Polyamorous Scheduling

Leszek Gąsieniec ✉ 🔗
University of Liverpool, UK

Benjamin Smith ✉ 🔗
University of Liverpool, UK

Sebastian Wild ✉ 🔗
University of Liverpool, UK

──── **Abstract** ────

Finding schedules for pairwise meetings between the members of a complex social group without creating interpersonal conflict is challenging, especially when different relationships have different needs. We formally define and study the underlying optimisation problem: Polyamorous Scheduling.

In Polyamorous Scheduling, we are given an edge-weighted graph and try to find a periodic schedule of matchings in this graph such that the maximal weighted waiting time between consecutive occurrences of the same edge is minimised. We show that the problem is NP-hard and that there is no efficient approximation algorithm with a better ratio than 4/3 unless P = NP. On the positive side, we obtain an $O(\log n)$-approximation algorithm; indeed, a $O(\log \Delta)$-approximation for Δ the maximum degree, i.e., the largest number of relationships of any individual. We also define a generalisation of density from the Pinwheel Scheduling Problem, "poly density", and ask whether there exists a poly-density threshold similar to the 5/6-density threshold for Pinwheel Scheduling [Kawamura, STOC 2024]. Polyamorous Scheduling is a natural generalisation of Pinwheel Scheduling with respect to its optimisation variant, Bamboo Garden Trimming.

Our work contributes the first nontrivial hardness-of-approximation reduction for any periodic scheduling problem, and opens up numerous avenues for further study of Polyamorous Scheduling.

1 Introduction

We study a natural periodic scheduling problem faced by groups of regularity-loving polyamorous people: Consider a set of persons and a set of pairwise relationships between them, each with a value representing its neediness, importance, or emotional weight. Find a periodic schedule of pairwise meetings between couples that minimizes the maximal weighted waiting time between such meetings, given that each person can meet with at most one of their partners on any particular day.

Before formally defining the Polyamorous Scheduling Problem (Poly Scheduling for short), we illustrate some features of the problem on an example. Figure 1 shows an instance using the natural graph-based representation: We have vertices for people and weighted (undirected) edges for relationships. It is easy to check that the schedule given at the bottom of Figure 1 never schedules more than one daily meeting for any of the 8 persons in the group; in the graph representation, the set of meetings for each day must form a matching.
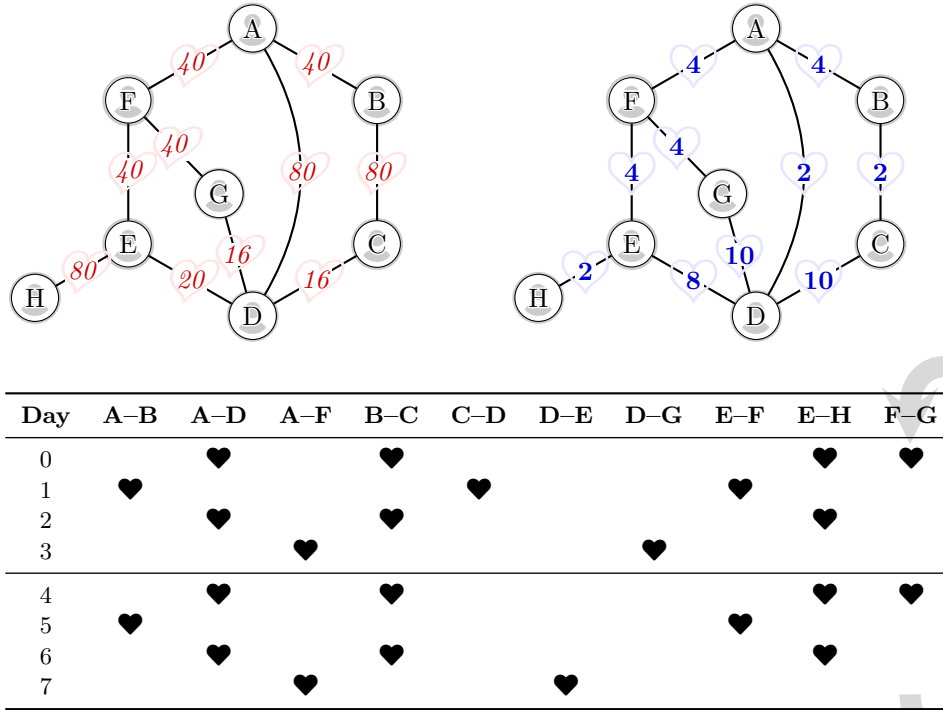
Day	A–B	A–D	A–F	B–C	C–D	D–E	D–G	E–F	E–H	F–G
0		♥		♥					♥	♥
1	♥			♥				♥	♥	
2		♥		♥					♥	
3			♥				♥			
4		♥		♥					♥	♥
5	♥							♥	♥	
6		♥		♥					♥	
7			♥			♥				

■ **Figure 1** An example Optimisation Polyamorous Scheduling instance with 8 persons: Alex, Brady, Charlie, Daisy, Eli, Frankie, Grace, and Holly. **Top left:** Graph representation with *edge labels* showing the weight (desire growth rates) of each pairwise relationship. **Bottom:** An optimal schedule for the instance. On each day, a set of meetings is scheduled as indicated by ♥s. The schedule has a period of 8 days: after day 7, we start from day 0 again. **Top right:** A decision version of the instance obtained for heat 160. The **edge labels** here are the frequencies with which edges have to be scheduled stay below heat 160.

Each day the mutual desire for a meeting experienced by each couple grows by the weight or *desire growth rate* of that relationship[1] – that is, until a meeting occurs and their desire is reset to zero. We will refer to the highest desire ever felt by any pair when following a given schedule as the *heat* of the schedule. The heat of the schedule in Figure 1 is 160: as the reader can verify, no pair ever feels a desire greater than 160 before meeting and resetting their desire to zero. Desire 160 is also attained; e.g., Alex and Daisy are scheduled to see each other every other day, but over the period of 2 days between subsequent meetings, their desire grows to $2 \cdot 80 = 160$.

For the instance in Figure 1, it is easy to show that no schedule with heat < 160 exists. For that, we first convert from desire growth rates to required *frequencies:* Under a heat-160 schedule, a pair with desire growth rate g must meet at least every $\lfloor 160/g \rfloor$ days. The top-right part of Figure 1 shows the result. It is easy to check that the given schedule indeed achieves these frequencies. However, any further reduction of the desired heat to $160 - \varepsilon$ would leave, e.g., Alex hopelessly overcommitted: the relation with Daisy would get frequency $\lfloor 80/(160 - \varepsilon) \rfloor = 1$, forcing them to meet *every* day; but then Brady and Frankie, each with frequency $\lfloor 40/(160 - \varepsilon) \rfloor \leq 3$ cannot be scheduled at all.

---

[1] "Remember, absence makes the heart grow fonder" [10].
(*https://getyarn.io/yarn-clip/ae628721-c1d1-49d1-bd7c-78cbffceabf0*)

While local arguments suffice for our small example, in general, Poly Scheduling is NP-hard (as shown below). We therefore focus this paper on approximation algorithms and inapproximability results.

1.1 Formal Problem Statement

We begin by defining a decision version of Poly Scheduling. In the *Decision Polyamorous Scheduling Problem*, we are given a set of people and pairwise relationships with "attendance frequencies" $f_{i,j}$, and we are trying to find a daily schedule of two-person meetings such that each couple $\{i, j\}$ meets at least every $f_{i,j}$ days. The only constraint on the number of meetings that can occur on any given day is that each person can only participate in (at most) one of them. A Decision Polyamorous Scheduling instance can naturally be modelled as a graph of people with the edges representing their relationships. Because each person can participate in at most one meeting per day, the edges scheduled on any given day must form a matching in this graph.

▶ **Definition 1.1** (Decision Polyamorous Scheduling (DPS)). *A DPS instance $\mathcal{P}_d = (P, R, f)$ (a "(decision) polycule") consists of an undirected graph (P, R) where the vertices $P = \{p_1, \ldots, p_n\}$ are n persons and the edges R are pairwise relationships, with integer frequencies $f : R \to \mathbb{N}$ for each relationship.*

The goal is to find an infinite schedule $S : \mathbb{N}_0 \to 2^R$, such that

(a) *(no conflicts) for all days $t \in \mathbb{N}_0$, $S(t)$ is a matching in $\mathcal{P}_d$, and*

(b) *(frequencies) for all $e \in R$ and $t \in \mathbb{N}_0$, we have $e \in S(t) \cup S(t+1) \cup \cdots \cup S(t + f(e) - 1)$;*

or to report that no such schedule exists. In the latter case, $\mathcal{P}_d$ is called infeasible.

We write $f_{i,j}$ and f_e as shorthands for $f(\{p_i, p_j\})$ resp. $f(e)$. An infinite schedule exists if and only if a *periodic* schedule exists, i.e., a schedule where there is a $T \in \mathbb{N}$ such that for all t, we have $S(t) = S(t + T)$: any feasible schedule corresponds to an infinite walk in the finite configuration graph of the problem (see Section 4), implying the existence of a finite cycle. A periodic schedule can be finitely described by listing $S(0), S(1), \ldots, S(T - 1)$.

By relaxing the hard maximum frequencies of couple meetings to "desire growth rates", we obtain the Optimisation Polyamorous Scheduling (OPS) Problem. Our objective is to find a schedule that minimizes the *"heat"*, i.e., the *worst pain of separation* ever felt in the polycule by any couple.

▶ **Definition 1.2** (Optimisation Polyamorous Scheduling). *An OPS instance (or "optimisation polycule") $\mathcal{P}_o = (P, R, g)$ consists of an undirected graph (P, R) along with a desire growth rate $g : R \to \mathbb{R}_{>0}$ for each relationship in R. An infinite schedule $S : \mathbb{N}_0 \to 2^R$ is valid if, for all days, $t \in \mathbb{N}_0$, $S(t)$ is a matching in $\mathcal{P}_o$.*

The goal is to find a valid schedule that minimizes the heat $h = h(S)$ *of the schedule where $h(S) = \max_{e \in R} h_e(S)$ and*

$$h_e(S) = \sup_{d \in \mathbb{N}} \begin{cases} (d+1) \cdot g(e) & \exists t \in \mathbb{N}_0 : e \notin S(t) \cup S(t+1) \cup \cdots \cup S(t+d-1); \\ g(e) & \text{otherwise.} \end{cases}$$

As for DPS, S can be assumed to be periodic without loss of generality, meaning that S is finitely representable.

1.2 Related Work

Polyamorous Scheduling itself has not been studied to our knowledge. Other variants of periodic scheduling have attracted considerable interest recently [25, 18, 1], including FUN [4].

The arguably simplest periodic scheduling problem is *Pinwheel Scheduling*. In Pinwheel Scheduling [19] we are given k positive integer *frequencies* $f_1 \leq f_2 \leq \cdots \leq f_k$, and the goal is to find a Pinwheel schedule, i.e., an infinite schedule of tasks $1, \ldots, k$ such that any contiguous time window of length f_i contains at least one occurrence of i, for $i = 1, \ldots, k$, (or to report the non-existence of such a schedule).

Pinwheel Scheduling is NP-hard [23], but unknown to be in NP [25], (see [14] for more discussion). Poly Scheduling inherits these properties.

The *density* of a Pinwheel Scheduling instance is given by $d = \sum_{i=1}^{k} 1/f_i$. It is easy to see that $d \leq 1$ is a necessary condition for A to be schedulable, but this is not sufficient, as the infeasible instance $(2, 3, M)$ with $d = \frac{5}{6} + 1/M$, for any $M \in \mathbb{N}$ shows. However, there is a threshold d^* so that $d \leq d^*$ implies schedulability: Whenever $d \leq \frac{1}{2}$, we can replace each frequency f_i by $2^{\lceil \lg(f_i) \rceil}$ without increasing d above 1; then a periodic Pinwheel schedule always exists using the largest frequency as period length. A long sequence of works [19, 6, 20, 28, 7, 13, 11, 14] successively improved bounds on d^*, culminating very recently in Kawamura's proof [25] that it is indeed a the sharp threshold, $d^* = \frac{5}{6}$, confirming the corresponding conjecture of Chan and Chin from 1993 [21]. Generalizations of Pinwheel Scheduling have also been studied, e.g., with jobs of different lengths [17, 12].

Pinwheel Scheduling is a special case of DPS, where the underlying graph (P, R) is a *star*, i.e., a centre connected to k pendant vertices with edges of frequencies $f_1, \ldots, f_k$. Note that it is not generally possible to obtain a polyamorous schedule by combining the local schedule of each person[2]; see for example a triangle with edge frequencies 2: In the DPS instance $(\{A, B, C\}, \{A{-}B, B{-}C, A{-}C\}, f)$ with $f(e) = 2$ for all edges, the local problem for each person is feasible by alternating between their two partners, but the global DPS instance has no solution. This example also shows that the simple strategy of replacing f_i by $2^{\lceil \lg(f_i) \rceil}$ is not sufficient to guarantee the existence of a schedule for Poly Scheduling. Indeed, it is unclear whether any such constant-factor scaling of frequencies exists which applies to all Poly Scheduling instances.

There are two natural optimisation variants of Pinwheel Scheduling. In *Windows Scheduling* [3] tasks with frequencies are given and the goal is to find a perpetual scheduling that minimizes the *number* of tasks that need to be done *simultaneously* while respecting all frequencies (i.e., the number of channels or servers needed to schedule all tasks). Efficient constant-factor approximation algorithms are known that use the connection to Bin Packing [2] (where we bin tasks by used channels), even when the sets of tasks to schedule changes over time [8].

The *Bamboo Garden Trimming (BGT) Problem* [16, 15] retains the restriction of one task per day, but converts the frequencies into *growth rates* $g_1 \leq \cdots \leq g_k$ (of k bamboo plants $1, \ldots, k$) and asks to find a perpetual schedule that minimizes the *height* ever reached by any plant. BGT also allows efficient constant-factor approximations whose approximation factor has seen a lively race of successively improvements over last few years: from 2 [16] over $\frac{12}{7} \approx 1.71$ [30], 1.6 [15], and 1.4 [18], down to the current record, $\frac{4}{3} \approx 1.33$, again by Kawamura [25]. As for the Windows Scheduling problem, no hardness of approximation results are known. It remains open whether it is possible to obtain a PTAS for the Bamboo

---

[2] The current state-of-the-art approach in practice, usually via Google Calendar.

Garden Trimming Problem [15] or the Windows Scheduling Problem. We show that the same is not true for Poly Scheduling (see Theorem 1.3 below).

As for Pinwheel Scheduling and DPS, Bamboo Garden Trimming is the special case of OPS on star graphs. Although BGT can be approximated well, since it is in general not possible to combine local schedules into a global schedule for a polycule (as noted above), it is not clear whether Poly Scheduling allows an efficient constant-factor approximation.

All mentioned problems above have simple *fractional* counterparts that are much easier to solve and hence provide necessary conditions. Indeed, this is the motivation for density in Pinwheel Scheduling: if we allow a schedule to spend arbitrary fractions of the day on different tasks, we obtain a schedule if and only if the density is at most 1. (Spending a $1/f_i$ fraction on task i each day is best possible). For Windows Scheduling, any valid schedule must partition the tasks into bins (channels/servers), so that each bin admits a Pinwheel schedule. Relaxing the latter constraint to "density at most 1" yields a standard bin packing problem, to which we can apply existing techniques; (packing bins only up to density 5/6 guarantees a Pinwheel schedule, at the expense of a 6/5 factor increase in channels). For Bamboo Garden Trimming, the optimal fractional schedule spends a G/g_i fraction of each day with task i, where G is the sum of all growth rates, thus achieving height exactly G. For Poly Scheduling, we can similarly define a fractional problem, but its structure is much richer (see Section 7).

There are further periodic scheduling problems with less direct connections to Poly Scheduling that received attention in the literature. Patrolling problems typically involve periodic schedules: for example, [1] finds schedules for a fleet of k identical robots to patrol (unweighted) points in a metric space, whereas the "Continuous BGT Problem" [15] sends a single robot to points with different frequencies requirements; [26] tasks k robots with patrolling a line or a circle. The underlying geometry in these problems requires different techniques from our work. The *Point Patrolling Problem* studied in [26] can be seen as a "covering version" of Pinwheel Scheduling: each day, we have to assign one of n workers to a single, daily recurring task, where worker i requires a break of a_i days before they can be made to work again. Yet another twist on a patrolling problem is the *Replenishment Problems with Fixed Turnover Times* given in [5], where vertices in a graph have to be visited with given frequencies, but instead of restricting the number of vertices that can be visited per day, the *length of a tour* to visit them (starting at a depot node) shall be minimized.

In the *Fair Hitting Sequence* Problem [9], we are given a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, each consisting of a subset of the set of elements $\mathcal{V} = \{v_1, \ldots, v_n\}$. Each set S_j has an urgency factor g_j, which is comparable to the growth rates in BGT instances with one key difference: A set S_j is hit whenever any $v_i \in S_j$ is scheduled. The goal is again similar to BGT; to find a perpetual schedule of elements $v_i \in \mathcal{V}$ that minimizes the time between visits to each set S_j, weighted by g_j. There is also a decision variant, similar to Pinwheel Scheduling in that growth rates are replaced by frequencies. We use a similar layering technique in our approximation algorithm (Section 6) as the $O(\log^2 n)$-approximation from [9], but we obtain a better approximation ratio for Poly Scheduling. Their $O(\log n)$-approximation based on randomized rounding does not extend to Poly Scheduling since the used linear program has exponentially many variables for Poly Scheduling (Section 7).

1.3   Our Results

Despite the recent flurry of results on periodic scheduling, Polyamorous Scheduling seems not to have been studied before. Apart from its immediate practical applications, some quirks make Polyamorous Scheduling an interesting combinatorial optimization problem

in its own right. *One of these quirks seems to be that love makes blind . . . Shortly after publishing the first version of this manuscript, we found a substantial simplification and strengthening of our hardness-of-approximation results. We keep all original results in this revised manuscript for the record and for future improvement, but point out where they have been modified or superseded in this version.*

Our originally strongest hardness-of-approximation result, which rules out the existence of a PTAS (polynomial-time approximation scheme) for Poly Scheduling, is based on a direct reduction from 3SAT.

▶ **Theorem 1.3** (SAT Hardness of approximation). *Unless P = NP, there is no polynomial-time $(1+\delta)$-approximation algorithm for the Optimisation Poly Scheduling problem for any $\delta < \frac{1}{12}$.*

Love may not always reduce to logic, but propositional logic provably reduces to *scheduling* love.

We also obtain a much simpler, and indeed stronger, inapproximability result, Theorem 1.4, from containing the 3-Regular Chromatic Index Problem as a special case.

▶ **Theorem 1.4** (Hardness of approximation). *Unless P = NP, there is no polynomial-time $(1+\delta)$-approximation algorithm for the Optimisation Poly Scheduling problem for any $\delta < \frac{1}{3}$.*

While in its current form, Theorem 1.3 follows from Theorem 1.4, the direct 3SAT reduction is significantly more versatile and we hope to improve the lower bound on the approximation ratio in future work. The core idea of the reduction in Theorem 1.3 is to force any valid schedule to have a periodic structure with a 3-day period, where edges scheduled on days t with $t \equiv 0 \pmod 3$ represent the value *True* and edges scheduled on days with $t \equiv 1 \pmod 3$ represent *False*; the remaining slots, $t \equiv 2 \pmod 3$, are required to enforce correct propagation along logic gadgets. Indeed, the actual construction uses a 6-day period throughout, where slots 2 and 5 are further distinguished. A variable is represented as a person with 4 relationships with frequencies $[3, 3, 6, 6]$. The two frequency-6 edges are used to connect all variables, which forces the frequency-3 edges to choose slot 0 or 1 (mod 3). The choice of which edge is scheduled in slot 0 encodes the variable assignment. Complications arise in the construction because some gadgets, e.g., the *logical or* ($\vee$), can only guarantee a weaker form of truth value encoding, which requires further gadgets (such as the "tensioning gadget") to preserve the reduction. Section 3 gives the detailed gadget constructions and proofs; Figure 2 (page 11) shows the resulting DPS instance corresponding to an example 3-CNF formula.

Theorem 1.3 of course implies the NP-hardness of Poly Scheduling; we overall have *3* independent reductions establishing the NP-hardness for Poly Scheduling (Section 4); the best known upper bound for the complexity is PSPACE.

We could thus call Poly Scheduling *very* NP-hard; yet efficient approximation algorithms are possible. A simple round-robin schedule using an edge colouring yields a good approximation if *both* maximum degree and ratio between smallest and largest desire growth rates are small (Theorem 1.5).

▶ **Theorem 1.5** (Colouring approximation). *For an Optimisation Poly Scheduling instance $\mathcal{P}_o = (P, R, g)$ set $g_{\min} = \min_{e \in R} g(e)$, $g_{\max} = \max_{e \in R} g(e)$, and let Δ be the maximum degree in (P, R) and h^* be the heat of an optimal schedule. There is an algorithm that computes in polynomial time a schedule S of heat h with $\frac{h}{h^*} \leq \min\{\frac{\Delta+1}{\Delta} \cdot \frac{g_{\max}}{g_{\min}}, \Delta + 1\}$.*

A fully general approximation seems only possible with much weaker ratios; we provide a $O(\log \Delta)$-approximation by applying Theorem 1.5 to groups with similar weight and interleaving the resulting schedules.

▶ **Theorem 1.6** (Layering approximation). *For an Optimisation Poly Scheduling instance* $\mathcal{P}_o = (P, R, g)$, *let* Δ *be the maximum degree in* (P, R) *and* h^* *be the heat of an optimal schedule. There is an algorithm that computes in polynomial time a schedule S of heat h with* $\frac{h}{h^*} \leq 3 \lg(\Delta + 1) = O(\log n)$, *where* $n = |P|$.

Finally, we generalize the notion of density to Poly Scheduling. As discussed above, density has proven instrumental in understanding the structure of Pinwheel Scheduling and in devising better approximation algorithms, by providing a simple, instance-specific lower bound. For Poly Scheduling, the fractional problem is much richer, and indeed remains nontrivial to solve. We devise a generalization of density for Poly Scheduling from the dual of the LP corresponding to a fractional variant of Poly Scheduling with gives the following instance-specific lower bound.

▶ **Theorem 1.7** (Fractional lower bound). *Given an OPS instance* $\mathcal{P}_o = (P, R, g)$ *with optimal heat* h^*. *For any values* $z_e \in [0, 1]$, $e \in R$, *with* $\sum_{e \in R} z_e = 1$, *we have*

$$h^* \geq \bar{h}(z) = \frac{1}{\displaystyle\max_{M \in \mathcal{M}} \sum_{e \in M} \frac{z_e}{g(e)}}.$$

with the maximum ranging over all maximal matchings in (P, R). *The largest value* $\bar{h}^*$ *of* $\bar{h}(z)$ *over all feasible z, is the* poly density *of* $\mathcal{P}_o$.

The bound implies (and formally establishes) simple ad-hoc bounds such as the following, which corresponds to the lower bound of G on the height in Bamboo Garden Trimming.

▶ **Corollary 1.8** (Total growth bound). *Given an OPS instance* $\mathcal{P}_o = (P, R, g)$ *with optimal heat* h^*. *Let* $G = \sum_{e \in R} g(e)$ *and m be the size of a maximum matching in* (P, R). *Then* $h^* \geq G/m$.

More importantly though, Theorem 1.7 allows us to define a *poly density* similarly to the Pinwheel Scheduling Problem, and allows us to formulate the most interesting open problem about Poly Scheduling. For a DPS instance $\mathcal{P}_d = (P, R, f)$, define the poly density of $\mathcal{P}_d$, $\bar{h}^*(\mathcal{P}_d)$, as the poly density of the OPS instance $\mathcal{P}_o = (P, R, 1/f)$ (see also Lemma 2.1).

▶ **Open Problem 1.9** (Poly Density Threshold?). *Is there a constant c such that every Decision Poly Scheduling instance* $\mathcal{P}_d = (P, R, f)$ *with poly density* $\bar{h}^*(\mathcal{P}_d) \leq c$ *admits a valid schedule?*

2   Preliminaries

In this section, we introduce some general notation and collect a few simple facts about Poly Scheduling used later.

We write $[n..m]$ for $\{n, n+1, \ldots, m\}$ and $[n]$ for $[1..n]$. For a set A, we denote its powerset by 2^A. All graphs in this paper are simple and undirected. We denote by $\mathcal{M} = \mathcal{M}(V, E)$ the set of *inclusion-maximal matchings* in graph (V, E), where matching has the usual meaning of an edge set with no two edges incident to the same vertex. By $\Delta = \Delta(V, E)$, we denote the *maximum degree* in (V, E). A *pendant vertex* is a vertex with degree 1. The *chromatic index* $\chi_1 = \chi_1(V, E)$ is the smallest number C of "colours" in a proper edge colouring of (V, E) (i.e., the number of disjoint matchings required to cover E); by Vizing's Theorem [31], we have $\Delta \leq \chi_1 \leq \Delta + 1$ for every graph. Misra and Gries provide a polynomial-time algorithm for edge colouring any graph using at most $\Delta + 1$ colours [29].

Given a schedule $S : \mathbb{N}_0 \to 2^R$ and an edge $e \in R$, we define the *(maximal) recurrence time* $r(e) = r_S(e)$ of e in S as the maximal time between consecutive occurrences of e in S, formally

$$r_S(e) \;=\; \sup_{d \in \mathbb{N}} \begin{cases} d + 1 & \exists t \in \mathbb{N}_0 : e \notin S(t) \cup S(t+1) \cup \cdots \cup S(t+d-1); \\ 0 & \text{otherwise.} \end{cases}$$

Using recurrence time, the heat $h = h(S)$ of a schedule S in an OPS instance (P, R, g) is $h(S) = \max_{e \in R} g(e) \cdot r(e)$. Clearly, for any schedule $S : \mathbb{N}_0 \to 2^R$, we can obtain $S' : \mathbb{N}_0 \to \mathcal{M}$ by adding edges to $S(t)$ until we have a maximal matching $S'(t) \supseteq S(t)$; then $r_{S'}(e) \le r_S(e)$ for all $e \in R$ and hence S' is a valid schedule for any DPS instance for which S is valid, and if S schedules an OPS instance with heat $h(S)$ then S' does too, with $h(S') \le h(S)$.

We use Lemma 2.1 to reduce OPS to DPS, and Lemma 2.2 to formalize how DPS solves OPS:

▶ **Lemma 2.1** (OPS to DPS). *For every combination of OPS instance $\mathcal{P}_o = (P, R, g)$ and heat value h, there exists a DPS instance $\mathcal{P}_d = (P, R, f)$ such that*

(a) *any feasible schedule $S : \mathbb{N}_0 \to 2^R$ for $\mathcal{P}_d$ is a schedule for $\mathcal{P}_o$ with heat $\le h$, and*

(b) *any schedule S' for $\mathcal{P}_o$ with heat $h' > h$ is not feasible for $\mathcal{P}_d$.*

Proof. Consider an OPS polycule $\mathcal{P}_o = (P, R, g)$; we set $\mathcal{P}_d = (P, R, f)$ where $f(e) = \left\lfloor \frac{h}{g(e)} \right\rfloor$ for all $e \in R$. Schedules satisfying $\mathcal{P}_d$ when applied to $\mathcal{P}_o$ will allow heat of at most $\max_{e \in R} g(e) \cdot f(e) = \max_{e \in R} g(e) \lfloor \frac{h}{g(e)} \rfloor \le h$.

Now consider a schedule S' for $\mathcal{P}_o$ with heat $h' > h$. By definition, $h' = \max_{e \in R} r_{S'}(e) \cdot g(e)$, where $r(e) = r_{S'}(e)$ is the recurrence time of e in S'. Assume towards a contradiction that $r(e) \le f(e)$ for all $e \in R$. This implies that $h' = \max_{e \in R} r(e) \cdot g(e) \le \max_{e \in R} \lfloor \frac{h}{g(e)} \rfloor \cdot g(e) \le h$, a contradiction to the assumption. ◀

▶ **Lemma 2.2** (DPS to OPS). *Let $\mathcal{P}_d = (P, R, f)$ be a DPS instance. Set $F = \max_{e \in R} f(e)$. There is an OPS instance $\mathcal{P}_o = (P, R, g)$ such that the following holds.*

(a) *If $\mathcal{P}_d$ is feasible, then $\mathcal{P}_o$ admits a schedule of height $h \le 1$.*

(b) *If $\mathcal{P}_d$ is infeasible, then the optimal heat h^* of $\mathcal{P}_o$ satisfies $h^* \ge (F+1)/F$.*

Proof. Let $\mathcal{P}_d = (P, R, f)$ be given; we set $\mathcal{P}_o = (P, R, g)$ with $g(e) = 1/f(e)$ for $e \in R$. Any feasible schedule S for $\mathcal{P}_d$ has recurrence time $r(e) \le f(e)$ by definition, so its heat in $\mathcal{P}_o$ is $h(S) = \max r(e) g(e) = \max r(e) \frac{1}{f(e)} \le 1$. If conversely $\mathcal{P}_d$ is infeasible, for every $S : \mathbb{N}_0 \to 2^R$ there exists an edge $e \in R$ where $r_S(e) > f(e)$, i.e., $r_S(e) \ge f(e) + 1$. For the heat of S in $\mathcal{P}_o$, this means $h(S) = \max_{e \in R} r_S(e) g(e) \ge \max_{e \in R} (f(e) + 1) \frac{1}{f(e)} \ge (F+1)/F$. ◀

We will often use the *Normal Form* of OPS instances in proofs; this can be assumed without loss of generality but is not generally useful for algorithms unless h^* is known:

▶ **Lemma 2.3** (Normal Form OPS). *For every OPS instance $\mathcal{P}_o = (P, R, g)$, there is an equivalent OPS instance $\mathcal{P}'_o = (P, R, g')$ with optimal heat 1 where $g' : R \to \mathcal{U}$ for $\mathcal{U} = \{1/m : m \in \mathbb{N}_{\ge 1}\}$, i.e., the set of unit fractions. More precisely, for every schedule $S : \mathbb{N}_0 \to 2^R$ holds: S has optimal heat h^* in $\mathcal{P}_o$ if and only if S has heat 1 in $\mathcal{P}'_o$.*

Proof. Let (P, R, g) be an arbitrary OPS instance with optimal heat h^*. Setting $\hat{g}(e) = g(e)/h^*$ yields OPS instance $(P, R, \hat{g})$ with optimal heat 1. We now start by setting $g'(e) = \hat{g}(e)$ for all $e \in R$. Fix an optimal schedule S. Suppose that for some relation $e \in R$, we have $g'(e) \notin \mathcal{U}$. In S, there is a maximal separation $r(e) = q \in \mathbb{N}$ between consecutive occurrences

of e with $q \cdot g(e) \leq h^*$. But then, increasing $g(e)$ to h^*/q would not affect the heat of S. We can thus set $g'(e) = 1/q$. By induction, we thus obtain $g' : R \to \mathcal{U}$ without affecting the heat of S. ◀

3 Inapproximability

In this section, we prove Theorem 1.3. We show via a reduction from MAX-3SAT that OPS does not allow efficient $(1 + \delta)$-approximation algorithms for $\delta < \frac{1}{12}$ unless P = NP.

3.1 Overview of Proof

The proof of Theorem 1.3 has two steps: The first step is a reduction from the decision version of MAX-3SAT (D-MAX-3SAT) to DPS. D-MAX-3SAT is the following problem: given a 3-CNF formula $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ with clauses $C = \{c_1, c_2, \ldots, c_m\}$ over variables $X = \{x_1, x_2, \ldots, x_{n'}\}$ and integer k, decide whether there is an assignment of Boolean values to the variables in X such that at least k clauses in C are satisfied.

▶ **Lemma 3.1** (D-MAX-3SAT $\leq_p$ DPS). *For any 3-CNF formula φ with m clauses and integer $k \leq m$, we can construct in polynomial time a decision polycule $\mathcal{P}_{d\varphi k}$ which has a valid schedule if and only if at least k clauses of φ can be simultaneously satisfied.*

Lemma 3.1 is our key technical contribution and its proof will be given over the course of the remainder of this section.

The second step in the proof of Theorem 1.3 is to convert the decision polycule $\mathcal{P}_{d\varphi k}$ from Lemma 3.1 to an optimisation polycule $\mathcal{P}_{o\varphi k}$ using Lemma 2.2. It will be immediate from the construction that the largest frequency in $\mathcal{P}_{d\varphi k}$ is $F = 12$. So by Lemma 2.2, $\mathcal{P}_{o\varphi k}$ has either $h^* = 1$, namely if $\mathcal{P}_{d\varphi k}$ is feasible, or $h^* \geq \frac{13}{12}$, otherwise.

Proof of Theorem 1.3. Assume that there is a polynomial-time approximation algorithm A for OPS with approximation ratio $\alpha < \frac{13}{12}$. If $\mathcal{P}_{o\varphi k}$ has optimal heat $h^* = 1$, A produces a schedule with heat $1 \leq h \leq \alpha < \frac{13}{12}$, whereas if $h^* \geq \frac{13}{12}$, A must produce a schedule of heat $\frac{13}{12} \leq h \leq \alpha \cdot \frac{13}{12}$. So, by running A, we are able to distinguish between $h^* \leq 1$ and $h^* \geq \frac{13}{12}$ for $\mathcal{P}_{o\varphi k}$, hence between feasibility or infeasibility of $\mathcal{P}_{d\varphi k}$ and, therefore between Yes and No instances of D-MAX-3SAT via the polynomial-time reduction from Lemma 3.1. As a generalisation of 3SAT, D-MAX-3SAT is NP-hard, hence P = NP follows. ◀

Let us denote by α^* the approximability threshold for Optimisation Poly Scheduling, that is: efficient polynomial-time approximation algorithms with approximation ratio α exist if and only if $\alpha \geq \alpha^*$ (assuming P $\neq$ NP). Theorem 1.3 shows that $\alpha^* \geq \frac{13}{12}$ and Theorem 1.6 shows that $\alpha^* = O(\log n)$, leaving a substantial gap. We conjecture that the constant $\frac{13}{12}$ can be improved by careful analysis of our construction, but we leave this to future work.

▶ **Conjecture 3.2.** $\alpha^* \geq \frac{4}{3}$.

▶ Remark 3.3 (Towards stronger inapproximability results). Our reduction includes additional degrees of freedom not currently used towards the proof of Lemma 3.1. In particular, for the current statement, a reduction for standard 3SAT would have sufficed, removing the sorting network from the construction. However, to find better constant or even superconstant lower bounds for α^* it seems likely that starting with a gapped MAX-3SAT problem can provide stronger gaps for the outcome. For that, we need that our construction allows us to specify freely how many clauses have to be satisfiable for $\mathcal{P}_{d\varphi k}$ to be feasible. While we leave this to

future work, we include here the required features in the construction which may facilitate these results.

3.2   Reduction Overview

We now give the proof of Lemma 3.1. For the remainder of this section, we assume a 3-CNF formula $\varphi = c_1 \wedge \cdots \wedge c_m$ over variables $X = \{x_1, \ldots, x_{n'}\}$ and integer k are given. We will describe how to construct the DPS instances $\mathcal{P}_{d\varphi k}$ that admits a schedule iff there is a variable assignment $v : X \to \{\text{True}, \text{False}\}$ that satisfies at least k clauses in $C = \{c_1, \ldots, c_m\}$. The construction is based on building components of Boolean formulas via DPS "gadgets":

- *variables* (Section 3.4),
- clauses (*OR gadgets*) (Section 3.6),
- a *sorting network*, comprised of *SWAP gadgets* (Section 3.7.5) to group satisfied outputs together, and
- a check for $\geq k$ true clauses, the *tension gadget* (Section 3.8).

To make those gadgets work, we require further auxiliary gadgets:

- a *"True Clock"* to break ties between symmetric choices for schedules (Section 3.3),
- slot duplication gadgets: D_3 *duplicators* (Section 3.5.1), D_6 *duplicators* (Section 3.5.2), D_{12} duplicators (Section 3.7.1), and
- *slot splitting gadgets S_{B6}, S_{B12}, S_{G12}* (Section 3.7.4).

The overall conversion algorithm is stated in Definition 3.4 below; a worked example is shown in Figure 2.

▶ **Definition 3.4** ($\mathcal{P}_{d\varphi k}$ polycules). *The decision polycule $\mathcal{P}_{d\varphi k} = (P, R, f)$ is constructed in layers as follows:*

(a) *Variable layer:*
The variable layer consists of a True Clock and a variable gadget with outputs 3_{iR} and 3_{iB} for each variable $x_i \in X$.

(b) *Duplication layer:*
The duplication layer duplicates outputs of the variable layer: D_3 duplicators create one 3_{iR} edge for each $x_i \in C$, and one 3_{iB} edge for each $\overline{x_i} \in C$. They also create as many 3_R and 3_B edges as are needed, while D_6 duplicators do the same for 6_G and 6_P edges. Unused edges are connected to pendent nodes.

(c) *Clause layer:*
The clause layer consists of one OR gadget for each clause $c_j \in C$. OR gadgets have three inputs, each corresponding to a literal in c_j: 3_{iR} for x_i, 3_{iB} for $\overline{x_i}$. For clauses with less than 3 literals, 3_B edges fill the OR gadget's unused inputs.

(d) *Sorting network:*
The next layer is a single gadget – a sorting network. The 12_O output edges of the clause layer will each be the input to one of m channels, each of which terminates with a 12_O output edge. This gadget also consumes 12_B and 12_G edges created by S_{B12} and S_{G12} gadgets.

(e) *Tension layer:*
The tensioning layer attaches tension gadgets to the leftmost k outputs of the sorting layer – any other outputs from this layer and any spare inputs to the tensioning layer are connected to pendant nodes.
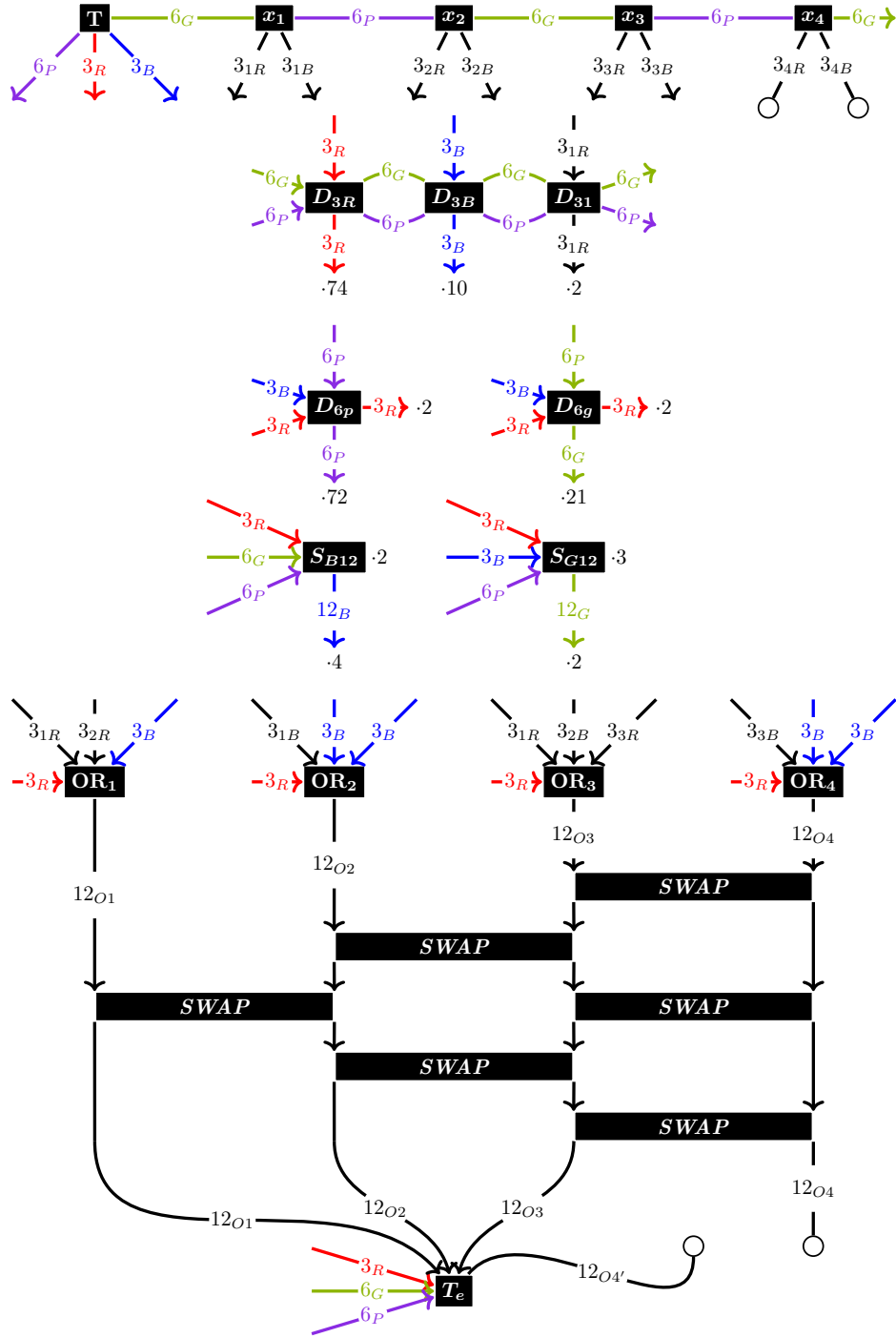
Figure 2 A DPS polycule which is schedulable iff there is some assignment that simultaneously satisfies at least 3 of $(x_1 \vee x_2), (\overline{x_1}), (x_1 \vee \overline{x_2} \vee x_3)$, and $(\overline{x_3})$ (possible in this case). Note that re-unifying the bottom-most 12_{O4} edge with the $12_{O4'}$ edge will make a polycule which is schedulable iff all 4 clauses can be satisfied (and hence has no valid schedule here). Similarly, breaking the 12_{O3} edge in the middle and connecting the two new ends to pendant nodes will create a polycule which is schedulable iff 2 clauses can be satisfied. Connections between layers are omitted for clarity but flow from top to bottom, starting with the variable layer, then three layers concerned with the duplication of variables, the OR layer, the sorting network, and the tensioning layer.

3.3 The True Clock & Colour Slots
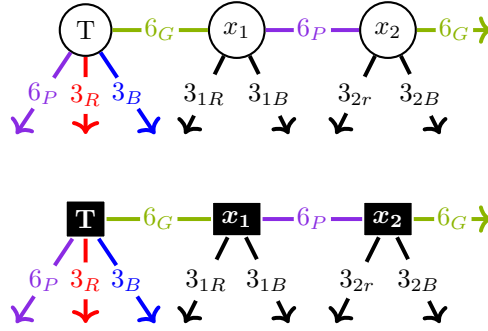
Figure 3 Gadgets for the True Clock and for sample variables x_1 and x_2 (top, left to right), with shorthand versions shown below. Gadgets are shown connected as they would be in a sample variable layer, and their colours and schedules are discussed in Section 3.3. Further variables can be added to the right, and must be added in pairs to conserve the 6_G output edge (though the final variable may be connected to pendant nodes if not otherwise needed).

Figure 3 introduces gadgets representing sample variables x_1 and x_2, as well as a special variable: the True Clock T, which acts as a drumbeat for the polycule as a whole. Variables, including T, have four relationships: $[3, 3, 6, 6]$, so their local schedules must all have the following form: $[3_a, 3_b, 6_a, 3_a, 3_b, 6_b]$. As schedules are cyclic, we can choose to start this schedule with the 3_a edge of T without loss of generality. We then assign names to these edges as dictated by the True Clock.

▶ **Definition 3.5** (Slots). *We call days $t \in \mathbb{N}_0$ with $t \equiv 0$ (mod 3) the red slots, days $t \equiv 1$ (mod 3) blue slots, days $t \equiv 2$ (mod 6) green slots, and days $t \equiv 5$ (mod 6) purple slots.*

The local schedule of the True Clock is therefore given by $[3_R, 3_B, 6_G, 3_R, 3_B, 6_P]$. As 3_R is scheduled on day 0, it will always be assigned in red slots, with 3_B, 6_G, and 6_P edges also restricted to slots of their respective colours. We will sometimes represent this by underlining elements or gaps in a schedule, e.g., $[\_, \_, \_, \_, \_, \_]$, or by referring to edges as being red, blue, green, or purple.

All gadgets introduced below will be constructed such that the lengths of their schedules are integer multiples of 6. In the final polycule $\mathcal{P}_{d\varphi k}$ they will be connected (usually through intermediaries) to the True Clock, such that their edges must stick to certain slots. To keep correctness proofs of individual gadgets readable, we call a schedule S that schedules all coloured edges in slots of the given colour *slot-respecting*.

Drawing Conventions

Gadgets are connected by input and output edges, represented by incoming and outgoing arrows respectively – each one being half of a relationship between two people from different gadgets. In addition to their frequencies, input and output edges share restrictions on their permissible schedules, carrying them from one gadget to another as discussed in the proofs associated with each gadget.

In shorthand gadgets, vertical incoming edges are the primary *input* to a gadget, encoding the value of some variable or logical function; horizontal inputs contain edges of fixed colour, which we will refer to as *constants*. Similarly, vertical outgoing edges represent primary *outputs* that encode the result of the gadget, while horizontal outgoing edges represent

incidentally created constants which may either be used by other gadgets or connected to pendent vertices.

Some incoming and outgoing edges will have end labels of the form "$\cdot i$", indicating i connections of the given type, each between different people.

3.4 Variables

Figure 3 also introduces the gadget for a sample variable, x_1, which again has four relationships: $[3_{1R}, 3_{1B}, 6_G, 6_P]$. The key property of variable gadgets is summarized in the following lemma.

▶ **Lemma 3.6** (Variable gadget schedules). *Any valid global schedule must yield a local schedule for the variable gadget x_i of the form $[3_a, 3_b, 6_G, 3_a, 3_b, 6_P]$.*

Proof. Note that x_i has local density $D = 1$, so 3_{iR} and 3_{iB} must be scheduled exactly once in each 3-day period, forcing every 3 days to be of the form $[3_a, 3_b, \_\_]$, and every 6-day schedule to be of the form $[3_a, 3_b, \_\_, 3_a, 3_b, \_\_]$; this leaves two remaining slots, which must contain 6_G and 6_P.

The 6_G edge of the first variable, x_1, is shared with T such that it must be green, so schedules for x_1 are of the form $[3_a, 3_b, 6_G, 3_a, 3_b, \_\_]$, which must be completed as $[3_a, 3_b, 6_G, 3_a, 3_b, 6_P]$. This proceeds for x_2, whose 6_P edge is shared with x_1 such that it must be purple, forcing the partial schedule $[3_a, 3_b, \_\_, 3_a, 3_b, 6_P]$ which likewise must be completed $[3_a, 3_b, 6_G, 3_a, 3_b, 6_P]$.

Further pairs of variables are each connected to the 6_G edge returned by the previous pair, so their schedules must be of the same form. Thus, by induction, schedules for each variable gadget x_i must be of the form $[3_a, 3_b, 6_G, 3_a, 3_b, 6_P]$. ◀

According to Lemma 3.6, the incident 6_G edge is used to restrict the valid schedules for x_1, leaving two possibilities: $[\underline{3_{1R}}, \underline{3_{1B}}, 6_G, \underline{3_{1R}}, \underline{3_{1B}}, 6_P]$ and $[\underline{3_{1B}}, \underline{3_{1R}}, 6_G, \underline{3_{1B}}, \underline{3_{1R}}, 6_P]$. The former schedule, where 3_{1R} is scheduled in red slots, corresponds to a variable assignment where x_1 is True, whereas the the second schedule corresponds to x_1 being assigned False.

This technique of connecting 3_R, 3_B, 6_G, or 6_P edges to people in a gadget to limit their valid local schedules and force relationships between edges, slots, and particular meanings will be used extensively in what follows.

Note that 3_{iB} has the opposite value to 3_{iR}, so using 3_{iB} edges in the polycule corresponds to the negated literal $\overline{x_i}$, just as 3_{iR} edges correspond to the literal x_i.

3.5 Duplication of Variables and Constants

Variables may appear in multiple clauses, while the constant 3_R, 3_B, 6_G, and 6_P edges are used in multiple gadgets, engendering a need for the duplication of variables and constants.

3.5.1 3-Duplicators

A gadget for duplicating edges with period 3 is shown in Figure 4 and proven to accurately reproduce its input by Lemma 3.7.

▶ **Lemma 3.7** (3-Duplicator gadget schedules). *Any slot-respecting schedule must yield a local schedule for each node in any 3-duplicator gadget D_3 of the same form: either*

$$[\underline{3_a}, \underline{9_b}, 6_G, \underline{3_a}, \underline{9_{b'}}, 6_P, \underline{3_a}, \underline{9_{b''}}, 6_G, \underline{3_a}, \underline{9_b}, 6_P, \underline{3_a}, \underline{9_{b'}}, 6_G, \underline{3_a}, \underline{9_{b''}}, 6_P] \quad or$$

$$[\underline{9_b}, \underline{3_a}, 6_G, \underline{9_{b'}}, \underline{3_a}, 6_P, \underline{9_{b''}}, \underline{3_a}, 6_G, \underline{9_b}, \underline{3_a}, 6_P, \underline{9_{b'}}, \underline{3_a}, 6_G, \underline{9_{b''}}, \underline{3_a}, 6_P].$$
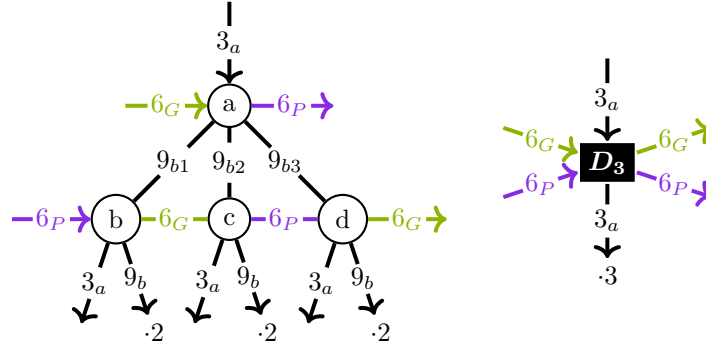
Figure 4 A gadget for duplicating input edges with frequency 3, with a shorthand version below. Note that the input can be duplicated indefinitely many times by repeating the second layer using the 9_b edges from the previous layer and a 6_P or 6_P edge from two layers above.

Proof. Each node in D_3 has tasks $[3_a, 6_P, 6_G, 9_{b1}, 9_{b2}, 9_{b3}]$ and has local density $D = 1$, so each task with frequency f must appear exactly once every f days. Further, in any slot-respecting schedule, tasks 6_P and 6_G are scheduled in purple and green slots respectively, forcing partial schedules of the form $[\_\_, \_\_, 6_G, \_\_, \_\_, 6_P]$.

Considering node a, note that if incident edge 3_a is scheduled on a combination of red and blue days then either it will appear more than once in some 3-day period or its constraint will be violated. This demonstrates that schedules must be of the form

$$[\underline{3_a}, \_\_, 6_G, \underline{3_a}, \_\_, 6_P, \underline{3_a}, \_\_, 6_G, \underline{3_a}, \_\_, 6_P, \underline{3_a}, \_\_, 6_G, \underline{3_a}, \_\_, 6_P] \quad \text{or}$$

$$[\_\_, \underline{3_a}, 6_G, \_\_, \underline{3_a}, 6_P, \_\_, \underline{3_a}, 6_G, \_\_, \underline{3_a}, 6_P, \_\_, \underline{3_a}, 6_G, \_\_, \underline{3_a}, 6_P],$$

depending on the colour of the incident 3_a edge. In either case, 9_{b1}, 9_{b2} and 9_{b3} must occupy the remaining slots, which match the schedules shown in the lemma for some mapping of 9_{b1}, 9_{b2} and 9_{b3} to 9_b, $9_{b'}$ and $9_{b''}$.

Now consider an arbitrary node $p \neq a$, with an incident 9_b node. If the 3_a edge incident to a is red, the schedule for n must be of the form

$$[\_\_, \underline{9_b}, 6_G, \_\_, \_\_, 6_P, \_\_, \_\_, 6_G, \_\_, \underline{9_b}, 6_P, \_\_, \_\_, 6_G, \_\_, \_\_, 6_P],$$

with $9_{b'}$, $9_{b''}$, and 3_a filling the remaining slots. If either $9_{b'}$ or $9_{b''}$ are ever scheduled in a red slot, the constraint of 3_a will be violated, therefore either partial schedule leads to the full schedule

$$[\underline{3_a}, \underline{9_b}, 6_G, \underline{3_a}, \underline{9_{b'}}, 6_P, \underline{3_a}, \underline{9_{b''}}, 6_G, \underline{3_a}, \underline{9_b}, 6_P \underline{3_a}, \underline{9_{b'}}, 6_G, \underline{3_a}, \underline{9_{b''}}, 6_P]$$

matching the schedule of a, and of the lemma. If the 3_a edge incident to a is blue, the same logic applies, with all 3_a edges also being blue and the 9_b nodes being red. ◀

3.5.2   6-Duplicators

Figure 5 and Lemma 3.8 introduce a gadget which duplicates incident 6_G or 6_P edges.

▶ **Lemma 3.8** (6-Duplicator gadget schedules). *Any slot-respecting schedule must yield a local schedule for each node in any 6-duplicator gadget D_6 of the form*

$$[3_R, 3_B, 12_{G1}, 3_R, 3_B, 6_P, 3_R, 3_B, 12_{G2}, 3_R, 3_B, 6_P].$$
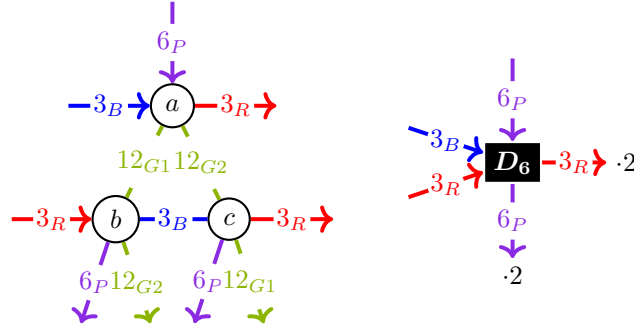
Figure 5 A gadget for duplicating 6_P input edges, with a shorthand version below. Note that if additional 6_P edges are needed, additional layers can be added, re-using constants from above. Also note that 6_G edges can be duplicated with the same gadget simply by replacing the topmost 6_P edge with a 6_G edge, forcing the 12_1 and 12_2 edges to be purple.

Proof. Each node in D_6 has tasks $[3_R, 3_B, 6_P, 12_{G1}, 12_{G2}]$. It also has density $D = 1$, so each task with frequency f must appear exactly once every f days.

Consider node a, which has inputs 3_B and 6_P. In any slot-respecting schedule these are scheduled in slots of their respective colours, which forces partial schedules for a to be of the form $[\_\_, 3_B, \_\_, \_\_, 3_B, 6_P]$. Exactly two of these slots must be filled by 3_R, and if these are not both red slots, the constraint of 3_R will be violated. Thus, the schedule for a must be of the form

$$[3_R, 3_B, \_\_, 3_R, 3_B, 6_P, 3_R, 3_B, \_\_, 3_R, 3_B, 6_P].$$

Two spaces remain, which must then contain 12_{G1} and 12_{G2}, as shown in the Lemma.

Now consider an arbitrary node other than a. All such nodes will have inputs 12_{Ga} and 3_R, forcing their partial schedules to be of the form

$$[3_R, \_\_, 12_{Ga}, 3_R, \_\_, \_\_, 3_R, \_\_, \_\_, 3_R, \_\_, \_\_].$$

As with a, exactly four of these slots must schedule 3_B edges, and these must be the blue spaces for the 3_B constraint not to be violated, forcing schedules of the form

$$[3_R, 3_B, 12_{Ga}, 3_R, 3_B, \_\_, 3_R, 3_B, \_\_, 3_R, 3_B, \_\_].$$

One of these slots must contain the 12_{Gb} edge, with the other two scheduling the 6_P edge. If the 12_{Gb} edge is not scheduled in the remaining green slot, the constraint on the 6_P edge will be violated, so the schedule must be as shown in the Lemma.   ◄

3.6   Clauses

Clauses in C are disjunctions of at most 3 literals, i.e., the logical OR of at most 3 variables, any of which may be negated. A gadget which determines the truth value of a clause given the values of its variables is shown in Figure 6. Lemma 3.9 and Remark 3.10 show that the output of this gadget (12_O) can be scheduled in blue slots iff the corresponding clause is evaluated to be True, though it can always be scheduled in green slots.

Recall the intuitive meaning of slots assigned to the outputs from variable gadgets: a schedule that schedules 3_{iR} in red slots corresponds to a variable assignment where x_i is True. Unfortunately, the output of the OR gadget has to be encoded with a different (and weaker) invariant.
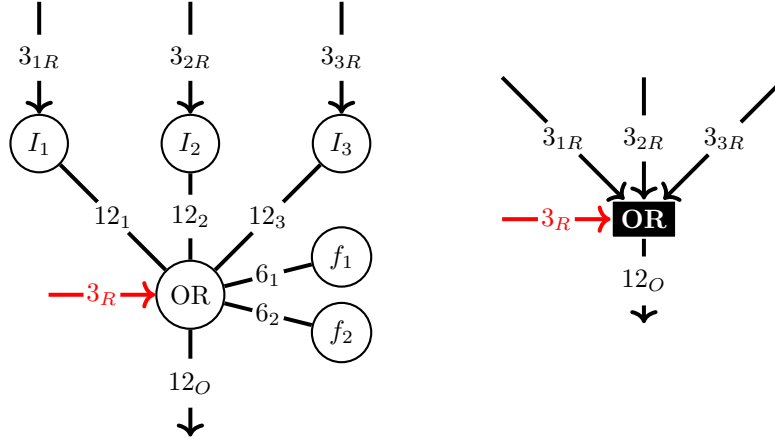
Figure 6 A gadget which computes $x_1 \vee x_2 \vee x_3$ (left), along with a shorthand version (right). To compute $x_1 \vee x_2 \vee \overline{x_3}$, replace the incoming 3_{3R} edge with a 3_{3B} edge. To instead compute $x_1 \vee x_2$, replace the incoming 3_{3R} edge with a 3_B edge. According to Lemma 3.9 and Remark 3.10, 12_O will be scheduled in green slots if all inputs are assigned False and may be scheduled in green or blue slots if any input is assigned True.

▶ **Lemma 3.9** (OR gadget schedules). *In any slot-respecting schedule where 3_{1R}, 3_{2R}, and 3_{3R} are scheduled in red or blue slots, the local schedule of the OR gadget has the following form:*

- *If all of 3_{1R}, 3_{2R}, and 3_{3R} are scheduled in blue slots,*
  then 12_O will be scheduled in green or purple slots.
- *If at least one of 3_{1R}, 3_{2R}, and 3_{3R} are scheduled in red slots,*
  then 12_O will be scheduled in green, purple, or blue slots.

The lemma only covers the clause $x_1 \vee x_2 \vee x_3$, but all other clauses can be handled similarly: For a literal x_j, we use 3_{jR} as input, and for negated literals $\overline{x_j}$, we use the 3_{jB} edge instead of 3_{jR}.

Proof. The node labelled OR has tasks $[3_R, 6_1, 6_2, 12_1, 12_2, 12_3, 12_O]$ and local density $D = 1$, so each task with frequency f must appear exactly once every f days. Any slot-respecting schedule must schedule the 3_R edge in red slots so schedules for the OR node must be of the form $[3_R, \_\_, \_\_, 3_R, \_\_, \_\_]$.

Consider an inverter node I_i, $i = 1, 2, 3$, which has tasks $[3_{iR}, 12_i]$ where 3_{iR} is red or blue by assumption. Given these constraints, if the input edge 3_{iR} is scheduled in red slots, then partial schedules for I_i must be of the form $[\underline{3_{iR}}, \_\_, \_\_, \underline{3_{iR}}, \_\_, \_\_]$ and the 12_i edge must be scheduled in either green, purple, or blue slots. Similarly, if the input edge 3_{iR} is scheduled in blue slots then partial schedules for I_i must be of the form $[\_\_, \underline{3_{iR}}, \_\_, \_\_, \underline{3_{iR}}, \_\_]$, restricting the 12_i edge to green, purple, or red slots. However, the 12_i edge is also connected to the OR node which has no empty red slots, further restricting it to green or purple slots.

Suppose that all inputs $3_{1R}, 3_{2R}, 3_{3R}$ are scheduled in blue slots. By the reasoning above, 12_1, 12_2, 12_3 are then scheduled in green or purple slots. This will force schedules for the OR node to be of the form

$$[3_R, \_\_, \underline{12_a}, 3_R, \_\_, \underline{12_b}, 3_R, \_\_, \underline{12_c}, 3_R, \_\_, \_\_] \quad \text{or}$$

$$[3_R, \_\_, \underline{12_a}, 3_R, \_\_, \underline{12_b}, 3_R, \_\_, \_\_, 3_R, \_\_, \underline{12_c}]$$

(for some mapping of 12_1, 12_2, and 12_3 onto 12_a, 12_b, and 12_c). Assume towards a contradiction that the 12_O edge is blue. This immediately causes a constraint violation when scheduling either 6_1 or 6_2, which demonstrates that the 12_O edge cannot be blue, and the schedule for the *OR* node must be of the form

$$[3_R, \underline{6_a}, \underline{12_a}, 3_R, \underline{6_b}, \underline{12_b}, 3_R, \underline{6_a}, \underline{12_c}, 3_R, \underline{6_b}, \underline{12_d}]$$

(likewise, for some mapping of 12_1, 12_2, and 12_3, and 12_O onto 12_a, 12_b, 12_c, and 12_d). This ensures that the 12_O edge must be scheduled in either green or purple slots if all input edges are blue.

Now suppose that one input edge, 3_{tR}, is scheduled in red slots, while the other two, 3_{iR} and $3_{i'R}$ may be either red or blue. Consider the schedule

$$[3_R, \underline{12_t}, \underline{6_a}, 3_R, \underline{6_b}, \underline{12_i}, 3_R, \underline{12_O}, \underline{6_a}, 3_R, \underline{6_b}, \underline{12_{i'}}],$$

where 12_O is scheduled in blue slots and no constraints are violated. However, even if 3_{tR} is assigned red, the corresponding edge 12_t may be also be scheduled in green, or purple slots, permitting schedules of the form

$$[3_R, \underline{6_a}, \underline{12_a}, 3_R, \underline{6_b}, \underline{12_b}, 3_R, \underline{6_a}, \underline{12_c}, 3_R, \underline{6_b}, \underline{12_d}],$$

in which 12_O is scheduled in green or purple slots. This shows that if one or more inputs are red, then 12_O may be scheduled in blue, green, or purple slots. ◀

▶ Remark 3.10 (No purple output for OR). Note that in $\mathcal{P}_{d\varphi k}$, the 12_O edge of an *OR* gadget will always be connected to the I node of a D_{12} gadget (discussed below, see Figure 7). Due to this node's 6_P edge, I has no empty purple slots, so the purple slots in Lemma 3.9 for 12_O are not actually possible once the gadgets is part of the overall polycule.

3.7 Sorting Networks

The previous subsection introduced gadgets whose output edges $(12_{O1}, 12_{O2}, \ldots, 12_{Om})$ each capture the truth value of one clause from φ by being blue *or* green if the clause can be satisfied, but restricted to green slots if it cannot. The green ambiguity will be resolved by a gadget that applies *Tension* to the system by forcing a subset of k of the 12_O edges to be scheduled in blue slots in any valid schedule (Section 3.8). However, this Tension gadget requires us to pick a *fixed* k-subset; to be able to capture the difference between *any* k clauses being potentially blue and at most $k-1$ being blue, we build a *sorting network* gadget. This moves edges scheduled in green slots to the right, which in turn moves edges scheduled in blue slots to the left. We can then apply tension to the leftmost k outputs to obtain a polycule which is schedulable iff the corresponding φ has at least k simultaneously satisfiable clauses.

Sorting networks have two components: *wires*, which carry data, and *comparators*[3], which compare two inputs a and b, re-ordering them if necessary. More specifically, the left output always contains $\max\{a, b\}$ and the right output $\min\{a, b\}$. Note that since we are sorting Boolean values, we can compute these as $a \vee b$ and $a \wedge b$, respectively.

---

[3] sometimes called modules or comparator modules

Figure 2 includes the sorting network for 4 inputs where the wires are represented by *channels* of 12_O edges, and comparators by *SWAP* gadgets (introduced below). We use the simple $\Theta(m^2)$-size insertion/bubble sort network [27, §5.3.4]. While asymptotically better sorting networks are obviously available, this simple method is sufficient for our reduction as we are only aiming for polynomial time overall. We give the general construction here for reference.

▶ **Definition 3.11** (DPS sorting networks). *Our DPS sorting network takes m input 12_O edges arranged in vertical channels which are connected by layers of SWAP nodes and terminate with output 12_O edges. 12_{O1} and 12_{O2} are connected by 1 SWAP node in layer 0, 12_{O2} and 12_{O3} are connected by 2 SWAP nodes in layers 1 and −1, and 12_{Ok} and $12_{O(k+1)}$ are connected by k SWAP nodes in layers $k-1, k-3, \ldots, 1-k$.*

The goal of the rest of this subsection is to establish the following lemma for the sorting network:

▶ **Lemma 3.12** (Sorting Network gadget schedules). *Consider a DPS sorting network with m each of input 12_O edges and output 12_O edges.*

(a) *(sufficient input) for all ℓ, every slot-respecting schedule in which the leftmost ℓ output edges are scheduled in blue slots must schedule at least ℓ input edges in blue slots.*

(b) *(sorted output) for any assignment $C : [m] \to \{green, blue\}$ of colours to input edges such that at least ℓ are blue, there exists a slot-respecting schedule S for the sorting Network that assigns in which the leftmost ℓ output edges are scheduled in blue slots.*

The proof relies on the *SWAP* gadgets, which in turn require a few auxiliary gadgets, so we introduce the latter first before we return to Lemma 3.12 in Section 3.7.6.

3.7.1   12-Edge Duplicator

We begin by introducing our first auxiliary gadget: a frequency-12 edge duplicator, D_{12}, shown in Figure 7.

▶ **Lemma 3.13** (12-duplicator gadget schedules). *Any slot-respecting schedule must yield a local schedule for the D_{12} gadget where all edges labelled 12_O are scheduled in slots of the same colour.*

Proof. Nodes I and D_{12} each have tasks $[3_R, 6_B, 6_P, 6_{\overline{O}}, 12_O, 12_{O'}]$ and density $D = 1$, so each task with frequency f must appear exactly once in every f day period in the schedules for either node. Further, tasks 3_R, 6_B, and 6_P are indirectly connected to the True Clock such that their partial schedules must either be of the form

$$[3_R, 6_B, \_\_, 3_R, \_\_, 6_P, 3_R, 6_B, \_\_, 3_R, \_\_, 6_P], \quad \text{or}$$

$$[3_R, \_\_, \_\_, 3_R, 6_b, 6_P, 3_R, \_\_, \_\_, 3_R, 6_B, 6_P].$$

Note that all empty slots in both schedules are either blue or green.

Assume towards a contradiction that the $6_{\overline{O}}$ edge is scheduled in a mixture of blue and green slots and note that, as it is also scheduled twice in each 12-day period, its constraint must be violated. This means that either the $6_{\overline{O}}$ edge is consistently green and all 12_O and $12_{O'}$ edges are blue, or the $6_{\overline{O}}$ edge is consistently blue and all 12_O and $12_{O'}$ edges are green. Either way, all 12_O and $12_{O'}$ edges are scheduled in slots of the same colour.   ◀
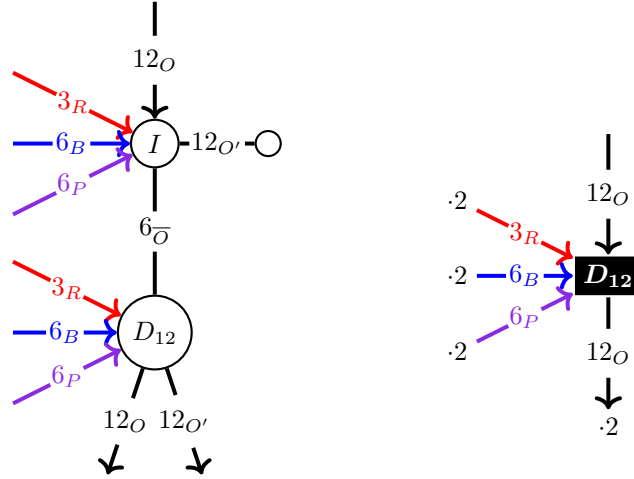
Figure 7 A gadget for duplicating input edges with frequency 12 (left), and a shorthand version (right). Note that while it can be easily modified to have three 12_O outputs, two are sufficient for our purposes. Also note that the 12_O outputs will not both be scheduled concurrently with the 12_O input, merely in a slot of the same colour (green or blue).

3.7.2 OR2 Gadget

Next, we introduce the $\vee$ gadget, shown in Figure 8. Note that the surrounding construction in the sorting layer provides weaker guarantees than for the OR gadget of the clause layer, requiring a slightly different approach.

▶ **Lemma 3.14** ($\vee$ gadget schedules). *Any slot-respecting schedule must yield a local schedule for the $\vee$ gadget where the $12_{O\vee}$ edge is either blue or green. Further, if the $12_{O\vee}$ edge is blue then either 12_{O1}, 12_{O2}, or both are also blue.*

Proof. Consider the node labelled "$\vee$" which, by Figure 8, has tasks $[3_R, 6_P, 6_S, 12_S, 12_{\vee'}, 12_{O1}, 12_{O2}]$ and density $D = 1$, so each task with frequency f must appear exactly once in every f day period. Further, 3_R and 6_P are indirectly connected to the True Clock such that partial schedules must be of the form

$$[3_R, \_\_, \_\_, 3_R, \_\_, 6_P, 3_R, \_\_, \_\_, 3_R, \_\_, 6_P].$$

Suppose that neither 12_{O1} nor 12_{O2} is blue – instead, schedule both in the remaining green slots. This forces partial schedules to be of the form

$$[3_R, \_\_, \underline{12_O}, 3_R, \_\_, 6_P, 3_R, \_\_, \underline{12_{O'}}, 3_R, \_\_, 6_P],$$

that is, schedules where all remaining slots are blue, including the slot assigned to $12_{\vee'}$.

Now consider the alternative, scheduling some 12_O input edges (12_{O1} or 12_{O2} or both) in blue slots. Under this assumption, partial schedules must be of the form

$$[3_R, \underline{12_O}, \_\_, 3_R, \_\_, 6_P, 3_R, \_\_, \_\_, 3_R, \_\_, 6_P] \quad \text{or}$$

$$[3_R, \_\_, \_\_, 3_R, \underline{12_O}, 6_P, 3_R, \_\_, \_\_, 3_R, \_\_, 6_P].$$

Note that either case allows schedules where $12_{\vee'}$ is green, and also schedules where $12_{\vee'}$ is blue. Thus it is always possible for $12_{\vee'}$ to be scheduled in blue slots, but if 12_{O1} or 12_{O2} are scheduled in blue slots then $12_{\vee'}$ must be scheduled in green slots.
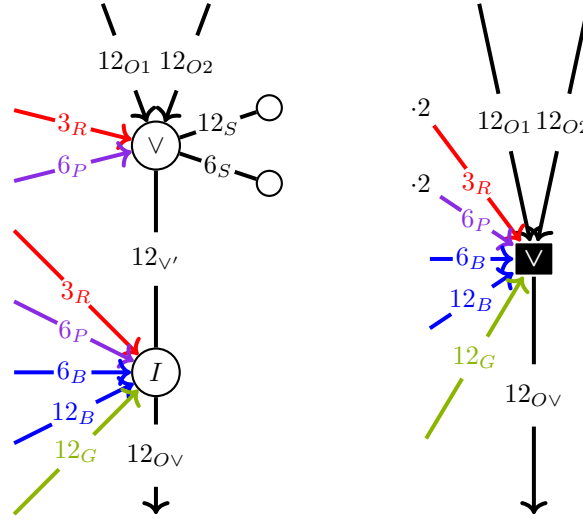
■ **Figure 8** A gadget for computing $y_1 \vee y_2$ (left), along with a shorthand version (right). 12_{O1}, 12_{O2}, and $12_{O\vee}$ can always be scheduled in green slots, but may be scheduled in blue slots only if the corresponding logical term is assigned True. Again, ambiguity introduced by True logical terms being scheduled in green slots will be addressed in Section 3.8. Note that while Figure 6 also calculates a logical OR, its incoming edges both have different frequencies and a different mapping from truth values to edge colours.

Now consider schedules for node I, which has tasks $[3_R, 6_P, 6_B, 12_B, 12_G, 12_{\vee'} 12_{O\vee}]$, and has density $D = 1$, with the same implication. Partial schedules for I have two free slots in each 12-day period, with all other slots taken by 3_R, 6_P, 6_B, 12_B, and 12_G – all of which must be scheduled in slots of their corresponding colour due to their indirect connections to the True Clock. These free slots are blue and green, so either $12_{\vee'}$ is blue and $12_{O\vee}$ must be green, or $12_{\vee'}$ is green and $12_{O\vee}$ must be blue.

It, therefore, follows that if both 12_{O1} and 12_{O2} are green then $12_{\vee'}$ will be blue and $12_{O\vee}$ must be green, while if either 12_{O1} or 12_{O2} are scheduled in blue slots then $12_{\vee'}$ and $12_{O\vee}$ may both be scheduled in either green or blue slots.   ◄

3.7.3   AND2 Gadget

We will now introduce the $\wedge$ gadget, shown in Figure 9.

▶ **Lemma 3.15** ($\wedge$ gadget schedules). *Any slot-respecting schedule must yield a local schedule for the $\wedge$ gadget where the $12_{O\wedge}$ edge is either blue or green. Further, if the $12_{O\wedge}$ edge is blue then both 12_{O1} and 12_{O2} must also be blue.*

Proof. Consider the node labelled "$\wedge$" in Figure 9, which has tasks $[3_R, 6_P, 6_\wedge, 12_{O1}, 12_{O2}, 12_S, 12_{S'}]$ and density $D = 1$, forcing each task with frequency f to appear exactly once in every f day period. Further, 3_R and 6_P are indirectly connected to the True Clock such that partial schedules must be of the form $[3_R, \_\_, \_\_, 3_R, \_\_, 6_P, 3_R, \_\_, \_\_, 3_R, \_\_, 6_P]$.

Assume towards a contradiction that the $6_\wedge$ edge is scheduled in a mixture of blue and green slots. Under this assumption, either $6_\wedge$ occurs more than once in some 6-day periods, or its constraint must violated. Thus, it must be scheduled consistently in either blue slots or green slots.
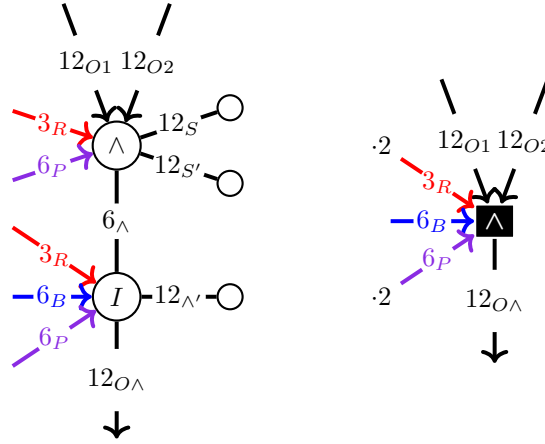
Figure 9 A gadget for computing $y_1 \wedge y_2$ (left), along with a shorthand version (right). Here, 12_{Oi} will be scheduled in blue or green slots if y_i is True and restricted to green slots if y_i is False. Similarly, $12_{O\wedge}$ will be scheduled in blue or green slots if $y_1 \wedge y_2$ is True and restricted to green slots otherwise. As with other such gadgets, ambiguities resulting from edges corresponding to True variables being scheduled in green slots will be addressed in Section 3.8.

Suppose that some 12_O edge, either 12_{O1} or 12_{O2}, is scheduled in a green slot. In this case, partial schedules for $\wedge$ will either be of the form

$$[3_R, \_\_, \underline{12_O}, 3_R, \_\_, 6_P, 3_R, \_\_, \_\_, 3_R, \_\_, 6_P] \quad \text{or}$$

$$[3_R, \_\_, \_\_, 3_R, \_\_, 6_P, 3_R, \_\_, \underline{12_O}, 3_R, \_\_, 6_P].$$

Either way, $6_\wedge$ must then be scheduled in a blue slot to avoid constraint violations. If we instead suppose that both 12_{O1} and 12_{O2} are scheduled in blue slots, then two valid schedules for $\wedge$ are

$$[3_R, \underline{6_\wedge}, \underline{12_S}, 3_R, \underline{12_{O1}}, 6_P, 3_R, \underline{6_\wedge}, \underline{12_{S'}}, 3_R, \underline{12_{O2}}, 6_P] \quad \text{and}$$

$$[3_R, \underline{12_{O1}}, \underline{6_\wedge}, 3_R, \underline{12_{O2}}, 6_P, 3_R, \underline{12_S}, \underline{6_\wedge}, 3_R, \underline{12_{S'}}, 6_P],$$

demonstrating that in this case 6_P may be scheduled in either blue or green slots.

Now consider the node labelled I, which has tasks $[3_R, 6_B, 6_P, 6_\wedge, 12_{O\wedge}, 12_{O\wedge'}]$, and density $D = 1$ with the same implication as above for the $\wedge$ node. Again, partial schedules of the coloured nodes (3_R, 6_B, and 6_P) are restricted by their indirect connections to the True Clock, and must either be of the form

$$[3_R, 6_B, \_\_, 3_R, \_\_, 6_P, 3_R, 6_B, \_\_, 3_R, \_\_, 6_P] \quad \text{or}$$

$$[3_R, \_\_, \_\_, 3_R, 6_B, 6_P, 3_R, \_\_, \_\_, 3_R, 6_B, 6_P].$$

In either case, if $6_\wedge$ is scheduled in blue slots then only green slots remain for $12_{O\wedge}$ and $12_{O\wedge'}$. Likewise, if $6_\wedge$ is scheduled in green slots then $12_{O\wedge}$ and $12_{O\wedge'}$ must be scheduled in the remaining blue slots. Thus $12_{O\wedge}$ must be either green or blue, and further can only be blue if $6_\wedge$ is green, which is only possible if both 12_{O1} and 12_{O2} are blue, as claimed.   ◀

3.7.4   Slot Splitting Gadgets

Several of the gadgets introduced in this section have constant input edges which are scheduled in slots with a specific colour, but with a larger frequency than those produced by the gadgets introduced in Section 3.5. These can be produced by the simple slot-splitter gadgets shown in Figure 10.
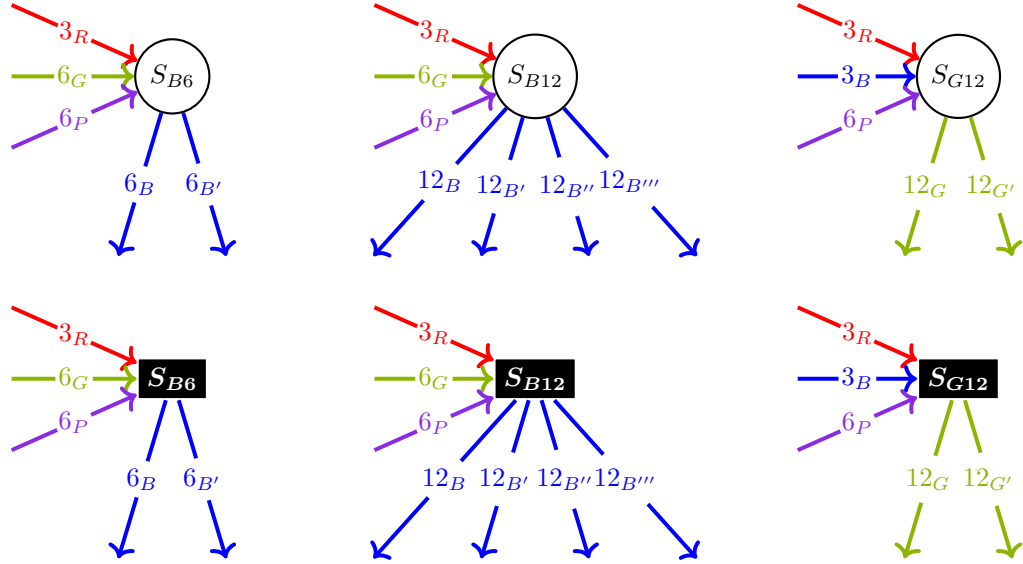
■ **Figure 10** Gadgets for generating 6_B edges (top left) and 12_B edges (top centre), and 12_G edges (top right), with shorthand versions below. Excess 6_B, 12_B, and 12_G edges should be connected to pendant nodes.

▶ **Lemma 3.16** (Slot splitting gadget schedules). *Any slot-respecting schedule must yield a local schedule for the slot splitting gadgets where 6_B, 12_B, and 12_G edges produced by S_{B6}, S_{B12}, and S_{G12} nodes must be scheduled in blue, blue, and green slots, respectively.*

Proof. In addition to their respective output edges, the S_{B6} and S_{B12} nodes each have 3_R, 6_G, and 6_P input edges, which are connected indirectly to the True Clock such that they must be scheduled in slots of their respective colour, with partial schedules of the form: $[3_R,$ __, 6_G, 3_R, __, 6_P, 3_R, __, 6_G, 3_R, __, $6_P]$. In these schedules, all remaining slots are blue, so the output edges (6_B and $6_{B'}$ for the S_{B6} node, and 12_B, $12_{B'}$, $12_{B''}$, $12_{B'''}$ for the S_{B12} node) must be scheduled in blue slots.

Similarly, the S_{G12} node has 3_R, 3_B, and 6_P input edges, and partial schedules of the form $[3_R, 3_B,$ __, $3_R, 3_B, 6_P, 3_R, 3_B,$ __, $3_R, 3_B, 6_P]$, where, as above, all remaining slots are green, so the output edges (12_G and $12_{G'}$) must be scheduled in green slots.   ◀

3.7.5   SWAP Gadgets

With these preparations, we are finally able to build the *SWAP* gadgets which act as comparators in the sorting network. The internal structure of *SWAP* gadgets is shown in Figure 11.

▶ **Lemma 3.17** (SWAP gadget schedules). *Any slot-respecting schedule must yield a local schedule for the SWAP gadget with the following properties:*
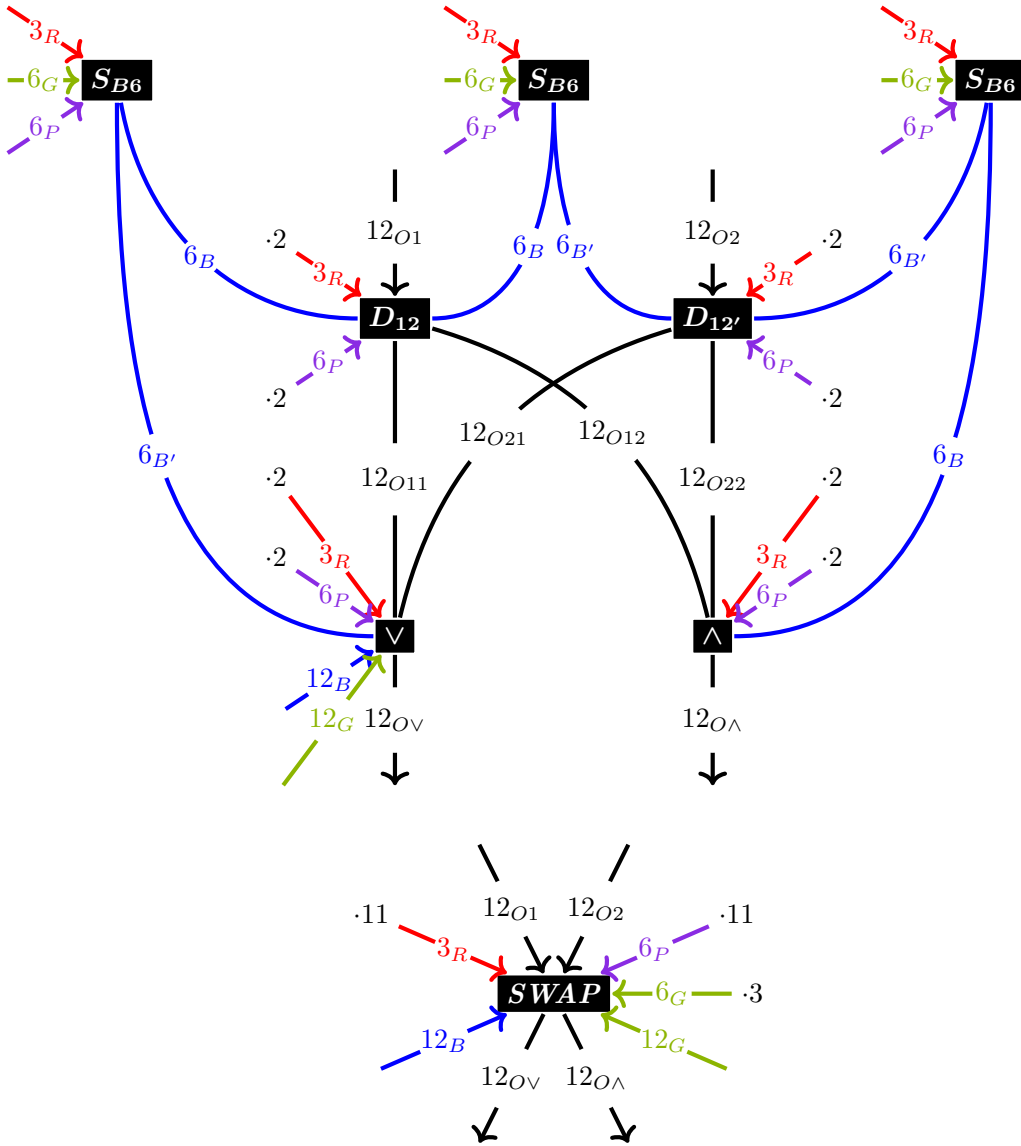
Figure 11 A gadget for comparing and re-ordering two input edges (top), and a shorthand version (bottom). The 6_B, 12_B, and 12_G edges used by $SWAP$ nodes come from slot splitting gadgets, which are not shown because they are shared between multiple $SWAP$ nodes. If the gadget has as many outputs as it has inputs, 12_{O1} and 12_{O2} must be scheduled in blue or green slots and $12_{O\vee}$ and $12_{O\wedge}$ will match them in all cases save one: if 12_{O1} is green and 12_{O2} is blue, $12_{O\vee}$ will be blue and $12_{O\wedge}$ will be green.

- *Both output edges, $12_{O\vee}$ and $12_{O\wedge}$, are scheduled in either green or blue slots.*
- *If $12_{O\vee}$ is blue, then either 12_{O1}, 12_{O2}, or both are blue.*
- *If $12_{O\wedge}$ is blue, then both 12_{O1} and 12_{O2} are blue.*

Note that this is again a weaker invariant than a true $\vee$ and $\wedge$; $SWAP$ correctly sorts its inputs, sending green to the right and blue to the left, but it is allowed to "swallow" a blue value, turning it green. Fortunately, this is good enough for our purposes, as we will go on to demonstrate.

Proof of Lemma 3.17. Assume a slot-respecting global schedule S is given, i.e., all coloured edges are scheduled in slots of their respective colours. At D_{12}, the 12_{O11} and 12_{O12} edges must be scheduled in slots of the same colour as 12_{O1}; similarly, at $D_{12'}$, edges 12_{O21} and 12_{O22} must be scheduled in slots of the same colour as 12_{O2} (both by Lemma 3.13). By Lemma 3.14, $12_{O\vee}$ must be green or blue, and can only be blue if at least one of its inputs, 12_{O11} and 12_{O21}, is also blue. Similarly, by Lemma 3.15, $12_{O\wedge}$ must be green or blue, and further can only be blue if both of its inputs, 12_{O12} and 12_{O22}, are also blue. ◄

3.7.6   Sorting Network Schedules

We now finally show that under the assumptions enforced on the schedule by the surrounding gadgets, our sorting network correctly groups green edges on the right, with enough blue edges on the left to suit our purposes.

Proof of Lemma 3.12 (page 18). (a) Assume a slot-respecting schedule S that schedules the leftmost ℓ output edges in blue slots. Then, starting at the outputs of the sorting network, move backwards through the sorting network, one $SWAP$ gadget at a time (sweeping from bottom to top in Figure 2). We will show by induction that while moving through the sorting network in this way, there are always at least ℓ blue edges between all channels. Most importantly, this holds at the inputs – implying the claim.

The inductive basis at the output level is true by assumption. When moving over each $SWAP$ gadget, we replace the two outputs of that $SWAP$ with its two inputs. By Lemma 3.17, there are several different valid colour combinations for the inputs, but we always have at least as many blue values among the inputs as are among the outputs: If both outputs are green or blue, this is obvious. If exactly one output is blue it will either be the $12_{O\vee}$ input (implying that at least one input is blue), or the $12_{O\wedge}$ input (implying that both inputs are blue, but the $\vee$ gadget chose a green output). Hence the number of blue values cannot drop when moving over a $SWAP$ gadget.

(b) We build the local slot-respecting schedule inductively by again moving through the sorting network one $SWAP$ node at a time, only now we will move forwards (that is, top to bottom in Figure 2). We will also restrict S to the special case S', where each $SWAP$ node has as many blue output 12_O edges as it has blue input 12_O edges (possible due to Lemma 3.17). Note that not all schedules satisfying the lemma will necessarily have this property, but it is sufficient to show that one such schedule exists.

In this class of schedules S', $SWAP$ gadgets behave exactly like the comparator modules in a sorting network for binary inputs: if both inputs are blue then both outputs will be blue, if both inputs are green then both outputs will be green, and if exactly one input is blue then the $12_{O\vee}$ output will be blue and the $12_{O\wedge}$ output will be green (all by Lemma 3.17). By the correctness of the insertion/bubble sort network, we hence end up with an output layer with exactly ℓ blue edges on the left, followed by $m - \ell$ green edges on the right. ◄

3.8   Tension

Lemma 3.12 assumes that the leftmost k output edges of a sorting network must be scheduled in blue slots – we ensure this by connecting Tension gadgets (shown in Figure 12) to these output edges.
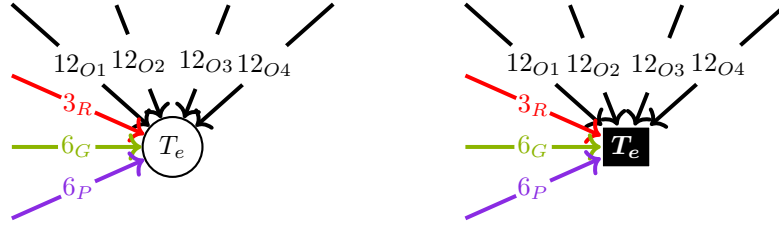
■ **Figure 12** A gadget (left) which applies tension to four inputs, ensuring that either 12_{O1}, 12_{O2}, 12_{O3}, and 12_{O4} are scheduled in blue slots or no schedule can be found for T_e. To apply tension to less than 4 inputs, connect any unneeded inputs to its own pendant node.

▶ **Lemma 3.18** (Tension gadget schedules). *Any slot-respecting schedule must yield a local schedule for a tension gadget T_e where each 12_O edge must be scheduled in blue slots.*

Proof. The single node of each tension gadget T_e, as shown in Figure 12, has tasks $[3_R, 6_G, 6_P, 12_{O1}, 12_{O2}, 12_{O3}, 12_{O4}]$ and density $D = 1$, so each task with frequency f must appear exactly once in every f day period. Further, the 3_R, 6_G, and 6_P edges are indirectly connected to the True Clock such that they must be scheduled in slots of their respective colours. Partial schedules for T_e nodes must therefore be of the form $[3_R, \_\_, 6_G, 3_R, \_\_, 6_P, 3_R, \_\_, 6_G, 3_R, \_\_, 6_P]$. All empty slots in schedules of this form are blue, so the remaining $12_{O1}, 12_{O2}, 12_{O3}, 12_{O4}$ edges must be scheduled in blue slots. ◀

3.9   Correctness Proof of Reduction

With these preparations we can, at long last, prove Lemma 3.1, and hence complete the proof of Theorem 1.3.

Proof of Lemma 3.1 (page 9). Consider a polycule $\mathcal{P}_{d\varphi k}$ built from some 3-CNF formula φ using the algorithm defined by Definition 3.4.

First, suppose that there is a variable assignment $v : X \to \{\text{True}, \text{False}\}$ such that clauses $c_{i_1}, \ldots, c_{i_k}$ evaluate to True under v ($1 \leq i_1 < i_2 < \cdots < i_k \leq m$). We construct a slot-respecting schedule S for $\mathcal{P}_{d\varphi k}$ as follows: At the variable gadget for each $x_i \in X$, schedule 3_{iR} in red slots if $v(x_i) = \text{True}$ and in the blue slots otherwise. This fixes a schedule for all edges in the variable and duplication layers (by Lemma 3.7 and Lemma 3.8). In the clause layer, we schedule the 12_O output edges of the OR gadgets for clauses c_{i_j}, $j = 1, \ldots, k$, in blue slots, and schedule all other 12_O output edges in green slots (Noting that, by Lemma 3.9, this yields a valid schedule for all OR gadgets). In the sorting layer, we now have at least k blue inputs and at most $m - k$ green inputs. By Lemma 3.12–(b), we can extend this schedule to the sorting layer such that the leftmost k output edges of the sorting layer are scheduled in blue slots. This then also yields a valid schedule for the Tension gadgets (Lemma 3.18). Overall, this shows that a schedule S for $\mathcal{P}_{d\varphi k}$ exists.

Now assume that we are given a schedule S for $\mathcal{P}_{d\varphi k}$. By applying Lemma 3.6 to the True Clock, we can assign coloured slots to S. It then follows from the construction of $\mathcal{P}_{d\varphi k}$ that

S must be slot-respecting. Set $v(x_i) = \text{True}$ if S schedules 3_{iR} in red slots and $v(x_i) = \text{False}$ otherwise. Next, show that v satisfies at least k clauses in φ: By Lemma 3.18, the k leftmost output edges of the sorting layer must all be scheduled in blue slots. By Lemma 3.12−(a), this means that there are (at least) k output edges of the clause layer that are scheduled in blue slots, say those for clauses $c_{i_1}, \dots, c_{i_k}$ for some $1 \le i_1 < i_2 < \cdots < i_k \le m$. Considering this along with Lemma 3.9 implies that for each c_{i_j}, at least one input is scheduled in a red slot. By the definition of v, this means that clauses $c_{i_1}, \dots, c_{i_k}$ all evaluate to True under v, and the claim follows.

It remains to argue that our reduction can be realised in polynomial time. The size of the polycule $\mathcal{P}_{d\varphi k}$ is clearly polynomial in the size of the formula, with the quadratic sorting network contributing the most persons. All other gadgets, including the *SWAP* nodes composing the sorting network, have constant size and thus are easy to implement in polynomial time.   ◀

<div align="center">∗   ∗   ∗   ∗   ∗   ∗</div>

4   Computational Complexity

One proof of the NP-hardness of the Decision Poly Scheduling (DPS) Problem is that it contains Pinwheel Scheduling as a special case, an NP-hard problem [23]. We show in Section 5 that OPS also contains the Chromatic Index problem as a special case, which gives another proof of the NP-hardness of DPS using the conversion in 2.1. Since all good things come in threes, our inapproximability result in Section 3 gives a third independent proof of NP-hardness by reducing 3SAT to DPS.

Upper bounds on the the complexity of DPS are much less clear. Similar to other periodic scheduling problems, the characterizing the computational complexity of Poly Scheduling is complicated by the fact that there are feasible instances that require an exponentially large schedule. It is therefore not clear whether Decision Poly Scheduling is in NP since no succinct Yes-certificates are known; this is unknown even for the more restricted Pinwheel Scheduling Problem [25].

The following simple algorithm shows that DPS is at least in PSPACE (see also [14], [26]): Given the polycule $\mathcal{P}_d = (P, R, f)$ with $|P| = n$ and $|R| = m$, construct the *configuration graph* $\mathcal{G}_c = (V, E)$, where V consists of "countdown vectors" listing for each edge e how many days remain before e has to be scheduled again. E has an edge for every maximal matching M in $\mathcal{M}(P, R)$, and leads to a successor configuration where all $e \in M$ have their urgency reset to $f(e)$ and all $e \notin M$ have their countdown decremented. Feasible schedules for $\mathcal{P}_d$ correspond to infinite walks in the finite $\mathcal{G}_c$, and hence must contain a cycle. Conversely, any cycle forms a valid periodic schedule. Our algorithm for DPS thus checks in time $O(|V| + |E|)$ whether $\mathcal{G}_c$ contains a cycle.

The configuration graph $\mathcal{G}_c$ has single exponential size: $V = \{(u_e)_{e \in R} : u_e \in [0..f(e)]\}$ and E has an edge for every matching in (P, R). So $|E| \le |V| \cdot 2^m$ (since we have at most $2^{|E|}$ matchings) and $|V| \le \prod_{e \in R} f(e)$. To further bound this, we use that all $f(e)$ need to be encoded explicitly in binary in the input. $\prod_{e \in R} f(e) \le \prod_{e \in R} 2^{|f_e|} = 2^{\sum |f_e|} \le 2^N$ for N the size of the encoding of the input.

To obtain a PSPACE algorithm, we use the polylog-space s-t-connectivity algorithm (using Savich's Theorem on the NL-algorithm that guesses the next vertex in the path) on $\mathcal{G}_c$, computing the required part of the graph on-the-fly when queried; this yields overall polynomial space.

5 Unweighted Poly Scheduling & Edge Coloring

Given an OPS instance $\mathcal{P}_o = (P, R, g)$, one can always obtain a feasible schedule from a proper *edge colouring* $c : E \to [C]$ of the graph (P, R): any round-robin schedule of the C colours is a valid schedule for $\mathcal{P}_o$, and the number of colours becomes the separation between visits. More formally, we can define a schedule S via $S(t) = \{e \in R : c(e) \equiv t \pmod{C}\}$. An example is shown in Figure 13.
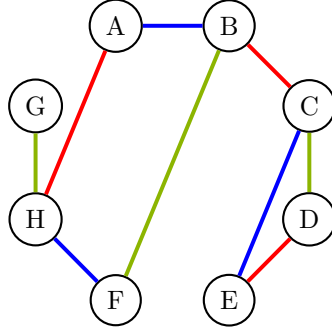
Figure 13 An unweighted polyamorous scheduling instance (that is, an OPS instance where all edges have growth rate 1). Edge colours show one optimal schedule, where every edge is visited exactly every three days: [3, 3, 3], i.e., all red edges are scheduled on days t with $t \equiv 0 \pmod 3$, all blue edges when $t \equiv 1 \pmod 3$ and green edges for $t \equiv 2 \pmod 3$.

Such a schedule can yield an arbitrarily bad solution to general instances of $\mathcal{P}_o$, but it gives optimal solutions for a special case: The non-hierarchical polycule $\mathcal{P}_u$, which is an OPS polycule where all growth rates are $g_{i,j} = 1$ (i.e., an unweighted graph).

Recall that any graph with maximal degree Δ can be edge-coloured with at most $\Delta + 1$ colours and clearly needs at least Δ colours.

▶ **Proposition 5.1** (Unweighted OPS = edge coloring). *An unweighted OPS problem admits a schedule with heat h if and only if the corresponding graph is h-edge-colourable.*

Proof. First note that any k-edge-colouring immediately corresponds to a schedule that visits every edge every k days, since we can schedule all edges e with $c(e) = i$ on days $t \equiv i \pmod C$. Moreover, any schedule with height h must visit every edge at least once within the first h days (otherwise it would grow to desire $> h \cdot 1$). We can therefore assign h colours according to these first h days of the schedule; some edges might receive more than one colour, but we can use any of these and retain a valid colouring using h colours. ◀

Since it is NP-complete to decide whether a graph has chromatic index Δ even when the graph is 3-regular [22], unweighted Poly Scheduling is NP-hard. This provides a second restricted special case of the problem that is NP-hard. This observation also gives us the inapproximability result stated in Theorem 1.4:

Proof of Theorem 1.4 (page 6). Assume that there is a polynomial-time algorithm A that achieves an approximation ratio of $\frac{4}{3} - \varepsilon$ for some $\varepsilon > 0$. Given an input (V, E) to the 3-Regular Chromatic Index Problem (i.e., given a 3-regular graph, decide whether $\chi_1(G) = 3$), we can apply A to (V, E, g), setting $g(e) = 1$ for all $e \in E$. By Proposition 5.1, A finds an edge colouring with $c \leq (\frac{\Delta+1}{\Delta} - \varepsilon) \cdot \chi_1(G)$ colours. If $\chi_1(G) = \Delta$, then $c \leq \Delta + 1 - \varepsilon\Delta < \Delta + 1$, so $c = \Delta$; if $\chi_1(G) = \Delta + 1$, then $c \geq \Delta + 1$. Comparing c to Δ thus determines $\chi_1(G)$

exactly in polynomial time; in particular, for every 3-regular graph, this decides whether $\chi_1(G) = 3$. Since 3-Regular Chromatic Index is NP-complete, it follows that P = NP.   ◄

We close this section with the remark that there are weighted DPS instances where any feasible schedule must "multi-colour" some edges, including the polycule shown in Figure 14. For the general problem, we thus cannot restrict our attention to edge colourings (though they may be a valuable tool for future work).
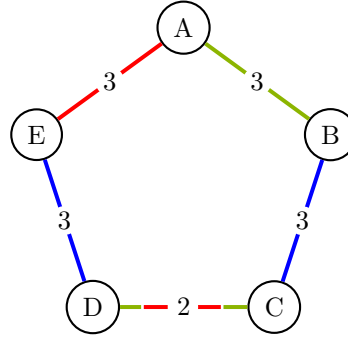
Figure 14 A discrete polyamorous scheduling instance which is solvable only by assigning multiple colours to the CD edge

6   Approximation Algorithms

In this section, we present two efficient polynomial-time approximation algorithms for Poly Scheduling, thereby proving Theorems 1.5 and 1.6. Throughout this section, we assume a fixed instance $\mathcal{P}_o = (P, R, g)$ of Optimisation Polyamorous Scheduling (OPS) is given.

6.1   Lower Bounds

We first collect a few simple lower bounds used in the analysis later; note that Section 7 has further lower bounds.

▶ **Lemma 6.1** (Simple lower bound). *Given an OPS instance $\mathcal{P}_o = (P, R, g)$, set $g_{\min} = \min_{e \in R} g(e)$, $g_{\max} = \max_{e \in R} g(e)$, and $\Delta = \max_{p \in P} \deg(p)$. Any periodic schedule for $\mathcal{P}_o$ has heat $h \geq \max\{\Delta \cdot g_{\min}, g_{\max}\}$.*

Proof. The chromatic number χ_1 of the unweighted graph (P, R) is $\chi_1 \in \{\Delta, \Delta + 1\}$. This means that under any periodic schedule, some edge desires will grow to at least to $\chi_1 \cdot g_{\min} \geq \Delta \cdot g_{\min}$, since we cannot schedule any two edges incident to a degree-Δ node on the same day. Moreover, we cannot prevent the weight-$g_{\max}$ edge from growing to heat $g_{\max}$.   ◄

A second observation is that the lower bound for any subset of the problem is also a lower bound for the problem as a whole:

▶ **Lemma 6.2** (Subset bound). *Given two OPS instances $\mathcal{P}_o = (P, R, g)$ and $\mathcal{P}'_o = (P, R', g')$ with $R' \subseteq R$ and $g(e) = g'(e)$ for all $e \in R'$, i.e., $\mathcal{P}'_o$ results from $\mathcal{P}_o$ by dropping some edges. Assume further that any schedule for $\mathcal{P}'_o$ has heat at least h^*. Then, any schedule for $\mathcal{P}_o$ also has heat at least h^*.*

Proof. Suppose there is a schedule S for $\mathcal{P}_o$ of heat $h' < h$. We obtain a schedule S' for $\mathcal{P}'_o$ by dropping all edges $e \notin R'$. (The resulting schedule may have empty days.) By construction, when using S' to schedule $\mathcal{P}'_o$, all edges in R' will grow to the same heat as in $\mathcal{P}_o$ under S, and hence also to heat $h' < h$. ◀

6.2 Approximation for Almost Equal Growth Rates

We first focus on a special case of OPS instances with "almost equal weights", which is used as base for our main algorithm. Let the edge weights satisfy $g_{\min} \leq g(e) \leq g_{\max}$ for all $e \in R$. We will show that scheduling a proper edge colouring round-robin gives a $\frac{\Delta+1}{\Delta} \cdot \frac{g_{\max}}{g_{\min}}$ approximation algorithm, establishing Theorem 1.5.

Proof of Theorem 1.5 (page 6). We compute a proper edge colouring for (P, R) with $\Delta+1$ colours using the algorithm from [29] and schedule these $\Delta + 1$ matchings in a round-robin schedule. No edge desire will grow higher than $(\Delta + 1) \cdot g_{\max}$ in this schedule. Lemma 6.1 shows that $\mathrm{OPT} \geq \max\{\Delta \cdot g_{\min}, g_{\max}\}$. The edge-colouring schedule is thus never more than a $\min\{\frac{\Delta+1}{\Delta} \cdot \frac{g_{\max}}{g_{\min}}, \ \Delta + 1\}$ factor worse than OPT. ◀

6.3 Layering Algorithm

The colouring-based algorithm from Theorem 1.5 can be arbitrarily bad if desire growth rates are vastly different and Δ is large. For these cases, a more sophisticated algorithm achieves a much better guarantee (Theorem 1.6). The algorithm consists of 3 steps:

1. breaking the graph into layers (by edge growth rates),
2. solving each layer using Theorem 1.5, and
3. interleaving the layer schedules into an overall schedule.

Let L be a parameter to be chosen later. We define layers of $\mathcal{P}_o = (P, R, g)$ as follows. For $i = 0, \ldots, L-1$, set $\mathcal{P}_i = (P, R_i, g)$ where

$$R_i = \left\{ e \in R \ : \ \frac{g_{\max}}{2^{i+1}} < g(e) \leq \frac{g_{\max}}{2^i} \right\}.$$

Moreover, $\mathcal{P}_L = (P, R_L, g)$ with $R_L = \left\{ e \in R : g(e) \leq \frac{g_{\max}}{2^L} \right\}$.

Denote by Δ_i, for $i = 0, \ldots, L$, the maximal degree in (P, R_i). Let S_i be the round-robin-$(\Delta_i + 1)$-colouring schedule from Theorem 1.5 applied on the OPS instance $\mathcal{P}_i$. If run in isolation on $\mathcal{P}_i$, schedule S_i has heat $h_i \leq (\Delta_i + 1)g_{\max}/2^i \leq (\Delta + 1)g_{\max}/2^i$ by the same argument as in Section 6.2. Moreover, for $i < L$, S_i is a $2\frac{\Delta_i+1}{\Delta_i}$-approximation (on $\mathcal{P}_i$ in isolation); for $i = L$, we can only guarantee a $(\Delta_L + 1)$-approximation.

To obtain an overall schedule S for $\mathcal{P}$, we schedule the $L + 1$ layers in round-robin fashion, and within each layer's allocated days, we advance through its schedule as before, i.e., $S(t) = S_{(t \bmod (L+1))}\big(\lfloor t/(L+1) \rfloor\big)$. Any advance in layer i is now delayed by a factor $(L+1)$. Hence S achieves heat at most

$$\overline{h} = \max_{i \in [0..L]} (L+1) \cdot h_i \ \leq \ \max_{i \in [0..L]} (L+1)(\Delta_i + 1) \cdot \frac{g_{\max}}{2^i}$$

Using Lemma 6.2 on the layers and Lemma 6.1, we obtain a lower bound for OPT of

$$\underline{h} = \max\left\{ \max_{i \in [0..L-1]} \Delta_i \cdot \frac{g_{\max}}{2^{i+1}}, \ g_{\max} \right\}$$

We now distinguish two cases for whether the maximum in $\overline{h}$ is attained for an $i < L$ or for $i = L$. First suppose $\overline{h} = (L+1)(\Delta_i + 1)g_{\max}/2^i$ for some $i < L$. Since we also have $\underline{h} \geq \Delta_i \cdot g_{\max}/2^{i+1}$, we obtain an approximation ratio of $2(L+1)\frac{\Delta_i+1}{\Delta_i} \leq 3(L+1)$ overall in this case.

For the other case, namely $\overline{h} = (L+1)(\Delta_L+1)g_{\max}/2^L > (L+1) \cdot (\Delta_i+1)g_{\max}/2^i$ for all $i < L$, we do not have lower bounds on the edge growth rates. But we still know $\underline{h} \geq g_{\max}$, so we obtain a $(L+1)(\Delta_L+1)/2^L$-approximation overall in this case.

Equating the two approximation ratios (and using $\Delta_L \leq \Delta$) suggests to choose $L = \lg(\Delta_L + 1) - \lg 3$, which gives an overall approximation ratio of $3\lg(\Delta_L + 1) - 3\lg(3/2) \leq 3\lg(\Delta + 1) \leq 3\lg n$. This concludes the proof of Theorem 1.6.

7   Fractional Poly Scheduling

In this section, we generalize the notion of density from Pinwheel Scheduling for the Polyamorous Scheduling Problem. For that, we consider the dual of the linear program corresponding to a fractional variant of Poly Scheduling.

7.1   Linear Programs for Poly Scheduling

In the fractional Poly Scheduling problem, instead of committing to a single matching M in (P, R) each day, we are allowed to devote an arbitrary *fraction* $y_M \in [0, 1]$ of our day to M, but then switch to other matchings without cost or delay for the rest of the day (a simple form of scheduling with preemption). The heat of a fractional schedule is again defined as $\max_{e \in R} r(e)g(e)$, but the recurrence time $r(e)$ now is the maximal time in S before the fraction of days devoted to matchings containing e sum to at least 1. (For a non-preemptive schedule with one matching per day, this coincides with the definition from Section 2.)

Schedules for the fractional problem are substantially easier because there is no need to have different fractions y_M for different days: the schedule obtained by always using the average fraction of time spent on each matching yields the same recurrence times. We can therefore assume without loss of generality that our schedule is given by $S = S(\{y_M\}_{M \in \mathcal{M}})$, with $y_M \in [0, 1]$ and $\sum_{M \in \mathcal{M}} y_M = 1$. S schedules the matchings in some arbitrary fixed order, each day devoting the same y_M fraction of the day to M. Then, recurrence times are simply given by $r_S(e) = 1/\sum_{M \in \mathcal{M}:e \in M} y_M$.

With these simplifications, we can state the fractional relaxation of Optimisation Poly Scheduling instance $\mathcal{P}_o = (P, R, g)$ as an optimisation problem as follows:

$$\min \quad \overline{h} \tag{1}$$

$$\text{s.t.} \qquad\qquad \sum_{M \in \mathcal{M}} y_M \ \leq \ 1 \tag{2}$$

$$\frac{1}{\sum_{M \in \mathcal{M}:e \in M} y_M} \cdot g_e \ \leq \ \overline{h} \qquad \forall e \in R \tag{3}$$

$$y_M \ \in \ [0, 1] \qquad \forall M \in \mathcal{M} \tag{4}$$

Substituting $\overline{h} = 1/\ell$, this is equivalent to the following linear program (LP):

$$\max \quad \ell \tag{5}$$

$$\text{s.t.} \qquad\qquad \sum_{M \in \mathcal{M}} y_M \ \leq \ 1 \tag{6}$$

$$\frac{1}{g_e} \sum_{M \in \mathcal{M}:e \in M} y_M \geq \ell \qquad \forall e \in R \tag{7}$$

$$y_M \geq 0 \qquad \forall M \in \mathcal{M} \tag{8}$$

The optimal objective value ℓ^* of this LP gives $\bar{h}^* = 1/\ell^*$, the optimal fractional heat.

▶ **Lemma 7.1** (Fractional lower bound). *Consider an OPS instance $\mathcal{P}_o = (P, R, g)$ with optimal heat h^* and let $\bar{h}^* = 1/\ell^*$ where ℓ^* is the optimal objective value of the fractional-problem LP from Equation* (5)*. Then $\bar{h}^* \leq h^*$.*

Proof. We use the same approach as in [9, §3]: For any schedule S, $h(S)$ is at least the heat $h_T(S)$ obtained during the first T days only, which in turn is at least $\max_e g(e) \cdot \bar{r}(e)$ for $\bar{r}(e)$ the *average* recurrence time of edge e during the first T days. A basic calculation shows that for the fractions y_M of time spent on matching M during the first T days there exists a value $1/\ell = h(S)(1 - o(T))$, so that we obtain a feasible solution of the LP (5). Hence $1/\ell^* \leq 1/\ell = h(S)(1 - o(T))$. Since these inequalities hold simultaneously for all T, taking the limit as $T \to \infty$, we obtain $1/\ell^* = \bar{h}^* \leq h(S)$. ◀

The immediate usefulness of Lemma 7.1 is limited since the number of matchings can be exponential in n.

▶ Remark 7.2 (Randomized-rounding approximation?). One could try to use this LP as the basis of a randomized-rounding approximation algorithm, but since it is not clear how to obtain an efficient algorithm from that, we do not pursue this route here. The simple route taken in [9] cannot achieve an approximation ratio better than $O(\log n)$, so Theorem 1.6 already provides an equally good deterministic algorithm.

We therefore proceed to the dual LP of Equation (5):

$$\min \quad x \tag{9}$$

$$\text{s.t.} \quad \sum_{e \in R} z_e \geq 1 \tag{10}$$

$$\sum_{e \in M} \frac{z_e}{g_e} \leq x \quad \forall M \in \mathcal{M} \tag{11}$$

$$z_e \geq 0 \quad \forall e \in R \tag{12}$$

While still exponentially large and thus not easy to solve exactly, the dual LP yields the versatile result from Theorem 1.7.

Proof of Theorem 1.7 (page 7). Using the given z_e and $x = \max_{M \in \mathcal{M}} \sum_{e \in M} \frac{z_e}{g(e)}$, we fulfil all constraints of Equation (9). The optimal objective value x^* is hence $x^* \leq x$. By the duality of LPs, we have $x^* \geq \ell^*$ for ℓ^* the optimal objective value of Equation (5). Together with Lemma 7.1, this means $h^* \geq \bar{h}^* = 1/\ell^* \geq 1/x^* \geq 1/x$. ◀

7.2 Poly Density

Theorem 1.7 gives a more explicit way to compute the *poly density* $\bar{h}^*$ than the primal LP, but it is unclear whether it can be computed exactly in polynomial time. Given the more intricate global structure of Poly Scheduling, $\bar{h}^*$ is necessarily more complicated than the density of Pinwheel Scheduling. The most interest open problem for Poly Scheduling hence is whether a sufficiently small poly density implies the existence of a valid (integral) schedule.

Specific choices for z_e in Theorem 1.7 yield several known bounds:
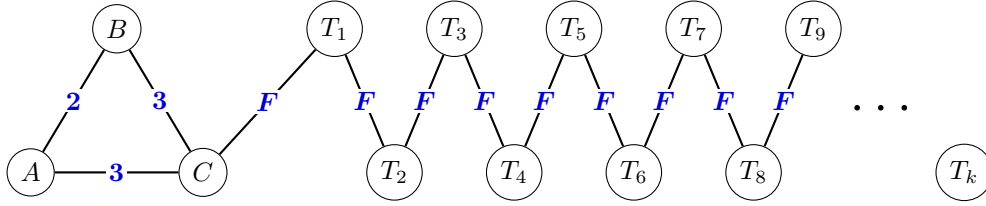
Figure 15 The tadpole family of DPS instances, defined for parameters $k \geq 0$ (tail length) and $F \geq 3$ (tail frequency). The total growth rate is $G = \frac{1}{2} + \frac{2}{3} + k \cdot 1/F = \frac{7}{6} + \frac{k}{F}$ and the size of a maximum matching is $m = 1 + \lfloor (k+1)/2 \rfloor$.

- Setting $z_e = g_e/G$ for $G = \sum_{e \in R} g_e$ yields Corollary 1.8.
- Fix any subset $R' \subseteq R$. Now set $z_e = g_e/C$ if $e \in R'$ and 0 otherwise, where $C = \sum_{e \in R'} g_e$. The maximum from Theorem 1.7 then simplifies to $\frac{1}{C} \max_{M \in \mathcal{M}} |M \cap R'|$, so

$$h^* \geq \frac{\sum_{e \in R'} g_e}{\max_{M \in \mathcal{M}} |M \cap R'|}.$$

- An immediate application of that observation with R' all edges incident at a person $p \in P$ yields the BGT bound:

▶ **Corollary 7.3** (Bamboo lower bound). *Given an OPS instance (P, R, g) and $p \in P$ with $g_1 \geq \cdots \geq g_d$ the desire growth rates for edges incident at p. Set $G_p = g_1 + \cdots + g_d$. Any periodic schedule for (P, R, g) has heat at least G_p.*

▶ Remark 7.4 (Better general bounds?). For the general case, it seems challenging to obtain other such simple bounds. The bound of G/m is easy to justify without the linear programs by a "preservation-of-mass argument": Assume a schedule S could achieve a heat $h < G/m$. Every day, the overall polycule's desire grows by G, and S can schedule at most m pairs to meet, whose desire is reset to 0 from some value $\leq h$. Every day, S thus removes only a total of $\leq mh < G$ desire units from the polycule, whereas the overall growth is G, a contradiction to the heat remaining bounded.

Note that the bound of G/m is tight for some instances, so we cannot hope for a strictly lower bound. On the other hand, the example from Figure 15 shows demonstrates that it can also be arbitrarily far from h^*.

Figure 15 shows the tadpole family of instances demonstrating the power of the dual-LP approach and Theorem 1.7. All DPS tadpoles (as shown in the figure) are infeasible since already the triangle $A-B-C$ does not admit a schedule obeying the given frequencies. The corresponding OPS instances (as given by Lemma 2.2) with $g(e) = 1/f(e)$ thus have $h^* > 1$; indeed $h^* = 4/3$ if $F \geq 2$. However, the simple lower bounds or local arguments do not detect this: (a) All local Pinwheel Scheduling instances (any person plus their neighbours) are feasible. (b) The mass-preservation bound (Corollary 1.8) is $G/m < 1$ for $k \geq 1$. Indeed, setting $F = k$ and letting $k \to \infty$, $G/m = O(1/k)$, giving an arbitrarily large gap to h^*. By contrast, consider the LP fractional lower bound. One can show that $\bar{h}^* = \frac{7}{6} > 1$ for any $k \geq 1$ and $F \geq 2$, so Theorem 1.7 correctly detects the infeasibility in this example.

▶ Remark 7.5 (Better Pinwheel density via dual LPs?). Since Poly Scheduling is a generalization of Pinwheel Scheduling resp. Bamboo Garden Trimming, we can apply Theorem 1.7 also to these problems. However, for this special case, the optimal objective value of the dual LPs is *always* $x^* = \ell^* = 1/G$ for G the G the sum of the growth rates, so we only obtain the trivial "biomass" lower bound of G for Bamboo Garden Trimming resp. the density ≤ 1 necessary

condition for Pinwheel Scheduling. The more complicated structure of matchings in non-star graphs makes fractional lower bounds in Poly Scheduling much richer and more powerful.

8   Open Problems & Future Directions

This paper opens up several avenues for future work. The most obvious open problem concerns efficient approximation algorithms: can we reduce the gap between our $\frac{13}{12}$ hardness of approximation lower bound and the $O(\log n)$ upper bound? As introduced by Conjecture 3.2, we expect that future works may demonstrate better inapproximability results for OPS in the general case; the true lower bound may even be super-constant. However, in light of our Theorem 1.6, a super-constant hardness of approximation result would have to use Poly Scheduling instances with super-constant degrees. Open Problem 1.9 will also have clear implications for OPS, as well as being interesting in its own right.

There is also interesting work to be done looking at specific classes of polycules. Bipartite polycules are particularly interesting, both for the likelihood that they will permit better approximations than are possible in the general case and for their applications (e.g., modelling the users and providers of some service).

Polyamorous scheduling has several interesting generalizations including Fungible Polyamorous Scheduling, whose decision version we define as:

▶ **Definition 8.1** (Fungible Decision Polyamorous Scheduling (FDPS)). *An FDPS instance $\mathcal{P}_{fd} = (P, R, s, f)$ (a "(fungible decision) polycule") consists of an undirected graph (P, R) where the vertices $P = \{p_1, \ldots, p_n\}$ are n classes of fungible persons and the edges R are pairwise relationships between those classes. Classes have integer sizes $s : P \to \mathbb{N}$ and relationships have integer frequencies $f : R \to \mathbb{N}$.*

The goal is find an infinite schedule $S : \mathbb{N}_0 \to 2^R$, such that

(a) *(no overflows) for all days $t \in \mathbb{N}_0$, $S(t)$ is a multiset with elements from P such that each node $p_i \in P$ appears at most $s(p)$ times, and*

(b) *(frequencies) for all $e \in R$ and $t \in \mathbb{N}_0$, we have $e \in S(t) \cup S(t+1) \cup \cdots \cup S(t+f(e)-1)$; or to report that no such schedule exists.*

FDPS also has an optimisation version, which again allows each person $p \in P$ to have at most $s(p)$ meetings each day. These problems have clear applications to the scheduling of staff and resources.

Another natural generalisation is Secure Polyamorous scheduling. Suppose that Adam is dating both Brady and Charlie, who are also dating each other. In a DPS or OPS polycule, on any day, Adam must choose to meet with either Brady or Charlie, who each face similar choices; but why can't he meet both?[4] The Secure Decision Polyamorous scheduling problem allows this:

▶ **Definition 8.2** (Secure Decision Polyamorous Scheduling (SDPS)). *An SDPS instance $\mathcal{P}_{sd} = (P, R, f)$ (a "(secure decision) polycule") consists of an undirected graph (P, R) where the vertices $P = \{p_1, \ldots, p_n\}$ are n persons, and the edges R are pairwise relationships, with integer frequencies $f : R \to \mathbb{N}$ for each relationship.*

The goal is find an infinite schedule $S : \mathbb{N}_0 \to 2^R$, such that

(a) *(no third-wheels) for all days $t \in \mathbb{N}_0$, $S(t)$ is a set of meetings between cliques of people in P in which each person appears at most once, and*

---

[4] A key part of polyamory [24]!

(b) *(frequencies) for all $e \in R$ and $t \in \mathbb{N}_0$, we have $e \in S(t) \cup S(t+1) \cup \cdots \cup S(t+f(e)-1)$; or to report that no such schedule exists.*

Again, this has a natural optimisation version.

Polyamorous Scheduling also motivates the study of several restricted versions of Pinwheel Scheduling and Bamboo Garden Trimming, including partial scheduling (wherein some portion of the schedule is fixed as part of the problem and the challenge is to find the remainder of the schedule), and fixed holidays (where the fixed part of the schedule consists of periodic gaps).

—— **References** ——————————————————————————————————————

1   P. Afshani, M. de Berg, K. Buchin, J. Gao, M. Löffler, A. Nayyeri, B. Raichel, R. Sarkar, H. Wang, and H.-T. Yang. On cyclic solutions to the min-max latency multi-robot patrolling problem. In *International Symposium on Computational Geometry (SoCG)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SOCG.2022.2.

2   A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted version of bin packing. *ACM Transactions on Algorithms*, 3(3):28–es, 2007. doi:10.1145/1273340.1273344.

3   A. Bar-Noy, J. S. Naor, and B. Schieber. Pushing dependent data in clients-providers-servers systems. *Wireless Networks*, 9(5):421–430, 2003. doi:10.1023/a:1024632031440.

4   D. Bilò, L. Gualà, S. Leucci, G. Proietti, and G. Scornavacca. Cutting Bamboo down to Size. In M. Farach-Colton, G. Prencipe, and R. Uehara, editors, *Fun with Algorithms (FUN)*, volume 157 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FUN.2021.5.

5   T. Bosman, M. van Ee, Y. Jiao, A. Marchetti-Spaccamela, R. Ravi, and L. Stougie. Approximation algorithms for replenishment problems with fixed turnover times. *Algorithmica*, 84(9):2597–2621, May 2022. URL: http://dx.doi.org/10.1007/s00453-022-00974-4, doi:10.1007/s00453-022-00974-4.

6   M. Y. Chan and F. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–462, 1993. doi:10.1007/BF01187034.

7   M. Y. Chan and F. Y. L. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Trans. Computers*, 41(6):755–768, 1992. doi:10.1109/12.144627.

8   W.-T. Chan and P. W. H. Wong. On-line windows scheduling of temporary items. In *International Symposium on Algorithms and Computation (ISAAC)*, page 259–270. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-30551-4_24.

9   S. Cicerone, G. Di Stefano, L. Gasieniec, T. Jurdzinski, A. Navarra, T. Radzik, and G. Stachowiak. Fair hitting sequence problem: Scheduling activities with varied frequency requirements. In *International Conference on Algorithms and Complexity (CIAC)*, page 174–186. Springer, 2019. doi:10.1007/978-3-030-17402-6_15.

10   L. Clemmons, K. Anderson, and V. Gerry. Robin hood (movie). *Walt Disney Productions*, 1973.

11   W. Ding. A branch-and-cut approach to examining the maximum density guarantee for pinwheel schedulability of low-dimensional vectors. *Real-Time Systems*, 56(3):293–314, 2020. doi:10.1007/s11241-020-09349-w.

12   E. A. Feinberg and M. T. Curry. Generalized pinwheel problem. *Math. Methods Oper. Res.*, 62(1):99–122, 2005. doi:10.1007/s00186-005-0443-4.

13   P. C. Fishburn and J. C. Lagarias. Pinwheel scheduling: Achievable densities. *Algorithmica*, 34(1):14–38, 2002. doi:10.1007/s00453-002-0938-9.

14   L. Gąsieniec, B. Smith, and S. Wild. Towards the 5/6-density conjecture of pinwheel scheduling. In C. A. Phillips and B. Speckmann, editors, *Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 91–103. SIAM, Jan. 2022. doi:10.1137/1.9781611977042.8.

15 L. Gąsieniec, T. Jurdziński, R. Klasing, C. Levcopoulos, A. Lingas, J. Min, and T. Radzik. Perpetual maintenance of machines with different urgency requirements. *Journal of Computer and System Sciences*, 139:103476, Feb. 2024. URL: `http://dx.doi.org/10.1016/j.jcss.2023.103476`, `doi:10.1016/j.jcss.2023.103476`.

16 L. Gąsieniec, R. Klasing, C. Levcopoulos, A. Lingas, M. Jie, and T. Radzik. *Bamboo Garden Trimming Problem*, volume 10139 of *Lecture Notes in Computer Science*. Springer, 2017. `doi:10.1007/978-3-319-51963-0`.

17 C.-C. Han and K.-J. Lin. Scheduling distance-constrained real-time tasks. In *Proceedings Real-Time Systems Symposium*. IEEE Comput. Soc. Press, 1992. `doi:10.1109/REAL.1992.242649`.

18 F. Höhne and R. van Stee. A 10/7-approximation for discrete bamboo garden trimming and continuous trimming on star graphs. In *Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.APPROX/RANDOM.2023.16`.

19 R. Holte, A. Mok, A. Rosier, I. Tulchinsky, and I. Varvel. The pinwheel: a real-time scheduling problem. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume II: Software Track*, volume 2, pages 693–702 vol.2, 1989. `doi:10.1109/HICSS.1989.48075`.

20 R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science*, 100(1):105–135, 1992. `doi:10.1016/0304-3975(92)90365-M`.

21 R. Holte, L. E. Rosier, I. Tulchinsky, and D. A. Varvel. Pinwheel scheduling with two distinct numbers. *Theor. Comput. Sci.*, 100(1):105–135, 1992. `doi:10.1016/0304-3975(92)90365-M`.

22 I. Holyer. The np-completeness of edge-coloring. *SIAM Journal on computing*, 10(4):718–720, 1981.

23 T. Jacobs and S. Longo. A new perspective on the windows scheduling problem. *coRR*, 2014. `arXiv:1410.7237`.

24 John_Threepwood. Why not both? / Why don't we have both?, Aug 2011. URL: `https://knowyourmeme.com/memes/why-not-both-why-dont-we-have-both`.

25 A. Kawamura. Proof of the density threshold conjecture for pinwheel scheduling. In *Symposium on Theory of Computing (STOC)*, 2024. URL: `https://www.kurims.kyoto-u.ac.jp/~kawamura/pinwheel/paper_e.pdf`.

26 A. Kawamura and M. Soejima. Simple strategies versus optimal schedules in multi-agent patrolling. *Theoretical Computer Science*, 839:195–206, Nov. 2020. `doi:10.1016/j.tcs.2020.07.037`.

27 D. E. Knuth. *The Art Of Computer Programming: Searching and Sorting*. Addison Wesley, 2nd edition, 1998.

28 S. Lin and K. Lin. A pinwheel scheduler for three distinct numbers with a tight schedulability bound. *Algorithmica*, 19(4):411–426, 1997. `doi:10.1007/PL00009181`.

29 J. Misra and D. Gries. A constructive proof of vizing's theorem. In *Information Processing Letters*. Citeseer, 1992.

30 M. van Ee. A 12/7-approximation algorithm for the discrete bamboo garden trimming problem. *Operations Research Letters*, 49(5):645–649, Sept. 2021. `doi:10.1016/j.orl.2021.07.001`.

31 V. G. Vizing. The chromatic class of a multigraph. *Cybernetics*, 1(3):32–41, 1965.