Proof of the Density Threshold Conjecture for Pinwheel Scheduling*

Akitoshi Kawamura
kawamura@kurims.kyoto-u.ac.jp
Kyoto University
Kyoto, Japan

ABSTRACT

In the pinwheel scheduling problem, each task i is associated with a positive integer a_i called its period, and we want to (perpetually) schedule one task per day so that each task i is performed at least once every a_i days. An obvious necessary condition for schedulability is that the density, i.e., the sum of the reciprocals $1/a_i$, not exceed 1. We prove that all instances with density not exceeding $\frac{5}{6}$ are schedulable, as was conjectured by Chan and Chin in 1993. Like some of the known partial progress towards the conjecture, our proof involves computer search for schedules for a large but finite set of instances. A key idea in our reduction to these finite cases is to generalize the problem to fractional (non-integer) periods in an appropriate way. As byproducts of our ideas, we obtain a simple proof that every instance with two distinct periods and density at most 1 is schedulable, as well as a fast algorithm for the bamboo garden trimming problem with approximation ratio $\frac{4}{3}$.

CCS CONCEPTS

• **Mathematics of computing** → **Combinatoric problems**; *Combinatorial algorithms*; • **Theory of computation** → **Scheduling algorithms**; *Packing and covering problems*; *Rounding techniques*; • **Computer systems organization** → *Real-time systems*; • **Applied computing** → *Industry and manufacturing*.

KEYWORDS

pinwheel scheduling, density, bamboo garden trimming

ACM Reference Format:
Akitoshi Kawamura. 2024. Proof of the Density Threshold Conjecture for Pinwheel Scheduling. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3618260.3649757

*This research was supported by JSPS KAKENHI (Grants-in-Aid for Scientific Research) JP20H00587 and JP23K28036 and by Royal Society International Exchanges IES\R1\191184. Preliminary announcements of some results in this work appeared at the 2022 IEICE General Conference [24] and IPSJ SIG on Algorithms [25, 26].

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649757

1 INTRODUCTION

In *pinwheel scheduling* (PS) [15], we are given k recurring tasks, each of which must be performed with a given frequency. Specifically, each task $i \in [k] = \{1, \dots, k\}$ must be scheduled at least once every a_i days, where a_i is the *period* of task i. We want to schedule the tasks, one per day, so that each task is performed at (at least) its requisite frequency. Thus, an instance of PS is a nonempty array $(a_i)_{i\in[k]} = (a_1, \dots, a_k)$ of positive integers, which we assume to be arranged in non-decreasing order, and we seek to find a *schedule* $S\colon \mathbb{Z} \to [k]$ (with $S(t)$ specifying the task performed on day t) satisfying, for all $i \in [k]$, the *frequency condition*:

> for each $m \in \mathbb{Z}$, there exists a day $t \in [m, m + a_i) \cap \mathbb{Z}$ such that $S(t) = i$.

An instance for which a schedule exists is said to be *schedulable*. For example, the instances $(3, 3, 3)$, $(2, 4, 8, 8)$, and $(3, 4, 5, 8)$ are schedulable, but all become non-schedulable if any one period is decreased. A schedule $S\colon \mathbb{Z} \to [4]$ for $(3, 4, 5, 8)$ is given by

$$S(t) = \begin{cases} 1 & \text{for } t \equiv 0, 3, 6, \\ 2 & \text{for } t \equiv 1, 5, \\ 3 & \text{for } t \equiv 2, 7, \\ 4 & \text{for } t \equiv 4 \pmod 8. \end{cases}$$

For an instance $A = (a_i)_{i\in[k]}$ to be schedulable, the condition that its *density*

$$\mathrm{D}(A) = \sum_{i\in[k]} \frac{1}{a_i}$$

be at most 1 is clearly necessary, but not sufficient. For example, $(2, 3, a_3)$ is non-schedulable for all values of a_3. On the other hand, it is relatively easy to see that any instance with a density of at most $\frac{1}{2} = 0.5$ is schedulable [15, Corollary 3.2]. The challenge of improving this sufficient condition has been taken up by several authors, who succeeded in increasing the bound to $0.66\dots$ [5], to 0.7 [6, Theorem 4.2], and then to 0.75 [10, Theorem 1]. It has been conjectured [5] that this value could be increased to $\frac{5}{6} = 0.83\dots$ (which is the best possible because of the instance $(2, 3, a_3)$ mentioned above), and this conjecture has been confirmed in a number of special cases: when A has three (or fewer) distinct period values [19, Theorem 4], when for each period value there are at least five tasks having that period [4, Theorem 3], when the smallest period a_1 is 2 [10, Theorem 2], and when the number k of tasks is ≤ 12 [13]. We resolve this conjecture affirmatively:

Theorem 1. *If a PS instance A, comprised of positive integers, satisfies $\mathrm{D}(A) \le \frac{5}{6}$, then A is schedulable.*

The reason for the explicit restriction to integers is that, in the following section, we extend the PS problem so that we allow periods to be positive real numbers, not just integers. This extension plays a key role in our proof of Theorem 1.

The condition $\mathrm{D}(A) \le 1$, which is obviously necessary for any instance A, has been shown also to be sufficient for instances A containing only two distinct periods [16, Corollary 4.9]. Our idea of fractional periods yields a simple proof of this statement (for possibly non-integer periods as well). That is, we (re)prove:

Theorem 2 ([16]). *If a PS instance A has at most two distinct period values, and $\mathrm{D}(A) \le 1$, then A is schedulable.*

Bamboo garden trimming (BGT) [11] is an optimization version of PS: in a grove of k bamboo plants, plant $i \in [k]$ grows in height at a daily rate $h_i \in \mathbb{N}$; each day (at a fixed time of day) we select one plant to be trimmed (i.e., reduced to height 0) with the goal of keeping the highest plant in the grove as low as possible. Clearly, a trimming schedule achieves $K \in \mathbb{N} \setminus \{0\}$ (i.e., ensures perpetually an overall grove height of at most K) if and only if it satisfies the PS instance $(\lfloor K/h_i \rfloor)_{i\in[k]}$. There have been efficient algorithms for BGT with approximation ratios 2 [12], 1.88... [8, Corollary 1], 1.71... [22, Corollary 1], 1.60... [11, Theorem 3], and 1.42... [14, Section 3]. We use Theorem 1, and ideas used in its proof, to obtain an improved approximation ratio $\frac{4}{3} = 1.33\ldots$.

In what follows, we first note in Section 1.1 some basic facts regarding the decision of schedulability, then review in Section 1.2 other related work. In Section 2 we discuss the extension of PS to non-integer periods and note some basic properties of this extension, which, as we pause to note, allow a simple proof of Theorem 2. We then use the extension to prove our main theorem (Theorem 1) in Section 3, after which we discuss BGT in Section 4.

1.1 Deciding schedulability

Although solutions to the PS problem are infinite schedules, the following discussion shows [15, Theorem 2.1] that any schedulable instance $A = (a_i)_{i\in[k]}$ has a schedule that repeats a finite sequence of tasks (thus justifying the term "pinwheel" scheduling).

The elements of $[a_1] \times \cdots \times [a_k]$ are called *states*; if, at the end of a day, we find ourselves in a state $(u_i)_{i\in[k]}$, this means that task i must be performed no later than u_i days from now. For a task $j \in [k]$ and two states $u = (u_i)_{i\in[k]}$ and $u' = (u'_i)_{i\in[k]}$, we write $u \vdash^j u'$ if

$$u'_i = \begin{cases} a_i & \text{if } i = j, \\ u_i - 1 & \text{otherwise.} \end{cases}$$

This means that if we are in state u on a given day, then performing task j on the next day will bring us into state u'. We write $u \vdash u'$ if $u \vdash^j u'$ for some j. An instance A is thus schedulable if and only if its *state graph*, i.e., the directed graph with states as nodes and the relation $\vdash$ as edges, admits an infinite walk, or equivalently, contains a cycle. This can be checked in polynomial space [15, Corollary 2.2], but the number of states is in general exponential in the size of the input A. It is unknown whether or not the problem of deciding schedulability is in NP.

This gives rise to some subtlety in discussing algorithms that are supposed to "output" a schedule: a general schedule is an infinite object, and even if we focus on repeating schedules, we cannot hope for a polynomial-time algorithm to always write out the whole repeating pattern. One (somewhat informal) way to formulate efficient scheduling in this context is to require that we can, given an instance, generate in polynomial time a *fast online scheduler* (FOLS), i.e., a program that efficiently computes which task to perform each day [15]. Many of the results about PS or BGT, including ours in Section 4, give such algorithms.

1.2 Related work

Our PS problem is perhaps the most basic among various settings for scheduling recurring tasks (such as monitoring, maintenance, or replenishment) that must be performed with sufficient frequencies. Some generalizations immediately present themselves: one might suppose that (a fixed number of) multiple tasks can be performed each day [1, 2], that different tasks require different lengths of time to perform [9], or that the tasks are placed on graphs or metric spaces so that moving between them takes some time or cost [7, 11].

The frequency condition for PS is that task i must be scheduled at least once in the interval $[m, m + a_i)$ starting on any day m. We could consider a somewhat simpler problem [3, 4, 20] where instead we require this condition only when the starting day m is an integer multiple of a_i. One may also consider variants in which (there can be days to which no task is assigned, and) each task i must be scheduled *precisely* once (rather than *at least* once) in a_i days [17, 21, 23].

The *packing*-style problem considered in this paper (in which the tasks are packed into $\mathbb{Z}$) is complementary to a *covering*-style variant of the problem, in which task i may only be scheduled *at most once* in a_i days. For the covering problem, there is a conjecture similar to Theorem 1 (namely, that any instance of integer-valued periods with a density of at least 1.26... suffices to cover $\mathbb{Z}$) whose status remains unresolved [18].

The class of problems including ours and the above variants, or a subclass thereof depending on the context, has been also called *windows scheduling* or *periodic scheduling*.

2 FRACTIONAL PERIODS

As noted above, an essential component of our argument is to extend the allowed values of periods from integers to real numbers. We do so by requiring that a task i with possibly non-integer period a_i be performed at least $\lfloor r/a_i \rfloor$ times during any r-day interval. Thus, the new frequency condition is that

> for each $r \in \mathbb{N}$ and $m \in \mathbb{Z}$, there exist at least $\lfloor r/a_i \rfloor$ values of $t \in [m, m + r) \cap \mathbb{Z}$ such that $S(t) = i$.

For integer-valued a_i this reduces to the frequency condition stated at the beginning of the paper, but for non-integer a_i it acquires new significance. For example, a period $a_i = \frac{7}{2}$ requires that task i be performed at least once every 4 days *and* at least twice every 7 days; this could be achieved if we schedule task i every week on Monday and Thursday, but not if we schedule it on Monday and Wednesday, or if we simply schedule it every 4 days.

One advantage of this extension is that it does not affect the validity of the basic properties of schedulability stated in the following lemma, which have been used, explicitly or implicitly, in previous studies as well (for integer periods). We write $A \sqcup B$ for the

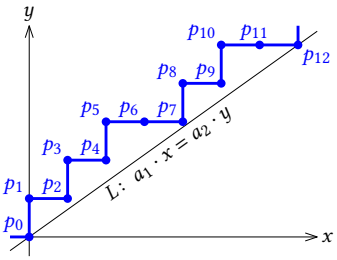

Figure 1: Sequence of lattice points $(p_t)_{t\in\mathbb{Z}}$, tracking a line L with slope a_1/a_2, for the case $a_1 = \frac{12}{7}$, $a_2 = \frac{12}{5}$. This gives the schedule that repeats the 12-day cycle of performing tasks 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1.

instance consisting simply of tasks in A and tasks in B; for example, $(6, 6) \sqcup (4, 4, 6) = (4, 4, 6, 6, 6)$.

Lemma 3. *If $A \sqcup (a)$ is schedulable, then so are*

(1) *$A \sqcup (b)$, for any period $b \geq a$ (monotonicity), and*
(2) *$A \sqcup (\underbrace{a \cdot q, \ldots, a \cdot q}_{q})$, for any positive integer q (partitioning).*

Proof. (1) Simply schedule the period-b task on the days reserved for the period-a task.
(2) Simply schedule the q new tasks, in sequence, on the days reserved for the period-a task. □

This is the only preparation we need to give a concise proof of Theorem 2, which we pause to state here (though it is not needed for Theorem 1).

Proof of Theorem 2. By the partitioning property of Lemma 3, we may assume we have just one task with each period value, i.e., our instance is $A = (a_1, a_2)$ with $1/a_1 + 1/a_2 \leq 1$. Using the monotonicity property, we may further assume that $1/a_1 + 1/a_2 = 1$.

For each $t \in \mathbb{Z}$, the line $\{ (x, y) \in \mathbb{R}^2 : x + y = t \}$ intersects the line $L = \{ (x, y) \in \mathbb{R}^2 : a_1 \cdot x = a_2 \cdot y \}$ at the point $(t/a_1, t/a_2)$; let

$$p_t = \left(\left\lfloor \frac{t}{a_1} \right\rfloor, \left\lceil \frac{t}{a_2} \right\rceil \right)$$

be the closest lattice point above and to the left of this intersection (Figure 1). The sequence $(p_t)_{t\in\mathbb{Z}}$ defines a trajectory of lattice points lying just above and to the left of line L. Now consider the schedule $S\colon \mathbb{Z} \to [2]$ defined by traversing this sequence of lattice points and performing task 1 or 2 for each horizontal or vertical line segment traversed, i.e.,

$$S(t) = \begin{cases} 1 & \text{if } p_{t+1} - p_t = (1, 0), \\ 2 & \text{if } p_{t+1} - p_t = (0, 1). \end{cases}$$

This satisfies the frequency condition, since the number of times task $i \in \{1, 2\}$ is performed during an r-day interval $[m, m + r) \cap \mathbb{Z}$ is the ith component of

$$p_{m+r} - p_m = \left(\left\lfloor \frac{m+r}{a_1} \right\rfloor - \left\lfloor \frac{m}{a_1} \right\rfloor, \left\lceil \frac{m+r}{a_2} \right\rceil - \left\lceil \frac{m}{a_2} \right\rceil \right),$$

which is $\geq \lfloor r/a_i \rfloor$. □

3 PROOF OF THEOREM 1

By repeatedly applying the procedures of Lemma 3 *in reverse*, a non-schedulable instance may be transformed into a non-schedulable instance with only small periods. Indeed, consider the following process which can be applied to any instance whose largest period a exceeds a threshold $\theta > 0$:

- if the instance contains only one task with period a, decrease its period to the second largest period $> \theta$, or to θ if a is the only period $> \theta$;
- if the instance contains more than one tasks with period a, replace two of them with a single task with period $a/2$.

Given an instance A and $\theta > 0$, every second application of this process reduces the number of periods $> \theta$; thus, after a finite number of steps, we obtain an instance $fold_\theta(A)$ containing only periods $\leq \theta$. For example, $fold_{22}(3, 4, 8, 17, 42, 55, 72) = (3, 4, 8, 13.75, 17)$.

Lemma 4. *For an arbitrary PS instance A and arbitrary $\theta > 0$,*

(1) *If A is non-schedulable, then $fold_\theta(A)$ is also non-schedulable.*
(2) *Any period in $fold_\theta(A)$ with a value $\leq \theta/2$ is in A as well.*
(3) *$\mathrm{D}(fold_\theta(A)) < \mathrm{D}(A) + 1/\theta$.*

Proof. The above operations used in turning A into $fold_\theta(A)$

(1) preserve non-schedulability by Lemma 3,
(2) never create periods $\leq \theta/2$ newly,
(3) and never increase the difference between the density and the reciprocal of the largest period. □

With this, to prove the schedulability of instance A in Theorem 1, it suffices to establish the schedulability of $fold_\theta(A)$ for some θ. In fact, it turns out that the choice $\theta = 22$ allows us to reduce Theorem 1 to exhaustive analysis of a finite number of instances:

Lemma 5. *Any instance $B = (b_i)_{i\in[k]}$ whose periods are integers less than 22 and which satisfies $\mathrm{D}'(B) < \frac{5}{6} + \frac{1}{22}$ is schedulable, where*

$$\mathrm{D}'(B) = \sum_{i\in[k]} \begin{cases} 1/b_i & \text{for } b_i < 11, \\ 1/(b_i + 1) & \text{for } b_i \geq 11. \end{cases}$$

Proof. We need only check (by computer) the schedulability of finitely many instances B, using the state graph method in Section 1.1 (and some techniques to implement it efficiently [13]). Details will appear in a full version of this paper. □

Proof of Theorem 1. Suppose that there is a non-schedulable instance A with integer-valued periods and a density of at most $\frac{5}{6}$. By Lemma 4, the instance $fold_{22}(A)$ is also non-schedulable, consists of integers ≤ 11 and real numbers between 11 and 22, and has density less than $\mathrm{D}(A) + \frac{1}{22} \leq \frac{5}{6} + \frac{1}{22}$. Now construct a new instance B from $fold_{22}(A)$ by retaining all periods ≤ 11 and replacing other periods a by $\lceil a \rceil - 1$. By the monotonicity property of Lemma 3, B is non-schedulable. On the other hand, any period $b \in B$ produced by this replacement has a value ≥ 11 and originates from a period in $fold_{22}(A)$ with a value $\leq b + 1$, whereupon $\mathrm{D}'(B) \leq \mathrm{D}(fold_{22}(A)) < \frac{5}{6} + \frac{1}{22}$. This contradicts Lemma 5. □

The values 11 and 22 in Lemma 5 are the smallest possible: replacing them by 10 and 20 would make the lemma false—for example, the instance $(3, 4, 7, 10, 15)$ is non-schedulable (as can be verified by the state graph method), even though $\frac{1}{3} + \frac{1}{4} + \frac{1}{7} + \frac{1}{11} + \frac{1}{16} < \frac{5}{6} + \frac{1}{20}$.

4 APPROXIMATION ALGORITHM FOR BGT

We will provide an (efficient, in the sense discussed at the end of Section 1.1) algorithm M that, given a PS instance $(a_i)_{i\in[k]}$, either

- declares correctly that it is non-schedulable, or
- outputs a schedule for the "relaxed" instance $\left(\left\lfloor \frac{4}{3}\cdot a_i\right\rfloor\right)_{i\in[k]}$;

note that M is allowed to choose the second branch when $(a_i)_{i\in[k]}$ is non-schedulable. This implies $\frac{4}{3}$-approximation for BGT: Given a BGT instance $H=(h_i)_{i\in[k]}$, binary search for a height K such that M applied to $(\lfloor K/h_i\rfloor)_{i\in[k]}$ outputs a schedule but not if K is replaced by $K-1$ (and hence the optimal value for H is at least K). Since this schedule satisfies the relaxed PS instance $\left(\left\lfloor \frac{4}{3}\cdot\lfloor K/h_i\rfloor\right\rfloor\right)_{i\in[k]}$ (and thus $\left(\left\lfloor \frac{4}{3}\cdot K/h_i\right\rfloor\right)_{i\in[k]}$), it achieves $\frac{4}{3}\cdot K$ for the BGT instance H.

Such an algorithm M would come easily if we aim for the ratio $\frac{3}{2}$ instead of $\frac{4}{3}$: Simply declare $(a_i)_{i\in[k]}$ non-schedulable if it has density >1. Otherwise, the relaxed instance $\left(\left\lfloor \frac{3}{2}\cdot a_i\right\rfloor\right)_{i\in[k]}$ has density $\le\frac{5}{6}$ (unless $k=1$), because elementwise we have

$$\left\lfloor\frac{3}{2}\cdot a\right\rfloor\ge\frac{3}{2}\cdot a-\frac{1}{2}\ge\frac{3}{2}\cdot a-\frac{1}{4}\cdot a=\frac{5}{4}\cdot a\ge\frac{6}{5}\cdot a$$

for any integer $a>1$. Thus it is schedulable by (the underlying algorithm for) Theorem 1 (or already by the earlier density bound $\frac{3}{4}$ [10] instead of our $\frac{5}{6}$, since in fact we have $\left\lfloor\frac{3}{2}\cdot a\right\rfloor\ge\frac{4}{3}\cdot a$).

For the ratio $\frac{4}{3}$, we need a better idea, since the analogous bound $\left\lfloor\frac{4}{3}\cdot a\right\rfloor\ge\frac{6}{5}\cdot a$ holds only for $a>2$ (as can be verified by

$$\left\lfloor\frac{4}{3}\cdot a\right\rfloor\ge\frac{4}{3}\cdot a-\frac{2}{3}\ge\frac{4}{3}\cdot a-\frac{2}{15}\cdot a=\frac{6}{5}\cdot a$$

for $a\ge5$ and individually for $a=3,4$). Thus, giving exceptional treatment to period 2, our algorithm M, given a PS instance $A=(a_i)_{i\in[k]}$, proceeds as follows:

(1) If $a_1=1$, do as follows.
   - If $k>1$, declare A non-schedulable.
   - If $k=1$, output the trivial schedule that performs task 1 every day.

(2) If $a_1=2$, recursively apply M itself to the instance $A'=\left(\left\lfloor\frac{1}{2}\cdot a_i\right\rfloor\right)_{i\in[k]\setminus\{1\}}$.
   - If M says A' is non-schedulable, declare A non-schedulable.
   - If M outputs a schedule $S'\colon\mathbb{Z}\to([k]\setminus\{1\})$, then output a schedule $S\colon\mathbb{Z}\to[k]$ defined by

$$S(t)=\begin{cases}1 & \text{for } t \text{ odd},\\ S'(t/2) & \text{for } t \text{ even}.\end{cases}$$

(3) If $a_1\ge3$, yield the schedule obtained by applying the underlying algorithm of Theorem 1 to the instance $\left(\left\lfloor\frac{4}{3}\cdot a_i\right\rfloor\right)_{i\in[k]}$, which has density $\le\frac{5}{6}$ by the above argument.

Correctness of the algorithm M may be shown by induction on k, with cases (1) and (3) being trivial or already taken care of. The first branch of case (2) is justified because, if there is a schedule for A, then simply eliminating all days on which it performs task 1 yields a schedule for A'. The second branch is justified because each task $i\ne1$ is scheduled at least once in $\left\lfloor\frac{4}{3}\cdot\left\lfloor\frac{1}{2}\cdot a_i\right\rfloor\right\rfloor$ days in S', and thus in twice as many days in S, which is $\le\left\lfloor\frac{4}{3}\cdot a_i\right\rfloor$.

Acknowledgements

The author is grateful to Leszek Gąsieniec, Yusuke Kobayashi, Igor Potapov, Benjamin Smith and Sebastian Wild for valuable comments on preliminary presentations of this work. He also thanks Keita Hiroshima for helping to verify the list of schedules in Lemma 5.

REFERENCES

[1] A. Bar-Noy and R. E. Ladner. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing*, 32(4), 1091–1113, 2003.
[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), Article 28, 2007.
[3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15, 600–625, 1996.
[4] S. K. Baruah and S. Lin. Pfair scheduling of generalized pinwheel task systems. *IEEE Transactions on Computers*, 47(7), 812–816, 1998.
[5] M. Y. Chan and F. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9, 425–462, 1993.
[6] M. Y. Chan and F. Y. L. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Transactions on Computers*, 41(6), 755–768, 1992.
[7] S. Coene, F. C. R. Spieksma, and G. J. Woeginger. Charlemagne's challenge: The periodic latency problem. *Operations Research*, 59(3), 674–683, 2011.
[8] F. Della Croce. An enhanced pinwheel algorithm for the bamboo garden trimming problem. ArXiv preprint arXiv:2003.12460, 2020.
[9] E. A. Feinberg and M. T. Curry. Generalized pinwheel problem. *Mathematical Methods of Operations Research*, 62, 99–122, 2005.
[10] P. C. Fishburn and J. C. Lagarias. Pinwheel scheduling: Achievable densities. *Algorithmica*, 34, 14–38, 2002.
[11] 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, 2024.
[12] L. Gąsieniec, R. Klasing, C. Levcopoulos, A. Lingas, J. Min, and T. Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In *Proc. 43rd International Conference on Current Trends in Theory and Practice of Computer Science* (SOFSEM), 229–240, 2017.
[13] L. Gąsieniec, B. Smith, and S. Wild. Towards the 5/6-density conjecture of pinwheel scheduling. In *Proc. SIAM Symposium on Algorithm Engineering and Experiments* (ALENEX), 91–103, 2022.
[14] F. Höhne and R. van Stee. A 10/7-approximation for discrete bamboo garden trimming and continuous trimming on star graphs. In *Proc. 26th International Conference on Approximation Algorithms for Combinatorial Optimization Problems* (APPROX), Article 16, Leibniz International Proceedings in Informatics (LIPIcs) 275, 2023.
[15] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A real-time scheduling problem. In *Proc. 22nd Annual Hawaii International Conference on System Sciences*, Volume II, 693–702, 1989.
[16] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science*, 100(1), 105–135, 1992.
[17] A. P. Huhn and L. Megyesi. On disjoint residue classes. *Discrete Mathematics*, 41(3), 327–330, 1982.
[18] A. Kawamura and M. Soejima. Simple strategies versus optimal schedules in multi-agent patrolling. *Theoretical Computer Science*, 839, 195–206, 2020.
[19] S.-S. Lin and K.-J. Lin. A pinwheel scheduler for three distinct numbers with a tight schedulability bound. *Algorithmica*, 19, 411–426, 1997.
[20] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), 46–61, 1973.
[21] Z.-W. Sun. On disjoint residue classes. *Discrete Mathematics*, 104(3), 321–326, 1992.
[22] M. van Ee. A 12/7-approximation algorithm for the discrete Bamboo Garden Trimming problem. *Operations Research Letters*, 49(5), 645–649, 2021.
[23] W. D. Wei and C. L. Liu. On a periodic maintenance problem. *Operations Research Letters*, 2(2), 90–93, 1983.
[24] 河村．輪番詰込の密度閾値について． 2022年電子情報通信学会総合大会講演論文集，D-1-11，令和4年3月．
[25] 河村．輪番割当6分の5予想の解決．情報処理学会研究報告2024-AL-196(1)（第196回アルゴリズム研究会），令和6年1月．
[26] 河村．竹叢伐採の近似率の改善．情報処理学会研究報告2024-AL-197(9)（第197回アルゴリズム研究会），令和6年3月．

Received 2023-11-13; accepted 2024-02-11