ISR TECHNICAL REPORT 2013-08

Scheduling Perfectly Periodic Services Quickly with Aggregation

Jeffrey W. Herrmann

Scheduling Perfectly Periodic Services Quickly with Aggregation

Jeffrey W. Herrmann
A. James Clark School of Engineering
2181 Martin Hall
University of Maryland
College Park, MD 20742
301-405-5433
jwh2@umd.edu

Abstract

The problem of scheduling periodic services that have different period lengths seeks to find a schedule in which the workload is nearly the same in every time unit. A time unit's workload is the sum of the workloads of the services scheduled for that time unit. A level workload minimizes the variability in the resources required and simplifies capacity and production planning. This paper considers the problem in which the schedule for each service must be perfectly periodic, and the schedule length is a multiple of the services' period lengths. The objective is to minimize the maximum workload. The problem is strongly NP-hard, but there exist heuristics that perform well when the number of services is large. Because many services will have the same period length, we developed a new aggregation approach that separates the problem into subproblems for each period length, uses the subproblem solutions to form aggregate services, schedules these, and then creates a solution to the original instance. We also developed an approach that separates the problem into subproblems based on a partition of the period lengths. Computational experiments show that using aggregation generates high-quality solutions and reduces computational effort. The quality of the partition approach depended upon the partition used.

Introduction

The problem of scheduling periodic services that have different period lengths is an important problem that occurs in production, maintenance, and other applications. Typically, a firm has agreed to provide a periodic service to a customer by performing a specific task at a specific frequency. Different customers have different requirements, which creates an interesting scheduling problem. The period length (the time between consecutive tasks) varies based on the customer's preferences, and the workload associated with each customer varies based on the task that needs to be accomplished each time. The time unit could be a week, so a period length equals a number of weeks, while the workload is in man-hours.

For example, a firm that maintains and sells access to a database of commercial real estate properties must verify and update the status of the information on each property so that they have accurate information. For each property, one of the firm's researchers periodically calls a knowledgeable source to determine what information, if any, has changed since the last update. Different types of properties require different amounts of time for updating the information, and the firm has set a frequency for each property (for instance, check every 2 months). Kazan *et al.* (2012) discuss the problem of planning industrial waste management services that has similar characteristics.

This paper, like Kazan *et al.* (2012), focuses on the problem of minimizing the maximum workload when the tasks associated with each service must be completed on a perfectly periodic schedule. A time unit's workload is the sum of the workloads of the services scheduled for that time unit. A level workload minimizes the variability in the resources required and simplifies capacity and production planning. This is known as the Perfectly Periodic Service Scheduling (PPSS) problem. This paper is not concerned with how the tasks scheduled for the same time unit will be performed. We assume that the workload associated with a service is independent of when it is scheduled and is independent of which other services are scheduled for the same time unit.

This paper presents an aggregation approach that (1) separates an instance by formulating and solving a set of subproblems, (2) creates aggregate services from the solutions to these subproblems, (3) schedules the aggregate services, and (4) disaggregates that solution to form a feasible solution to the original instance. The objective of this paper is to show that aggregation is a computationally efficient method for generating high-quality policies. The paper precisely defines the aggregation approach, presents performance bounds, and discusses computational results that demonstrate its performance.

This paper also presents an approach that separates the problem into subproblems by partitioning the set of period lengths, which partitions the set of services. After each subproblem is solved, either the solutions are combined directly or the solutions are used to form aggregate services that are scheduled as in the aggregation approach.

The remainder of the paper proceeds as follows: we will discuss related work, formulate the PPSS problem, and then present a schedule construction heuristic. Then, we discuss the algorithms for aggregating an instance and disaggregating a solution for an aggregate instance and present performance bounds and a lower bound that will be used to evaluate the quality of the solutions. We then discuss the results of computational experiments designed to evaluate the effectiveness of the heuristics, the aggregation approach, and the partition approach.

Related Work

Motivated by the problem of planning industrial waste management services, Kazan *et al.* (2012) proved that the PPSS problem is strongly NP-hard and introduced a heuristic approach (the BestFit algorithm) for generating good schedules. They used the BestFit algorithm to generate schedules to real-world instances that ranged in size from 264 services to 4274 services. They provided a worst-case performance bound for the BestFit algorithm.

Park and Yun (1985) presented an integer linear programming model and separated it into subproblems using the Chinese Remainder Theorem, with one subproblem for each set of period lengths. As Kazan *et al.* (2012) point out, this approach would not reduce the size of instances (like those used here) in which the period lengths are {2, 3, 4, 6, 8, 12, 16, 24, 48}.

The Periodic Maintenance Scheduling Problem, discussed by Wei and Liu (1983), is a version of the PPSS problem in which all of the services have the same workload, and there is a constraint that limits the workload that can be scheduled in any time unit. The problem is to determine if any feasible schedule exists. Bar-Noy *et al*. (2002a) discussed a generalized maintenance scheduling problem that seeks to minimize the total cost of maintaining and operating a set of machines.

Herrmann (2009) considered the response time variability (RTV) problem when multiple servers, working in parallel, are available, and presented a specific aggregation approach. The results showed that, in most cases, combining aggregation with other heuristics does dramatically reduce both RTV and computation time compared to using the heuristics without aggregation.

Other work has considered problem in which a single resource must perform certain tasks perfectly periodically or as close to the ideal as possible. This include Bar-Noy *et al*. (2002b), Campbell and Hardin (2005), Corominas *et al*. (2007), Waldspurger and Weihl (1995), Hajek (1985), Altman *et al*. (2000), Sano *et al*. (2004). Kubiak (2004) provided a good overview of the need for fair sequences in different domains and presented results for multiple related problems, including the product rate variation problem, generalized pinwheel scheduling, the hard real-time periodic scheduling problem, the periodic maintenance scheduling problem, stride scheduling, minimizing response time variability (RTV), and peer-to-peer fair scheduling. See also Kubiak (2009) and Kazan *et al.* (2012) for additional references to other work on cyclic scheduling.

Aggregation is a well-known and valuable technique for solving optimization problems, especially large-scale mathematical programming problems. Model aggregation replaces a large optimization problem with a smaller, auxiliary problem that is easier to solve (Rogers *et al.*, 1991). The solution to the auxiliary model is then disaggregated to form a solution to the original problem. Model aggregation has been applied to a variety of production and distribution problems, including machine scheduling problems. For example, Rock and Schmidt (1983) and Nowicki and Smutnicki (1989) aggregated the machines in a flow shop scheduling problem to form a two-machine problem. Previous work has developed and studied aggregation approaches for the RTV problem, the waiting time problem (WTP), and the balanced word problem (BWP) (Herrmann, 2007, 2008, 2009, 2010, 2011a, b, 2012). Those problems seek to minimize the deviation of the schedule from a perfectly periodic one, whereas the PPSS problem considered here seeks to minimize the variability of workload in a perfectly periodic schedule.

PPSS Problem Formulation

We are given a set of n services that require scheduling. A service may correspond to a particular customer or location or equipment that needs periodic service like cleaning or maintenance. Thus, each service generates a set of tasks that must be done periodically. Service i has a period length p_i and workload w_i. A task must be scheduled every p_i time units, and the task adds w_i to the workload for the time unit in which it is scheduled. The schedule length equals J time units, where J is the least common multiple of $p_1, \ldots, p_n$. Thus, service i will require tasks in J / p_i time units and will add w_i to the workload for the time units in which its tasks are scheduled. The first occurrence of a task for service *i* must be in one of the first p_i time units. Once this is determined, the schedule for the remaining tasks for this service follows. Note that any services with $p_i = 1$ are trivial to schedule. Thus, we will consider only services with $p_i \geq 2$.

Let s_i be the time unit in which the first task of service *i* is scheduled. A schedule is completely specified when all of the s_i are determined. Clearly, $1 \leq s_i \leq p_i$. Let S_j be the set of services that have tasks scheduled in time unit *j*, and let W_j be the total workload of time unit *j*, for $j = 1, \ldots, J$. Then, $i \in S_j$ if and only if $j = s_i (\text{mod}\, p_i)$. $W_j = \sum_{i \in S_j} w_i$.

The objective function is to minimize max {W_1, …, W_J }.

Thus, we can describe the PPSS problem as follows: Given an instance {(p_i, w_i), $i = 1, \ldots, n$ }, find a schedule of length *J* that minimizes max {W_1, …, W_J } subject to the constraints that $1 \leq s_i \leq p_i$ for $i = 1, \ldots, n$.

Kazan *et al.* (2012) presented an integer programming formulation of the PPSS problem and proved that the PPSS problem is strongly NP-hard.

Given an instance, let W^* be the optimal value of max {W_1, …, W_J }. Then, because the total workload over all *J* time units equals $\sum_{i=1}^{n} w_i J / p_i$, it is easy to see that $W^* \geq \sum_{i=1}^{n} w_i / p_i$. We denote this lower bound as LB.

It will be convenient to define the following for an instance of the PPSS problem: let N_{PL} be the number of distinct values of the period length, let p'_k be the *k*-th value, with $k = 1, ..., N_{PL}$, and let A_k be the set of services that have $p_i = p'_k$.

Consider the following special case: if each and every set A_k had exactly p'_k services and the workloads of all of the services in A_k were equal, then an optimal schedule can be found easily by assigning each of the services in A_k to a different time period. This would create a schedule in which every time period has the same total workload. The aggregation approach described later attempts to create such an instance.

BestFit Heuristic

Kazan *et al.* (2012) presented the BestFit heuristic and showed that it can generate high-quality solutions, especially when the number of services is large (over 1000). The worst-case computational effort is $O\left(n \log n + nJ\right)$.

BestFit algorithm

The BestFit algorithm can be described as follows. The input is an instance $\{\left(p_i, w_i\right)$, $i = 1, \ldots, n\}$. Let *J* be the least common multiple of $p_1, \ldots, p_n$.

```
1. Sort the services and renumber so that $w_1 \geq w_2 \geq \cdots \geq w_n$.
2. Set $W_j = 0$ for $j = 1, \ldots, J$.
3. For $i = 1, \ldots, n$, perform the following steps:
    a. Set $W'_j = \max\left\{W_k : k = j, p_i + j, \ldots, J - p_i + j\right\}$ for $j = 1, \ldots, p_i$.
    b. Find j* such that $W'_{j*} = \min\left\{W'_1, \ldots, W'_{p_i}\right\}$.
    c. Set $s_i = j*$ and add $w_i$ to $W_{j*}, \ldots, W_{J-p_i+j*}$.
4. Return $s_1, ..., s_n$ as the schedule.
```

Note that Kazan *et al.* (2012) considered other ways to sort the services; our results showed that sorting by workload had the best performance.

Let W^* be the optimal maximum workload, let H be the maximum workload of the PPSS schedule generated by the BestFit heuristic, and let p be the period length of the service that determines the maximum workload. Then, if $p^2 \le J$, $H / W^* \le J / p$. Otherwise, $H / W^* \le 1 + J(p-1)/p^2$ (Kazan *et al.*, 2012).

Aggregation Approach

To improve the performance of this heuristic, we developed and tested an aggregation approach that (1) separates an instance by formulating and solving a set of subproblems, (2) creates aggregate services from the solutions to these subproblems, (3) schedules the aggregate services, and (4) disaggregates that solution to form a feasible solution to the original instance. Before giving the details of the approach, we will consider some of the ideas that motivated and justify the approach.

First, if, for period length p'_k, one could find a schedule for the services in A_k in which the total workload was the same in every time period, the total workload in every time period that schedule would equal $\frac{1}{p'_k}\sum_{i \in A_k} w_i$. (As Kazan *et al*. (2012) remarked, when all period lengths are equal, then the PPSS problem is equivalent to the problem of minimizing the makespan of a schedule for parallel machines.) If such a schedule could be found for every period length ($k = 1, \ldots, N_{PL}$), then these schedules, when combined, would form a complete, feasible schedule for the entire set of services, and the total workload in every time period would equal $\sum_{k=1}^{N_{PL}} \frac{1}{p'_k} \sum_{i \in A_k} w_i = \sum_{i=1}^{n} w_i / p_i$, which is the lower bound LB, so this ideal schedule must be optimal.

In addition, we note that, in any feasible solution to the PPSS problem, a time period may have multiple tasks that are generated by services that have the same period length (say p). The tasks generated by this set of services will occur in *J*/*p* (evenly spaced) time periods. Thus, in this schedule, these services are equivalent to one (aggregate) service that has a workload equal to the sum of these corresponding workloads and the same period length.

Thus, the ideal schedule mentioned in the previous paragraph is equivalent to a schedule in which there are p'_k (aggregate) services that have period length p'_k and all of the (aggregate) services with the same period length have the same workload. Thus, we have the special case that was identified earlier.

These ideas motivated the aggregation approach, which transforms any instance into another instance that nearly fits the conditions of the special case. The aggregation approach first separates an instance of the PPSS problem into subproblems. There is one subproblem for each distinct period length in the instance, and each subproblem has all of the services with that period length. Each of the subproblems is a parallel machine scheduling problem in which the number of machines equals *p*, the associated period length. A solution to any subproblem can be viewed as *p* aggregate services, one for each machine, where the workload of each aggregate service is the total processing time on the corresponding machine. Note that each aggregate service has the workload of one or more services from the original PPSS instance.

An aggregate instance of the PPSS problem can be created with all the aggregate services from all of the subproblems. The total workload of the aggregate instance equals the total workload of the original instance, but the total number of services has been reduced. Any solution for the aggregate instance can be transformed into a solution for the original instance by starting each service in the time unit in which its aggregate service starts.

More precisely, the aggregation algorithm proposed here involves separation (splitting the PPSS problem into subproblems), aggregation (combining services into aggregate services), and disaggregation (constructing a solution for the original instance from a solution for the aggregate instance).

The notation used in the algorithm that follows enables us to keep track of the services in order to describe the disaggregation of a schedule precisely. Let I_0 be the original instance, let N_{PL} be the number of distinct values of the period length, and let p'_k be the *k*-th value, with $k = 1, ..., N_{PL}$. For $k = 1, ..., N_{PL}$, let I_k be the *k*-th subproblem generated from I_0 and let n'_k be the number of services with $p_i = p'_k$. Let I_A be the aggregate instance. Let B_{kh} be the set of services that form an aggregate service, $k = 1, ..., N_{PL}$, $h = 1, ..., p'_k$.

As the aggregation algorithm is presented, we describe its operation on the following nine-service example: I_0 = {(2, 6), (2, 4), (2, 3), (2, 2), (2, 2), (3, 8), (3, 6), (3, 5), (3, 2)}, n = 9, $N_{PL} = 2$, and J = 6.

Aggregation. Given: an instance I_0 with $\{ (p_i, w_i),\ i = 1, \ldots, n \}$.

1. For $k = 1, ..., N_{PL}$, perform the following steps:

   a. If $n'_k \le p'_k$, go to the next value of *k*.

   b. Create an instance I_k of P//Cmax as follows: the number of machines equals p'_k. There are n'_k jobs, with one job for every service *i* with $p_i = p'_k$; the processing time of that job equals w_i.

   c. Generate a solution to the instance of P//Cmax. For $h = 1, ..., p'_k$, let B_{kh} be services whose jobs are scheduled on machine *h*.

      Example. With *k* = 1, p'_k = 2, n'_k = 5. If the solution to the two-machine problem schedules services 1 and 5 on machine 1 and services 2, 3, and 4 on machine 2, then

$B_{11} = \{1, 5\}$ and $B_{12} = \{2, 3, 4\}$. With k = 2, p'_k = 3, n'_k = 4. If the solution to the three-machine problem schedules service 6 on machine 1, service 7 on machine 2, and services 8 and 9 on machine 3, then $B_{21} = \{6\}$, $B_{22} = \{7\}$, and $B_{23} = \{8, 9\}$.

2. For $k = 1, ..., N_{PL}$, perform the following steps:

a. If $n'_k \le p'_k$, then create n'_k aggregate services as follows: for $h = 1, ..., n'_k$, B_{kh} = {[*h*]}, the period length equals p'_k, and the workload of the aggregate service is $w'_{kh} = w_{[h]}$, where [*h*] is the index of the *h*-th service with a period length equal to p'_k. Go to the next value of *k*.

b. Consider the solution to instance I_k. For $h = 1, ..., p'_k$, create an aggregate service, set the period length to p'_k, and set the workload $w'_{kh} = \sum_{i \in B_{kh}} w_i$.

Example. With k = 1, p'_k = 2. Create two aggregate services with workloads $w'_{11} = w_1 + w_5 = 8$ and $w'_{12} = w_2 + w_3 + w_4 = 9$. With k = 2, p'_k = 3. Create three aggregate services with workloads $w'_{21} = w_6 = 8$, $w'_{22} = w_7 = 6$, and $w'_{23} = w_8 + w_9 = 7$.

3. Create an aggregate instance I_A = $\{ \left(p'_k, w'_{kh} \right),\ k = 1, ..., N_{PL},\ h = 1, ..., \min\left\{ n'_k, p'_k \right\} \}$ of the PPSS problem. The total number of aggregate services $N_{AS} = \sum_{k=1}^{N_{PL}} \min\left\{ n'_k, p'_k \right\}$. Generate a schedule for the aggregate instance that specifies the start time s'_{kh} of each aggregate service.

Example. The aggregate instance has 5 aggregate services: {(2, 8), (2, 9), (3, 8), (3, 6), (3, 7)} Note that *J* still equals 6. A feasible schedule for this aggregate instance has $s'_{11} = 2$, $s'_{12} = 1$, $s'_{21} = 1$, $s'_{22} = 3$, and $s'_{23} = 2$. The workloads are 17, 15, 15, 16, 16, and 14.

4. Generate a schedule for the original instance as follows: for $k = 1, ..., N_{PL}$, $h = 1, ..., \min\{n'_k, p'_k\}$, and $i \in B_{kh}$, set $s_i = s'_{kh}$.

Example. $B_{11} = \{1, 5\}$, so $s_1 = s_5 = s'_{11} = 2$. $B_{12} = \{2, 3, 4\}$, so $s_2 = s_3 = s_4 = s'_{12} = 1$. $B_{21} = \{6\}$, so $s_6 = s'_{21} = 1$. $B_{22} = \{7\}$, so $s_7 = s'_{22} = 3$. $B_{23} = \{8, 9\}$, so $s_8 = s_9 = s'_{23} = 2$.

The worst-case computational effort of the aggregation procedure depends upon the algorithms used to solve the subproblems (which are parallel machine scheduling problems) and the aggregate instance. If a list scheduling procedure is used for the parallel machine scheduling problems, then, although the worst-case computational effort of each one is $O\left(n'_k \log n'_k\right)$, the worst-case computational effort required for all of them is $O\left(\sum_{k=1}^{N_{PL}} n'_k \log n'_k\right)$, which is less than $O\left(n \log n\right)$.

If BestFit is used to create a schedule for the aggregate instance, the worst-case computational effort of that procedure is $O\left(N_{AS} \log N_{AS} + N_{AS} J\right)$. Because $N_{AS} \leq \sum_{k=1}^{N_{PL}} p'_k$, then the computational effort of solving the aggregate problem remains constant as the number of services increases beyond $\sum_{k=1}^{N_{PL}} p'_k$ if the set of period lengths remains constant. The disaggregation of the schedule for the aggregate instance requires $O\left(n\right)$ time. Thus, if list scheduling and BestFit are used, the worst-case computational effort of the aggregation procedure is not worse than that worst-case computational effort of BestFit. (We also note that the parallel machine subproblems could be solved in parallel.) This fact and the small increase (at most 1/3) in the worst-case relative performance (discussed in the next section) indicate that the aggregation procedure should, in general, be faster than the BestFit procedure but may possibly generate slightly lower-quality solutions.

Performance Bounds

This section will present bounds on the worst-case performance of the aggregation procedure. First, we consider the parallel machine scheduling problem $P \| C_{\max}$. Let m be the number of machines, let $C^*_{\max}$ be the optimal makespan for an instance, and let $C^{LPT}_{\max}$ be the makespan of the schedule found using the longest processing time (LPT) first list scheduling rule. The worst case performance of the LPT first list scheduling rule is $C^{LPT}_{\max} / C^*_{\max} \le \frac{4}{3} - \frac{1}{3m}$ (Graham, 1969).

Now, consider an instance of PPSS. Let W^* be the optimal maximum workload, let W^A be the maximum workload of the schedule generated by the aggregation procedure, and let p and w' be the period length and workload of the aggregate service that determines the maximum workload of this schedule (the deciding aggregate service). (Note that the following proof follows the ideas of Graham, 1969, and Kazan *et al*., 2012.)

Theorem 1. The worst-case performance guarantee of the aggregation procedure is $W^A / W^* \le J / p$ if $p^2 \le J$, and $W^A / W^* \le \frac{J}{p} + \left(1 - \frac{J}{p^2}\right)\left(\frac{4}{3} - \frac{1}{3p}\right)$ if $p^2 > J$.

Proof. The aggregate service was determined by solving a parallel machine scheduling problem using the LPT first list scheduling rule. The scheduling problem had p machines, and the workload w' of the aggregate service is the total processing time of the jobs on one machine. Thus, if we let $C^*_{\max}$ be the optimal makespan for this problem, $w' \le C^{LPT}_{\max} \le \left(\frac{4}{3} - \frac{1}{3m}\right) C^*_{\max}$. Moreover, because all of the jobs in the parallel machine scheduling problem are services in the PPSS instance, it is clear that $C^*_{\max} \le W^*$. Thus, $w' \le \left(\frac{4}{3} - \frac{1}{3m}\right) W^*$.

Recall that, when the deciding aggregate service is added, $W'_{j*} = \min\left\{W'_1, \ldots, W'_p\right\}$. Thus, there are at least p time periods that have a workload of at least W'_{j*}. Thus, $W^* \geq \left(pW'_{j*} + \frac{J}{p}w'\right)/J$, which implies that $W'_{j*} \leq \frac{J}{p}\left(W^* - \frac{1}{p}w'\right)$. Because $W^A = W'_{j*} + w'$, $W^A \leq \frac{J}{p}W^* + \left(1 - \frac{J}{p^2}\right)w'$.

If $p^2 \leq J$, then $W^A / W^* \leq J/p$.

If $p^2 > J$, then because $w' \leq \left(\frac{4}{3} - \frac{1}{3m}\right)W^*$, $W^A / W^* \leq \frac{J}{p} + \left(1 - \frac{J}{p^2}\right)\left(\frac{4}{3} - \frac{1}{3p}\right)$. Q.E.D.

This performance guarantee allows one to bound the relative error by considering the worst-case value of *p*. For the cases considered in this paper, the bound decreases as *p* increases. When *J* = 48 and the smallest value of *p* = 2, then we know only that $W^A / W^* \leq J/p = 24$. For instances with no small period lengths, the bound will decrease. If, for instance, the set of period lengths is {8, 12, 16, 24, 48}, then *J* = 48, the smallest value of *p* = 8, and this performance guarantee shows that $W^A / W^* \leq \frac{J}{p} + \left(1 - \frac{J}{p^2}\right)\left(\frac{4}{3} - \frac{1}{3p}\right) = 6\frac{31}{96}$. If the set of period lengths is only {24, 48}, then this performance guarantee shows that $W^A / W^* \leq \frac{J}{p} + \left(1 - \frac{J}{p^2}\right)\left(\frac{4}{3} - \frac{1}{3p}\right) = 3\frac{181}{864}$.

Partitioning Approach

We also considered a partitioning approach that generalizes the aggregation approach. Instead of considering one period length at a time, this partitioning approach considered multiple period lengths simultaneously. Given a partition of the set of period lengths into multiple subsets so that the period lengths in any one subset were multiples of each other, the first step of the aggregation approach created one instance of the PPSS problem for each subset. This instance contained only the services with the period lengths in that subset. The schedule length was the least common multiple of the period lengths in that subset. The partition approach used the BestFit heuristic to construct a schedule for that instance (subproblem).

Two versions of combining the solutions to the subproblems were considered. The first, which we call *partition-aggregation*, formed aggregate services from each solution. The number of aggregate services (and each one's period length) was the smallest period length of any service in that subproblem. The workload of the aggregate service was the maximum workload of the time units separated by the period length. For example, if the period lengths were 2, 4, and 8, a schedule has 8 time units. The first aggregate service corresponds to time units 1, 3, 5, and 7, so its workload is the maximum workload of these time units, and the aggregate service represents all of the services scheduled in these time units; likewise, the second aggregate service corresponds to time units 2, 4, 6, and 8 and the services scheduled in these time units.

Then, like the aggregation approach, this approach used the BestFit heuristic to construct a schedule for the aggregate instance and disaggregated the schedule to create a solution for the original instance. Note that using the aggregation approach is equivalent to using the partition-aggregation approach with a partition in which each period length is its own subset.

Partition-Aggregation. Given: an instance I_0 with $\{ (p_i, w_i),\ i = 1, \ldots, n \}$ and a partition of the period lengths into N_{PT} subsets such that period length k is in subset P_k, where $P_k \in \{1, \ldots, N_{PT}\}$ for $k = 1, ..., N_{PL}$.

1. For $r = 1, ..., N_{PT}$, perform the following steps:

a. Create an instance I_r of PPSS: $I_r = \{ (p_i, w_i) :\ i \in S_k$ and $P_k = r \}$. This instance will have $\sum_{k:P_k = r} n'_k$ services. Set J'_r to be the least common multiple of $\{ p'_k : P_k = r \}$ and set $p_{\min,r} = \min \{ p'_k : P_k = r \}$.

b. Generate a solution to this instance. Let W_{rj} denote the scheduled workload for time unit *j* for $j = 1, ..., J'_r$. For service *i* in I_r, let $\overline{s}_{ri}$ denote the time unit of the first scheduled task, and set $\delta_{ri} = p_{\min,r} \left\lfloor \left(\overline{s}_{ri} - 1 \right) / p_{\min,r} \right\rfloor$, where $\lfloor x \rfloor$ is the greatest integer less than or equal to *x*. This value is the interval between the scheduled start time and the time unit for the corresponding aggregate service. Note that $1 \le \overline{s}_{ri} - \delta_{ri} \le p_{\min,r}$.

2. For $r = 1, ..., N_{PT}$, perform the following step:

a. Consider the solution to instance I_r. For $h = 1, ..., p_{\min,r}$, create an aggregate service, set the period length to $p_{\min,r}$, and set the workload $w'_{rh} = \max \left\{ W_{rj} : j = h, p_{\min,r} + h, \ldots, J'_r - p_{\min,r} + h \right\}$.

3. Create an aggregate instance I_A = $\{ \left(p_{\min,r}, w'_{rh} \right),\ r = 1, ..., N_{PT},\ h = 1, ..., p_{\min,r} \}$ of the PPSS problem. The total number of aggregate services equals $\sum_{r=1}^{N_{PT}} p_{\min,r}$. The schedule length J'_A is the least common multiple of $\left\{ p_{\min,r} : r = 1, \ldots, N_{PT} \right\}$. Generate a schedule for the aggregate instance that specifies the start time s'_{rh} of each aggregate service.

4. For $r = 1, ..., N_{PT}$, perform the following step:

a. For every service *i* in I_r, set $h' = \overline{s}_{ri} - \delta_{ri}$ and then $s_i = s'_{rh'} + \delta_{ri}$.

Example. To demonstrate this algorithm, we describe its operation on the following eleven-service example: I_0 = {(4, 5), (2, 6), (2, 1), (2, 4), (6, 1), (2, 2), (4, 1), (6, 2), (3, 5), (3, 4), (3, 3)}, *n* = 11, $N_{PL} = 4$, and *J* = 12. The partition is {2, 4} and {3, 6}, so $N_{PT} = 2$.

Step 1. The first subproblem corresponds to the first subset in the partition: I_1 = {(4, 5), (2, 6), (2, 1), (2, 4), (2, 2), (4, 1)}, $J_1' = 4$, and $p_{\min,1} = 2$. A feasible solution to this subproblem has the following start times: $\bar{s}_{11} = 2$, $\bar{s}_{12} = 1$, $\bar{s}_{13} = 1$, $\bar{s}_{14} = 2$, $\bar{s}_{16} = 1$, $\bar{s}_{17} = 4$. $\delta_{1i} = 0$ for all services except $\delta_{17} = 2\lfloor (4-1)/2 \rfloor = 2$ because service 7 starts in time unit 4 but will be part of the second aggregate service from this subproblem. The time unit workloads are (9, 9, 9, 5).

The second subproblem corresponds to the second subset in the partition: I_1 = {(6, 1), (6, 2), (3, 5), (3, 4), (3, 3)}, $J_2' = 6$, and $p_{\min,2} = 3$. A feasible solution to this subproblem has the following start times: $\bar{s}_{25} = 6$, $\bar{s}_{28} = 3$, $\bar{s}_{29} = 1$, $\bar{s}_{2,10} = 2$, $\bar{s}_{2,11} = 3$. $\delta_{2i} = 0$ for all services except $\delta_{25} = 3\lfloor (6-1)/3 \rfloor = 3$ because service 5 starts in time unit 6 but will be part of the third aggregate service from this subproblem. The time unit workloads are (5, 4, 5, 5, 4, 4).

Step 2. With r = 1, $p_{\min,1}$ = 2. Create two aggregate services with workloads $w_{11}' = \max\{W_{11}, W_{13}\} = 9$ and $w_{12}' = \max\{W_{12}, W_{14}\} = 9$. Their period length equals 2.

With r = 2, $p_{\min,2}$ = 3. Create three aggregate services with workloads $w_{21}' = \max\{W_{21}, W_{24}\} = 5$ and $w_{22}' = \max\{W_{22}, W_{25}\} = 4$, and $w_{23}' = \max\{W_{23}, W_{26}\} = 5$. Their period length equals 3.

Step 3. The aggregate instance has 5 aggregate services: {(2, 9), (2, 9), (3, 5), (3, 4), (3, 5)} Note that J_A' equals 12 (which is shorter than the actual schedule length). A feasible schedule for this aggregate instance has $s_{11}' = 1$, $s_{12}' = 2$, $s_{21}' = 1$, $s_{22}' = 3$, and $s_{23}' = 2$.

Step 4. With r = 1, $s_1 = s_4 = s_{12}' = 2$, $s_2 = s_3 = s_6 = s_{11}' = 1$, $s_7 = s_{12}' + \delta_{17} = 4$.

With r = 2, $s_5 = s_{23}' + \delta_{25} = 5$, $s_8 = s_{11} = s_{13}' = 2$, $s_9 = s_{31}' = 1$, $s_{10} = s_{22}' = 3$.

The time unit workloads are (14, 14, 13, 10, 13, 13, 14, 10, 13, 14, 13, 9).

Partition-Stacking. The second version of the partition approach, which we call *partition-stacking*, does not create aggregate services or solve an aggregate instance. Instead, after Step 1, this algorithm simply “stacks” the schedules for each subproblem to construct a complete solution for the original instance. Each service’s start time in its subproblem solution becomes the start time in the complete solution. The workload in each time unit is the sum of the workloads from the subproblem solutions (after repeating their schedules to fill the entire schedule length).

Consider the eleven-service example again. The solution to the first subproblem specifies feasible start times for services 1, 2, 3, 4, 6, and 7. The solution to the second subproblem specifies feasible start times for services 5, 8, 9, 10, and 11. These two solutions form a solution to the original instance, and the time unit workloads are (14, 13, 14, 10, 13, 13, 14, 9, 14, 14, 13, 9). Table 1 shows how the schedules are “stacked.”

Table 1. “Stacking” solutions to two subproblems. The entries show the workload in each time period for the solutions to both subproblems and the combined schedule for the original eleven-service example.

Time Unit:	1	2	3	4	5	6	7	8	9	10	11	12
Schedule 1:	9	9	9	5	9	9	9	5	9	9	9	5
Schedule 2:	5	4	5	5	4	4	5	4	5	5	4	4
Combined Schedule	14	13	14	10	13	13	14	9	14	14	13	9

Computational Experiments

The purpose of the computational experiments was to compare the performance of the aggregation approach, the partition approaching, and the BestFit heuristic. All of the algorithms were implemented in Matlab and executed using Matlab R2006b on a Dell Optiplex GX745 with Intel Core2Duo CPU 6600 @ 2.40 GHz and 2.00 GB RAM running Microsoft Windows XP Professional Version 2002 Service Pack 3.

We slightly modified the scheme of Kazan *et al.* (2012) and generated 81 new instances as follows.

We used nine values of *n*: 100, 250, 500, 1,000, 2,000, 3,000, 4,000, 5,000, and 10,000. The period lengths were chosen from the set {2, 3, 4, 6, 8, 12, 16, 24, 48}. $J = 48$ in all instances.

For each value of *n*, we generated nine instances with that many services, one instance of each of nine types. Each type had different distributions for period length and workload.

In types 1, 2, 3, 4, 8, and 9, all of the period lengths were equally likely. In type 5, smaller period lengths were more likely. In type 6, larger period lengths were more likely. In type 7, period lengths of 6, 8, and 12 were more likely. Table 2 lists the probability of each period length by instance type.

Table 2. Probability of each period length by instance type.

Period	Instance Type			
Length	1, 2, 3, 4, 8, 9	5	6	7
2	1/9	1/6	1/18	1/12
3	1/9	1/6	1/18	1/12
4	1/9	1/6	1/18	1/12
6	1/9	1/9	1/9	1/6
8	1/9	1/9	1/9	1/6
12	1/9	1/9	1/9	1/6
16	1/9	1/18	1/6	1/12
24	1/9	1/18	1/6	1/12
48	1/9	1/18	1/6	1/12

In addition, different probability distributions were used for the workloads. For types 1, 5, 6, and 7, the distribution was a uniform distribution on the range 0 to 200. For type 2, the distribution was a triangular distribution with a minimum of 0, a mode of 0, and a maximum of 200. For type 3, the distribution was a triangular distribution with a minimum of 0, a mode of 200, and a maximum of 200. For type 4, the distribution was a uniform distribution on the range 0 to 1000. For type 8 and 9, the distribution depended upon the period length of the service. For type 8, the distribution was a uniform distribution on the range 0 to $100 - p_i$ (so services with longer period lengths have smaller workloads). For type 9, the distribution was a uniform distribution on the range p_i to $p_i + 50$ (so services with shorter period lengths have smaller workloads).

For each combination of *n* and the instance type, we generated an instance by generating *n* random period lengths using the period length distribution for that type and *n* random workloads using the workload distribution for that type.

For testing the partitioning approach, nine partitions were considered:

A: {2, 4}, {3, 6}, {8, 16}, {12, 24}, and {48};

B: {2, 4}, {3, 6}, {8, 16}, and {12, 24, 48};

C: {2, 4, 8}, {3, 6, 12, 24}, {16, 48};

D: {2, 4, 8, 16}, {3, 6, 12}, and {24, 48};

E: {2, 4, 8, 16} and {3, 6, 12, 24, 48};

F: {2, 4, 8, 16, 48} and {3, 6, 12, 24};

G: {2, 4, 12}, {3, 6}, {8, 16}, and {24, 48};

H: {2, 4, 12, 24}, {3, 6}, and {8, 16, 48}; and

I: {2, 4, 12, 24, 48}, {3, 6}, and {8, 16}.

For each instance, we evaluated the lower bound LB, used the BestFit heuristic to construct a schedule, and used the aggregation algorithm to construct a schedule. We used LPT list scheduling to generate solutions to the parallel machine scheduling subproblems and the BestFit heuristic to construct a solution for the aggregate instance.

Table 3. Average number of aggregate services for the instances.

N	Average number of aggregate services
100	64
250	101
500	119
1,000	123
2,000	123
3,000	123
4,000	123
5,000	123
10,000	123

Before discussing the results of the heuristics, we consider first how many aggregate services were generated. Table 3 shows that, as *n* increases, the average number of aggregate services increases. For all instances with at least 1000 services, the number of aggregate services $N_{AS} = 123$, the sum of the period lengths, which is at least an order of magnitude less than *n*. Recall that all of these instances shared the same nine distinct values of period length ($N_{PL} = 9$).

To compare the schedule quality, for each instance, we report the relative deviation between the max workload of each schedule constructed and the lower bound (see Table 4).

Table 4. Average values of the relative deviation between the max workload and the lower bound for the schedules generated by the BestFit heuristic, the aggregation algorithm, and the partition approach using Partition E.

n	BestFit	Aggregation	Partition E-Stacking	Partition E-Aggregation
100	2.9%	3.6%	3.8%	3.8%
250	0.7%	0.9%	0.7%	0.7%
500	0.2%	0.6%	0.2%	0.2%
1,000	0.1%	0.2%	0.1%	0.1%
2,000	0.0%	0.0%	0.0%	0.0%
3,000	0.0%	0.0%	0.0%	0.0%
4,000	0.0%	0.0%	0.0%	0.0%
5,000	0.0%	0.0%	0.0%	0.0%
10,000	0.0%	0.0%	0.0%	0.0%

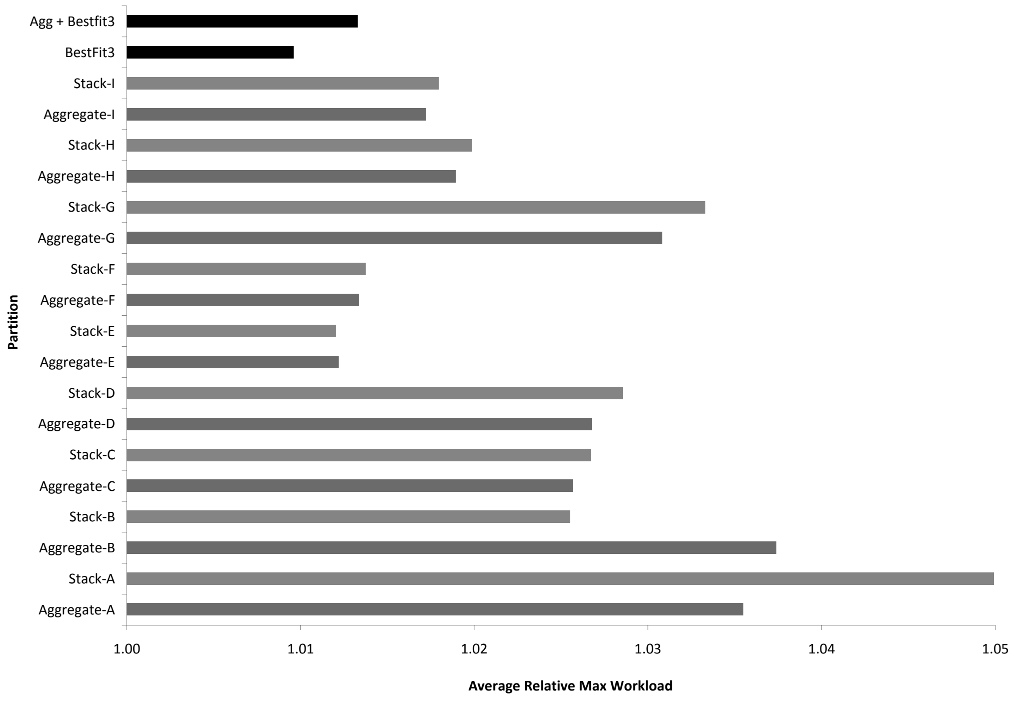

Figure 1. Schedule quality of the aggregation approach, the BestFit heuristic, the partition-aggregation approach, and the partition-stacking approach on instances with 100, 250, 500, and 1,000 services.

For the BestFit heuristic, the aggregation algorithm, and the partition approach, the gap from the lower bound decreases as the number of services increases. All of the algorithms generate near-optimal solutions for instances with at least 1,000 services.

For the smaller instances, the BestFit heuristic generates better schedules on average.

In a smaller instance, a large period length is unlikely to have enough services; that is, it is likely that $n'_k < p'_k$ for larger p'_k. Thus, in any schedule, these services will cause great variability in the workloads. Because the aggregate instance has fewer services than the original instance, the BestFit heuristic has less flexibility to smooth this variability.

As shown in Figure 1, for the instances with 100, 250, 500, and 1,000 services, the partitioning approach generated the best schedules using Partition E. Partitions E and F (which also generated better schedules) have only two subsets. Using Partitions A, B, and G (which have four or five subsets) on these instances generated schedules that were not as good. In general, for these instances, the partitioning approach generated schedules that were not as good on average as those generated by the aggregation approach and the BestFit heuristic.

The BestFit heuristic corresponds to a partition with only one subset, and the aggregation approach corresponds to a partition in which each period length has its own subproblem. These two special cases generated better schedules. Partitions A to I all involved forming aggregate services that approximated (indeed, overestimated) the actual workload, whereas the aggregate services in the aggregate approach did not use such approximations.

The various types of instances had little impact on the performance of the procedures. Not surprisingly, the quality of the schedules (measured as the gap from the lower bound) was better for type 8 instances (services with longer period lengths have smaller workloads) than for type 9 instances (services with shorter period lengths have smaller workloads), however. For type 8 instances, the average relative deviation between the max workload and the lower bound was 0.8% for the schedules generated by the BestFit heuristic and 0.6% for the schedules generated by the aggregation algorithm. For type 9 instances, the average relative deviation between the max workload and the lower bound was 2.1% for the schedules generated by the BestFit heuristic and 2.0% for the schedules generated by the aggregation algorithm.

Interestingly, the aggregation algorithm performed better than the BestFit heuristic and the partitioning approach on type 9 instances with fewer services. In particular, for type 9 instances with 100, 250, and 500 services, the average relative gap between the max workload and the lower bound was 3.0% for the schedules constructed by aggregation, 3.7% for the schedules constructed by the BestFit heuristic, 5.5% for the schedules constructed using partition E and stacking, and 5.6% for the schedules constructed using partition E and aggregation. In the type 9 instances, the services with long period lengths have greater workloads. Aggregation combines many services with short period lengths (but smaller workloads) into services with much greater workloads that can be used to create balanced schedules.

We also measured the clock time needed to generate these schedules. Table 5 shows the average time needed to generate schedules for the BestFit heuristic and the aggregation algorithm for different values of *n*. These are averages over nine instances. As *n* increased, the time required increased. The average time for the BestFit heuristic is proportional to *n*. The average time for the aggregation algorithm increased more slowly than *n* increased. Because its subproblem solution approach (LPT list scheduling) is simpler and quicker than the BestFit heuristic, the aggregation algorithm is faster than the BestFit heuristic and the partitioning approaches, and its relative performance increased as *n* increased. The partitioning approaches were not faster than then BestFit heuristic. Although the subproblems are small, solving them still requires using BestFit on every service.

Table 5. Average time required to generate a schedule by the BestFit heuristic, the aggregation algorithm, and the partitioning approach (in 1/1000 seconds).

n	BestFit	Aggregation	Partition-Stacking	Partition-Aggregation
100	4	4	5	7
250	11	8	11	14
500	21	12	21	25
1,000	43	15	43	48
2,000	86	20	85	93
3,000	127	25	125	137
4,000	170	30	167	183
5,000	213	35	209	228
10,000	422	61	413	448

Summary and Conclusions

This paper presented an aggregation approach for the problem of scheduling periodic services that have different period lengths, analyzed its computational effort, and presented a worst case performance bound. We combined this approach with existing heuristics in order to determine when aggregation is useful.

The results show that using aggregation reduces the computational effort needed to construct a schedule and the schedules are equally good for larger instances. For smaller instances, the quality of the schedules generated using aggregation are not as good. For smaller instances, the quality of the schedules generated using the partitioning approach depended upon the partition used but were not as good as those generated by the aggregation approach and did not reduce the computational effort. Thus, for the PPSS problem, we recommend using aggregation when the instances are large (greater than 1,000 services).

The results indicate that separating a large problem into subproblems will be faster only if the subproblems can be solved with faster algorithms that exploit the structure of the subproblems. This was done in the aggregation approach but not in the more general partitioning approach.

Because aggregation creates a smaller instance, it could be employed with an exact approach that requires too much computational effort to run on the original instance but is still reasonable on the smaller instance. Of course, an optimal solution to the smaller, aggregate instance would not necessarily yield, after disaggregation, an optimal solution to the original instance.

References

Altman, E., Gaujal, B., Hordijk, A. (2000). Balanced sequences and optimal routing. *Journal of the ACM*, 47(4), 752–775.

Bar-Noy, A., Bhatia, R., Naor, J., Schieber, B. (2002a). Minimizing service an operation costs of periodic scheduling. *Mathematics of Operations Research*, 27, 518-544.

Bar-Noy, Amotz, Aviv Nisgah, and Boaz Patt-Shamir (2002b), Nearly Optimal Perfectly Periodic Schedules, *Distributed Computing*, 15, 207-220.

Campbell, Ann Melissa, and Jill R. Hardin (2005), Vehicle Minimization for Periodic Deliveries, *European Journal of Operational Research*, 165, 668–684.

Corominas, A., Kubiak, W., Palli, N.M. (2007). Response time variability. *Journal of Scheduling*, 10, 97-110.

Graham, R.L. (1969). Bounds on Multiprocessing Timing Anomalies, *SIAM Journal on Applied Mathematics*, 17, 263-269.

Hajek, B. (1985). Extremal splittings of point processes. *Mathematics of Operations Research*, 10, 543-556.

Herrmann, J.W. (2007). Generating cyclic fair sequences using aggregation and stride scheduling. Technical Report 2007-12, Institute for Systems Research, University of Maryland, College Park. http://hdl.handle.net/1903/7082. Accessed 1 November 2010.

Herrmann, J.W. (2008). Constructing perfect aggregations to eliminate response time variability in cyclic fair sequences. Technical Report 2008-29, Institute for Systems Research, University of Maryland, College Park. http://hdl.handle.net/1903/8643. Accessed 1 November 2010.

Herrmann, J.W. (2009). Generating cyclic fair sequences for multiple servers. MISTA 2009, Dublin, Ireland, August 10-12, 2009.

Herrmann, J.W. (2010). “Using Aggregation to Construct Periodic Policies for Routing Jobs to Parallel Servers with Deterministic Service Times,” to appear in *Journal of Scheduling*. DOI: 10.1007/s10951-010-0209-6. Published online on November 24, 2010.

Herrmann, J.W. (2011a). Using aggregation to reduce response time variability in cyclic fair sequences. *Journal of Scheduling*, Volume 14, Number 1, pages 39-55, 2011.

Herrmann, J.W. (2011b), “Generating Better Cyclic Fair Sequences Faster with Aggregation,” Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2011), Phoenix, Arizona, August 9-12, 2011.

Herrmann, J.W. (2012), “Finding Optimally Balanced Words for Production Planning and Maintenance Scheduling,” *IIE Transactions*, Volume 44, Number 3, pages 215-229.

Kazan, Osman, Milind Dawande, Chelliah Sriskandarajah, and Kathryn E. Stecke (2012). “Balancing Perfectly Periodic Service Schedules: An Application from Recycling and Waste Management,” *Naval Research Logistics*, Volume 59, Issue 2, pages 160-171.

Kubiak, W. (2004). Fair sequences. In J.Y-T. Leung (Ed.), *Handbook of Scheduling: Algorithms, Models and Performance Analysis* (pp. 1-21). Boca Raton, Florida: Chapman & Hall/CRC.

Kubiak, W. (2009). *Proportional Optimization and Fairness*, New York: Springer.

Nowicki, E., Smutnicki, C. (1989). Worst-case analysis of an approximation algorithm for flow shop scheduling. *Operations Research Letters*, 8, 171-177.

Park, K.S., and D.K. Yun (1985). “Optimal Scheduling of Periodic Activities,” *Operations Research*, Volume 33, pages 690-696.

Rock, H., Schmidt, G. (1983). Machine aggregation heuristics in shop scheduling. *Methods of Operations Research*, 45, 303-314.

Rogers, D.F., Plante, R.D., Wong, R.T., Evans, J.R. (1991). Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39(4), 553-582.

Sano, S., Miyoshi, N., Kataoka, R. (2004). "*m*-balanced words: a generalization of balanced words. *Theoretical Computer Science*, 314(1-2), 97-120.

Waldspurger, C.A., Weihl, W.E. (1995). Stride scheduling: deterministic proportional-share resource management. Technical Memorandum MIT/LCS/TM-528, MIT Laboratory for Computer Science, Cambridge, Massachusetts.

Wei, W.D., Liu, C.L. (1983). On a periodic maintenance problem. *Operations Research Letters*, 2(2), 90-93.