Optimal Cyclic Scheduling on Parallel Processors with Special Precedence Constrains

Natalia Grigoreva[0000−0002−6621−0911]

St. Petersburg State University,
Universitetskaja nab. 7/9, 199034 St. Petersburg, Russia,
n.s.grig@gmail.com

Abstract. We consider the multiprocessors scheduling problem when a set of jobs V is done on m identical parallel processors and set of jobs V is to be repeated an infinitely number of times. Precedence constraints between jobs are represented by an uniform graph G. The goal is to generate 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. We consider four special Periodic Scheduling on Identical Processors problems. We propose algorithms for four problems: the problem with unit processing times and three problem with arbitrary processing times.
The problem with unit processing times and the problem with preemptions can be solved in polynomial time. Algorithms for two problems with arbitrary processing times (with precedence constrains and without its) generates heuristic schedules.

Keywords: Cyclic scheduling problem · Precedence constraints · Parallel processors

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, 4], communications and transport or multiprocessor computing[6].

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. One of the first papers which investigated parallel identical processors problems is the paper [5]. The problem with some additional hypothesis on the structure of uniform constraints is considered in [11].

In [15] 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 [14] authors investigates periodic schedules for cyclic scheduling problems with resources 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, and additional restrictions on the structure of uniform constraints.

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 a cyclic version Periodic Scheduling on Identical Processors problem with loops.

We propose the algorithm for cyclic version the problem $P|prec, p_j = 1|C_{\max}$ in section 4. The algorithm for cyclic version the problem with preemptions $P|pmtp, prec|C_{\max}$ are provided in section 5. The algorithm for cyclic version the problem with independent jobs $P||C_{\max}$ are provided in section 6. The cyclic version of problem $P|prec|C_{\max}$ we consider in section 7. Section 8 contains a summary of this paper.

2 Multiprocessor cyclic scheduling problem

First we consider a non-cyclic multiprocessor system of tasks $V = \{v_1, v_2, ..., v_n\}$. 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 $v_i \prec v_j$. The expression $v_i \prec v_j$ means that the task v_j may be initiated only after completion of the task v_i.

The execution time of each job $p(v_i)$ is known. Set of tasks is performed on m 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 $C_{\max}(S) = \max\{t(v_i) + p(v_i)|v_i \in V\}$.

We assume that the set of tasks $V = \{v_1, ..., v_n\}$ is a template which we wish to repeat infinitely, 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 w time units (the cycle time) and the same schedule of tasks is completed for each project. Assuming the time to complete each individual project $C_{\max}(S)$ is greater than w.

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

Let $V = \{v_1, ..., v_n\}$ 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 $< v_j, k+H_{ij} >$ 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 (i, i) with $Lii = p(v_i)$ and $H_{ii} = 1$ to E.

Definition 1. *A schedule is called periodic with cycle time w, if $t(v_i; k) = t(v_i; 1) + (k - 1)w$ for all $v_i \in V$, $k \geq 1 \in N$.*

The goal is to minimize a cycle time w (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 makespan. The goal is to define the optimal cycle time w_{opt}, the starting time of each occurrence of operation v_i for all $v_i \in V$ and processor $f(v_i)$. 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 $t_i = t(v_i; 1)$.

A periodic schedule $\sigma = (t, w)$ is defined by the vector $t(v_i), v_i \in V$ and the cycle time $w \geq 0$.

Definition 2. *A schedule $\sigma = (t, w, f)$ is called resource-periodic with period K if $f(v_i, k + K) = f(v_i, k).k \geq 1 \in N$.*

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.

Consider a circuit μ in graph G. Let $L(\mu) = \sum_{(i,j)\in\mu} p(v_i)$ and $H(\mu) = \sum_{(i,j)\in\mu} Hij$. Then $z(\mu) := \frac{L(\mu)}{H(\mu)}$ is called the value of μ. 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 $LB = \max\{\sum p(v_i)/m, z(G)\}$ [6] is a lower bound on the cycle time.

PSIP is a cyclic version of the classical m-processor makespan minimization problem. PSIP is NP-hard in the strong sense [11]. The special case of PSIP with unit processing time in which there are only two available processors is polynomial [11]. We are interesting in a polynomial special cases of Periodic Scheduling on Identical Processors problem, so we study Periodic Scheduling on Identical Processors with loops.

3 Periodic scheduling problem with loops

We consider the cyclic case of the problem of minimizing the makespan while scheduling jobs to parallel identical processors. This is a classical combinatorial optimization problem. Following the 3-field classification scheme proposed by Graham *et al.* [8], this problem is denoted by $P|prec|C_{max}$. This problem is NP-hard [7].

Consider the special case of Periodic scheduling problem. Let $V = \{v_1, ..., v_n\}$ be a set of generic operations.

The execution time of each task positive integer$p(v_i)$ is known. Precedence constructions between jobs are represented by a directed acyclic task graph $G = \langle V, E\rangle$. E is a set of directed arcs, an arc $e = (v_i, v_j) \in E$ if and only if $v_i \prec v_j$. The expression $v_i \prec v_j$ means that the job v_j may be initiated only after completion of job v_i.

Set of tasks is performed on m parallel identical processors, any job can run on any processor and each processor can perform no more than one job at a time. Job preemption is not allowed.

Each arc $(v_i, v_j) \in E$ is supplied by two values $L_{ij} = p(v_i)$ and $H_{ij} = 0$. Now we have defined non-cyclic multiprocessor scheduling problem.

We also prepare 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 E_1 (i, i) with $L_{ii} = p(v_i)$ and $H_{ii} = 1$ to E and we have uniform graph $G^* = (V, E \cup E_1)$.

We limit our study to a subclass of uniform graphs: graph G is acyclic graph with loops E_1.

This special LPSIP with can be written as:

$$\min \alpha$$

$$t(v_i; k) = t(v_i; 1) + (k-1)\alpha, \forall v_i \in V,$$

$$t(v_i; k) + p(v_i) \le t(v_i; k+1), \forall v_i \in V, \forall k \ge 1 \in N.$$

$$t(v_i;k) + p(v_i) \leq t(v_j;k), \forall (v_i, v_j) \in E.$$

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 $LB = \max\{\lceil(\sum p(v_i)/m)\rceil, 1\}$ is a lower bound for the optimal cycle time.

We consider four special LPSIP problems: problem with unit processing times, problem with preemptions, problem with independent jobs and problem with arbitrary processing times without preemptions. Although the problem is NP-hard, we show that the problem with unit processing times and problem with preemptions can be solved in polynomial time.

4 Periodic scheduling problem with unit processing times

We consider the problems, where operation v_i has a unit processing time.

This LPSIP with unit processing can be written as:

$$\min \alpha$$

$$t(v_i;k) = t(v_i;0) + k\alpha, \forall v_i \in V,$$

$$t(v_i;k) + 1 \leq t(v_i;k+1), \forall v_i \in V, \forall k \geq 1 \in Z.$$

$$t(v_i;k) + 1 \leq t(v_j;k), \forall (v_i, v_j) \in E.$$

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 $LB = \max\{\lceil(n/m, 1\rceil\}$ is a lower bound for the optimal cycle time.

4.1 Algorithm for Periodic scheduling problem with unit processing time

We now consider the non-cyclic problem $P|p_j = 1, prec|C_{\max}$ in which unit time jobs with precedence constraints are to be scheduled on identical parallel machines. This non-cyclic problem can be solved in $O(n^2)$ time by Coffman-Graham algorithm [8]. The algorithm is a list scheduling algorithm and has two steps.

First, the jobs are labeled in priority list and at each step the available job with the highest ranking on a priority list is assigned to the free processor.The job is available if all its predecessors have already been processed.

We use this algorithm by procedure $ListCG(G^*, S_L, C_{\max})$, where graph G^* is uniform graph G without loops, S_L is the schedule constructed by Coffman-Graham algorithm, $C_{\max}$ is the value of the objective function. Lower bound for cycle time equal $LB = \lceil n/m \rceil$.

We build a feasible cyclic schedule reaching the lower bound on the cycle time.

If the optimal cyclic schedule has a period $C_{\max}$ and for a given number of processors m, it is not possible to overlap the execution of individual projects.

4.2 Algorithm A_1

Consider graph $G^* = (V, E)$ and m processors. $V = \{v_1, ..., v_n\}$.

Step 1. Define lower bound LB for the optimal cycle time z_{opt}.
$LB = \lceil n/m \rceil$.

Step 2. Define the occurrence vector $\alpha(v_i) := 0$;

Step 3. Find schedule S_L, find start times $t(v_i)$ and makespan $C_{\max}$,
use procedure ListCG $(G; S_L, C_{\max})$.

Step 4. If $C_{\max} = LB$ then $z_{opt} = C_{\max}$ and optimal cyclic schedule is $\sigma = (t, z_{opt})$, goto step 19 else set $z = LB, k := 0$

Step 5. $k := k + 1$. Define two sets of jobs $D_k(z) = \{v_i \mid t(v_i) < z\}$ and $F_k(z) = \{v_i \mid t(v_i) \geq z\}$.

Step 6. Set $\alpha(v_i) := \alpha(v_i) + 1$ and set $t(v_i) = 0$ for $v_i \in F_k(z)$.

Step 7. Create a new graph: $G_k = (F_k(z), E_k)$,where G_k is a subgraph of $G^* = (V, E)$. $(v_i, v_j) \in E_k$ if and only if $((v_i, v_j) \in E) \& (v_i, v_j \in F_k(z))$.

Step 8. Jobs $F_(z)$ are ordered in a priority list, by procedure ListCG $(G_k; S_L, C_{\max})$.

Step 9. At each step the available job with the highest ranking on a priority list is assigned to the free interval in schedule S_L.

Step 10. Define new S_L and $C_{\max}$. If $C_{\max} = LB$ then $z_{opt} = C_{\max}$ and optimal cyclic schedule is $\sigma = (t, z_{opt}, \alpha(v_i))$, goto step 11 else goto step 5.

Step 11. Algorithm generates the optimal schedule $t(v_i) := t(v_i) + z_{opt}\alpha(v_i)$.

Example 1. Consider example: there are the graph $G = (V, E)$, $n = 9; m = 3$ (Fig.1).

Step 1. Generate schedule S_L use algorithm $ListCG(G, S_L, C_{\max})$(Fig.1)

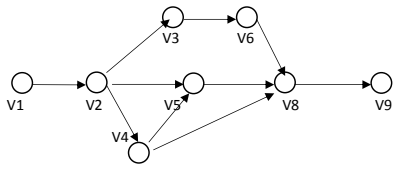

Fig. 1. Task graph $G = (V, E)$, $t(V_i) = 1$;

We can see schedule S_L (Fig.2), $C_{\max} = 7$.

Step 2. Define lower bound $LB = n/m = 3$. Define $\alpha(V) = (0, 0, 0, 0, 0, 0, 0, 0, 0)$.

Step 3. Define $F_1(z) = \{v_i \in V | t_i \geq z\} = \{v5, v6, v7, v8, v9\}$ and $D_1(z) = \{v1, v2, v3, v4\}$.
Define $\alpha(V) = (0, 0, 0, 0, 1, 1, 1, 1, 1)$.

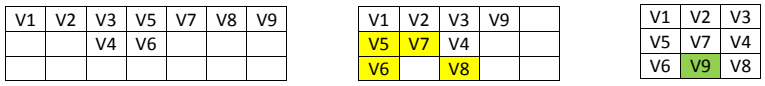

Fig. 2. The schedule S_L, $C_{\max} = 7$; the schedule S_2, $C_{\max} = 4$; the cyclic schedule .

Step 4. Create the new graphs $G_1 = (F_1(z), E_1)$

Step 5. Generate schedule S_2 use algorithm $ListCG(G_1, S_2, C_{\max})$ and put jobs $F_1(z)$ in free slots of S_L. (Fig.2.).

$t(v5) = t(v6) = 0; t(v7) = 1; t(v8) = 1; t(v9) = 3$.
$C_{\max} = 4$. and $C_{\max} > z$.

Step 6. Define $F_2(z) = \{v_i \in V | t_i \geq z\} = \{v9\}$ and $D_2(z) = V \setminus v9$.
Define $\alpha(V) = (0, 0, 0, 0, 1, 1, 1, 1, 2)$

Step 7. Put job $v9$ in free slot $t(v9) := 2$.

Step 8. This is the schedule pattern for optimal cyclic schedule (Fig.2.) $t(V) = (0, 1, 2, 2, 4, 4, 5, 6, 8)$. $z_{opt} = 3$.

Theorem 1. *Algorithm A_1 generates an optimal cyclic schedule.*

Proof. Let J be the number of idle periods of processors in schedule S_L in interval $[0, z)$, then $|J| \geq |F_1(z)|$.

The k-th execution of jobs from $D_k(z)$ are execution before the k-th execution of jobs from $F_k(z)$ on each k-th iteration of the algorithm. The precedence constraints induced by subgraphs G and G_k are met. The algorithm generates a feasible schedule S_z with minimum cycle time $z_{opt} = LB$. The algorithm installs to the processor at least one job from set $F_k(z)$ on each k-th iteration, then the number of iterations does not exceed n. This cyclic problem can be solved in $O(n^3)$ time.

5 Optimal cyclic scheduling with preemptions

We consider a non-cyclic multiprocessor system of jobs $V = \{v_1, v_2, ..., v_n\}$. The execution time of each job $p(v_i)$ is known . Precedence constraints between jobs are represented by a directed acyclic task graph $G = (V, E)$.

Set of jobs is performed on parallel identical processors, jobs preemptions are allowed.

$$\min \alpha$$

$$t(v_i;k) = t(v_i;1) + (k-1)\alpha, \forall v_i \in V,$$

$$t(v_i;k) + p(v_i) \leq t(v_i;k+1), \forall v_i \in V, \forall k \geq 1 \in N.$$

$$t(v_i;k) + p(v_i) \leq t(v_j;k), \forall (v_i, v_j) \in E.$$

Preemptions can be useful because they can shorten the schedule.

We consider a non-cyclic multiprocessor system of tasks $V = \{v_1, v_2, ..., v_n\}$. The execution time of each task $p(v_i)$ is known. Precedence constraints between tasks are represented by a directed acyclic task graph $G = (V, E)$.

Algorithm by Muntz and Coffman [16] uses the level of job v_i, which is the longest path between v_i and a terminal job. The algorithm uses a notion of a processor shared schedule, in which a job receives some fraction β of the processing capacity of processor.

Let algorithm by Muntz and Coffman constructs a schedule S_p. Each job v_i can execute in k_i intervals. For each job v_i the algorithm defines the start time $t_j(v_i)$ in j-th bloc of execution job v_i, where $j \in 1 : k_i$. Job v_i is processing $p_j(v_i)$ moments of time in interval $[t_j(v_i), t_j(v_i) + p_j(v_i)]$.

$\sum_{j=1}^{k_i} p_j(v_i) = p(v_i)$. If some job v_i is not interrupted in a schedule S_p then $k_i = 1$ and $p_1(v_i) = p(v_i)$. We use algorithm Muntz and Coffman by procedure ListMC $(G; S_p, C_{\max})$.

5.1 Algorithm A_2

Step1. Define lower bound LB for the optimal cycle time z_{opt}.
$LB = \sum_{i=1}^{n} p(v_i))/m$.

Step 2. Find schedule S_p and find start times $t_j(v_i)$
and makespan $C_{\max}$, use ListMC $(G; S_p, C_{\max})$.

Step 3. If $C_{\max} = LB$ then schedule S_p is optimal cyclic schedule and cycle time is equal $C_{\max}$, then goto Step 10 else $z := LB, k := 0$.

Step 4. For each job v_i the algorithm defines the start time $t_j(v_i)$ in j-th bloc of execution job v_i, where $j \in 1 : k_i$.

Step 5. $k := k + 1$. Define three sets of jobs:
$D_k(z) = \{v_i \mid (t_{k_i}(v_i) + p_{k_i}(v_i) \leq zk)\}$,

$B_k(z) = \{v_i \mid (t_j(v_i) < zk) \& (t_j(v_i) + p_j(v_i) > zk)\}$,
$F_k(z) = \{v_i \mid (t_j(v_i) \geq zk) \& (v_i \notin B_k(z))\}$.

Step 6. The set $D_k(z)$ consist of jobs, which are ended before zk. We interrupt all jobs from $B_k(z)$, which are processing at moment of time zk.

Step 7. For each job v_i from $B_k(z)$ find j_0, such that
$(t_{j_0}(v_i) < zk) \& (t_{j_0}(v_i) + p_{j_0}(v_i) > zk)$.

Find new processing times for jobs $B_k(z)$.
$p(v_i) = \sum_{j_0}^{k_i} p_j(v_i) + t_{j_0}(v_i) - zk$. $p_{j_0}(v_i) = zk - t_{j_0}(v_i)$.

Step 8. Find new processing times for jobs $F_(z)$.
Let $I(v_i) = \{j \in 1 : k_i \mid t_j(v_i) \geq z\}$ then $p(v_i) = \sum_{j \in I(v_i)} p_j(v_i)$

Step 9. Create the new graph: $G_k(V_k, E_k)$
where G_k is subgraph of $G = (V, E)$. $V_k = F_k(z) \cup B_k(z)$.

Step 10. Set $t(v_i) = 0$, for $v_i \in V_k$.

Step 11. Create a copy of schedule S_L from interval $[z(k-1), zk)$ to interval $[zk, z(k+1))$.

Step 12. Find idle periods of processors in copy S_L in interval $[zk, z(k+1))$.

Step 13. Jobs $V_k(z)$ are ordered in a priority list, by procedure ListMC $(G_k; S_p, C_{\max})$.

Step 14. At each step the available job from $V_k(z)$ with the highest ranking on a priority list is assigned to the free interval in $[zk, z(k+1))$ by procedure ListMC $(G_k; S_p, C_{\max})$.

Step 15. Repeat this procedure for all free intervals in $[zk, z(k+1))$.

Step 16. Define new S_p and $C_{\max}$. If $C_{\max} \leq z(k+1)$ then the optimal cyclic schedule is $\sigma = (t, z)$ else goto step 4.

Theorem 2. *Algorithm A_2 generate optimal cyclic schedule.*

Proof. Denote $V_k = F_k(z) \cup B_k(z)$, $P(V_k) = \sum_{v_i \in V_k} p(v_i)$ Let J be the number of idle periods of processors in schedule S_p in interval $[0, z)$,and $L(j)$ is the length of idle interval j, then $P(J) = \sum_{j \in J} L(j)$ and $P(V_k) \leq P(J)$. Procedure ListMC $(G; S_p, C_{\max})$ generates a feasible schedule S_p, we save this schedule in interval $[0, z)$.

The k-th execution of jobs from $D_k(z)$ are execution before the k-th execution of jobs from $V_k(z)$ on each k-th iteration of the algorithm. The precedence constraints induced by subgraphs G and G_k are met. The algorithm generates a feasible schedule S_z with minimum cycle time $z_{opt} = LB$.

Procedure ListMC $(G; S_p, C_{\max})$ can be implemented to run in $O(n^2)$ time [16]. The number of idle periods of processors J in schedule S_p does not exceed n. The algorithm fills at least one idle interval on each k-th iteration then the number of iterations does not exceed n. This cyclic problem can be solved in $O(n^3)$ time.

6 Cyclic scheduling on parallel processors

Set of tasks is performed on parallel identical processors. Task preemptions are not allowed.

$$\min \alpha$$

$$t(v_i; k) = t(v_i; 0) + k\alpha, \forall v_i \in V,$$

$$t(v_i; k) + 1 \leq t(v_i; k+1), \forall v_i \in V, \forall k \geq 1 \in Z.$$

$$t(v_i; k) + p(v_i) \leq t(v_j; k), \forall (v_i, v_j) \in E.$$

The usual objective function is completion time of the scheduled task graph also referred to as makespan or schedule length.

We propose a heuristic algorithm to generate cyclic schedule. We use CP(critical path) algorithm for constructing schedule for the problem. CP algorithm uses the level of jobs v_i in graph G, which is the sum of processing times (including $p(v_i)$) of the longest path between v_i and a terminal job.

6.1 Algorithm A3

.

Step1. Define lower bound LB for the optimal cycle time z_{opt}.
$LB = \lceil (\sum_{i=1}^{n} p(v_i))/m \rceil$.

Step 2. Define the occurrence vector $\alpha(v_i) := 0$;

Step 3. Find schedule S and find start times $t(v_i)$
and makespan $C_{\max}$, use CP $(G; S_L, C_{\max})$.

Step 4. If $C_{\max} = LB$ then schedule S_L is optimal cyclic schedule and cycle time is equal $C_{\max}$, goto Step 10 else $z := LB$.

Step 5. Define two sets of jobs $D(z) = \{v_i \ | t(v_i) + p(v_i) \leq z\}$ and
$F(z) = \{v_i \mid t(v_i) + p(v_i) > z\}$.

Step 6. Set $\alpha(v_i) := \alpha(v_i) + 1$ for $v_i \in F(z)$.

Step 7. Create two new graphs: $G_1 = (D(z), E_1)$ and $G_2 = (F(z), E_2)$
where G_1 and G_2 are subgraphs of $G = (V, E)$.

Arc $(v_i, v_j) \in E_1$ if and only if $v_i, v_j \in D(z)$ and arc $(v_i, v_j) \in E_2$ if and only if $v_i, v_j \in F(z)$.

Step 8. Construct the new graph $G_{new}(V_{new}, E_{new})$,
where $V_{new} = D(z) \cup F(z)$. $E_{new} = E_1 \cup E_2$.

Step 9. We construct the schedule S_z using algorithm CP [**?**] by the procedure $CP(G_{new}, S_z, C_{\max})$.

Step 10. $z = C_{\max}$ and the cyclic schedule is $\sigma = (t, z, \alpha(v_i))$.
$t(v_i) := t(v_i) + z\alpha(v_i)$.

Example 2. Consider the example at Fig.3, there are task a,$V_1, V_2, ..., V_m$, W and $m-1$ independent tasks U_i.

This tasks are executed on m processors.

Any list scheduling algorithm build the schedule 1 of Fig.4 with makespan $C_{\max} = 2m - 1$ [9]. The optimal schedule has makespan $C_{\max} = m + \varepsilon$.

Step1. Define lower bound LB for the optimal cycle time z_{opt}.
$LB = \lceil (\sum_{i=1}^{n} p(v_i))/m \rceil. = m + \epsilon/m$.

Step 2. Define the occurrence vector $\alpha(v_i) := 0$;

Step 3. Generate schedule S_L use algorithm $CP(G, S_L, C_{\max})$(Fig.5).
$C_{\max} = 2m - 1$.
$C_{\max} > LB$ then $z := m + \epsilon/m$.

Step 4. Define two sets of jobs:
$D(z) = \{v_i \ | t(v_i) + p(v_i) \leq z\}$ and $F(z) = \{v_i \mid t(v_i) + p(v_i) > z\}$.
$D(z) = \{a, v_1, \ldots, v_m, u_1, \ldots, u_{m-1}\}, F(z) = w$.

Step 5. Set $\alpha(v_i) := \alpha(v_i) + 1$ for $v_i \in F(z)$. $\alpha(w) := 1$.

Step 6. We construct the schedule S_z using algorithm CP by the procedure $CP(G_{new}, S_z, C_{\max})$.

We can see the graph G_{new} on Fig.5 and schedule S_z (Fig.6), $C_{\max} = m + \epsilon$.
Algorithm A_3 generates the optimal schedule S_z $z := m + \epsilon$.(Fig.6)
The algorithm can be implemented to run in $O(n^2)$ time.

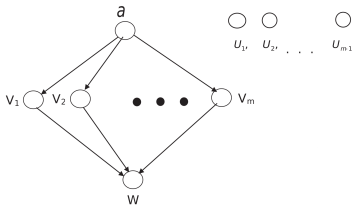

Fig. 3. Task graph $t(a) = \varepsilon$; $t(V_i) = 1$; $t(U_i) = m - 1$; $t(W) = m - 1$

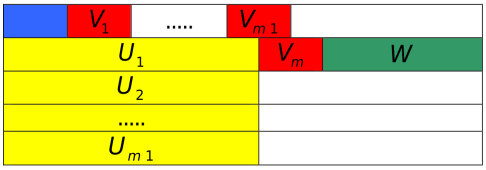

Fig. 4. The schedule 1, makespan $C_{\max} = 2m - 1$

7 Cyclic scheduling independent jobs on parallel processors

We consider problem $P||C_{max}$, where a set of independent jobs is to be scheduled on identical processors in order to minimize schedule length. The problem is NP-hard. One of the simplest algorithms is the LPT algorithm in which the tasks are arranged in order of non-increasing $p(v_i)$. Then whenever a processor is free the longest job which not yet processed is assigns to the processor. The LPT algorithm is a $4/3 - 1/3m$ approximation algorithm [10]. The worst case example of LPT algorithm has m processors and $2m + 1$ jobs with processing times $2m - 1, 2m - 1, 2m - 2, 2m - 2, \ldots, m + 1, m + 1, m, m, m$.

We propose the heuristic algorithm A_4, which generates a heuristic cyclic schedule for the problem.

Fig. 5. Task graph G_{new} $t(a) = \varepsilon$; $t(V_i) = 1$; $t(U_i) = m - 1$; $t(W) = m - 1$

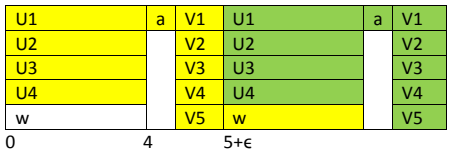

Fig. 6. The schedule S_z, cycle time $z := m + \epsilon$.(Fig.5)

7.1 Algorithm A_4.

Step1. Define lower bound LB for the optimal cycle time z_{opt}.
$LB = \lceil (\sum_{i=1}^{n} p(v_i))/m \rceil$.

Step 2. Find schedule S_L for the problem and find start times $t(v_i)$ and makespan $C_{\max}$, use $LPT(V; S_L, C_{\max})$.

Step 3. If $C_{\max} = LB$ then schedule S_L is optimal cyclic schedule and cycle time is equal $C_{\max}$, goto Step 10.

Step 4. Define the sum of processing times of every processors $s_1, s_2, \ldots, s_m$. Lets $\tau(j)$ is the start time of processor j.

Step 5. Renumber processors in non-increasing order: $s_1 \geq s_2 \geq \ldots, \geq s_m$. Define $z = \max\{(s_j + s_{m-j+1})/2 \mid j \in 1 : m\}$.

Step 6. Define $f(v_i)$ processor for job v_i.

Step 7. Define the start time of processor j $\tau(1) := 0$, $\tau(j) = (s_1 - s_j)/2$, $j \in 2 : m$.

Step 8. If $f(v_i) = j$ then $t(v_i) := t(v_i) + \tau(j)$.

Step 9. Algorithm generates the cyclic schedule $\sigma = (t, z)$.

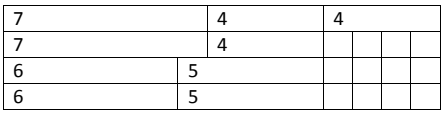

Fig. 7. The schedule S_L, $C_{\max} = 15$

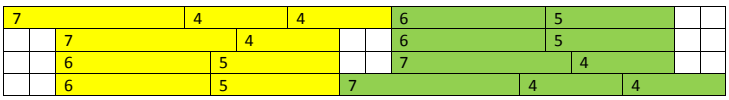

Fig. 8. The cycle schedule S_z, cycle time $z := 13$

Example 3. Consider the worst case example of LPT algorithm with $m = 4$,$n = 9$

Processing times of jobs $p(V) = (7, 7, 6, 6, 5, 5, 4, 4, 4)$.

LPT algorithm generates the schedule S_L , $C_{\max} = 15$(Fig. 7) The optimal schedule has makespan $C_{opt} = 12$. $t(V) = (0, 0, 0, 0, 6, 6, 7, 7, 11)$ and $f(V) = (1, 2, 3, 4, 4, 3, 2, 1, 1)$.

Create the cyclic schedule S_c. $s_1 = 15, s_2 = 11, s_3 = 11, s_4 = 11$

Define the start time of processors $\tau_1 = 0, \tau_2 = 2, \tau_3 = 2, \tau_4 = 2$.

Define t(V)=(0,2,2,2,8,8,9,7,11). $z = 13$. The cycle schedule S_z is resource-periodic with period 2: $f(v_i, k) = f(v_i, k + 2)$ (Fig.8).

8 Conclution

We consider the multiprocessor cyclic scheduling problems with loops, where the goal is to find a periodic schedule that minimizes the cycle time under precedence constraints. We propose algorithms for four problems. The problem with unit processing times and the problem with preemptions can be solved in polynomial time. Algorithms for two problems with arbitrary processing times (with precedence constrains and without its) generates heuristic schedules.

References

1. E. Levner, V. Kats, and V.E. Levit, An improved algorithm for a cyclic robotic scheduling problem, European Journal of Operational Research 97, (1997) 500-508.
2. V. Kats and E. Levner, Cyclic scheduling on a robotic production line, Journal of Scheduling, 5, (2002) 23-41.
3. W. Kubiak, Solution of the Liu-Layland problem via bottleneck just-in-time sequencing, Journal of Scheduling, 8(4), (2005) 295 - 302.
4. M. Fink , T. B. Rahhou, L.Houssin. A New Procedure for the Cyclic Job Shop Problem. In p Proceedings of the 14th IFAC Symposium on Information Control Problems in Manufacturing Bucharest, Romania, May 23-25, 2012, 69—74.
5. Hanen Claire, Munier Alix. Cyclic scheduling on parallel processors: an overview. In: Chrtienne Philippe, Coffman Edward G, Lenstra Jan Karel, Liu Zhen, editors. Scheduling theory and its applications. New York: John Wiley and Sons; 1994.
6. Hanen, C., Munier, A.: A study of the cyclic scheduling problem on parallel processors. Discrete Appl. Math. 57, (1995) 167-192.
7. J. Ullman. NP-complete scheduling problems *J. Comp. Sys. Sci.* **171**, pp.394-394, (1975).
8. Graham, R.L., Lawner, E.L., Rinnoy Kan,A.H.G.: Optimization and approximation in deterministic sequencing and scheduling. A survey. Ann. of Disc. Math. **5** (10), 287–326 (1979)
9. Graham, R. L., Bounds for certain multiprocessing anomalies. Bell System Technical Journal, 45, 15631581 (1966).
10. Graham, R. L., Bounds on multiprocessing timing anomalies. SIAM Journal of Applied Mathematics, 17, 416426
11. Munier, A. The complexity of a cyclic scheduling problem with identical machines and precedence constraints. Eur. J. Oper. Res. 91(3), (1996) 471 - 480.
12. Brucker Peter, Kampmeyer Thomas. A general model for cyclic machine scheduling problems. Discret. Appl. Math. 156(13) (2008) 2561-72.
13. Coffman E.G. and Graham R.L. Optimal schedule for two-processor systems/ Acta Informat. 1 (1972) 200-213.
14. Benot Dupont de Dinechin a, Alix Munier Kordon b,n Converging to periodic schedules for cyclic scheduling problems with resources and deadlines. Computers & Operations Research 51 (2014) 227236
15. Nargess Shirvani a,n, Rubn Ruiz b, Shahram Shadrokh Cyclic scheduling of perishable products in parallel machine with release dates, due dates and deadline.Int. J. Production Economics 156 (2014) 1 12
16. R. Muntz, E. G. Coffman, Jr., Preemptive scheduling of real time tasks on multiprocessor systems, /. Assoc. Comput. Mach. 17, 1970, 324-338