Periodic Multiprocessor Scheduling

Jan Korst[1], Emile Aarts[1,2], Jan Karel Lenstra[2,3] and Jaap Wessels[2]

1. Philips Research Laboratories, P.O. Box 80.000, 5600 JA Eindhoven, the Netherlands
2. Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, the Netherlands
3. CWI, P.O. Box 4079, 1009 AB Amsterdam, the Netherlands

Abstract

A number of scheduling and assignment problems are presented involving the execution of periodic operations in a multiprocessor environment. We consider the computational complexity of these problems and propose approximation algorithms for operations with identical periods as well as for operations with arbitrary integer periods.

Keywords: periodic scheduling, periodic assignment, cyclic scheduling, nonpreemptive scheduling

1 Introduction

This paper deals with the problem of scheduling periodic operations, i.e., operations that have to be repeated at a constant rate over an infinite time horizon. Periodic scheduling problems naturally arise in such diverse areas as real-time processing, process control, vehicle scheduling, personnel scheduling and preventive maintenance scheduling; see Section 2 for references. Our interests in periodic scheduling originate from the field of real-time video signal processing, where the samples of a video signal have to be processed at a constant high frequency (10 - 100 MHz) on a network of processors. Due to the high frequencies, the processing of successive samples necessarily overlaps in time. The intrinsic periodic nature of video signal processing gives rise to a periodic scheduling formulation. This application area poses some specific constraints, resulting in a class of optimization problems that so far have received little attention in the literature. In this paper we discuss this class of problems by examining their computational complexity, introducing approximation algorithms, and indicating relevant results presented in the literature.

We aim to keep the discussion as general as possible by proposing solution strategies that are also applicable in other application areas. Many papers on periodic scheduling are concerned with specific applications, proposing solution strategies that are often strongly tailored to the application at hand, a notable exception being the paper by Serafini & Ukovich [1989], which presents a general mathematical model for periodic scheduling problems. However, their emphasis is on periodic scheduling subject to precedence constraints. In our paper, the emphasis is on periodic scheduling subject to resource constraints. In that respect, our work is complementary to theirs.

The organization of the paper is as follows. Section 2 briefly surveys the literature on periodic scheduling. Section 3 gives a mathematical model of periodic scheduling, from which a number of interrelated optimization problems are derived. The computational complexity of these problems is examined in Section 4. Section 5 gives approximation algorithms and bounds on their worst-case performance, if available. Section 6 contains some concluding remarks.

2 Survey of the Literature

In the literature, the notion 'scheduling' refers to planning in time as well as planning in time and space. In this paper, we take the latter interpretation. We divide the literature on scheduling periodic operations into two main areas of interest, namely

(i) *Periodic Scheduling*: assigning start times and processors to periodic operations so as to minimize the number of processors, possibly subject to precedence constraints, and

(ii) *Periodic Assignment*: assigning processors to periodic operations so as to minimize the number of processors for periodic operations with fixed start times.

Clearly, periodic assignment is a subproblem in periodic scheduling. Next, we briefly describe some results obtained in both areas. We do not aim to give a complete overview.

2.1 Periodic Scheduling

Most of the literature on scheduling periodic operations in time is restricted to preemptive scheduling. Preemptive scheduling allows interruption of an execution on a given processor at some time and its resumption at the same time on a different processor or at a later time on any processor.

2.1.1 Preemptive Periodic Scheduling

Preemptive periodic scheduling problems are usually modelled as follows. Given a set of operations $O = \{o_1, \ldots, o_n\}$, any operation $o_i \in O$ is periodically requested to be executed with a given period $p(o_i)$ between two successive requests of operation o_i. Once requested at time t an execution of o_i is required to be completed at time $t + d(o_i)$, called its deadline. The objective is then to find a feasible schedule that requires a minimal number of processors, where a schedule is called feasible if all deadlines are met. Leung & Merrill [1980] prove that the problem of deciding whether a feasible schedule exists on m processors is NP-complete, even for $m = 1$. However, this problem can be solved in polynomial time if the deadline of each execution coincides with the next request for the operation. For $m = 1$, Liu & Layland [1973] and Labetoulle [1974] prove that, if a feasible schedule exists, then it is obtained by the so-called *deadline driven algorithm*, which is a dynamic-priority algorithm that schedules executions with earliest deadlines as soon as possible. Liu & Layland also give a fixed-priority scheduling algorithm for $m = 1$, known as *rate-monotonic priority assignment*, which is optimal in the sense that the algorithm finds a feasible schedule whenever a feasible fixed-priority schedule exists. Dhall & Liu [1978] present two fixed-priority scheduling algorithms for $m \geq 1$, and discuss their worst-case performance. Leung & Whitehead [1982] study the complexity of preemptive fixed-priority scheduling. Lawler & Martel [1981] show that a feasible preemptive schedule exists if and only if a feasible periodic schedule exists with a period equal to the least common multiple of the periods of the individual operations. Bertossi & Bonuccelli [1983, 1985] consider preemptive scheduling on multiprocessor systems consisting of 'processors of different speeds'. Scheduling periodic operations together with 'sporadic time-critical operations' is examined by Chetto & Chetto [1989].

2.1.2 Nonpreemptive Periodic Scheduling

So far, nonpreemptive periodic scheduling has received little attention in the literature. To schedule periodic operations nonpreemptively, it is usually assumed that the operations have to be executed with a fixed time between successive executions of the same operation. Gonzalez & Soh propose an optimization algorithm for nonpreemptively scheduling periodic operations for the rather special case that the period of the ith operation is half the period of the $(i+1)$th one. Serafini & Ukovich [1989] discuss nonpreemptive periodic scheduling subject to precedence constraints and show that this problem is NP-complete. Park & Yun [1985] give an ILP formulation of a nonpreemptive scheduling problem. They consider a set of independent periodic operations, where each execution requires a given number of resources during one unit of time, and aim to minimize the maximum required amount of resources. They show how this problem can be partitioned into a set of independent subproblems, which can be optimized independently. The partitioning divides the operations into subsets such that the periods of operations in different subsets are relatively prime. A problem related to nonpreemptive periodic scheduling is the problem of inscribing regular polygons in a circle so as to maximize the minimum distance between two vertices on the circle. Burkard [1986] solves this problem for a set of regular polygons that includes only two different types of polygons. Vince [1989] presents a more general approach to this problem.

2.2 Periodic Assignment

Periodic assignment deals with the problem of assigning the executions of periodic operations to a minimal number of processors, assuming that the executions are fixed in time. As we show in the next sections, this problem is closely related to that of colouring circular arcs. Circular-arc colouring has been studied by several authors. Garey, Johnson, Miller & Papadimitriou [1980] prove that circular-arc graph colouring is NP-hard. Tucker [1975] gives upper bounds on the number of colours needed to colour various types of circular-arc graphs. Orlin, Bonuccelli & Bovet [1981] and Shih & Hsu [1989] give efficient algorithms for the polynomially solvable subproblem of colouring proper circular-arc graphs.

Bartholdi, Orlin & Ratliff [1980] consider the periodic assignment problem under the assumption that the availability of resources is also periodic. This problem naturally arises in the area of personnel scheduling, where periodic jobs have to be assigned to persons having periodic working hours. Bartholdi [1981] proposes a linear programming round-off algorithm and gives its worst-case deviation from optimum. Orlin [1982] discusses the periodic assignment problem under the assumption that processors require a setup time s_{ij} to switch from execution i to execution j. This problem naturally arises in the area of vehicle scheduling, where a vehicle has to be transported from the end point of route i to the starting point of route j before it can start traversing route j.

3 Problem Description

In this section we give a formal description of a number of interrelated periodic scheduling and assignment problems. We restrict ourselves to nonpreemptive scheduling and do not consider precedence constraints.

Let $O = \{o_1, \ldots, o_n\}$ be a set of n periodic operations. For each $o \in O$ an execution time $e(o) \in \mathbb{N}$ and a period $p(o) \in \mathbb{N}$ are given. We assume that $p(o) \geq 1$ and $e(o) \leq p(o)$ for each $o \in O$. Once an execution of an operation o is started at a time unit $t \in \mathbb{Z}$, it is completed without interruption on the same processor. Note that in this paper time is measured in time units, i.e., time periods of equal length. If an operation o with execution time $e(o)$ is said to start at time unit t, it starts at the beginning of time unit t and completes at the end of time unit $t + e(o) - 1$. Similarly, a time interval $[t_1, t_2]$ denotes a set of consecutive time units, given by $\{t_1, t_1 + 1, \ldots, t_2\}$. The kth execution of operation o is denoted by $o[k]$. If execution $o[k]$ is started at time unit t, then execution $o[k+1]$ is started at time unit $t+p(o)$. The set of all executions is given by

$$E = \{o[k] \mid o \in O,\ k \in \mathbb{Z}\}.$$

So, each operation $o \in O$ is started exactly every $p(o)$ time units. Consequently, if for an operation o the start time of an arbitrary execution is fixed, then all executions of o are fixed in time. Without loss of generality, the executions of operation o are uniquely specified by a start time $s(o)$, with $0 \leq s(o) < p(o)$. Hence, a schedule $\mathcal{S}$ of the operations in O is uniquely determined by an n-tuple $(s(o_1), s(o_2), \ldots, s(o_n))$, with $0 \leq s(o_i) < p(o_i)$ for all $o_i \in O$. Furthermore, the operations are considered independent, i.e., there are no precedence constraints between executions of different operations.

Scheduling periodic operations naturally leads to periodic schedules. A schedule $\mathcal{S}$ is called periodic with period P if for each time unit $t \in \mathbb{Z}$ and each $o \in O$ the following holds:

operation o is executed at time unit t if and only if it is executed at time unit $t + P$.

Clearly, in order for a schedule to be periodic with period P, it is required that $p(o) \mid P$, for each $o \in O$. Consequently, the minimal period P of a schedule is given by $\operatorname{lcm}(p(o_1), \ldots, p(o_n))$, i.e., the least common multiple of the periods of the individual operations.

Let M denote the set of processors. The processors are supposed to be identical, i.e., each operation $o \in O$ can be executed on any processor $m \in M$ and the time to execute operation o does not depend on the processor. Furthermore, a processor can only execute one operation at a time. We aim to minimize the number of processors necessary for the execution of the operations in O. Given a schedule $\mathcal{S}$, we can define the *thickness function* $T_{\mathcal{S}} : \mathbb{Z} \rightarrow \mathbb{N}$ which assigns to each $t \in \mathbb{Z}$ the number of operations that are being executed at that time unit. Since a processor can only execute one operation at a time, $\max_t T_{\mathcal{S}}(t)$ gives, for a given schedule $\mathcal{S}$, a lower bound on the number of processors that is required to carry out schedule $\mathcal{S}$. If schedule $\mathcal{S}$ is periodic with period P, then the thickness function $T_{\mathcal{S}}$ is also periodic with a period P', for which $P' \mid P$. Hence, to determine $\max_t T_{\mathcal{S}}(t)$, it suffices to consider time units $t \in \{1, \ldots, P\}$.

With respect to the assignment of executions to processors we consider two different cases, namely

(i) the *constrained* case, where all executions of an operation o have to be assigned to the same processor, for all $o \in O$, i.e., an assignment from O to M is required, and

(ii) the *unconstrained* case, where each execution $o[k]$ can be assigned to a different processor, i.e., an assignment from E to M is required.

An assignment of each execution in E to a processor in M may be difficult to specify, since E is a (countably) infinite set. We therefore restrict ourselves to periodic assignments. An assignment is called periodic with period $P \in \mathbb{N}$ if for each time unit $t \in \mathbb{Z}$, each $o \in O$, and for each $m \in M$ the following holds:

m executes o at time unit t if and only if m executes o at time unit $t + P$.

If for a periodic schedule S with period P the corresponding assignment is periodic with period P', then necessarily $P \mid P'$. In the constrained case, i.e., if all executions of an operation are assigned to the same processor, the assignment is necessarily periodic with period $P' = P$. For the unconstrained case, restricting oneself to periodic assignments does not lead to the use of extra processors as long as the length of period P' is not restricted. This is shown in the following theorem.

Theorem 1 *For each periodic schedule S a periodic assignment exists requiring only* $\max_t \mathrm{T}_S(\mathrm{t})$ *processors.*

Proof We have seen that $\max_t T_S(t)$ gives a lower bound on the required number of processors. Now a finite set of executions can be optimally assigned to $\max_t T_S(t)$ processors, using an $O(n \log n)$ algorithm [Hashimoto & Stevens, 1971; Gupta, Lee & Leung, 1979], where n denotes the number of executions. The algorithm assigns the executions in order of increasing start times to the first available processor, i.e., to the available processor with the smallest index number. Let us consider the assignment of a finite set of executions, namely the set of all executions in the time interval $[0, mP-1]$, with $m \in \mathbb{N}$ and $P = \mathrm{lcm}(p(o_1), \ldots, p(o_n))$. We show that, if m is chosen sufficiently large, the assignment necessarily becomes periodic with some period $m'P$, $m' < m$. Let us examine the assignment in intervals $[lP, (l+1)P-1]$, with $0 \leq l < m$. The assignment can attain only a finite set of different solutions in such an interval $[lP, (l+1)P-1]$, since a finite set of executions can be assigned to a finite set of processors. Consequently, if m is chosen sufficiently large, then in two intervals $[lP, (l+1)P-1]$ and $[l'P, (l'+1)P-1]$, with $0 \leq l < l' < m$, the assignment must necessarily be identical. Hence, the assignment necessarily becomes periodic with period $(l'-l)P$, using only $\max_t T_S(t)$ processors, which completes the proof of the theorem. ∎

The minimum period for which a periodic assignment uses $\max_t T_S(t)$ processors may generally be very large. For reasons of simplicity, we restrict ourselves in this paper to periodic assignments with periods of minimal length, i.e., with a period $P = \mathrm{lcm}(p(o_1), \ldots, p(o_n))$. In this way, an operation o is executed on at most $P/p(o)$ different processors. For the unconstrained case, an assignment is thus completely specified if the processor is given for $P/p(o)$ successive executions of each operation $o \in O$, denoted by $o[1], o[2], \ldots, o[P/p(o)]$, where $o[1]$ is defined to be the first execution starting at a time unit $t \geq 0$.

Given the definitions and assumptions described above, we can define the following periodic assignment problems. We formulate these problems as decision problems.

Unconstrained Periodic Assignment (UPA)
Given a schedule S for a set O of periodic operations with an execution time $e(o) \in \mathbb{N}$ and a period $p(o) \in \mathbb{N}$ for each $o \in O$, and an integer k, does an unconstrained periodic assignment with period $P = \mathrm{lcm}(p(o_1), \ldots, p(o_n))$ exist that uses at most k processors?

Constrained Periodic Assignment (CPA)
Given a schedule S for a set O of periodic operations with an execution time $e(o) \in \mathbb{N}$ and a period $p(o) \in \mathbb{N}$ for each $o \in O$, and an integer k, does a constrained periodic assignment exist that uses at most k processors?

Likewise, we define the following periodic scheduling problems.

Unconstrained Periodic Scheduling (UPS)
Given a set O of periodic operations with an execution time $e(o) \in \mathbb{N}$ and a period $p(o) \in \mathbb{N}$ for each $o \in O$, and an integer k, does a schedule exist for which an unconstrained periodic assignment with period $P = \mathrm{lcm}(p(o_1), \ldots, p(o_n))$ uses at most k *processors?*

Constrained Periodic Scheduling (CPS)
Given a set O of periodic operations with an execution time $e(o) \in \mathbb{N}$ and a period $p(o) \in \mathbb{N}$ for each $o \in O$, and an integer k, does a schedule exist for which a constrained periodic assignment uses at most k processors?

With respect to CPS the following theorem gives a necessary and sufficient condition for scheduling the executions of two operations on the same processor.

Theorem 2 *The executions of two periodic operations o_i and o_j can be scheduled on the same processor if and only if*

$$\gcd(p(o_i), p(o_j)) \geq e(o_i) + e(o_j). \qquad (1)$$

Proof Let $g = \gcd(p(o_i), p(o_j))$. We first prove that (1) is a sufficient condition. This is shown as follows. Choosing the start times $s(o_i) = 0$ and $s(o_j) = e(o_i)$, operation o_i is executed in a subset of the set I_i of intervals, defined by $[lg, lg + e(o_i) - 1]$, $l \in \mathbb{Z}$, and operation o_j is executed in a subset of the set I_j of intervals, defined by $[lg + e(o_i), lg + e(o_i) + e(o_j) - 1]$, $l \in \mathbb{Z}$. Hence, if $g \geq e(o_i) + e(o_j)$, then no intervals of I_i and I_j overlap, which proves the sufficiency of (1).

We prove the necessity of (1) by showing that, if $g < e(o_i) + e(o_j)$, operation o_i and o_j cannot be scheduled on the same processor. So, assume that $g < e(o_i) + e(o_j)$. Without loss of generality we may assume that $s(o_i) = 0$. We now have to prove that integers x, y exist for which

$$[xp(o_i), xp(o_i) + e(o_i) - 1] \cap [s(o_j) + yp(o_j), s(o_j) + yp(o_j) + e(o_j) - 1] \neq \emptyset$$

or, equivalently,

$$[xp(o_i) - yp(o_j), xp(o_i) - yp(o_j) + e(o_i) - 1] \cap [s(o_j), s(o_j) + e(o_j) - 1] \neq \emptyset.$$

From elementary number theory it is known that integers w, z exist for which $wp(o_i) + zp(o_j) = g$. If we choose $x = lw$ and $y = -lz$, with $l \in \mathbb{Z}$, it suffices to show that for some integer l

$$[lg, lg + e(o_i) - 1] \cap [s(o_j), s(o_j) + e(o_j) - 1] \neq \emptyset.$$

Clearly, this must be the case since the free intervals between the intervals $[lg, lg + e(o_i) - 1]$, $l = 0, 1, \ldots,$ are of length $g - e(o_i)$, while the intervals $[s(o_j), s(o_j) + e(o_j) - 1]$ are of length $e(o_j)$. Hence, the assumption that $g < e(o_i) + e(o_j)$ implies that some integer l necessarily exists for which $[lg, lg + e(o_i) - 1]$ and $[s(o_j), s(o_j) + e(o_j) - 1]$ overlap. This completes the proof of the theorem. ∎

A similar condition can be derived for CPA, as is shown in the following theorem.

Theorem 3 *For CPA, two periodic operations o_i and o_j with given start times $s(o_i)$ and $s(o_j)$, can be executed on the same processor if and only if*

$$e(o_i) \leq (s(o_j) - s(o_i)) \bmod g \leq g - e(o_j), \tag{2}$$

where $g = \gcd(p(o_i), p(o_j))$.

Proof Without loss of generality we may assume that $s(o_i) = 0$. This is true since, if $s(o_i) \neq 0$, then the start times of o_i and o_j can be shifted such that $s(o_i)$ becomes zero, without affecting possible overlap. The sufficiency of (2) is shown as follows. Let us consider time intervals $[0 + kg, g - 1 + kg]$, with $k \in \mathbb{Z}$. The first $e(o_i)$ time units of each of these intervals can be allocated for executions of o_i, and the remaining $g - e(o_i)$ time units for executions of o_j. Now, if (2) holds, then the allocated time units surely suffices to execute o_i and o_j. The first $e(o_i)$ time units of the intervals are only used to execute o_i once every $p(o_i)/g$ intervals. The remaining $g - e(o_i)$ time units are only (partly) used to execute o_j once every $p(o_j)/g$ intervals.

The necessity of (2) is shown as follows. Let us again consider the time intervals $[0+kg, g - 1 + kg]$, with $k \in \mathbb{Z}$. If (2) does not hold then the execution of o_j overlaps the first $e(o_i)$ time units once every $p(o_j)/g$ time intervals. We have already seen that the first $e(o_i)$ time units of the intervals are used for the execution of o_i once every $p(o_i)/g$ time units. Now, by definition, $\gcd(p(o_i)/g, p(o_j)/g) = 1$. Hence, if (2) does not hold, then operations o_i and o_j cannot be executed on the same processor. This completes the proof of the theorem. ■

Note that Theorem 2 can be considered a corollary of Theorem 3, since (1) directly follows from (2). In the next section we examine the computational complexity of the problems defined above.

4 Computational Complexity

To examine the complexity of the periodic assignment problems CPA and UPA, we focus our attention on the subset of problem instances for which $p(o) = p$ for all $o \in O$. Note that under this restriction CPA and UPA are identical. If we prove that this subset of instances is NP-complete, then both CPA and UPA have been proved to be NP-complete.

Theorem 4 *CPA and UPA are NP-complete.*

Proof It is easily verified that CPA and UPA are in $\mathcal{NP}$. Now the NP-completeness is proved by a reduction from circular-arc colouring, which has been shown to be NP-complete by Garey, Johnson, Miller & Papadimitriou [1980]. We first define circular-arc colouring. Let a set of circular arcs $A = \{a_1, \ldots, a_n\}$ be given, where each arc a_i, specified by an ordered pair (l_i, r_i), with $l_i, r_i \in \{0, 1, \ldots, 2n-1\}$, is an arc on a circle with circumference $2n$ that stretches clockwise from point l_i to point r_i, containing both endpoints, and let an integer k be given. The problem is now: is A k-colourable, i.e., does a function $f : A \rightarrow \{1, \ldots, k\}$ exist such that $f(a_i) \neq f(a_j)$ whenever a_i and a_j overlap? Any instance of circular-arc colouring can be transformed to a periodic assignment instance as follows. For each arc a_i we define a periodic operation with period $p(o_i) = 2n$, start time $s(o_i) = l_i$, and execution time $e(o_i) = r_i - l_i + 1$ if $r_i \geq l_i$ and $e(o_i) = r_i - l_i + 2n + 1$ if $r_i < l_i$. Now two periodic operations can be assigned to the same processor if and only if the corresponding circular arcs can be coloured with the same colour. Consequently, the circular arcs can be coloured using k colours if and only if the periodic operations can be assigned to k processors. Evidently, this is a polynomial-time transformation, which completes the proof of the theorem. ■

Note that the transformation from circular-arc colouring defines an equivalence between circular-arc colouring and the problem of assigning operations with identical periods, which we will use in Section 5.1.

To consider the complexity of CPS and UPS we again focus our attention on the subset of problem instances for which $p(o) = p$ for all $o \in O$. Again notice that this subset is in the intersection of the CPS and UPS problem instances.

Theorem 5 *CPS and UPS are NP-complete in the strong sense.*

Proof It is easily verified that CPS and UPS belong to $\mathcal{NP}$. We now prove the NP-completeness by a reduction from bin packing, which is NP-complete in the strong sense [Garey & Johnson, 1979]. An instance of bin packing is specified as follows. Let a finite set $A = \{a_1, \ldots, a_n\}$ of items be given, with for each item $a_i \in A$ a positive integer size $s(a_i)$, a positive bin capacity B and a positive integer k. Can A be partitioned into k disjoint subsets $A_1, \ldots, A_k$, such that the sum of the sizes in each subset A_i does not exceed the bin capacity B? Any instance of bin packing can be directly transformed into an instance of CPS or UPS as follows. For each item a_i we define a periodic operation o_i with execution time $e(o_i) = s(a_i)$ and period $p(o_i) = B$. Clearly, a number of periodic operations can be executed on the same processor if the corresponding items can be packed in one bin, and vice versa. Hence, the items $a_1, \ldots, a_n$ can be packed into k bins if and only if the operations $o_1, \ldots, o_n$ can be scheduled on k processors. Since the above transformation is polynomial, CPS and UPS are both NP-complete in the strong sense. ■

An alternative reduction from 3-partition can be constructed, showing that the problems remain NP-complete in the strong sense for the case that only one processor is available. Hence, this gives a stronger result. We have chosen, however, to give the reduction from bin packing since this reduction defines an equivalence between bin packing and the problem of scheduling periodic operations with identical periods, which we will use in Section 5.1.

5 Approximation Algorithms

All problems presented in Section 3 are NP-complete. This means that, unless $\mathcal{P} = \mathcal{NP}$, efficient optimization algorithms do not exist for these problems. We therefore focus our attention on approximation algorithms, i.e., algorithms which do not guarantee to find an optimal solution for every instance but attempt to find near-optimal solutions. In the remainder of this paper we present approximation algorithms for the periodic scheduling and assignment problems presented in Section 3 and, to some extent, analyse their performance. An interesting subclass of problems arises if we assume that the operations all have identical periods. We first consider approximation algorithms for this subclass of problems.

5.1 Periodic Operations with Identical Periods

In Section 4 we already indicated the equivalence between bin packing and the problem of scheduling periodic operations with identical periods. Hence, approximation algorithms for bin packing can be directly applied to this problem. A large number of approximation algorithms exist for bin packing, ranging from simple approximation algorithms called *first fit* and *first fit decreasing*, which have asymptotic performance ratios of $\frac{17}{10}$ and $\frac{11}{9}$, respectively, to approximation schemes. An extensive survey of the literature on approximation algorithms for bin packing is given by Coffmann, Garey & Johnson [1984]. A bin packing algorithm gives a partitioning of the operations into subsets such that the operations in the same subset can be assigned to the same processor. A feasible schedule can then easily be obtained by scheduling the operations in each subset one after the other, in some arbitrary order. The wealth of approximation algorithms for bin packing provided by the literature surely suffices to effectively handle this subclass of periodic scheduling problems.

To present approximation algorithms for the assignment of periodic operations with identical periods we refer to its equivalence with the problem of colouring circular arcs, as indicated in Section 4. To the best of our knowledge, Tucker [1975] is the only author who considers the subject of approximation algorithms for colouring circular arcs, in order to give an upper bound on the number of colours necessary for colouring circular arcs. Elaborating on this result, we present the following 2-step approximation algorithm for colouring circular arcs, called *sort&match*.

1. Partition the set of arcs into two subsets A and B, where A contains all arcs that cover one specific point $t \in \{0, 1, \ldots, 2n-1\}$ for which the thickness function attains a minimum value, and B contains all remaining arcs. Consequently, $|A| = \min_t T_S(t)$. Now the arcs in B can be optimally assigned using the assignment algorithms of Hashimoto & Stevens [1971] or Gupta, Lee & Leung [1979] using $\max_t T_S(t)$ colours: the arcs a_i in B are sorted in order of their starting point l_i and they are assigned in this order to the first available colour, i.e., the available colour with the smallest index number.

2. Determine a maximum subset A' of arcs in A which can be coloured with a colour that is already used in step 1 to colour arcs in B. This problem can be formulated as a maximum-cardinality matching problem in a bipartite graph, which can be solved efficiently using an augmenting path algorithm [Edmonds, 1965; Hopcroft & Karp, 1973]. Finally, each remaining arc in $A - A'$ is given a different free colour.

Tucker [1975] only considers the first step of the algorithm presented above. Clearly the algorithm requires at most $\max_t T_S(t) + \min_t T_S(t)$ colours. Since $\max_t T_S(t)$ is a lower bound on the number of required colours, *sort&match* has a worst-case performance ratio of 2. This worst-case performance ratio already holds for the first step of the algorithm (assuming that all arcs in A are given a different free colour), which Tucker already showed. The worst-case performance bound can be shown to be tight [Korst, Aarts, Lenstra & Wessels, 1991]. The average-case performance of *sort&match* is much better. Experimental results indicate that the algorithm almost always finds solutions that are within 10% of the optimum for randomly generated instances [Korst, Aarts, Lenstra & Wessels, 1991].

5.2 Periodic Operations with Arbitrary Periods

In this subsection we discuss possible approximation algorithms for the UPA, CPA, UPS and CPS problems, for the case that operations have arbitrary integer periods.

Approximation Algorithm for UPA

Sort&match, presented in Section 5.1, can also be used as an approximation algorithm for UPA by associating an arc with each execution that is contained in a time window of length $P = \mathrm{lcm}(p(o_1), \ldots, p(o_n))$. Note, however, that here the number of arcs is not polynomially bounded by the number of operations. The performance bound of *sort&match* clearly remains unaffected. Circular arcs can be efficiently coloured if they are proper, i.e., if no arc is completely contained in another arc [Orlin, Bonuccelli & Bovet, 1981;Shih & Hsu, 1989]. Hence, if periodic operations all have identical execution times, they can be optimally assigned to processors in a time that is polynomial in the number of executions.

Approximation Algorithms for CPA

Using Theorem 3 we can easily determine for each pair of periodic operations whether they can be assigned to the same processor. Consequently, we can define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each $v_i \in \mathcal{V}$ is associated with a periodic operation o_i. Two vertices v_i and v_j are adjacent if the associated operations o_i and o_j cannot be assigned to the same processor. The resulting graph $\mathcal{G}$ is called a *periodic-interval graph.* Now it is easy to see that solving a CPA instance is identical to colouring the vertices of the corresponding periodic-interval graph with a minimum number of colours. A periodic-interval graph can be considered to be a generalization of a circular-arc graph in the case that all periods are identical. To the best of our knowledge no graph colouring algorithms are presented in the literature that are tailored to colouring periodic-interval graphs. However, approximation algorithms for colouring arbitrary graphs might give satisfactory results in practice.

Approximation Algorithms for UPS

Experimental results indicate that *sort&match* is able to find solutions for UPA that are often close to $\max_t T_S(t)$. It therefore seems tempting to handle UPS using the following two-step approach:

1. first determine start times for the operations such that $\max_t T_S(t)$ is minimized, and

2. next use *sort&match* to find a feasible assignment.

Now the problem of finding a schedule such that $\max_t T_S(t)$ is minimized can be shown to be NP-complete. This immediately follows from the fact that UPS remains NP-complete for the single processor case. Consequently, we can restrict ourselves to constructing an approximation algorithm for the problem of finding start times that minimize $\max_t T_S(t)$. Note that for a set O' of periodic operations with $\gcd(p(o_i), p(o_j)) = 1$ for all $o_i, o_j \in O'$, we have $\max_t T_S(t) = |O'|$ for any possible choice of start times. This is a corollary of Theorem 2; see also [Park & Yun, 1985]. Consequently, the set of periodic operations O can be partitioned into a number of disjoint subsets $O_1, O_2, \ldots, O_l$ such that $\gcd(p(o_i), p(o_j)) = 1$ for each pair of operations o_i, o_j that have been assigned to different subsets, and $\max_t T_{S(O_i)}(t)$ can be minimized independently for each subset O_i. The total thickness $\max_t T_S(t)$ is then given by $\sum_{O_i} \max_t T_{S(O_i)}$. This partitioning approach will reduce the size of the problem.

We now restrict ourselves to minimizing $\max_t T_S(t)$ for a given subset O_i. This can be done as follows. First select a subset $O_{i'}$ of O_i, for which $\gcd(p(o_i), p(o_j)) = 1$ for all $o_i, o_j \in O_{i'}$, such that $O_{i'}$ is as large as possible. This is done by using some independent set heuristic. The operations in $O_{i'}$ are given arbitrary start times. Next, the remaining operations must be given start times subject to the start times of the operations in $O_{i'}$. If the number of operations in $O_i - O_{i'}$ is small, an enumeration is most appropriate. Otherwise, some constructive or local search approach can be used.

Approximation Algorithm for CPS

In the case of CPS we observe the following. If one or more periodic operations are assigned to a processor, then the time that the processor remains idle can be expressed as one or more periodic intervals, each with a period and a duration. For example, if a periodic operation o_i with period $p(o_i)$ and execution time $e(o_i)$ is assigned to an idle processor, then the remaining idle time can be expressed as a periodic interval with period $p(o_i)$ and a duration $p(o_i) - e(o_i)$. We can thus consider the problem of assigning periodic operations to processors as the problem of assigning periodic operations to periodic intervals. For reasons of simplicity we denote a periodic operation o_i with period $p(o_i)$ and execution time $e(o_i)$ by the ordered pair (p_i, e_i) and a periodic interval with period p_j and duration d_j by the ordered pair $[\![p_j, d_j]\!]$. From Theorem 2 we derive that a periodic operation (p_i, e_i) can be assigned to a periodic interval $[\![p_j, d_j]\!]$ if and only if $\gcd(p_i, p_j) \geq e_i + (p_j - d_j)$. Let $g = \gcd(p_i, p_j)$ and $e_j = p_j - d_j$; then by assigning periodic operation (p_i, e_i) to periodic interval $[\![p_j, d_j]\!]$, the remaining idle time can be expressed as a set of periodic intervals in a number of alternative ways. We assume that a periodic operation is always started at the begin of the periodic interval to which it is assigned. Consequently, the remaining idle time can be expressed as one of the following three alternatives.

1.	$p_i/g - 1$	periodic intervals	$[\![p_i, g - e_j]\!]$,
	$p_j/g - 1$	periodic intervals	$[\![p_j, e_j]\!]$, and
	1	periodic interval	$[\![p_i, g - e_i - e_j]\!]$
2.	$p_i/g - 1$	periodic intervals	$[\![p_i, e_i]\!]$,
	$p_j/g - 1$	periodic intervals	$[\![p_j, e_j]\!]$, and
	1	periodic interval	$[\![g, g - e_i - e_j]\!]$
3.	$p_i/g - 1$	periodic intervals	$[\![p_i, e_i]\!]$,
	$p_j/g - 1$	periodic intervals	$[\![p_j, g - e_i]\!]$, and
	1	periodic interval	$[\![p_j, g - e_i - e_j]\!]$

In all three cases the number of periodic intervals is given by

$$\frac{p_i + p_j}{\gcd(p_i, p_j)} - 1.$$

Note that, if $p_i = p_j$, the three alternatives are identical, leading to only one periodic interval. Otherwise, if $p_i | p_j$ or $p_j | p_i$, then the three alternatives reduce to two essentially different ones.

Based on this observation, we propose the following iterative approximation algorithm. In each iteration all possible assignments of periodic operations to periodic intervals are considered and the one that is considered best is selected to be scheduled. The 'goodness' of an operation-to-interval assignment is defined by the amount of idle time that remains after assigning the periodic operation to the periodic interval. In each iteration the assignment of (p_i, e_i) to $[\![p_j, d_j]\!]$ is selected for which $d_j/p_j - e_i/p_i$ is minimal, provided that the assignment is feasible. Clearly, the amount of idle time that remains after assigning an operation (p_i, e_i) to an idle processor is given by $1 - e_i/p_i$. Consequently, the algorithm will not assign a periodic operation to an idle processor as long as the periodic operation can be assigned to a periodic interval of a processor that is already in use. After each iteration, the remaining idle time is expressed as one or more periodic intervals using one of the three alternatives mentioned above. Which alternative is selected is determined by considering how well the unassigned operations fit in the periodic intervals. This can be considered as a maximum-weight matching problem on a bipartite graph, which can be handled efficiently.

A detailed analysis of the algorithm is beyond the scope of the paper. We mention that, in the case of periodic operations with identical periods, solutions are found that are identical to the ones obtained by *first fit decreasing* for bin packing.

6 Conclusions

A number of closely interrelated optimization problems have been discussed from the field of nonpreemptive periodic scheduling. The complexity of these problems has been examined. We have derived Necessary and sufficient conditions for executing two periodic operations on a single processor. Finally, approximation algorithms have been proposed for periodic scheduling and periodic assignment problems, for the constrained case as well as the unconstrained case.
The material presented in this paper leaves the following open problems:

- Which constraints do we have to impose on the problems discussed in this paper to allow for efficient optimization algorithms?
- Do approximation algorithms exist for colouring periodic-interval graphs that have a constant worst-case performance ratio?
- Do approximation algorithms exist for colouring circular-arc graphs with a worst-case performance ratio smaller than two?
- Is it possible to give a constant worst-case performance ratio for the approximation algorithms for CPS and UPS?

Bibliography

Bartholdi, J.J. [1981], A guaranteed-accuracy round-off algorithm for cyclic scheduling and set covering, *Operations Research* **29**, 501-510.

Bartholdi, J.J., J.B. Orlin, and H.D. Ratliff [1980], Cyclic scheduling via integer programs with circular ones, *Operations Research* **28**, 1074-1085.

Bertossi, A.A. and M.A. Bonuccelli [1983], Preemptive scheduling of periodic jobs in uniform multiprocessor systems, *Information Processing Letters* **16**, 3-6.

Bertossi, A.A. and M.A. Bonuccelli [1985], A polynomial feasibility test for preemptive periodic scheduling of unrelated processors, *Discrete Applied Mathematics* **12**, 195-201.

Burkard, R.E. [1986], Optimal schedules for periodically recurring events, *Discrete Applied Mathematics* **15**, 167-180.

Chetto, H. and M. Chetto [1989], Scheduling periodic and sporadic tasks in a real-time system, *Information Processing Letters* **30**, 177-184.

Coffmann, E.G., Jr., M.R. Garey, and D.S. Johnson [1984], Approximation algorithms for bin packing - an updated survey, in: G. Ausiello, M. Lucertini, and P. Serafini (Eds.), *Algorithms Design and Computer System Design*, CISM Courses and Lectures 284, Springer, Vienna, 49-106.

Dhall, S.K. and C.L. Liu [1978], On a real-time scheduling problem, *Operations Research* **26**, 127-140.

Edmonds, J. [1965], Paths, trees and flowers, *Canadian Journal of Mathematics* **17**, 449-467.

Garey, M.R. and D.S. Johnson [1979], *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco.

Garey, M.R., D.S. Johnson, G.L. Miller, and C.H. Papadimitriou [1980], The complexity of coloring circular arcs and chords, *SIAM Journal on Algebraic and Discrete Methods* **1**, 216-227.

Gonzalez, M.J. and J.W. Soh [1975], Periodic job scheduling in a distributed processor system, *IEEE Transactions on Aerospace and Electronic Systems* **12**, 530-536.

Gupta, U.I., D.T. Lee, and J.Y.-T. Leung [1979], An optimal solution for the channel-assignment problem, *IEEE Transaction on Computers* **28**, 807-810.

Hashimoto, A. and J. Stevens [1971], Wire routing by optimizing channel assignment with large apertures, *Proceedings of the 8th Design Automation Conference*, 155-169.

Hopcroft, J.E. and R.M. Karp [1973], An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, *SIAM Journal on Computing* **2**, 225-231.

Korst, J.H.M., E.H.L. Aarts, J.K. Lenstra, and J. Wessels [1991], Periodic Assignment and Graph Colouring, *Philips Research Manuscript.*

Labetoulle, J. [1974], Some theorems on real time scheduling, in: E. Gelenbe and R. Mahl (Eds.), *Computer Architecture and Networks*, North-Holland, Amsterdam, 285-293.

Lawler, E.L. and C.U. Martel [1981], Scheduling periodically occurring tasks on multiple processors, *Information Processing Letters* **12**, 9-12.

Leung, J.Y.-T. and M.L. Merrill [1980], A note on preemptive scheduling of periodic, real-time tasks, *Information Processing Letters* **11**, 115-118.

Leung, J.Y.-T. and J. Whitehead [1982], On the complexity of fixed-priority scheduling of periodic, real-time tasks, *Performance Evaluation* **2**, 237-250.

Liu, C.L. and J.W. Layland [1973], Scheduling algorithms for multiprogramming in a hard real-time environment, *Journal of the Association for Computing Machinery* **20**, 46-61.

Orlin, J.B. [1982], Minimizing the number of vehicles to meet a fixed periodic schedule: an application of periodic posets, *Operations Research* **30**, 760-776.

Orlin, J.B., M.A. Bonuccelli, and D.P. Bovet [1981], An $O(n^2)$ algorithm for coloring proper circular arc graphs, *SIAM Journal on Algebraic and Discrete Methods* **2**, 88-93.

Park, K.S. and D.K. Yun [1985], Optimal scheduling of periodic activities, *Operations Research* **33**, 690-695.

Serafini, P. and W. Ukovich [1989], A mathematical model for periodic scheduling problems, *SIAM Journal on Discrete Mathematics* **2**, 550-581.

Shih, W.-K. and W.-L. Hsu [1989], An $O(n^{1.5})$ algorithm to color proper circular arcs, *Discrete Applied Mathematics* **25**, 321-323.

Tucker, A [1975], Coloring a family of circular arcs, *SIAM Journal on Applied Mathematics* **29**, 493-552.

Vince, J. [1989], Scheduling periodic events, *Discrete Applied Mathematics* **25**, 299-310.