Cyclic scheduling for parallel processors with precedence constrains

N Grigoreva
St. Petersburg State University, Universitetskaya nab. 7/9, St. Petersburg, Russia
E-mail: n.s.grig@gmail.com

Abstract. The goal of this paper is to prepare algorithms of the cyclic scheduling problem, in which some set of jobs \( V \) is to be repeated an infinite number of times. We consider the multiprocessors problem when a set of jobs is done on identical parallel processors. Cyclic scheduling problems arise (for example) in manufacturing, timesharing of processors in embedded systems. The goal is to find a periodic schedule that minimizes the cycle time under precedence constraints. Although the problem is NP-hard, we show that the special case, where the precedence graph \( G \) is a tree or there are only two processors and all jobs have a unit processing time, can be solved in polynomial time.

1. Introduction and related work
In classical scheduling, a set of jobs \( V \) is executed once, and the goal is to generate an optimal schedule. The usual objective function is completion time of the scheduled tasks also referred to as makespan and the goal is to minimize the makespan.

A cyclic scheduling problem is a scheduling problem in which some set of tasks \( V \) is to be repeated an infinitely number of times. These approaches are also applicable if the number of loop repetitions is large enough. Cyclic scheduling has multiple applications, such as robotics [1, 2], manufacturing systems [3], communications and transport or multiprocessor computing [4].

Cyclic scheduling applications usually deal with a periodic schedule, which is a schedule of one iteration that is repeated within a fixed time interval called the period (or cycle time). The aim of cyclic scheduling is to find a periodic schedule with the minimum period. Cyclic scheduling is not less difficult than non-cyclic scheduling since any non-cyclic scheduling problem polynomially reduces to a cyclic problem where successive iterations must not overlap.

Cyclic scheduling problems have been studied from several points. We are interested in parallel identical processors problems.

In [5] authors developed a new method to build periodic schedules with cumulative resource constraints, periodic release dates and deadlines. This paper deals with a realistic cyclic scheduling problem in the food industry environment in which parallel machines jobs with given release dates, due dates, and deadlines.

In this paper, we limit our study to periodic schedules with the objective of minimizing the period, which is equivalent to maximizing the throughput.

This paper is organized as follows. We will start with a brief description of the now standard constraint formulation of Multiprocessor scheduling problem in section 2, and then go on to present Periodic Scheduling on Identical Processors problem. In section 3 we consider Periodic Scheduling on Identical Processors problem with unit processing time. We propose the algorithm for the problem,
where the precedence constraints graph is a tree in section 4. Algorithm for the two parallel processors problem are provided in section 5. Section 6 contains a summary of this paper.

2. Multiprocessor cyclic scheduling problem

First, we consider a non-cyclic multiprocessor system of tasks $V = \{v_1, v_2, \ldots, v_n\}$. The execution time of each task $p(v_i)$ is known. Precedence constraints between tasks are represented by a directed acyclic task graph $G = (V, E)$. $E$ is a set of directed arcs, an arc $e = (v_i, v_j) \in E$, if and only if $v_i < v_j$. The expression $v_i < v_j$ means that the task $v_j$ may be initiated only after completion of the task $v_i$.

Set of tasks is performed on parallel identical processors, any task can run on any processor and each processor can perform no more than one task at a time. Task preemption is not allowed. The usual objective function is completion time of the scheduled task graph also referred to as makespan and the goal is to minimize the makespan. A schedule for the set $V$ is the mapping of each task $v_i \in V$ to a start time $t(v_i)$ and a processor $f(v_i)$. Makespan of schedule $S$ is the quantity

$$C_{\text{max}}(S) = \max\{t(v_i) + p(v_i) \mid v_i \in V\}.$$ 

We assume that the set of tasks $V = \{v_1, v_2, \ldots, v_n\}$ is a template which we wish to repeat indefinitely, for example the tasks represent the steps to build a single project and we wish to construct a schedule for a project factory. Then we can overlap the manufacture of multiple projects. A cyclic schedule for the problem is one in which a new project is begun every $z$ time units (the cycle time) and the same schedule of tasks is completed for each project. Assuming the time to complete each individual project $C_{\text{max}}(S)$ is greater than $z$.

The PSIP (Periodic scheduling identical Processors problem) [6] can be described as follows:

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of generic operations.

We denote by $< v_i; k >$ the $k$-th occurrence of the generic operation $v_i$.

Precedence relations are defined by a graph $G = (V, E)$ with vertex set $V$ and arc set $E$. Each arc $(v_i, v_j) \in E$ is supplied by two values $L_{ij} = p(v_i)$ and $H_{ij}$. $H_{ij}$ is called the height (or distance).

If $(v_i, v_j) \in E$ is a generic precedence constraint then for each iteration $k > 0$, the task $< v_i; k >$ must be completed before the task $< v_j; k + h >$ starts being performed. Let $m$ identical processors are available to execute the tasks. As usual, each task $< v_i; k >$ is performed by one processor and, at any instant, one processor may perform at most one task.

A schedule for the set $V$ is the mapping of each task $v_i \in V$ a start time $t(v_i; k)$ and a processor $f(v_i)$.

This graph leads to the following uniform precedence constraints

$$t(v_i; k) + p(v_i) \leq t(v_j; k + H_{ij}).$$

We also postulate that the $k + 1$-th occurrence of operation $v_i$ can only start if the $k$-th occurrence is finished. Thus, we get the following constraint

$$t(v_i; k) + p(v_i) \leq t(v_i; k + 1).$$

In the following we assume that the constraints are included in graph $G$ by adding loops $(v_i, v_i)$ with $L_{ii} = p(v_i)$ and $H_{ii} = 1$ to $E$.

**Definition 1** A schedule is called periodic with cycle time $z$, if $t(v_i; k) = t(v_i; 0) + kz$, for all $v_i \in V$, $k \in Z$.

The goal is to minimize a cycle time $z$ (sometimes called cycle length or period), which is the time between starting the first job in a cycle and starting the first job in the next cycle. Cycle time is roughly equivalent to static scheduling makespan. The goal is to define the optimal cycle time $z_{\text{opt}}$, and the starting time $t(v_i; k)$ of each occurrence of operation $v_i$ for all $v_i \in V$. As the starting time of the $k$-th
occurrence depends only on the starting time of the 0-th occurrence of job \( v_i \), it is sufficient to compute the starting time \( t_i = t(v_i; 0) \).

A periodic schedule \( \sigma = (t, z) \) is defined by the vector \( t(v_i), v_i \in V \) and the cycle time \( z \geq 0 \). The problem is to determine a feasible time-periodic schedule with a minimum cycle time. The cyclic scheduling problem is a cyclic version of the non-cyclic scheduling problem, the solution of which is characterized by the critical paths of the precedence graph \( G \). Critical circuits generalize this notion for uniform graph.[6]

Consider a circuit \( \mu \) in graph \( G \). Let \( L(\mu) = \sum_{(v_i,v_j) \in \mu} L_{ij} \) and \( H(\mu) = \sum_{(v_i,v_j) \in \mu} H_{ij} \). Then \( z(\mu) = L(\mu) / H(\mu) \) is called the value of \( \mu \). The circuits with the maximum value and positive height are called critical circuits. Then the value of a critical circuit \( z(G) \) is a lower bound for the optimal cycle time and \( LB = \max \{ \sum_{v_i \in V} p(v_i) / m, z(G) \} \) is a lower bound on the cycle time [7].

**Definition 2** Let \( \sigma = (t, z, f) \) is resource-periodic schedule with period \( z \) if \( f(v_i; k + z) = f(v_i; k) \).

Thus, a cyclic schedule is denoted by \( \sigma = (t, z, f) \).

**Definition 3** Let \( \sigma = (t, z, f) \) is a cyclic schedule. The occurrence vector \( \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \) of \( \sigma \) is defined by \( \alpha_i = (t_i - t_{i-1}) \mod z \). The pattern \( \pi = (\pi, z, f) \) of \( \sigma \) is defined as follows \( \pi_i = (\pi_1, \pi_2, \ldots, \pi_n) \), where \( \pi_1 = (t_1 - t_0) \mod z \).

So, the pattern represents the schedule of an interval of length \( z \) that is executed repeatedly in the steady state.

**Definition 4** The pattern \( \pi = (\pi, z, f) \) of \( \sigma \) is a feasible pattern if there exists a schedule \( \sigma = (t, z, f) \).

PSIP is a cyclic version of the classical \( m \)-processor makespan minimization problem. PSIP is NP-hard in the strong sense [8]. The special case of PSIP in which there are only two available processors is NP-hard. A polynomial special case the uniform circuit [8]. We are interested in a polynomial special cases of Periodic Scheduling on Identical Processors problem, so we study Periodic Scheduling on Identical Processors with unit processing time.

### 3. Periodic scheduling problem with unit processing time

Let \( V = \{v_1, v_2, \ldots, v_n\} \) be a set of generic operations.

Precedence relations are defined by acyclic graph \( G = (V, E) \) with vertex set \( V \) and arc set \( E \). Each arc \((v_i,v_j) \in E\) is supplied by two values \( L_{ij} = 1 \) and \( H_{ij} = 0 \). We consider the problems, where operation \( v_i \) has a unit processing time. We also postulate that the \( k + 1 \)-th occurrence of operation \( v_i \) can only start if the \( k \)-th occurrence is finished. We assume that the constraints are included in graph \( G \) by adding loops \((v_i,v_i)\) with \( L_{ii} = 1 \) and \( H_{ii} = 1 \). to \( E \). The PSIP with unit processing can be written as:

\[
\min z
\]

\[
t(v_i; k) = t(v_i; 0) + kz, \ \forall \ v_i \in V, \forall k \in Z
\]

\[
t(v_i; k) + 1 \leq t(v_i; k + 1), \ \forall \ v_i \in V, \forall k \in Z
\]

\[
t(v_i; k) + 1 \leq t(v_j; k), \ \forall (v_i, v_j) \in E, \forall k \in Z
\]

The circuits with the maximum value (critical circuits) are loops \((v_i, v_i)\) in \( G \). Then the value of a critical circuit equal 1 and \( LB = \max \{ \sum_{v_i \in V} p(v_i) / m, 1 \} \) is a lower bound for the optimal cycle time.

Although the problem is NP-hard, we show that the special cases where \( G \) is a tree or \( m = 2 \) can be solved in polynomial time.

#### 3.1 Multiprocessor scheduling problem with unit processing time
We now consider the non-cyclic problem in which unit time jobs with precedence constraints are to be scheduled on identical parallel machines. The precedence constraints are either in-trees or out-trees. The list algorithm proposed by Hu [9] generates an optimal schedule for this problem.

List algorithms, which select the ready tasks to be assigned processors from a static priority list, are certainly the most studied algorithms in this field. As a simplest combinatorial algorithm, list scheduling (LS) has been well known for almost half a century. The algorithm of Hu is a list scheduling algorithm and has two steps. In the first step, tasks are ordered in a priority list. In the second step, when a processor is free, algorithm selects the first available task in the priority list and maps on the processor. The job is available if all its predecessors have already been processed.

In next section we propose the algorithm, which generates an optimal cyclic schedule if graph G is a tree.

4. Algorithm for Periodic scheduling problem with unit processing time and the precedence constraints are in-tree

We use Hu algorithm to build a non-cyclic schedule as an auxiliary procedure. We denote this procedure by ListHu \((T; S, C_{\text{max}})\) where graph \(T\) is a tree or forest, \(S\) is the schedule constructed by Hu algorithm, \(C_{\text{max}}\) is the value of the objective function.

We build a pattern reaching the lower bound on the cycle time, and then prove that this pattern defines an optimal cyclic schedule. As a result of the algorithm, a feasible pattern \(\pi = (\pi, z, f)\) and the occurrence vector \(\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)\) are built. Based on these data, an optimal schedule is created.

The algorithm generates a pattern in one iteration, if \(z \geq C_{\text{max}} / 2\), and in \(p\) iterations if \(z < C_{\text{max}} / 2\), where \(p = \lceil \frac{C_{\text{opt}}}{z} \rceil\). The algorithm captures the value of the period \(z\) and checks whether a schedule of a given length exists. If it is built, then this is the optimal schedule; otherwise, increase \(z\) by 1 and continue.

Algorithm LC.
Consider graph \(T = (V, E)\) and \(m\) processors. \(T\) is a tree and \(V = \{v_1, v_2, \ldots, v_n\}\).

Step 1. Define lower bound \(LB = \lceil n/m \rceil\) for the optimal cycle time \(z_{\text{opt}}\).
Step 2. Define the occurrence vector \(\alpha (v_i) = 0\);
Step 3. Find schedule \(S\) and makespan \(C_{\text{max}}\), use procedure ListHu \((T; S, C_{\text{max}})\).
Step 4. If \(C_{\text{max}} = LB\) then \(z_{\text{opt}} = C_{\text{max}}\) and optimal cyclic schedule is \(\sigma = (t, z_{\text{opt}})\), go to 18.

For a given number of processors \(m\), it is not possible to overlap the execution of individual projects.

Step 5. \(z = LB\)
Step 6. If \(z < C_{\text{max}} / 2\), go to Step 12.
Step 7. Define two sets of jobs \(D(z) = \{v_i \mid t(v_i) < z\}\) and \(F(z) = \{v_i \mid t(v_i) \geq z\}\).
Step 8. Set \(\alpha (v_i) = \alpha (v_i) + 1; \) for \(v_i \in F(z)\).

Step 9. Create two new graphs: \(T_1 = D(z), E_1)\) and \(T_2 = (F(z), E_2)\), where \(T_1\) and \(T_2\) are subgraphs of \(T = (V, E)\). Construct the new tree \(T_{\text{new}}(V_{\text{new}}, E_{\text{new}})\), where \(V_{\text{new}} = D(z) \cup F(z) \cup \{w\}\). Add vertex \(w\), as a root to the new tree. \(E_{\text{new}} = E_1 \cup E_2 \cup E_3\), where arcs \(E_3 = \{e = (v_i, w)\}\), where \(v_i\) has no successors in \(T_1\) or \(T_2\).

Step 10. Construct the schedule \(S_z\) using the procedure ListHu \((T_{\text{new}}; S_z, C_{\text{max}}(z))\).
Delete vertex \(w\) from \(S_z\).
Step 11. if \(C_{\text{max}}(z) - 1 = z\), then \(S_{\text{opt}} = S_z\) and \(z_{\text{opt}} = z\) else
if \(C_{\text{max}}(z) - 1 = z + 1\), then \(S_{\text{opt}} = S_z\) and \(z_{\text{opt}} = z + 1\) go to Step 18
else \(z = z + 1\) and go to Step 6.
Step 12. \(p = \lceil C_{\text{max}} / z \rceil\).
Step 13. \(k = p\).
Step 14. \(D(zk) = \{v_i \mid t(v_i) < zk\}\) and \(F(zk) = \{v_i \mid t(v_i) \geq zk\}\). Create two new trees \(T_1 = (D(zk), E_1)\) and \(T_2 = (F(zk), E_2)\). Construct the new tree \(T_{\text{new}}(V_{\text{new}}, E_{\text{new}})\).
Step 15. Set $\alpha (v_i) = \alpha (v_i) + 1$; for $v_i \in F(zk)$.
Step 16. Construct the schedule $S_z$ using the procedure ListHu ($T_{\text{new}}; S_z, C_{\text{max}}(z)$).
Step 17. if $C_{\text{max}}(z) > zk + 1$ then $z := z + 1$ go to Step 6. If $k = 1$ then $S_{\text{opt}} = S_z$ and $z_{\text{opt}} = z$
else $k := k - 1$ go to Step 14.
Step 18. Algorithm generates the optimal schedule $S_{\text{opt}}$: $t(v_i) := t(v_i) + z_{\text{opt}} \alpha (v_i)$.

4.1. Example 1
Consider example: there are the graph $G$ (Fig.1), $n = 12$; $m = 3$.
Step 1. Define the occurrence vector $\alpha (v_i) = 0$;
Define lower bound. $LB = \lceil n/m \rceil = 4$.
Step 2. Generate schedule $S_L$ use algorithm Hu by ListHu ($G; S_L, C_{\text{max}}$).

![Figure 1. Graph G.](image1)

![Figure 2. Optimal schedule $S_L$.](image2)

We can see the tree $G$ and priority label on figure 1 and optimal schedule $S_L$ (figure 2), $C_{\text{max}} = 6$.
Step 2. $z = 4 > C_{\text{max}}/2$.
Step 6. Define $F(z) = \{v_i \mid t(v_i) \geq z\} = \{l, q\}$, $D(z) = V \backslash \{l, q\}$. Delete arcs: $(k, q)$ and $(p, q)$.
Step 7. Create three new graphs $T_1 = (D(z), E_1)$ and $T_2 = (F(z), E_2)$ and $T_3 = ((l, q), (l, q))$ Graph $T_1$ is a forest, it consists of three trees. We construct new tree by adding a vertex $w$, as a root to the new tree and four new arcs: $(k, w), (h, w), (q, w), (p, w)$. We have the new tree $T_{\text{new}}$ (figure 3).

![Figure 3. Graph $T_{\text{new}}$.](image3)

![Figure 4. Optimal schedule $S_L$.](image4)
Step 11. Generate schedule $S_L$ use algorithm Hu by ListHu $(T_{new}; S_L, C_{max})$ (Fig.4.). Delete root w, then $C_{max} = 4 = LB$. This is the schedule pattern for optimal cyclic schedule $S_{opt}$, the optimal period $z_{opt} = 4$. (figure 5.)

Figure 5. Optimal cyclic schedule the optimal period $z_{opt} = 4$.

4.2. Example 2 ($m = 3$ and graph $G$)
Step 1. Generate schedule $S_L$ use algorithm Hu by Lists $(G; S_L, C_{max})$ We can see the tree $G$ on Fig.6. and optimal schedule $S_L$ (figure 7), $C_{max} = 5$. LB=2.

Figure 6. Graph $G$. Figure 7. Optimal schedule $S_L$.

Step 3. $z < C_{max}/2$, then algorithm generates an optimal schedule by two iterations.
Step 12. $p_1 = \lceil C_{max}/z \rceil$. We must consider $k = 2$, 1, algorithm takes 2 iteration.
Step 14-17. $k = 2$. Delete jobs such that $t(v_i) \geq kp$. It is the job $f$ and construct graph $G_1$ (figure 9) adding the root vertex w. Then we generate schedule $S_1$ (figure 10). $C_{max} = 4$.

Figure 9. Graph $G_1$. Figure 10. Optimal schedule $S_L$.

Step 18. $k := 1$. Go to Step 6.
Step 6. Delete jobs such that $t(v) \geq 2$ from graph $G_1$ and create graph $G_2$ (figure 11).

Step 3. $z < C_{max}/2$, then algorithm generates an optimal schedule by two iterations.
Step 16. Construct optimal schedule (figure 12).
Step 17. $C_{\text{max}} = 2$, this is the optimal cyclic schedule $z_{\text{opt}} = 2$ (figure 13).

Theorem 1 Algorithm LC generate an optimal cyclic schedule if graph $G$ is a tree.

5. Cyclic Scheduling problem for two processors system

Now we consider a graph $G$ and two parallel identical processors.

This non-cyclic problem can be solved in $O(n^2)$ time by Coffman-Graham algorithm [10]. First, the jobs are labelled in priority list and at each step the available job with the highest ranking on a priority list is assigned to the free processor. We use this algorithm by procedure ListCG ($G; S_L; C_{\text{max}}$).

Lower bound for cycle time equal $LB = \frac{n}{2}$, and algorithm generates schedule by single iteration.

Algorithm LC2
Step 1. Define lower bound $LB$ for the optimal cycle time $z_{\text{opt}}$. $LB = \frac{n}{2}$.
Step 2. Find schedule $S_L$ and find start times $t(v_i)$ and makespan $C_{\text{max}}$, use ListCG ($G; S_L; C_{\text{max}}$).
Step 3. If $C_{\text{max}} = LB$ then schedule $S_L$ is optimal cyclic schedule and cycle time is equal $z_{\text{opt}} = C_{\text{max}}$, go to Step 10.
Step 4. $z := LB$.
Step 5. Define two sets of jobs $D(z) = \{ v_i \mid t(v_i) < z \}$ and $F(z) = \{ v_i \mid t(v_i) \geq z \}$.
Step 6. Create two new graphs: $G_1 = (D(z), E_1)$ and $G_2 = (F(z), E_1)$, where $G_1$ and $G_2$ are subgraphs of $G = (V,E)$.
Step 7. Construct the new graph $G_{\text{new}} (V_{\text{new}}, E_{\text{new}})$, where $V_{\text{new}} = D(z) \cup F(z)$, $E_{\text{new}} = E_1 \cup E_2$.
Step 8. Construct the schedule $S_z$ using the procedure ListCG ($G_{\text{new}}; S_z; C_{\text{max}}(z)$).
Step 9. if $C_{\text{max}}(S_z) = z$, then $S_{\text{opt}} = S_z$ and $z_{\text{opt}} = z$ else $z := z + 1$ and go to Step 5.
Step 10. Algorithm generates the optimal schedule $S_{\text{opt}}$. 

Figure 11. Graph $G_2$.

Figure 12. Optimal schedule $S_L$.

Figure 13. Cyclic optimal schedule for $G$ and $z_{\text{opt}} = 2$. 

| f | b | w |
|---|---|---|
| c | d |
| a | e |

| f | b | f | b | f | b | f | b | f | b |
|---|---|---|---|---|---|---|---|---|---|
| c | d | c | d | c | d | c | d |
| a | e | a | e | a | e | a | e | a | e |
**Theorem 3** Algorithm LC2 generate an optimal cyclic schedule if \( m = 2 \).

**6. Conclusion**

We consider the multiprocessor cyclic scheduling problem with unit processing time, where the goal is to find a periodic schedule that minimizes the cycle time under precedence constraints. Although the problem is NP-hard, we show that the two special cases, where graph \( G \) is a tree or \( m = 2 \), can be solved in polynomial time.

**References**

[1] Levner E, Kats V and Levit V E 1997 An improved algorithm for a cyclic robotic scheduling problem *Eur. J. of Oper. Res.* 97 500–8

[2] Kats V and Levner E 2002 Cyclic scheduling on a robotic production line *J. of Scheduling* 5 23–41

[3] Kubiak W 2005 Solution of the Liu-Layland problem via bottleneck just-in-time sequencing *J. of Scheduling* 8 (4) 295–302

[4] Sucha P and Hanzlek Z 2011 Deadline constrained cyclic scheduling on pipelined dedicated processors considering multiprocessor tasks and changeover times *Math. Comput. Model.* 47 (910) 925–42

[5] Dupont B and Munier A 2014 Converging to periodic schedules for cyclic scheduling problems with resources and deadlines *Comput. Oper. Res.* 51 227–36

[6] Hanen C and Munier A 1995 A study of the cyclic scheduling problem on parallel processors *Discrete Appl. Math.* 57 1167–92

[7] Brucker P and Kampmeyer T 2008 A general model for cyclic machine scheduling problems. *Discret. Appl. Math.* 156 (13) 2561–72

[8] Munier A 1996 The complexity of a cyclic scheduling problem with identical machines and precedence constraints. *Eur. J. Oper. Res.* 91 (3) 471–80

[9] Hu T S 1961 Parallel sequencing, and assembly line problems *Oper. Res.* 9 841–48

[10] Coffman E G and Graham R L 1972 Optimal schedule for two-processor systems *Acta Informat.* 1 200–13