THE PINWHEEL: A REAL-TIME SCHEDULING PROBLEM*

Robert Holte, Al Mok, Louis Rosier, Igor Tulchinsky, and Donald Varvel

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

Abstract

Some satellites transmit a piece of information for a set duration, then proceed with another piece of information. A ground station receiving from several such satellites and wishing to avoid data loss faces a real-time scheduling problem. "The pinwheel" is a formalization of this problem. Given a multiset A of integers = $\{a_1, a_2, ..., a_n\}$, a successful schedule S is an infinite sequence over $\{1, 2, ..., n\}$ such that any subsequence of a_i $(1 \le i \le n)$ consecutive entries ("slots") contains at least one i. The "pinwheel decision problem" concerns whether a "useful" representation of the corresponding schedule exists. We have thus far established several facts about these problems. First, if a schedule exists, there exists a cyclic schedule (hence, "pinwheel") of period no greater than $\prod_{i=1}^{n} a_i$. Exponential length cyclic schedules are often necessary; hence the scheduling problem requires time exponential in the length of the input. The decision problem is in PSPACE. The "density" of an instance is defined as $\sum_{i=1}^{n} 1/a_i$. Instances with a density of over 1.0 are impossible to schedule, always requiring more slots than exist in any given cycle length. Instances consisting solely of multiples (i.e., $i < j \Rightarrow a_i \mid a_j$) with density ≤ 1.0 are schedulable. (Note that this includes instances consisting solely of powers of the same base with density ≤ 1.0.) Densities ≤ 0.5 may always be scheduled by reduction to instances consisting solely of powers of 2. "Dense" instances are those whose density is 1.0. Our most comprehensive results concern dense instances, where we investigate subclasses where there exists polynomial time strategies for the construction of "fast online schedulers" — i.e., programs that generate the actual scheduling sequence in constant time per item scheduled.

1. Introduction

The "pinwheel" problem is motivated by the performance requirements of a ground station that processes data from a number of satellites (or mobile sensors). Now the ground station can process data from only one satellite at a time, no preemption of processing is allowed, and the time necessary for processing data from a satellite is exactly one "time unit." The following protocol is utilized to ensure that no data is lost. Each satellite may commence sending data at any time, but must repeat sending the same data for a set interval (a specified number of "time units"). Suppose the interval specified for satellite *x* is *a* "time units." Then the ground station can ensure the processing of data from satellite *x* by having a time slot assigned to service satellite *x* in *any* interval of length *a*, i.e., by making sure that no two consecutive slots assigned to servicing satellite *x* are more than *a* "time units" apart.

The pinwheel is a formalization of this problem. Given a multiset A of integers = $\{a_1, a_2, ..., a_n\}$, a successful schedule S is an infinite sequence $j_1, j_2, ...$ over $\{1, 2, ..., n\}$ such that any subsequence of a_i $(1 \le i \le n)$ consecutive entries ("slots") contains at least one i. The interpretation is that during the k^{th} "time unit," the ground station is servicing satellite j_k. For example, "1 2 1 2 ..." is a schedule for A = {2, 3}. Notice that the 1st (2nd) satellite is serviced at least once within any interval consisting of 2 or more "time units." The "pinwheel decision problem" concerns whether such a schedule exists. The "pinwheel scheduling problem" involves producing a "useful" representation of the corresponding schedule.

The pinwheel is one of a growing family of hard-real-time scheduling problems and is a special case of the *latency scheduling* problem for *sporadic processes* [Mok 83]. In the process model of real-time systems (e.g., [Liu & Layland 73], [Leung & Merrill 80], [Mok & Sutanthavibul 85]), each process is parameterized by an ordered pair (*c*, *p*) where *c*, and p are respectively the computation time and period of the process. The scheduling problem is to ensure that every process is executed in every period, i.e., the process (*c*, *p*) requests service at time = $i*p, i \ge 0$, and must complete execution by time = $(i+1)*p$. The satellites in the pinwheel problem may be modeled as processes. However, a process in this case may request service at any time, subject to the condition that two consecutive requests from the same process must be at least *p* time units apart. In the real-time scheduling terminology, the satellites in the the pinwheel problem are *sporadic processes* where the deadline and minimum separation parameters are the same. Another difference is

* This work was supported in part by U.S. Office of Naval Research Grant Nos. N00014-85-K-0117 and N00014-86-K-0763, and National Science Foundation Grant No. CCR-8711579.

0073-1129/89/0000/0693$01.00 © 1989 IEEE

that the pinwheel problem does not allow preemption of processes.

Let $\{a_1, a_2, ..., a_n\}$ be an instance of the pinwheel problem. Clearly, without loss of generality, we may assume that $a_1 \le a_2 \le ... \le a_n$. *Hence, we do so throughout this paper*. We have thus far established several facts. First, if a schedule exists, there exists a cyclic (hence, "pinwheel") schedule of period no greater than $\prod_{i=1}^{n} a_i$. A cyclic or pinwheel schedule for A is a sequence S, of finite length, over $\{1, ..., n\}$, such that S^{ω} — S concatenated to itself a countable number of times — is a schedule for A. A cyclic schedule for $A = \{2, 3\}$ is "1 2." Exponential length cyclic schedules are often necessary; hence the "pinwheel scheduling problem" — if we assume an enumeration of the cyclic schedule as output (an assumption we later reject) — requires time exponential in the length of the input. The decision problem is in PSPACE. The "density" of an instance is defined as $\sum_{i=1}^{n} 1/a_i$. Instances with a density of over 1.0 are impossible to schedule, always requiring more slots than exist in any given cycle length. Instances consisting solely of multiples (i.e., $i < j \Rightarrow a_i \mid a_j$) with density $\le$ 1.0 are schedulable via a simple greedy strategy. Densities $\le$ 0.5 may always be scheduled by reduction to instances consisting solely of powers of 2. In general, our simple greedy strategy does not solve the scheduling problem.

Our most comprehensive results concern "dense" instances — those instances whose density is 1.0. For "nondense" instances — those instances whose density is < 1.0 — we know of no *a priori* way to ascertain the exact length of a valid cyclic schedule — short of producing it. However, cycle lengths of $LCM(a_1, a_2, ..., a_n)$ are both necessary and sufficient for schedulable dense instances. It is not clear whether LCM is even an upper bound on the minimum sufficient length of cyclic schedules for nondense instances. In terms of special cases we consider dense instances A that contain only 2 or 3 distinct integers. In the case of 2 distinct integers, we show that A can always be scheduled. In the case of 3 distinct integers, we illustrate that the "pinwheel decision problem" is in PTIME. The best we can show for general dense instances is that the decision problem is in NP. We also show the problem to be NP-hard assuming a more succinct representation of the input. Unfortunately, at this time we are unable to reconcile the two proofs into a single theorem. The reduction used in the latter proof seems novel in that it makes use of a fundamental theorem in number theory concerning the distribution of primes.

Throughout this paper we are concerned with finding "useful" representations of the schedules — providing, of course, that they exist. What's really needed is not the schedule itself but a "fast online scheduler" (a FOLS) — a program that generates the scheduling sequence in constant time per item generated. Such programs would be suitable to act as drivers for the respective ground stations. From a particular cyclic schedule one can easily build a corresponding FOLS as a finite state machine. Unfortunately, since the cyclic schedules are, in general, of exponential length, this seems to be an impractical or unrealistic approach. A better approach would be to construct polynomial time program generators (PTPGs) that take as input an instance of the pinwheel problem and produce as output a corresponding FOLS — providing one exists. In light of the negative results mentioned earlier, we do not expect that a PTPG exists that will produce FOLSs for all schedulable instances. However, they might exist for classes of instances where the "pinwheel decision problem" is in PTIME. For many classes described in this paper we show just such a result.

The remainder of the paper is organized as follows. In Section 2, we explore some facts concerning general instances — as well as provide an operational definition for FOLSs. In Section 3, we examine some restricted instances where schedules always exist. In Section 4, we discuss the results we've obtained with respect to dense instances. Finally, in Section 5, we give some concluding remarks.

2. Some Facts Concerning General Instances

In this section, we present a number of relatively simple results concerning general instances of the pinwheel problem. The utilitarian value of these results will become clear later when we prove some of the more difficult theorems concerning restricted instances. These results also suggest an agenda of questions yet to be considered.

Let $A = \{a_1, ..., a_n\}$ be an instance of the pinwheel problem. Our first result illustrates that if a schedule for A exists then there exists a cyclic schedule for A whose period is exponential in the size of A.

Theorem 2.1. If $A = \{a_1, ..., a_n\}$ has a schedule then A has a cyclic schedule whose period is no greater than $\prod_{i=1}^{n} a_i$.

Proof. Let $m = \prod_{i=1}^{n} a_i$. Let $S = j_0 j_1 ...$ be an infinite schedule for A. Consider the prefix of S, $S' = j_0 j_1 ... j_{2m}$. Each slot in S' can be represented by a vector $<c_1, ..., c_n>$ where c_j, $1 \le j \le n$, denotes the number slots since the last occurrence of j. For the slots indexed from m to 2m these vectors are well defined — i.e., for each such vector each position j, $1 \le j \le n$, in the vector contains a value between 0 and $a_j - 1$. Now there are $\prod_{i=1}^{n} a_i$ such unique vectors. Hence, there exist indices s and t, $m \le s < t \le 2m$ such that slots s and t are represented by the same vector. It is now easy to see that "$j_s \dots j_{t-1}$" is a cyclic schedule for A whose length is no greater than $\prod_{i=1}^{n} a_i$. ❑

Exponential size schedules are, in general, necessary. Consider the instance $A = \{2, 4, 8, 16, ..., 2^{n-1}, 2^n, 2^n\}$. Note that A is dense. It will be shown in the next section that A is schedulable — and in the last section that any cyclic schedule for A must have a period of at least 2^n.

The next corollary follows directly from the proof of Theorem 2.1.

Corollary 2.2. The "pinwheel decision problem" is in PSPACE.

An immediate question to consider is whether the "pinwheel decision problem" is PSPACE-complete or whether it is somewhat easier. Although the pinwheel schedules themselves are, in general, exponential in length, this does not preclude the existence, say, of a PTPG that:

(1) decides whether a given instance is schedulable, **and**

(2) if so outputs a FOLS representation of a respective valid pinwheel schedule.

Recall that the FOLS is a program that will generate the scheduling sequence in constant time per item generated — and hence suitable to act as a driver for the ground station. For example, it might take the form of a program P:

```
P: $\alpha$;
   Do
     $\beta$
   forever
```

where α is an initialization code segment and β is a "simple" segment of straight-line code that can be made to run in precisely a "time unit." (We assume here that β may contain instructions within the repertoire of most assembly languages including those available on some vector machines.) Each iteration of β would cause the servicing of a single satellite; and the sequence of satellites serviced would be the schedule. Unfortunately, the algorithm in question probably does not exist — at least with respect to general instances. In the last section of this paper, we illustrate that the "pinwheel decision problem" is NP-hard with respect to dense instances — assuming a particular concise input representation. Hence, the existence of the proposed algorithm would make it likely that P = NP. There are, however, a number of instance classes for which PTPGs do exist. Examples of these are shown throughout the paper.

The existence of PTPGs that will produce FOLSs for a class of scheduling problems in fact defines a complexity class. We term this class "S-P-C" for Scheduling-Polynomial-Constant and define it formally as follows:

Definition. A scheduling problem is in S-P-C if and only if there exists a program that runs in polynomial time that determines whether a schedule exists, and if so generates a scheduler that runs in constant time per item scheduled.

Note that related scheduling classes may be defined as well, such as S-NP-C, S-PSPACE-NL, and S-P-NC. We consider S-P-C to be of greatest importance to the pinwheel problem, but the remaining classes may prove useful in approaching other scheduling problems and may lead to interesting theoretical problems.

Before concluding this section, we state an easy result concerning the density of an instance.

Theorem 2.3. Any instance whose density is greater than one cannot be scheduled.

Theorem 2.3 follows simply from observing that for $A = \{a_1, ..., a_n\}$ each i, $1 \le i \le n$, must be scheduled at least $1/a_i$ of the time. Clearly, this cannot be done for every i if $\sum_{i=1}^{n} 1/a_i > 1$.

3. Some Restricted Instance Classes

In this section, we illustrate two restricted instance classes for which simple schedules always exist. Specifically, we consider the following classes:

$$\mathbb{C}_M = \{A \mid A = \{a_1, ..., a_n\} \text{ where } i < j \Rightarrow a_i \mid a_j \text{ and } \sum_{i=1}^{n} 1/a_i \le 1\}$$

$$\mathbb{C}_{.5} = \{A \mid A = \{a_1, ..., a_n\} \text{ where } \sum_{i=1}^{n} 1/a_i \le .5\}$$

$\mathbb{C}_M$ are those instances consisting solely of multiples with density ≤ 1.0. $\mathbb{C}_{.5}$ are those instances whose density is no greater than 0.5. In each case we show that a simple "greedy" scheduling strategy can be used to schedule all instances in $\mathbb{C}_M$ and $\mathbb{C}_{.5}$.

We begin by proposing the following **SimpleGreedy** algorithm.

```
SimpleGreedy ($A = \{a_1, ..., a_n\}$)
    */ Recall that $a_1 \le a_2 \le ... \le a_n$ */
    m := $\prod_{i=1}^{n} a_i$;
    Set up a sequence of empty slots indexed 0
    through 2m – 1;
        For i := 1 to n do
            j := smallest index of an empty slot;
        Repeat
            While $slot_j$ is not empty do
                j := j–1;
                if j < 0 then
                output("cannot schedule");
                halt endif
            endwhile
            put i into slot j;
            j := j + $a_i$;
        Until j ≥ 2m
    endfor
    Assign to each slot j a vector $c_j = <c_{j1}, ..., c_{jn}>$
    where $c_{jl}$, $1 \le l \le n$, denotes the number of
    slots since the last occurrence of $l$.
    Locate indices s and t, $m \le s < t \le 2m - 1$, that
    have been assigned identical vectors.
    Delete all empty slots.
    Output the contents of $slot_s$ through $slot_{t-1}$.
End
```

It is fairly easy to see that **SimpleGreedy** cannot be used to schedule all instances that are schedulable — even in the dense case. Consider, for example A = {2, 8, 8, 12, 12, 12}. **SimpleGreedy** can, however, always be used to find a pinwheel schedule for instances in $\mathbf{C}_M$ and $\mathbf{C}_{.5}$.

Theorem 3.1. If A = {a_1, ..., a_n} is in $\mathbf{C}_M$, then **SimpleGreedy** will find a cyclic schedule for it.

Proof. Suppose A = {a_1, ..., a_n} is in $\mathbf{C}_M$. The essential observation to be made for instances in $\mathbf{C}_M$ is that the body of the **While** loop never gets executed. Hence, the occurrences of a given integer i, $1 \le i \le n$, end up exactly a_i slots apart.

Let j_i, $1 \le i \le n$, be the index of the first slot that **SimpleGreedy** assigns to i. We can conclude the theorem if we can establish for i = 1, ..., n that:

(1) $0 \le j_i \le a_i - 1$, and

(2) there do not exist integers k, l, and c, $1 \le k < l \le i$, $c \ge 0$, such that $(j_l + ca_l)$ modulo $a_k = j_k$ — thus insuring that the body of the **While** loop will not be executed during the first i iterations of the **For** loop.

The proof proceeds by showing (1) and (2) to be the invariant of the **For** loop. The proof is an induction. Clearly, $j_1 = 0$, and thus all slots whose indices are equal to 0 modulo a_1 contain 1. Both (1) and (2) are true when i = 1. Suppose then that (1) and (2) are true for 1, ..., i. Consider now the i + 1st iteration of the **For** loop.

We first show that j_{i+1}, $i + 1 \le n$, can be chosen consistent with (1). Suppose for a contradiction that it cannot — i.e., slots indexed 0 through $a_{i+1} - 1$ are all nonempty. Since $a_i \mid a_{i+1}$, slots indexed 0 through $a_i - 1$ are also nonempty. Since $1 \le k \le i$ implies that $a_k \mid a_i$ we can see that if slot s contains k that slot s + a_i also contains k. Hence, all slots are nonempty. Furthermore, slots 0 through m–1 form a schedule for {a_1, ..., a_i} where exactly m/a_k of the slots contain k, $1 \le k \le i$. Hence, $m = \sum_{k=1}^{i} m/a_k = m \sum_{k=1}^{i} 1/a_k$. We then get that $\sum_{k=1}^{i} 1/a_k = 1$ — a contradiction. We now have that (1) holds.

Given j_{i+1} suppose now that (2) fails. Again we derive a contradiction. Since (2) held for i, we conclude that $l = i+1$. Since j_{i+1} (> j_k) was empty we have that j_{i+1} modulo $a_k \ne j_k$. But since $a_k \mid a_{i+1}$, $j_{i+1} + ca_{i+1}$ modulo $a_k = j_{i+1}$ modulo a_k. An obvious contradiction results. Hence (2) holds. □

Theorem 3.1 would be considerably strengthened if it could be shown to hold for $\mathbf{C}_{M'}$ = {A ∣ A = {a_1, ..., a_n} where $1 < i \Rightarrow a_1 \mid a_i$ and $\sum_{i=1}^{n} 1/a_i \le 1$}. Unfortunately, not all members of $\mathbf{C}_{M'}$ are schedulable. A = {2, 4, 6, 12} is an example thereof. (See Section 4 for strategies of proving such facts.) Furthermore, **SimpleGreedy** fails to schedule some schedulable members of $\mathbf{C}_{M'}$ — like A = {2, 8, 8, 12, 12, 12} mentioned earlier.

Instances in $\mathbf{C}_{.5}$ can always be scheduled by reduction to instances consisting solely of powers of 2.

Corollary 3.2. If A = {a_1, ..., a_n} is in $\mathbf{C}_{.5}$, then **SimpleGreedy** can be used to find a pinwheel schedule for it.

Proof. Suppose A = {a_1, ..., a_n} is in $\mathbf{C}_{.5}$. Let B = {b_1, ..., b_n}, $b_i = 2^j$ where $2^j \le a_i < 2^{j+1}$. Note that any schedule for B is also a schedule for A. Now $\sum_{i=1}^{n} 1/a_i \le .5$ implies that $\sum_{i=1}^{n} 1/b_i \le 1.0$. Hence, B is in $\mathbf{C}_M$ and thus schedulable. □

4. Results Concerning Dense Instances

In this section, we prove a number of results concerning dense instances. We first illustrate a useful constraint on the cycle lengths of potential dense schedules. We then consider dense instances A = {a_1, ..., a_n} where A contains only 2 or 3 distinct integers. In the case of 2 distinct integers, we show that A can always be scheduled. In the case of 3 distinct integers, we illustrate a polynomial time algorithm to decide the pinwheel decision problem. PTPGs — as described in Section 2 — are given in both cases. Such algorithms might be useful since it seems likely that many satellites could be clones of one another and hence have identical periods. We then show that the pinwheel decision problem for general dense instances is in NP. For a concise input representation, we show the problem to be NP-hard. The reduction used in the NP-hardness proof seems novel in that it makes use of a fundamental theorem of number theory concerning the distribution of primes.

In general, we know that if $A = \{a_1, ..., a_n\}$ is schedulable, that it can be scheduled via a pinwheel schedule of length $\le \prod_{i=1}^{n} a_i$. We know of no *a priori* way to ascertain the exact length of a valid cyclic schedule — short of producing it. This is not the case for dense instances. In fact, for schedulable dense instances, cycle lengths of $LCM(a_1, ..., a_n)$ are both necessary and sufficient.

Lemma 4.1. Let $A = \{a_1, ..., a_n\}$ be a dense instance. Let S be a cyclic schedule for A of length m. Then each i must occur exactly m/a_i times in S. Furthermore, successive occurrences of i in S must be exactly a_i slots apart. Lastly, m must be a multiple of $LCM(a_1, ..., a_n)$.

Proof. Clearly each a_i must occur at least $\lceil m/a_i \rceil$ times in S. Because each i must occur at least $\lceil m/a_i \rceil$ times in S we get m — the length of S — to be $\ge \sum_{i=1}^{n} \lceil m/a_i \rceil$. But $m = m \sum_{i=1}^{n} 1/a_i = \sum_{i=1}^{n} m/a_i$. Hence, we get a contradiction if for any i, $\lceil m/a_i \rceil > m/a_i$. Thus each i must occur exactly m/a_i times in S. As a result, we see that m/a_i must be integral for every i and hence $LCM(a_1, ..., a_n)$ must divide m. ❑

Theorem 4.2. If $A = \{a_1, ..., a_n\}$ is dense and schedulable then A has a cyclic schedule whose length is $LCM(a_1, ..., a_n)$. Furthermore, shorter schedules for A are not possible.

Proof. Let S be a cyclic schedule for A of length m. From Lemma 4.1 it follows that $m = r\, LCM(a_1, ..., a_n)$. Hence, shorter schedules for A are not possible. Now divide S into r identical segments, each of size $LCM(a_1, ..., a_n)$. Consider an arbitrary slot k in the first segment. Suppose this slot contains i. Since the k^{th} slot of the j^{th} segment is the $k + a_i (j\, LCM(a_1, ..., a_n)/a_i)^{th}$ slot of S, it follows from Lemma 4.1 that this slot also contains i. Hence, S consists of r identical segments, each of which by itself constitutes a valid schedule for A. ❑

For dense instances $A = \{a_1, ..., a_n\}$ we know *a priori* certain properties about their schedules. Hence, we have some tools for showing that such schedules do not exist. For example, consider the instance $A = \{4, 4, 4, 6, 12\}$. Since $LCM(4, 6, 12) = 12$, A will have a schedule iff it has a cyclic schedule of length 12. Consider then a cyclic schedule for A of length 12 — slots 0 through 11. Since all the periods are even, the indices of all slots assigned the same integer must be equal modulo 2. There are a total of 6 even numbered slots and 6 odd numbered slots. Integers 1, 2, and 3 have a period of 4 and thus will each occupy 3 slots of a given value modulo 2. Integers 4 and 5 will occupy 2 and 1 slots, respectively, of a given value modulo 2. Hence, without loss of generality, we may assume that integers 1 and 2 occupy the even numbered slots and that integers 3, 4, and 5 occupy the odd numbered slots. Scheduling 1 and 2 in the even numbered slots is easy — but scheduling 3, 4, and 5 in the odd numbered slots is equivalent to finding a schedule for $A = \{2, 3, 6\}$. Such a schedule cannot exist since $GCD(2, 3) = 1$. This fact is formally proven in the next theorem.

Theorem 4.3. Let $A = \{a_1, ..., a_n\}$ be dense and schedulable. Then for every i,j, $1 \le i, j \le n$, $GCD(a_i, a_j) > 1$.

Proof. For a contradiction suppose that A is schedulable and that for some i and j $GCD(a_i, a_j) = 1$. Let S be a schedule for A. By Lemma 4.1 each occurrence of i (j) in S is a_i (a_j) slots from the previous occurrence. Hence if slot k_i (k_j) is the first slot allocated to i (j) then the slots allocated to i (j) are precisely those indexed $k_i + c\, a_i$ ($k_j + c\, a_j$) for all $c \ge 0$ — i.e., all slots whose indices are equivalent to k_i modulo a_i (k_j modulo a_j). But since $GCD(a_i, a_j) = 1$ there exist slots whose indices are simultaneously equivalent to both k_i modulo a_i and k_j modulo a_j. In order to see this, note that finding nonnegative integers c, c' for $k_i + c\, a_i = k_j + c'\, a_j$ is equivalent to solving the linear diophantine equation $c\, a_i - c'\, a_j = k_j - k_i$. This equation has nonnegative integer solutions whenever $GCD(a_i, a_j)$ divides $k_j - k_i$ [Stewart 64]. Hence $GCD(a_i, a_j) > 1$. ❑

Now let us turn our attention to the scheduling of dense instances $A = \{a_1, ..., a_n\}$ where there is a lot of repetition among the entries. Clearly, if A is composed of n identical integers, A can be scheduled. One simply constructs the pinwheel "1 2 3 ... n," since each $a_i = n$. If A is dense and composed of exactly 2 distinct integers, then it too can be scheduled. The proof of this fact is a bit more involved. Our exposition requires the following technical lemma.

Lemma 4.4. Suppose x_1 and x_2 are positive integers. Then there exist positive integer solutions to $a/x_1 + b/x_2 = 1$ iff $d = GCD(x_1, x_2) > 1$. Furthermore, if $x_1 = dy_1$ and $x_2 = dy_2$ then every solution to $a/x_1 + b/x_2 = 1$ is such that both a/y_1 and b/y_2 are integral.

Proof. Let x_1 and x_2 be positive integers. Let $d = GCD(x_1, x_2)$, $y_1 = x_1/d$, and $y_2 = x_2/d$. Rewrite the equation $a/x_1 + b/x_2 = 1$ as $ax_2 + bx_1 = x_1x_2$ — a linear diophantine equation in 2 unknowns a, b. From number theory — see e.g. [Stewart 64] — we know that all integer solutions in a, b to $ax_2 + bx_1 = x_1x_2$ are of the form $a = A + ty_1$, $b = B - ty_2$ where t is an arbitrary integer and A, B is a particular solution. One particular solution is $A = 0$, $B = dy_2$. Thus all positive integer solutions have the form $a = ty_1$, $b = dy_2 - ty_2$. It is now easy to see that a/y_1 and b/y_2 are integral — and that there are no positive solutions if $d = 1$. ❑

Lemma 4.4 gives rise to a very natural scheduling strategy for all dense instances that are composed of exactly 2 distinct numbers. The algorithm **DenseScheduler2** will find a pinwheel schedule for all such instances. **DenseScheduler2** runs in exponential time as it enumerates the pinwheel schedule. We include it because it is easy to understand. Later we illustrate how the same ideas can be used to construct a "fast" implementation satisfying the constraints laid out in Section 2. An understanding of **DenseScheduler2** will be helpful at that time.

DenseScheduler2 introduces a concept that we will use several times in the remainder of the paper: the subschedule. Assume that we have a valid cyclic schedule S, of length *l*, for instance A. Further assume that d divides *l*. One can then project S into d smaller cyclic schedules ("subschedules") $S_0, ..., S_{d-1}$ each of length *l*/d by considering the slots in S modulo d — i.e., S_i, $0 \leq i < d$, comprises the sequence of integers from S whose slot indices are equal to i modulo d. For example, the schedule S = "12131214" has two subschedules modulo 2 — S_0 = 1111" and S_1 = "2324". In what follows we'll refer to $S_0, ..., S_{d-1}$ as the subschedules for S modulo d.

```
DenseScheduler2 ($A = \{a_1, ..., a_n\}$, where A is dense and composed of 2 distinct integers $x_1$ and $x_2$)
        Let $X_1 = \{i \mid a_i = x_1\}$, $X_2 = \{i \mid a_i = x_2\}$, $a = |X_1|$, $b = |X_2|$, $d = GCD(x_1, x_2)$, $y_1 = x_1/d$ and $y_2 = x_2/d$. Note that $a/y_1$ and $b/y_2$ are both integers, that $LCM(x_1, x_2) = dy_1 y_2$, and that $a/y_1 + b/y_2 = d$.

    (1) Construct d pinwheels — $a/y_1$ ($b/y_2$) allocated to indices from $X_1$ ($X_2$) — of length $y_1 y_2$ as follows:

        On each pinwheel allocated to indices from $X_1$ ($X_2$) schedule exactly $y_1$ ($y_2$) of the indices — each occurrence of the same index being exactly $y_1$ ($y_2$) slots apart.

    (2) Expand each of the d pinwheels to size $dy_1 y_2$ by inserting $d - 1$ empty slots between each filled slot.

    (3) Construct a single pinwheel of length $dy_1 y_2$ by rotating and superimposing the d pinwheels on each other so that each slot contains exactly one index.

    Output the resulting pinwheel

End
```

Theorem 4.5. If $A = \{a_1, ..., a_n\}$ is dense and composed of 2 distinct integers, then **DenseScheduler2** will find a cyclic schedule for it.[1]

The fact that **DenseScheduler2** works follows almost directly from Lemma 4.4. The key is to note that (1) is doable (and completely fills all d pinwheels) because a/y_1 and b/y_2 are integral. Figure 1 illustrates the algorithm's intermediate as well as final result on the input A = {4, 4, 6, 6, 6}.

Suppose now that $A = \{a_1, ..., a_n\}$ is dense and composed of 3 distinct integers — x_1, x_2, and x_3. The situation is much more involved than it was for 2 distinct integers — but still tractable. For 2 distinct integers — x_1 and x_2 — the strategy revolved around scheduling the indices corresponding to periods of x_1 and x_2 on $GCD(x_1, x_2)$ different wheels. Lemma 4.3 illustrated that dense instances possess exactly the needed properties — one of which was that $GCD(x_1, x_2) > 1$ — for such a strategy to work. A natural question to ask in the case of 3 distinct numbers is whether $GCD(x_1, x_2, x_3) > 1$ is a necessary and sufficient condition for a schedule to exist. The condition is not sufficient — the instance A = {4, 4, 4, 6, 12} discussed earlier attests to this. The next lemma, however, does establish the necessity of the condition.

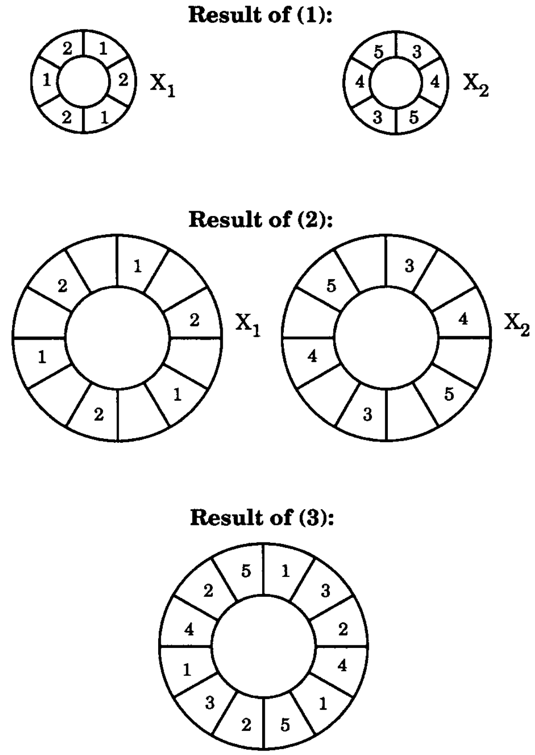


Figure 1. An illustration of how **DenseScheduler2** behaves on input A = {4, 4, 6, 6, 6}. Note here that $x_1 = 4$, $x_2 = 6$, $X_1 = \{1, 2\}$, $X_2 = \{3, 4, 5\}$, $a = 2$, $b = 3$, $d = 2$, $y_1 = 2$, and $y_2 = 3$.

[1] We have just recently discovered how to prove a similar result for nondense instances as well. It will appear in a later version of this paper.

Lemma 4.6. Let $A = \{a_1, ..., a_n\}$ be dense and composed of 3 distinct integers x_1, x_2, and x_3. If $GCD(x_1, x_2, x_3) = 1$ then A cannot be scheduled.

Proof. Suppose for a contradiction that A is dense, schedulable, composed of a x_1s, b x_2s, and c x_3s, and $GCD(x_1, x_2, x_3) = 1$. Let c_{ij}, $i \neq j$, $1 \leq i,j \leq 3$, be $GCD(x_i,x_j)$. Then:

$$x_1 = c_{12}\, c_{13}\, y_1,$$
$$x_2 = c_{12}\, c_{23}\, y_2, \text{ and}$$
$$x_3 = c_{13}\, c_{23}\, y_3,$$

for some y_1, y_2, and y_3. Note that:

$$\forall_{i,j,k,l},\ i \neq k \text{ or } j \neq l \Rightarrow GCD(c_{ij}, c_{kl}) = 1,$$
$$\forall_{i,j},\ i \neq j \Rightarrow GCD(y_i, y_j) = 1, \text{ and}$$
$$\forall_{i,j,k},\ i \neq j \text{ and } i \neq k \Rightarrow GCD(y_i, c_{jk}) = 1.$$

Hence, $LCM(x_1, x_2, x_3) = c_{12}\, c_{13}\, c_{23}\, y_1\, y_2\, y_3$ is the length of the shortest cyclic schedule S for A — Theorem 4.2.

Let us consider the subschedules for S modulo c_{12} — S_0, ..., $S_{c_{12}-1}$ — of length $c_{13}\, c_{23}\, y_1\, y_2\, y_3$. Since $c_{12} \mid x_1$ and $c_{12} \mid x_2$ each index whose period is x_1 or x_2 occupies a single subschedule — spaced $c_{13}\, y_1$ or $c_{23}\, y_2$ slots apart, respectively. Since $c_{13}\, y_1$ and $c_{23}\, y_2$ are relatively prime, no subschedule contains two indices whose periods are x_1 and x_2, respectively.

Consider any index whose period is x_3. Suppose that it first occupies slot j in S. Then it occupies all slots indexed $j + k\, c_{13}\, c_{23}\, y_3$, for $0 \leq k \leq c_{12}\, y_1\, y_2 - 1$. $y_1\, y_2$ slots are occupied by this index on each of S_0, ..., $S_{c_{12}-1}$. Since there are c indices whose period is x_3 we have that $c\, y_1\, y_2$ slots on each of S_0, ..., $S_{c_{12}-1}$ are occupied by indices whose period is x_3.

The key to the remainder of the proof is that the same number of slots remain for indices — whose periods are x_1 or x_2 — on each subschedule S_0, ..., $S_{c_{12}-1}$. Now recall that indices whose periods are x_1 or x_2 are placed on a single subschedule and that no subschedule contains two indices, one with period x_1, the other with period x_2. Thus the number of subschedules containing indices, whose period is x_1 (x_2), must divide a (b).

The number of subschedules is c_{12}. These must be allocated in the ratio of a/x_1 to b/x_2; that is, if the number of subschedules containing x_1s is w_1 and the number of subschedules containing x_2s is w_2, $w_1 = c_{12}\, (a/x_1) / (b/x_2) = ax_2\, c_{12}/bx_1 = ax_2/b\, c_{13}\, y_1$, and $w_2 = bx_1/a\, c_{23}\, y_2$. Now w_1 must divide a and w_2 must divide b. $ax_2/bc_{13}\, y_1$ divides a iff $bc_{13}\, y_1/x_2$ is an integer. Since x_2 and c_{13} are relative prime, x_2 must divide by_1. But $x_2 = c_{12}\, c_{23}\, y_2$, and y_1 is relative prime to $c_{23}\, y_2$. Therefore, $c_{23}\, y_2$ divides b; and hence c_{23} divides b. Similarly, from w_2 divides b we get that c_{13} divides a.

By symmetry using the subschedules for S modulo c_{13} and c_{23} we get that c_{23} divides c, c_{12} divides a, c_{13} divides c, and c_{12} divides b. Hence, $a = a_r\, c_{12}\, c_{13}$, $b = b_r\, c_{12}\, c_{23}$, and $c = c_r\, c_{13}\, c_{23}$. Now $a/x_1 + b/x_2 + c/x_3 = 1$ becomes:

$$\frac{a_r\, c_{12}\, c_{13}}{y_1\, c_{12}\, c_{13}} + \frac{b_r\, c_{12}\, c_{23}}{y_2\, c_{12}\, c_{23}} + \frac{c_r\, c_{13}\, c_{23}}{y_3\, c_{13}\, c_{23}} = 1, \text{ or}$$

$$\frac{a_r}{y_1} + \frac{b_r}{y_2} + \frac{c_r}{y_3} = 1 \qquad (1)$$

If $a_r/y_1 + b_r/y_2 + c_r/y_3 = 1$ has no solutions in a_r, b_r, and c_r over the natural numbers, we will have a contradiction. Multiplying through by $y_1\, y_2\, y_3$, we get $a_r\, y_2\, y_3 + b_r\, y_1\, y_3 + c_r\, y_1\, y_2 = y_1\, y_2\, y_3$. From number theory — see e.g. [Stewart 64] — we see that the diophantine equation $u_1v_1 + u_2v_2 + u_3v_3 = w$ has integral solutions in v_1, v_2, and v_3 iff $w \equiv u_3v_3$ modulo $GCD(u_1,u_2)$. Substituting a_r (b_r, c_r, $y_2\, y_3$, $y_1\, y_3$, $y_1\, y_2$, $y_1\, y_2\, y_3$, respectively) for v_1 (v_2, v_3, u_1, u_2, u_3, w, respectively), we get that $y_1\, y_2\, y_3 \equiv y_1\, y_2\, c_r$ modulo y_3. But $y_1\, y_2\, y_3 \equiv 0$ modulo y_3, hence $y_1\, y_2\, c_r \equiv 0$ modulo y_3. Since y_1 and y_2 are relatively prime to y_3, we have that c_r is a multiple of y_3. Let $c_r = ty_3$. Substituting into (1), we get

$$\frac{a_r}{y_1} + \frac{b_r}{y_2} + \frac{ty_3}{y_3} = 1 \text{ which is equivalent to } \frac{a_r}{y_1} + \frac{b_r}{y_2} + t = 1,$$

which has no solutions for natural a_r, b_r, t. □

Let $A = \{a_1, ..., a_n\}$ be dense and composed of 3 distinct integers — x_1, x_2, and x_3. If A is schedulable then A is schedulable via a pinwheel schedule S of length $d\, y_1\, y_2\, y_3$, where $d = GCD(x_1, x_2, x_3) > 1$, $y_1 = x_i/d$, $y_2 = x_2/d$, and $y_3 = x_3/d$. One can then project S into d cyclic subschedules S_0, ..., S_{d-1} each of length $y_1\, y_2\, y_3$. One can easily see that each occurrence of a given integer in S must occur in the same subschedule S_i. If in S the integers are x_j slots apart, on S_i they will be y_j slots apart. Notice also that, by Lemma 4.6, 3 integers possessing periods x_1, x_2, and x_3, respectively, cannot all end up in the same subschedule. Hence, the problem of scheduling A can be solved by partitioning $\{a_1/d, ..., a_n/d\}$ into d dense instances, A_0, ..., A_{d-1} (each A_i containing at most 2 distinct integers), finding schedules S_0, ..., S_{d-1} for each, and then folding them together to obtain S. The next lemma follows from the preceding discussion.

Lemma 4.7. Let $A = \{a_1, ..., a_n\}$ be dense and composed of 3 distinct integers x_1, x_2, and x_3. Suppose a (b, c, respectively) entries in A are equal to x_1 (x_2, x_3, respectively), $GCD(x_1, x_2, x_3) = d$, $y_1 = x_1/d$, $y_2 = x_2/d$, and $y_3 = x_3/d$. Then A is schedulable iff the multiset containing a y_1s, b y_2s, and c y_3s can be partitioned into d multisets such that each multiset is dense and composed of at most 2 distinct integers.

An algorithm that will schedule all schedulable dense instances with 3 distinct integers is **DenseScheduler3**. **DenseScheduler3** also runs in exponential time as it too enumerates the pinwheel schedule. However, the ideas within can be utilized to construct "fast" implementations as was the case with **DenseScheduler2.**

DenseScheduler3 (A = {a_1, ..., a_n}, where A is dense and composed of 3 distinct integers — x_1, x_2, and x_3.)

Let X_1 = {i | $a_i = x_1$}, X_2 = {i | $a_i = x_2$}, X_3 = {i | a_i = x_3}, a = |X_1|, b = |X_2|, c = |X_3|, and d = GCD(x_1, x_2, x_3).

(1) Partition {a_1/d, ..., a_n/d} into dense instances A_0, ..., A_{d-1} each containing at most 2 distinct integers — maintaining of course the original indices.

(2) Construct — using **DenseScheduler2** if necessary — pinwheel schedules of length LCM(x_1, x_2, x_3)/d for each of A_0, ..., A_{d-1}.

(3) Expand each of the d pinwheels, constructed in (2), by inserting d–1 empty slots between each of the filled slots.

(4) Construct a single pinwheel of length LCM(x_1, x_2, x_3) by rotating and superimposing the d pinwheels on each other so that each slot contains exactly one integer.

Output the resulting pinwheel.
End

The intent reader will find the following example instructive when trying to understand the behavior of **DenseScheduler3**. Figure 2 illustrates the algorithm's intermediate as well as final result on the input A = {8, 12, 12, 12, 12, 12, 12, 12, 24, 24, 24, 24, 24, 24, 24}.

Theorem 4.8. If A = {a_1, ..., a_n} is dense and composed of 3 distinct integers, then **DenseScheduler3** will find a cyclic schedule for it if one exists.

The fact that **DenseScheduler3** works follows from Theorem 4.5, Lemma 4.7, and the surrounding discussion. Notice that the decision of whether a schedule exists is determined in step (1) of **DenseScheduler3** — as prescribed by Lemma 4.7. Although **DenseScheduler3** runs in exponential time (because it produces an exponential length schedule), we can show that the decision step — step (1) — can be implemented in polynomial time. Hence, we obtain:

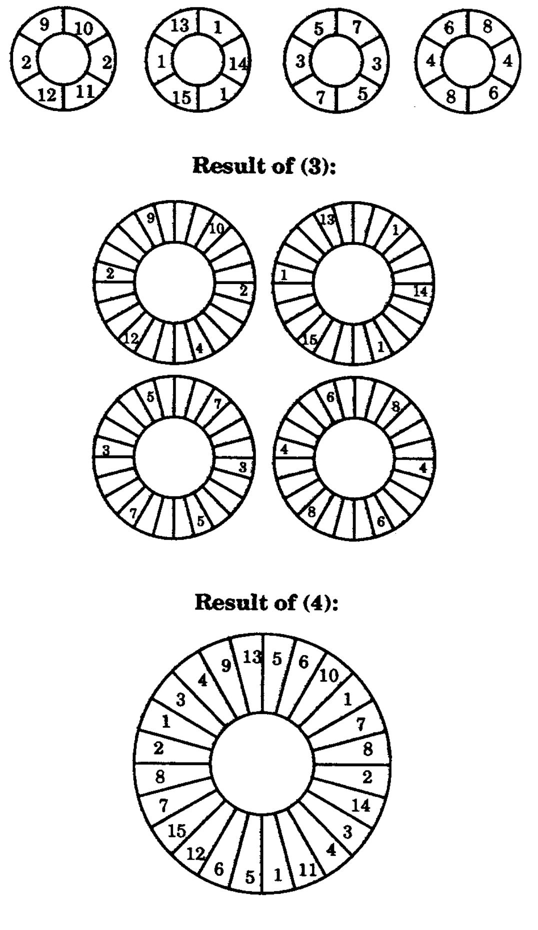
Theorem 4.9. Let $\mathfrak{C}_{d,1,2,3}$ denote the class of dense instances that are composed of no more than 3 distinct integers. Then the "pinwheel decision problem" for $\mathfrak{C}_{d,1,2,3}$ is in PTIME.

Proof. For instances with 1 or 2 distinct integers the algorithm is easy — it simply answers "yes." The proof for instances with 3 distinct integers is more involved and relies on the characterization given in Lemma 4.7.

Result of (1):

A_0 = {3, 6, 6, 6, 6} corresponding to indices 2, 9, 10, 11, 12, A_1 = {2, 6, 6, 6} corresponding to indices 1, 13, 14, 15, A_2 = {3, 3, 3} corresponding to indices 3, 5, 7, and A_3 = {3, 3, 3} corresponding to indices 4, 6, 8.

Result of (2):



Result of (3):

Result of (4):

Figure 2. An illustration of how **DenseScheduler3** behaves on input A = {8, 12, 12, 12, 12, 12, 12, 12, 24, 24, 24, 24, 24, 24, 24}. Note here that x_1 = 8, x_2 = 12, x_3 = 24, a = 1, b = 7, c = 7, d = 4, and LCM(8, 12, 24) = 24.

Let A = $\{a_1, ..., a_n\}$ be dense and composed of a x_1s, b x_2s, and c x_3s. Let $GCD(x_1, x_2, x_3) = d$, $y_1 = x_1/d$, $y_2 = x_2/d$, and $y_3 = x_3/d$. Then a multiset containing a y_1s, b y_2s, and c y_3s can be partitioned into d multisets each of which is dense and composed of at most 2 distinct integers iff the following system of equations has integer solutions in a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3, d_1, d_2, d_3, d_4, d_5, d_6.

(1) $a_1/y_1 + b_1/y_2 = d_1$ (if GCD $(y_1, y_2) > 1$)
(2) $b_2/y_2 + c_1/y_3 = d_2$ (if GCD $(y_2, y_3) > 1$)
(3) $a_2/y_1 + c_2/y_3 = d_3$ (if GCD $(y_1, y_3) > 1$)
(4) $a_3/y_1 = d_4$
(5) $b_3/y_2 = d_5$
(6) $c_3/y_3 = d_6$
(7) $a_1 + a_2 + a_3 = a$
(8) $b_1 + b_2 + b_3 = b$
(9) $c_1 + c_2 + c_3 = c$
(10) $d_1 + d_2 + d_3 + d_4 + d_5 + d_6 = d$
(11) $a_i \geq 0, b_i \geq 0, c_i \geq 0, 1 \leq i \leq 3$
(12) $d_i \geq 0, 1 \leq i \leq 6$
(13) $d_i \leq 1, 1 \leq i \leq 3$

Theorem 4.7 states that each subschedule modulo d may contain no more than two of y_1, y_2, and y_3. There are, therefore, six possible kinds of subschedules: those containing elements whose periods are y_1 and y_2, y_2 and y_3, y_1 and y_3, only y_1, only y_2, and only y_3. These correspond to equations (1) – (6), respectively. The total number of subschedules is d, as specified in (10). The total number of occurrences of y_1, y_2, and y_3 must be a, b, and c, respectively, as specified in (7) – (9). (11) – (13) provide the bounds for variables. A solution to (1) – (12) for which (13) does not hold exists only if there is also a solution in which (13) holds. This limits the number of each variety of "mixed subschedules" to 1, which will be useful later.

Furthermore, a solution to this system of equations corresponds directly to a desirable multiset partition. Also note that the above system of equations can easily be rewritten as an instance of integer linear programming — in 15 variables. Hence, solutions can be found in deterministic polynomial time [Lenstra 83].□

A perhaps useful observation that comes from a careful examination of the proofs of Theorems 4.5, 4.8, and 4.9 concerns the existence of PTPGs — as described in Section 2. For example, let A = $\{a_1, ..., a_n\}$ be dense and composed of exactly a x_1s and b x_2s. (Assume that $a_1, ..., a_a$ ($a_{a+1}, ..., a_{a+b}$) are equal to x_1 (x_2).) Let $d = GCD(x_1, x_2)$, $x_1 = dy_1$, and $x_2 = dy_2$. Then with a bit of work one can see that the following program P constitutes a FOLS representation of a pinwheel schedule for A:

```
P: i := 0;
    Do
        (q, r) := (quotient(i/d), remainder(i/d));
        /* We want to schedule the satellite corresponding to the integer in slot q in subschedule r. */
        if $0 \leq r < a/y_1$ then
            (q', r') := (quotient($q/y_1$), remainder($q/y_1$));
            service the $(ry_1 + 1+r')^{st}$ satellite
        else
            r := r − $a/y_1$;
            (q',r') := (quotient($q/y_2$), remainder($q/y_2$));
            service the $(a + ry_2 + 1 + r')^{st}$ satellite;
        endif
        i := i + 1 modulo $dy_1y_2$;
        wait for the "time unit" to fully elapse
    forever
```

Furthermore, P can be produced in polynomial time given only a, x_1, b, x_2. FOLS representations of pinwheel schedules can also be produced in polynomial time for dense instances composed of 3 distinct integers. The production of such implementations depends heavily on a solution to the set of equations given in Theorem 4.9.

For example, let A = {8, 12, 12, 12, 12, 12, 12, 12, 24, 24, 24, 24, 24, 24, 24} as in Figure 2. Here is the construction of a FOLS for A.

The first step is to find a solution to the equations in Theorem 4.9. Here $x_1 = 8$, $x_2 = 12$, $x_3 = 24$, a = 1, b = 7, and c = 7. d = GCD(8, 12, 24) = 4, so $y_1 = x_1/d = 2$, $y_2 = x_2/d = 3$, and $y_3 = x_3/d = 6$. A solution is:

$a_1 = 0$	$a_2 = 1$	$a_3 = 0$			
$b_1 = 0$	$b_2 = 1$	$b_3 = 6$			
$c_1 = 4$	$c_2 = 3$	$c_3 = 0$			
$d_1 = 1$	$d_2 = 1$	$d_3 = 0$	$d_4 = 0$	$d_5 = 2$	$d_6 = 0$

Only equations (2), (3), and (5) now have nonzero terms. They describe the subschedules we will use:

(2) $1/3 + 4/6 = 1$
(3) $1/2 + 3/6 = 1$
(5) $6/3 = 2$

Thus there will be 1 + 1 + 2 = 4 subschedules of three distinct sorts. Each will consist of six slots. The indices we schedule into these slots will correspond to the indices in A : $a_1 = 8$, $a_2 = 12$, etc., so for example, index 1 must be scheduled every 2 slots in its subschedule. The resulting subschedules are those of Figure 2, but the result is a FOLS rather than an actual schedule. The FOLS P follows:

```
P : $p_0$ := 0; $p_1$ := 0; $p_2$ := 0; $c_0$ := 9; $c_1$ := 13; $b_2$ := 3;
    Do
          if $p_0$ modulo 3 = 0
          then schedule satellite 2          /* 1 of the 12's */
          else schedule satellite $c_0$;     /* 4 of the 24's */
  $A_0$ {       $c_0 := c_0 + 1$;
              if $c_0 = 13$ then $c_0 := 9$ endif
          endif
          $p_0 := (p_0 + 1)$ modulo 6;
          Wait for the "time unit" to fully elapse;
```

```
     / if $p_1$ modulo 2 = 0
    |  then schedule satellite 1                 /* The 8 */
    |  else schedule satellite $c_1$;          /* 3 of the 24's */
A1 <     $c_1 := c_1 + 1$;
    |    if $c_1 = 16$ then $c_1 := 13$ endif
    |  endif
    |  $p_1 := (p_1 + 1)$ modulo 6;
     \ Wait for the "time unit" to fully elapse;
     / for i = 1 to 2 do       /* The remaining 6 12's */
A2  |    schedule satellite $b_2$;
&  <     $b_2 := b_2 + 1$;
A3  |    if $b_2 = 9$ then $b_2 := 3$;
    |    Wait for the "time unit" to fully elapse
     \ endfor
   forever
```

Note that d satellites (one from each subschedule) are scheduled each time through the **do** ... **forever** loop.

The construction for the general case is tedious but not difficult and hence is left to the reader.

The GCD of A, and therefore the number of subschedules, is bounded by the magnitude of the numbers in A, which is exponential in the length of their representation. The number of code segments is limited to six, however. The segments, corresponding to equations (4), (5), and (6), involve **for** loops which allow for multiple similar subschedules. In this way the FOLS may be generated in polynomial time. Hence:

Theorem 4.10. $\mathbb{C}_{d,1,2,3}$ is in S-P-C.

We would like to extend the techniques utilized here to dense instances with k distinct integers — for any constant k. The scheduling strategies we've employed so far seem to depend on the GCD being greater than 1. But there are schedulable dense instances with 4 or more distinct integers where this is not the case. For example, $A = \{6, 6, 10, 10, 10, 15, 15, 30, 30, 30, 30, 30, 30, 30\}$ is dense and can be shown to be schedulable. However, $GCD(6, 10, 15, 30) = 1$.

We now turn our attention to the case of general dense instances. Let $\mathbb{C}_d = \{A \mid A = \{a_1, ..., a_n\}$ is dense$\}$. We now show that the pinwheel decision problem for $\mathbb{C}_d$ is in NP.

Theorem 4.11. The "pinwheel decision problem" for $\mathbb{C}_d$ is in NP.

Proof. Let $A = \{a_1, ..., a_n\}$ be dense. Consider the construction of an infinite schedule for A. We need to select starting slots for each index that will insure the avoidance of collisions. By Lemma 4.1, we know that if integer i first appears in slot p_i then it will occupy slots $p_i + c\, a_i$, for every integer $c \geq 0$. Starting slots permissible for integer i are $0, ..., a_i-1$. Hence, A is schedulable iff:

$$\exists\, p_1, ..., p_n,\ 0 \leq p_i < a_i, \text{ such that}$$
$$\forall i,j,\ 1 \leq i, j \leq n,\ \neg[\exists a,b \geq 0 \text{ such that } p_i + a\, x_i = p_j + b\, x_j].$$

The question "$\exists a,b \geq 0$ such that $p_i + a\, x_i = p_j + b\, x_j$?" is an instance of integer linear programming in 2 variables and hence is solvable in deterministic polynomial time [Feit 84, Kannan 80, Lenstra 83]. A nondeterministic polynomial time algorithm should now be apparent. ❑

Suppose that a dense instance A composed of a_1 x_1s, a_2 x_2s, ..., a_n x_ns is represented using repetition factors — i.e., as $\{(a_1, x_1), (a_2, x_2), ..., (a_n, x_n)\}$. Notice that Theorem 4.9, as well as the claims made in the ensuing discussion, hold for inputs represented this way. For such representations, we can show that the "pinwheel decision problem," for dense instances, is NP-hard. We cannot, however, show this problem to be in NP. Neither can we show the problem, as described in Theorem 4.11, to be NP-hard.

The proof of the next theorem is long and somewhat tedious. It relies on the fact that primes (n^3) — the number of primes between 1 and n^3 — is greater than or equal to n, for $n \geq 8$ [Knuth 73]. We omit the proof from this version of the paper but will include it in a later version.

Theorem 4.12. The "pinwheel decision problem" for $\mathbb{C}_d$, whose instances are represented using repetition factors, is NP-hard.

References

[Cook 71] S. Cook, "The Complexity of Theorem Proving Procedures," Proceedings of the Third Annual ACM Symposium on the Theory of Computing, pp. 151-158.

[Feit 84] S. Feit, "A Fast Algorithm for the Two-Variable Integer Programming Problem," *J. ACM*, Vol. 31, No. 1, pp. 99-113, January 1984.

[Kannan 80] R. Kannan, "A Polynomial Algorithm for the Two-Variable Integer Programming Problem," *J. ACM*, Vol. 27, No. 1, pp. 118-122, January 1980.

[Knuth 73] D. Knuth, **The Art of Computer Programming: Vol. 2, Seminumerical Algorithms**, *Addison-Wesley*, 1973.

[Lenstra 83] H. Lenstra, "Integer Programming with a Fixed Number of Variables," *Mathematics of Operations Research*, Vol. 8, No. 4, pp. 538-548, November 1983.

[Leung & Merrill 80] J. Y-T. Leung and M. L. Merrill, "A Note on Preemptive Scheduling of Periodic, Real-Time Tasks," *Information Processing Letters*, Vol. 11, pp. 115-118.

[Liu & Layland 73] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming Systems in a Hard-Real-Time Environment," *JACM*, Vol. 20, pp. 46-61, January 1973.

[Mok 83] A. K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," *Ph.D thesis*, MIT Laboratory for Computer Science, May 1983.

[Mok & Sutanthavibul 85] A. K. Mok and S. Sutanthavibul, "Modeling and Scheduling of Dataflow Real-Time Systems," *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 178-187, December 1985.

[Stewart 64] B. Stewart, **Theory of Numbers**, *The Macmillan Company*, 1964.