule A_k. Let $D = \max_i\{D_i\}$. Let

$$n_1 = \left\lceil \frac{2^k D}{4T(S_{k-1})} \right\rceil \text{ and } n_2 = \left\lceil \frac{2n_1T(S_{k-1})}{T(A_k)} \right\rceil.$$

Property 1: The period of S_k is

$$T(S_k) = n_1T(S_{k-1}) + n_2T(A_k)$$

which is finite since n_1 and n_2 are finite.

Property 2: The frequency of task i in S_k is at least

$$\frac{n_1 T(S_{k-1})(f_i - 2^{-(k-2)}) + n_2 T(A_k)(f_i - 2^{-k})}{n_1 T(S_{k-1}) + n_2 T(A_k)} =$$

$$f_i - 2^{-(k-1)} - \frac{n_1 T(S_{k-1}) 2^{-(k-1)} - n_2 T(A_k) 2^{-k}}{n_1 T(S_{k-1}) + n_2 T(A_k)}.$$

We wish to show that the above quantity is at least $f_i - 2^{-(k-1)}$. This simplifies to

$$n_2 \geq \frac{2 n_1 T(S_{k-1})}{T(S_k)},$$

a condition which is satisfied by our choice of n_1 and n_2.

Property 3: Since $n_1 \geq 1$, it follows that S_{k-1} is a prefix of S_k.

Property 4: Since S_{k-1} is a prefix of S_k it follows that in the infinite schedule $S_k S_k S_k \cdots$, for any task $1 \leq i \leq n$ and time $t > T(S_k)$,

$$\begin{aligned} f_i^{(t)}(S_k S_k S_k \cdots) &= (f_i - 2^{-(k-1)}) T(S_k) + (f_i - 2^{-(k-2)})[t - T(S_k)] - D_i \\ &= (f_i - 2^{-(k-2)}) t + 2^{-(k-1)} T(S_k) - D_i. \end{aligned}$$

We wish to show that this is at least $t(f_i - 2^{-(k-2)})$. This condition simplifies to

$$2^{k-1} D_i \leq T(S_k) = n_1 T(S_{k-1}) + n_2 T(S_k).$$

Once again, the choice of n_1 and n_2 satisfies this condition.

Property 5: For any task $1 \leq i \leq n$ and time $T(S_{k-1}) < t \leq n_1 T(S_{k-1})$, Property 4 of schedule S_{k-1} guarantees that $f_i^{(t)}(S_k) \geq f_i - 2^{-(k-3)}$. Now, consider $t > n_1 T(S_{k-1})$. The number of times a task i is scheduled in S_k by time t is at least

$$(f - 2^{-(k-2)}) n_1 T(S_{k-1}) + (f - 2^{-k})(t - n_1 T(S_{k-1})) - D_i =$$

$$(f - 2^{-(k-3)}) t + 2^{-(k-2)} n_1 T(S_{k-1}) + 7 \cdot 2^{-k} (t - n_1 T(S_{k-1})) - D_i.$$

We wish to show that this quantity is at least $t(f - 2^{-(k-3)})$. This inequality is implied by the condition $4 n_1 T(S_{k-1}) \geq 2^k D_i$, which is satisfied by the choice of n_1.

We use the sequences $S_1, \ldots, S_k, \ldots$ to define an infinite sequence S (which is essentially the limiting element of the sequence $\{S_i\}$). To determine which independent set to schedule at time t in S, we let k be the smallest index such that $T(S_k) \geq t$. We schedule the independent set scheduled by S_k at time t.

To see that S realizes the desired frequency vector $\hat{f}$, we prove that for every $\epsilon > 0$, there exists $T < \infty$, such that for all $t \geq T$ and for all tasks $1 \leq i \leq n$, $f_i^{(t)}(S) \geq f_i - \epsilon$.

Given $\epsilon > 0$, let k be the minimum integer such that $2^{-(k-2)} \leq \epsilon$ and let $T = T(S_k) + 1$. Given $t \geq T$, let k' be the largest index such that $t > T(S_{k'})$. Clearly, $k' \geq k$. Observe that for any $j < \infty$, S_j is a prefix of S. Thus, the prefix of schedule S up to time t is a prefix of $S_{k'+1}$. By Property 5 of $S_{k'+1}$, $f_i^{(t)}(S_{k'+1}) \geq f_i - 2^{-(k'-2)} \geq f_i - 2^{-(k-2)} \geq f_i - \epsilon$. □