Math. Meth. Oper. Res (2005) 62: 99–122
DOI 10.1007/s00186-005-0443-4

ORIGINAL ARTICLE

Eugene A. Feinberg · Michael T. Curry

Generalized Pinwheel Problem

Received: November 2004 / Revised: April 2005

Abstract This paper studies a non-preemptive infinite-horizon scheduling problem with a single server and a fixed set of recurring jobs. Each job is characterized by two given positive numbers: job duration and maximum allowable time between the job completion and its next start. We show that for a feasible problem there exists a periodic schedule. We also provide necessary conditions for the feasibility, formulate an algorithm based on dynamic programming, and, since this problem is NP-hard, formulate and study heuristic algorithms. In particular, by applying the theory of Markov Decision Process, we establish natural necessary conditions for feasibility and develop heuristics, called frequency based algorithms, that outperform standard scheduling heuristics.

1 Introduction

Suppose there are $n \in \mathbb{N} = \{1, 2, \dots\}$ recurring jobs. Each job $i = 1, \dots, n$ is characterized by two positive numbers: τ_i, the duration of job i, and u_i, the maximum amount of the time that can transpire between epochs when job i is completed and is started again. We call them the duration and revisit time respectively. These jobs are to be completed by a single server. There is no preemption and setups are instantaneous. A schedule is a sequence in which the jobs should be performed. A schedule is feasible if each time job i is completed, it will be started again within

E.A. Feinberg (✉)
Department of Applied Mathematics and Statistics,
State University of New York,
Stony Brook, NY 11794-3600, USA
E-mail: efeinberg@notes.cc.sunysb.edu

M.T. Curry
Department of Applied Mathematics and Statistics,
State University of New York,
Stony Brook, NY 11794-3600, USA
E-mail: curry@ams.sunysb.edu

u_i units of time. A problem is feasible if a feasible schedule exists. We call this problem the Generalized Pinwheel Problem, (GPP). The GPP is to find a feasible schedule or to prove that it does not exist.

Our interest in this problem was initiated by applications. In particular, Feinberg, et al [12] studied the so-called radar sensor management problem, the mathematical formulation of which is the GPP. Since the GPP is generic, there are various other engineering applications such as mobile communications, wired and wireless networks, satellite transmissions, database support. In addition, though the GPP is a deterministic problem, the methods of Markov Decision Processes (MDPs), an area to which Professor Ulrich Rieder made many important contributions, are useful to study this problem. From the methodological point of view, this paper continues the line of research initiated by Filar and Krass [14] that deals with applications of MDP methods to discrete optimization problems; see also [1,5,8,11] and references therein.

If all of the τ_i are equal to 1, the problem becomes the Pinwheel Problem (PP) introduced by Holte, et al [16,17]. In particular, it was shown there that the PP is infeasible when the density $\rho^* := \sum_{i=1}^{n} \frac{1}{1+u_i} > 1$. Holte, et al [16,17] also proved that any instance with $\rho^* \leq .5$ can be scheduled in polynomial time. Chan and Chin [7,6] improved this result to $\frac{2}{3}$ and then to .7. For only three distinct numbers of revisit times, Lin and Lin [20] proved a similar result for $\rho^* < .83$. Holte, et al [16] claimed that the PP is NP-hard when $\rho^* = 1$. However, the proof of this fact has not been published as far as we know. We notice that, in general, none of the known algorithms and methodologies for the PP are applicable to the GPP.

In this paper we show that if the GPP is feasible, a periodic schedule exists. The GPP is NP-hard, at least in the case when $\rho := \sum_{i=1}^{n} \frac{\tau_i}{\tau_i+u_i} = 1$. We provide a relaxation of the GPP based on constrained Semi-Markov Decision Processes (SMDP). This relaxation implies that if $\rho > 1$ then the problem is infeasible. Unlike the PP, the GPP may not be feasible when ρ is small. We also find sufficient conditions when the GPP is infeasible and $\rho < 1$.

Since the GPP is NP-hard, at least for $\rho = 1$, we study heuristic approaches in this paper. In particular, we develop a so-called frequency based algorithm that uses a constrained SMDP corresponding to the GPP as a starting point. In fact, we introduce a sequence of approximations of the GPP by SMDPs. The approximation is tighter if the system remembers a larger sequence of jobs previously performed. The first and simplest heuristic of this type deals with the situation when the system has no memory.

This paper is organized in the following way. Section 2 describes some preliminary results as well as the strong NP-hardness of the GPP. In that section we prove the existence of periodic schedules. Section 3 describes some necessary conditions for the GPP to be scheduled and introduces some structural properties of GPPs. We establish the link between the GPP and a particular deterministic SMDP. By using this link, we apply the linear programming approach to the constrained SMDPs with average rewards per unit time [9] and prove that $\rho \leq 1$ is a necessary condition for feasibility.

In Section 4 we provide a tighter relaxation based on a larger SMDP than the relaxation described in Section 3. Unlike the relaxation in Section 3, where stationary policies have no memory, we consider an SMDP when the stationary policies remember the last completed job. We provide a tighter linear programming

relaxation of the GPP. For example, this relaxation also implies that $\rho \leq 1$ is a necessary condition for feasibility. However, the new relaxation provides more detailed linear constraints that can indicate the conditions for $\rho < 1$ and the GPP being infeasible. In Section 5 we consider the SMDP models when stationary policies remember the k last jobs performed, $k \in \mathbb{N}$. We show that, if we do not exclude from the state space the possible last k-job subsequences that clearly cannot be a part of a feasible schedule, then the relaxations for different positive k are equivalent.

In Section 6 we provide a dynamic programming formulation. We show that a schedule can be presented as a function of the vector of residual times and formulate a value iteration algorithm based on Negative Dynamic Programming. We prove that this algorithm converges in a finite number of iterations. In particular, we apply this dynamic programming algorithm and construct an example of a PP that is infeasible when $\rho^* < 1$.

Section 7 formulates standard scheduling heuristics: due date, critical ratio, and round robin algorithms. In Section 8 we develop a frequency based heuristic based on the SMDP representation of the GPP and on the time-sharing approach for constructing nonrandomized optimal policies for a Markov Decision Process (MDP) with constraints [23,2].

In Section 9 we construct linear constraints that are tighter when the parameter k increases. We also formulate the frequency based algorithm for an arbitrary k. In addition, we provide an exact algorithm, which is based on frequency based algorithms with increasing k, in Section 9.

2 Structural results

First, we show that if a feasible schedule exists, it can be expressed as a periodic feasible schedule. A finite sequence of jobs $\bar{i} = i_1, \ldots, i_m$ is called a period of a schedule s if $s = \bar{i}, \bar{i}, \bar{i} \ldots$. A schedule is called periodic if it has a period.

Theorem 2.1 *For a feasible problem there exists a feasible periodic schedule.*

Proof Let t be a positive number. We say that a finite sequence $i_1, \ldots, i_\ell$ with values in $\{1, \ldots, n\}$ crosses the level t if $\sum_{k=1}^{\ell} \tau_{i_k} \geq t$ and $\sum_{k=1}^{\ell-1} \tau_{i_k} < t$. Since all $\tau_i > 0$, the set of all sequences that cross a given level $t > 0$ is finite.

Now let $u = \max\{u_1, \ldots, u_n\}$. Consider a feasible schedule $b = b_1, b_2, \ldots$. For each $i \in \mathbb{N}$ we consider a unique finite sequence $b^i = b_i, b_{i+1}, \ldots, b_{i+k(i)}$ such that b^i crosses the level u. Since the number of all sequences that cross the level u is finite, there are two integers i and j with the following properties: (i) $j > i + k(i)$, and (ii) $b^i = b^j$.

Let $\bar{i} = b^i, b_{i+k(i)+1}, \ldots, b_{j-1}$, where b_i is an instance of a job. Then $\bar{i}, \bar{i}, \bar{i}, \ldots$ is a feasible schedule. We remark that, in fact, this proof provides an algorithm that converts a feasible schedule into a periodic feasible schedule. □

Next we formulate the complexity result. We recall that a class of problems is strongly NP-hard if it remains NP-hard even when all numbers in the input are bounded by some polynomial in the length of the input.

Theorem 2.2 *[[12, Theorem 2.2]] The GPP is strongly NP-hard when* $\rho = 1$.

If ρ is small, a PP has a feasible solution and can be scheduled in polynomial time [6,7,17,16]. The following example illustrates that the GPP may not have a feasible schedule for any small ρ. We do not know if the GPP is polynomial for small ρ.

Example 2.3 Let $n = 2$, $\tau_1 = 2\epsilon$, $\tau_2 = \epsilon^2$, $u_1 = 2$, and $u_2 = \epsilon$, where $\epsilon > 0$. Then $\rho = 2\epsilon/(\epsilon + 1) \to 0$ as $\epsilon \to 0$. However, since $\tau_1 > u_2$, there is no feasible schedule.

3 Necessary condition for schedulability

Since, in view of Theorem 2.1, a feasible problem always has a periodic schedule, we are interested only in finding periodic schedules. For periodic schedules, there exist average intervals between the same job executions. We re-formulate the problem and will try to find a schedule in which the average intervals between the job is completed and started again is limited above by $\bar{u}_i$, where $\bar{u}_i \le u_i$, $i = 1, \dots, n$. This is a reasonable re-formulation of the original problem. Indeed, if a feasible schedule exists, for its periodic version these average variables exist and they are not greater than u_i. When $\bar{u}_i = u_i$ for all $i = 1, \dots, n$, the new problem is a relaxation of the original one.

Let $s = i_1 i_2, \dots$ be a periodic schedule containing all the jobs $1, \dots, n$ and let $i_1, \dots, i_m$ be its period. Consider any job $i = 1, \dots, n$ and let $K^s(i)$ be the number of instances of job i in the period,

$$K^s(i) = \sum_{j=1}^{m} \delta_{i_j,i}, \tag{1}$$

where $\delta_{i,j}$ is the Kronneker symbol.

Let w_i^s be the average interval between the epochs when job i is completed and started again. Since s is periodic,

$$w_i^s = \lim_{N\to\infty} \left[\sum_{j=1}^{N} \tau_{i_j}(1 - \delta_{i_j,i}) \right] / \sum_{j=1}^{N} \delta_{i_j,i} = \left[\sum_{j=1}^{m} \tau_{i_j} - \tau_i K^s(i) \right] / K^s(i). \tag{2}$$

Consider a weaker problem: find a periodic schedule s such that it contains all the jobs $1, \dots, n$ and, in addition, $w_i^s \le \bar{u}_i$ for all $i = 1, \dots, n$.

Let

$$v_i^s = \lim_{N\to\infty} \left[\sum_{j=1}^{N} \tau_i \delta_{i_j,i} \right] / \sum_{j=1}^{N} \tau_{i_j} = \left[\sum_{j=1}^{m} \tau_i \delta_{i_j,i} \right] / \sum_{j=1}^{m} \tau_{i_j} = \left[\tau_i K^s(i) / \sum_{j=1}^{m} \tau_{i_j} \right] \tag{3}$$

be the proportion of time performing job i. Straightforward calculations applied to (2) and (3) imply that

$$v_i^s = \tau_i / (w_i^s + \tau_i). \tag{4}$$

The following lemma follows from (4).

Lemma 3.1 *Let s be a periodic schedule containing a job $i = 1, \ldots, n$. Then $w_i^s \le \bar{u}_i$ if and only if $v_i^s \ge \tau_i/(\bar{u}_i + \tau_i)$. Therefore, finding a periodic schedule s containing all the jobs $1, \ldots, n$ and such that $w_i^s \le \bar{u}_i$ for all $i = 1, \ldots, n$ is equivalent to finding a periodic schedule s satisfying*

$$v_i^s \ge \tau_i/(\bar{u}_i + \tau_i), \ i = 1, \ldots, n. \tag{5}$$

If, in addition to periodic schedules, we consider schedules generated by randomized and past-dependent procedures, the weaker version of our problem can be naturally formulated as a constrained SMDP; see [9]. Though SMDPs are stochastic systems, here we deal with a particular case of a deterministic SMDP that, of course, can have stochastic trajectories when a randomized policy is implemented. The results on constrained SMDPs needed for this paper are presented in the end of this section.

Now we describe the constrained SMDP for the GPP. The state space I consists of one state. Thus, there are no transition probabilities. We omit the notations for this state, e.g., $r_k(i, a) = r_k(a)$. The set of actions $A = \{1, \ldots, n\}$, i.e. it is the set of jobs. If a job j is selected, the deterministic sojourn time is τ_j. The number of constraints $K = n$ and $r_k(j) = \delta_{k,j}\tau_j$. Since we do not have an objective function, we set $r_0(j) = 0$. For this SMDP, we consider the values v_k^π defined in (10).

Relaxed GPP. Find a policy π such that

$$v_i^\pi \ge \tau_i/(\bar{u}_i + \tau_i), \ i = 1, \ldots, n. \tag{6}$$

In view of Lemma 3.1, the relaxed GPP is consistent with finding a possibly randomized strategy which is feasible in terms of average revisit times. We consider equations (13 – 17) for this problem with l_k set to equal $\tau_k/(\bar{u}_k + \tau_k)$, $k = 1, \ldots, n$. Since the objective function is 0 for all policies, our goal is to find a feasible solution. Thus, (13) should be dropped. Equations (14) become the identities $z_i = z_i$, $i = 1, \ldots, n$, and should be dropped as well. The remaining constraints become

$$z_i \ge (\bar{u}_i + \tau_i)^{-1}, \qquad i = 1, \ldots, n, \tag{7}$$

$$\sum_{i=1}^{n} \tau_i z_i = 1. \tag{8}$$

Let

$$\rho = \sum_{i=1}^{n} \tau_i/(u_i + \tau_i). \tag{9}$$

If $\bar{u}_i = u_i$ for all i, it is easy to see that the constraints (7, 8) are feasible if and only if $\rho \le 1$. This observation and the above explanation imply the following result.

Theorem 3.2 *The relaxed GPP is feasible for $\bar{u}_i = u_i$, $i = 1, \ldots, n$, if and only if $\rho \le 1$. Therefore, the GPP is infeasible when $\rho > 1$.*

We present in a concentrated form the major concepts and results on constrained Semi-Markov Decision Processes (SMDPs) with average rewards per unit time used in this paper. The details can be found in [9].

A finite state and action constrained SMDP with average rewards per unit time is defined by the set $\{I, A, A(\cdot), p, \tau, K, r., l.\}$, where

I is a finite state set;

A is a finite action set;

$A(i)$ are action sets available in states $i \in I$, $A(i) \subset A$;

$p(j|i, a)$ is the probability that the next state is $j \in I$ if an action $a \in A(i)$ is selected at state $i \in I$;

$\tau_{i,a} > 0$ is the expectation of the time spent in state i if an action $a \in A(i)$ is selected;

K is the number of constraints (the number of reward criteria is $(K + 1)$);

$r_k(i, a)$ is the reward for criterion $k = 0, 1, \dots K$ when an action $a \in A(i)$ is selected in a state i;

l_k is the constraint for criterion $k = 1, \dots, K$.

In general, for an SMDP, a system changes states in jumps and an action is selected at a jump epoch. Unlike MDPs, the sojourn times may not be identically equal to 1. Let $h_N = i_0, a_0, \xi_0, i_1, a_1, \xi_1, \dots, i_N$ be the history up to and including Nth jump, when i_j are states, a_j are selected actions, and ξ_j are sojourn times respectively. Let H_N be the set of all histories up to and including Nth jump; $H_0 = I$. A (randomized) strategy $\pi = \{\pi_0, \pi_1, \pi_2, \dots\}$ is a sequence of regular transition probabilities from H_N to I such that $\pi_N(A(x_N)|h_N) = 1$, $N = 0, 1, \dots$. A strategy is called a randomized stationary policy if $\{\pi_0, \pi_1, \pi_2, \dots\} = \{\pi, \pi, \pi, \dots\}$, where $\pi_N(\cdot|h_N) = \pi(\cdot|x_N)$, $N = 0, 1, \dots$. Here, as it is standard for MDPs, we denote by π both the π the conditional probability and the sequence of identical transition probabilities. A stationary policy is a mapping φ from I to A such that $\varphi(i) \in A(i)$.

A strategy π and an initial state i define average rewards per unit time for criteria $k = 0, 1, \dots, K$,

$$v_k^\pi(i) = \liminf_{t\to\infty} t^{-1} E_i^\pi \sum_{j=0}^{N(t)} r_k(i_j, a_j), \tag{10}$$

where E_i^π is the expectation operator defined by the initial state i and strategy π, and $N(t)$ is the number of jumps up to and including time t. In addition, if π is a randomized stationary policy, then the limits exist in (10).

The problem is: for a given initial state i

$$\text{maximize } v_0^\pi(i) \tag{11}$$

$$s.t. v_k^\pi(i) \geq l_k, \quad k = 1, 2, \dots, K. \tag{12}$$

A problem is called unichain if any nonrandomized stationary policy defines a Markov chain with one ergodic class. According to Theorem 9.2 in [9], a unichain problem is feasible if and only if the following Linear Program (LP)

$$\text{maximize } \sum_{i\in I} \sum_{a\in A(i)} r_0(i, a) z_{i,a} \tag{13}$$

subject to

$$\sum_{a\in A(j)} z_{j,a} - \sum_{i\in I}\sum_{a\in A(i)} p(j|i,a)z_{i,a} = 0, \qquad j \in I, \tag{14}$$

$$\sum_{i\in I}\sum_{a\in A(i)} r_k(i,a)z_{i,a} \geq l_k, \qquad k = 1,\ldots,K, \tag{15}$$

$$\sum_{i\in I}\sum_{a\in A(i)} \tau(i,a)z_{i,a} = 1, \tag{16}$$

$$z_{i,a} \geq 0, \qquad i \in I,\ a \in A(i). \tag{17}$$

is feasible. In addition, if this LP is feasible then there exists an optimal strategy which is a randomized stationary policy. This policy does not depend on the initial state i in (11) and (12). If z is an optimal solution of the LP (13 – 17) then the formula

$$\pi(a|i) = \begin{cases} z_{i,a}/z_i, & \text{if } z_i > 0; \\ \text{arbitrary}, & \text{otherwise}; \end{cases} \tag{18}$$

where

$$z_i = \sum_{a\in A(i)} z_{i,a}, \qquad i \in I, \tag{19}$$

defines a stationary optimal policy.

In fact in this paper we are interested in solving a feasibility problem. Therefore, the objective function (13) is unimportant. For MDPs, $\tau(i,a) = 1$ for all i and a. Let

$$x_{i,a} = z_{i,a} / \sum_{j\in I}\sum_{b\in A(j)} z_{j,b}, \qquad i \in I. \tag{20}$$

Which is equivalent to

$$z_{i,a} = x_{i,a} / \sum_{j\in I}\sum_{b\in A(j)} x_{j,b}, \qquad i \in I. \tag{21}$$

Then, (14–17) can be rewritten as

$$\sum_{a\in A(j)} x_{j,a} - \sum_{i\in I}\sum_{a\in A(i)} p(j|i,a)x_{i,a} = 0, \qquad j \in I, \tag{22}$$

$$\sum_{i\in I}\sum_{a\in A(i)} r'_k(i,a)x_{i,a} \geq 0, \qquad k = 1,\ldots,K, \tag{23}$$

$$\sum_{i\in I}\sum_{a\in A(i)} x_{i,a} = 1, \tag{24}$$

$$x_{i,a} \geq 0, \qquad i \in I,\ a \in A(i). \tag{25}$$

where $r'_k(i,a) = r_k(i,a) - l_k\tau_{i,a}$. This is an LP constraint for a similar problem with the process being an MDP instead of an SMDP.

In fact, utilizing equation (20), (18) can be rewritten as

$$\pi(a|i) = \begin{cases} x_{i,a}/x_i, & \text{if } x_i > 0; \\ \text{arbitrary}, & \text{otherwise.} \end{cases} \tag{26}$$

Note that x satisfies (22–25), if and only if z satisfies (13–17).

4 First order relaxation

In the previous sections we considered a simple relaxation of the GPP when the state of the corresponding SMDP consists of one point. For randomized stationary policies, no information can be used regarding previously selected jobs. Now we consider the situation when the decision maker remembers the last executed job. We call the relaxation resulted in this approach the first order relaxation. We remark that the decision maker could implement the policies that remember the last job performed in the SMDP described in the previous section. However, these policies are not stationary and stationary policies do not use this information.

Obviously, if the problem is feasible, a periodic schedule $s = (i_1, i_2, \dots, i_m)$, where m is the number of jobs in the period, can be selected in a way that sequential jobs are distinct, i.e. $i_{j+1} \neq i_j$, $j = 1, \dots, m-1$, and $i_1 \neq i_m$. Indeed, if a feasible schedule contains repeated jobs, i.e. $i_{j+1} = i_j$, the job i_{j+1} can be removed in all such instances and the remaining schedule is feasible and it does not have identical consecutive jobs. The model when the last job is remembered leads to stationary policies that do not repeat jobs consecutively.

We introduce an SMDP with stationary policies remembering the last executed job; see Section 3 for general SMDP definitions. The state and action sets are $I = A = \{1, \dots, n\}$. The set of actions available at state i is $A(i) = A \setminus \{i\}$. If an action j is selected at state i, the system spends time τ_j at i and the next state is j. We recall that the state of the system in our approach is the last completed job. The rewards, for the k^{th} criterion are defined by $r_k(i,j) = \delta_{k,j}\tau_j$. We also set $l_k = \tau_k/(\bar{u}_k + \tau_k)$, $k = 1, \dots, n$.

We formulate the LP for this approach, where the formulas (27 – 30) are the formulas (13 – 17) applied to this SMDP:

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} z_{ij} = \sum_{\substack{j=1 \\ j\neq i}}^{n} z_{ji}, \qquad i = 1, \dots, n, \tag{27}$$

$$\sum_{j=1}^{n} \tau_j \sum_{\substack{i=1 \\ i\neq j}}^{n} z_{ij} = 1, \tag{28}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} z_{ij} \geq \frac{1}{\bar{u}_j + \tau_j}, \qquad j = 1, \dots, n, \tag{29}$$

$$z_{ij} \geq 0, \qquad i, j = 1, \ldots, n, \ j \neq i. \tag{30}$$

We cannot apply directly the results of Section 3 to justify that (27 – 30) provide a relaxation of our problem because this SMDP may not be unichain (see example 4.3). However, we shall provide an indirect proof by using the results of Filar and Guo [13] on communicating MDPs.

Theorem 4.1 *If the GPP is feasible then the system of linear constraints (27 – 30) with* $\bar{u}_j = u_j, \ j = 1, \ldots, n$, *has a solution.*

Proof Consider a feasible schedule. According to theorem 2.1, this schedule has a period $i_1, \ldots, i_m$. Therefore, this schedule can be represented as some nonrandomized policy s in the SMDP with state space I described above. The policy s remembers $m-1$ last states including the current state and it reproduces the given periodic schedule. This policy s is formally defined by $s(i_0, i_1, \ldots, i_t) = i_{t+1}$ when $t < m-1$ and by $s(i_{t-m+2}, \ldots, i_t) = i_{\ell+1}$, $t \geq m-1$, where $\ell = t \pmod m$. We observe that

$$v_k^s = \frac{\sum_{j=1}^m \tau_k \delta_{i_j,k}}{\sum_{j=1}^m \tau_{j_k}} \geq \frac{\tau_k}{\tau_k + u_k}, \qquad k = 1, \ldots, n, \tag{31}$$

where the equality follows from (3) and the inequality follows from lemma 3.1. The inequality in (31) is equivalent to

$$\sum_{j=1}^m \tau_k \left(\delta_{i_j,k} - \tau_{j_k}/(u_k + \tau_k)\right) \geq 0, \tag{32}$$

which implies

$$\lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^N \tau_k \left(\delta_{i_j,k} - \tau_{j_k}/(u_k + \tau_k)\right) = \frac{1}{m} \sum_{j=1}^m \tau_k \left(\delta_{i_j,k} - \tau_{j_k}/(u_k + \tau_k)\right) \geq 0. \tag{33}$$

Therefore, the policy s is feasible for the MDP with the state space I, action sets $A(i)$, $i \in I$, deterministic transitions from state i to j when the action j is selected at i, the one-step rewards $r'_k(i,a) = \tau_k \left(\delta_{i,k} - \frac{\tau_k}{\tau_k + u_k}\right)$, $k = 1, \ldots, n$, and n non-negativity constraints. For each reward function r'_k, consider average rewards per unit time. Since it is possible to move this MDP from any state $i \in I$ to any state $j \in I$, $j \neq i$, in one step, this MDP is communicating; see Filar and Guo [13] for details on communicating MDPs. Theorem 4.1 and Lemma 3.1 in Filar and Guo [13] imply that the problem (11, 12) is feasible for a communicating MDP if and only if the system of linear constraints (22 – 25) is feasible. Formula (21) transforms (22 – 25) into constraints (14–17) which are (27 – 30) in our case. Since s is a feasible policy, constraints (27–30) are feasible. □

We remark that if $\rho > 1$ then Theorems 3.2 and 4.1 imply that the system of constraints (27 – 30) is infeasible. However, if this system is infeasible, then the GPP is infeasible even when $\rho \leq 1$. The following example illustrates that constraints (27 – 30) can provide a better relaxation of the problem than constraints (7, 8).

Example 4.2 Let $n = 2$ and $\bar{u}_i = u_i$, i = 1,2. According to Theorem 3.2, the feasibility of constraints (7, 8) is equivalent to $\rho \leq 1$. This is a necessary condition for the feasibility of the GPP. According to Example 2.3, this condition is not sufficient. Obviously, for $n = 2$ the GPP is feasible if and only if the round robin schedule is feasible. This is equivalent to $u_1 \geq \tau_2$ and $u_2 \geq \tau_1$. Simple calculations imply that the feasibility of (27 – 30) is equivalent to the validity of these two inequalities. Thus, for $n = 2$ the feasibility of (7, 8) is the necessary condition for the feasibility of the GPP while the feasibility of (27 – 30) is the necessary and sufficient condition.

The following example illustrates that the SMDP described in this section indeed may not be unichain. Therefore, if we have a vector z_i satisfying constraints (27 – 30), the results from [9] do not imply that the stationary policy defined by (18) is feasible.

Example 4.3 Consider a PP ($\tau_i \equiv 1$) with 4 jobs. We want to schedule them in a way that the time between executions of the same job does not exceed 3. We relax this problem and try to find a policy that satisfies these constraints on average. This leads us to (27 – 30) with $\tau_i = 1$ and $\bar{u}_i = 3$, $i = 1, 2, 3, 4$. Let $z_{12} = z_{21} = z_{34} = z_{43} = 1/4$ and the remaining $z_{ij} = 0$, $i, j = 1, 2, 3, 4$. This vector z satisfies (27 – 30). Formula (18) defines a stationary policy that prescribes action 2 in state 1, action 1 in state 2, action 4 in state 3, and action 3 in state 4. If the system is in state 1, jobs 3 and 4 will never be executed. If the system is in state 3, jobs 1 and 2 will never be executed. In fact, the solution z of (27 – 30), that defines a unichain Markov chain in the search algorithm proposed later, is $z_{ij} = 1/12$ when $i \neq j$.

Theorem 4.4 *Let a vector z satisfy the constraints (27 – 30). Consider a randomized stationary policy π defined by (18) with the parameter a substituted with j, $j \neq i$, $j = 1, \ldots, n$. Define a Markov chain on the state space I with the transition probabilities $\pi(j|i)$ from i to j, $i, j = 1, \ldots, n$. If this Markov chain is unichain (i.e. has one recurrent class) then for this chain the average time between consecutive visits to each state $i = 1, \ldots, n$ is less than or equal to $\bar{u}_i$.*

Proof Let $z_i = \sum_j z_{ij}$. Formulas (27) and (18) imply that $\sum_i \pi(j|i) z_i = z_j$, $j = 1, \ldots, n$. Therefore $\tilde{z}_i = z_i / \sum_{j=1}^{n} z_j$, $i = 1, \ldots, n$, is the stationary distribution of the Markov chain with the transition probabilities $\pi(j|i)$ defined in Section 3. In view of (29), $\tilde{z}_i > 0$ for all i. Therefore, this Markov chain has no transient states. From (2 – 4) and standard renewal theory arguments we have that the constraints (29) imply that the average revisit time for each job $j = 1, \ldots, n$ is not greater than $\bar{u}_j$. □

5 Higher order relaxations

In Section 4 we developed the linear constraints when the stationary policy remembers the last performed job. We have demonstrated that the solution for this approach is tighter than the solution when the decision maker has no information about previous jobs. It is natural to ask whether further improvement can be found by remembering the sequence of the last k jobs where $k > 1$. In this

section we develop this concept. In order to utilize the results of Section 3 we need a unichain SMDP. However, as Example 4.3 demonstrates, unlike the case $k = 0$, the unichain condition for $k \geq 1$ usually does not hold. We remark that the unichain condition holds when $n \leq 3$ and $k = 1$.

Similarly to the previous section we define an SMDP that remembers the last k executed jobs. This SMDP has a state space $I = I_k$, where I_k is the set of finite sequences $i_1 i_2, \ldots i_k$ which elements take values in $\{1, \ldots, n\}$ and such that $i_j \neq i_{j+1}$, $j = 1, \ldots, k-1$. The set I_k consists of at most $n(n-1)^{k-1}$ elements. The action set is $A = \{1, \ldots, n\}$. The set $A(\bar{i})$ of actions available at state $\bar{i} = (i_1, \ldots, i_k) \in I_k$ is $A \setminus \{i_k\}$. If an action j is selected at state i_k of $\bar{i}$, the system spends time τ_j in this state and then it moves to the state $i_2, \ldots, i_k, j$. The rewards are $r_\ell(i_k, j) = \delta_{\ell,j}\tau_j$. We also consider $l_\ell = \tau_\ell/(\bar{u}_\ell + \tau_\ell)$. Similarly to (27 – 30) when we dealt with $k = 1$, we write the constraints (13 – 17) for this SMDP:

$$\sum_{\substack{j=1 \\ j \neq i_k}}^{n} z_{\bar{i}j} = \sum_{\substack{j=1 \\ j \neq i_1}}^{n} z_{j\bar{i}}, \qquad \bar{i} \in I_k, \tag{34}$$

$$\sum_{\bar{i}=(i_1,i_2,\ldots,i_k)\in I_k} \sum_{\substack{j=1 \\ j \neq i_k}}^{n} z_{\bar{i}j}\tau_j = 1, \tag{35}$$

$$\sum_{\substack{\bar{i} = (i_1, i_2, \ldots, i_k) \in I_k \\ i_k \neq j}} z_{\bar{i}j} \geq \frac{1}{\bar{u}_j + \tau_j}, \qquad j = 1, \ldots, n, \tag{36}$$

$$z_{\bar{i}j} \geq 0, \qquad \bar{i} = (i_1, \ldots, i_k) \in I_k, \quad j = 1, \ldots, n, \ j \neq i_k. \tag{37}$$

The natural question is whether the constraints (34 – 37) for $k \geq 2$ are tighter relaxations of the original problem than the similar constraints for $k-1$. The answer is somewhat surprising: these constraints are equivalent. However, in Section 9 we shall correct the constraints (34 – 37) in a way that higher values of k lead to tighter relaxations.

Given the variables $z_{\bar{i}j}$ when $\bar{i} \in I_k$, we define the variables

$$z_{i_2 i_3 \ldots i_k j} = \sum_{\substack{i_1 = 1 \\ i_1 \neq i_2}}^{n} z_{i_1 i_2 \ldots i_k j} \qquad (i_2, \ldots, i_k) \in I_{k-1}, \text{ and } j \neq i_k. \tag{38}$$

Straightforward calculations yield that, if $z_{i_1 i_2 \ldots i_k, j}$ satisfy the constraints (34 – 37) for $i_1 i_2 \ldots i_k \in I_k$, $j = 1, \ldots, n$, and $j \neq i_k$, then the variable $z_{i_2 \ldots i_k, j}$, defined in (38) satisfy the same constraints for $k-1$. The following theorem presents this fact.

Theorem 5.1 *If z is a solution of the linear constraints (34 – 37) for some $k = 2, 3, \dots$ then formula (38) defines a solution of the same linear constraints for $k-1$.*

Let $z_{i_1 i_2 \dots i_k i_{k+1}}$ be a solution to the linear constraints (34 – 37) for some $k \in \mathbb{N}$. For $i \in \mathbb{N} \setminus \{i_{k+1}\}$ we define

$$z_{i_1 i_2 \dots i_{k+1} j} = z_{i_1 i_2 \dots i_k i_{k+1}} \cdot z_{i_2 \dots i_k i_{k+1}, j} / z_{i_2 \dots i_{k+1}}, \tag{39}$$

where

$$z_{i_2 \dots i_{k+1}} = \sum_{\substack{j=1 \\ j \neq i_{k+1}}}^{n} z_{i_2 \dots i_{k+1} j}. \tag{40}$$

Straightforward calculations yield that if $z_{i_1 i_2 \dots i_k j}$ satisfy the constraints (34 – 37) for $i_1 i_2 \dots i_k \in I_k$, $j = 1, \dots, n$, and $j \neq i_k$ then the variables $z_{i_2 \dots i_{k+1} j}$, defined in (38), satisfy the same constraints for $k+1$. Thus we have the following result.

Theorem 5.2 *If the variables $z_{i_1 i_2 \dots i_k j}$ satisfy the linear constraints (34 – 37) for any $k \in \mathbb{N}$ then the variables $z_{i_1 i_2 \dots i_{k+1} j}$ defined in (39) satisfy the same linear constraints for k+1.*

Theorems 5.1 and 5.2 imply the following corollary.

Corollary 5.3 *The system of linear constraints (34 – 34) is feasible if and only if this system for $k = 1$, formulated as (27 – 30), is feasible.*

Theorem 4.1 and Corollary 5.3 yield the following corollary.

Corollary 5.4 *If the GPP is feasible then the system of linear constraints (34 – 37) is feasible for any $k \in \mathbb{N}$, with $\bar{u}_j = u_j$, $j = 1, \dots, n$.*

We conclude this section with the theorem that is an extension of Theorem 4.4 for $k > 1$. The proof of this theorem is similar to the proof of Theorem 4.4.

Theorem 5.5 *Let a vector z satisfy the constraints (34 – 37) for some $k = 2, 3, \dots$. Consider a randomized stationary policy π defined by (18),*

$$\pi(j|i_1, \dots, i_k) = \begin{cases} z_{i_1 \dots i_k j} / z_{i_1 \dots i_k}, & \text{if } z_i > 0; \\ arbitrary, & \text{otherwise}; \end{cases} \tag{41}$$

where

$$z_{i_1 \dots i_k} = \sum_{\substack{j=1 \\ j \neq i_k}}^{n} z_{i_1 \dots i_k j}, \qquad i \in I. \tag{42}$$

Define a Markov chain on the state space I_k with the transition probabilities $p_{\bar{i}\bar{j}}$ from $\bar{i} \in I_k$ to $\bar{j} \in I_k$,

$$p_{\bar{i}\bar{j}} = \begin{cases} \pi(j|i_1, \dots, i_k), & \text{if } \bar{i} = (i_1, i_2, \dots, i_k),\ \bar{j} = (i_2, \dots, i_k, j)\ ; \\ 0 & \text{otherwise}. \end{cases} \tag{43}$$

If this Markov chain is unichain (i.e. has one recurrent class) then for this chain the average time between two consecutive occurrences of $i = 1, \ldots, n$ *in each trajectory* $i_1, i_2, \ldots$ *is less than or equal to* $\bar{u}_i$.

At first glance, the results of this section are disappointing because there is no advantage to considering the constraints for $k > 1$. However, in Section 9 we shall reduce the state spaces I_k and develop a sequence of relaxations that are tighter for higher values of k.

In the next section, we provide the first of two exact algorithms presented in this paper. The second, which makes use of the results of this section, can be found in Section 9.

6 Dynamic programming approach

A feasible schedule, if it exists, can be defined as a function of a residual vector. Each coordinate $x(i)$ of a residual vector $x = (x(1), \ldots, x(n))$ is $u_i - t_i$, where t_i is the time elapsed since the last completion of job i. By setting $x_0 = (u_1, \ldots, u_n)$ in the beginning, any schedule defines a sequence of residual vectors. A schedule is feasible if and only if the corresponding sequence of residual vectors consists of vectors with nonnegative coordinates.

Let R^n be the n-dimensional Euclidean space. For $i = 1, \ldots, n$ and for $x \in R^n$ we define $g(x, i) = y \in R^n$, where $y = (y(1), \ldots y(n))$ with $y(k) = (1 - \delta_{i,k})(x(k) - \tau_k) + \delta_{i,k} u_k$, where δ is the Kronneker symbol.

Let a schedule $s = i_1, i_2, \ldots$ be given. Starting with x_0, the sequence of residual times x_k is defined by $x_k = g(x_{k-1}, i_k)$. Thus, indeed, any schedule $i_1, i_2, \ldots$ defines a sequence of residual vectors $x_0, x_1, \ldots$.

We provide the characterization of feasible schedules by using a deterministic version of Negative Dynamic Programming with a finite set of actions; see [24, 21, 10]. Here we construct a Negative Dynamic Program for the GPP.

Let $U^n = \times_{i=1}^n [0, u_i]$. We define the state set $X = U^n \cup \{\mathbf{g}\}$, where $\mathbf{g} \in R^n$ and $\mathbf{g} \notin U^n$. The state $\mathbf{g}$ is an absorbing state. If the underlying process generated by a schedule hits the state $\mathbf{g}$, this schedule is not feasible. We also define the action set $A = \{1, \ldots, n\}$ which is also an action set available at each state $x \in X$. If the state is $x \in U^n$, it means that the residual vector is $(x(1), \ldots, x(n))$.

The transition function f is defined by $f(x, a) = g(x, a)$. if $x \in U^n$ and $g(x, i) \in U^n$. Otherwise, $f(x, a) = \mathbf{g}$. In other words, if the decision to process job i is feasible, the residual time changes in a way explained above (all coordinates, except the coordinate i, decrease by the corresponding job duration and the coordinate i becomes u_i). If the decision is not feasible or an infeasible decision was used earlier, the system moves to or remains in $\mathbf{g}$.

The reward function is $r(x, a, x') = -1$, if $x \in U^n$ and $x' = \mathbf{g}$. Otherwise, $r(x, a, x') = 0$. In other words, the reward is 0 as long as the system stays in U^n, the reward is -1 when the system jumps to $\mathbf{g}$, and the reward is 0 again at $\mathbf{g}$.

Any schedule can be defined recursively by selecting a job i_1 as a function of $x_0 = (u_1, \ldots, u_n)$ and then by selecting job i_{k+1} as a function of the history $(i_1, \ldots, i_k)$, $k > 1$. Thus, any schedule defines a nonrandomized policy, say π, for the defined dynamic programming problem with the initial state x_0. Let $v(x, \pi)$ be the infinite-horizon expected total reward for a policy π and initial

state π. Let $V(x) = \sup_\pi v(x,\pi)$. Then the schedule is feasible if and only if $v(x_0,\pi) = 0$. Since $V(x) \le 0$ for all $x \in X$, $V(x_0) = 0$ if and only if there exists a stationary policy φ with $v(x_0,\varphi) = 0$; Theorem 6.2(iv). In our case, a stationary policy is a mapping of X to A. We observe that any stationary policy φ with $v(x_0,\varphi) = 0$ defines a feasible schedule $i_1, i_2, ...$, where $i_1 = \varphi(x_0)$ and $i_{k+1} = \varphi(x_{k+1}), x_{k+1} = f(x_k, i_{k+1})$, $k = 0, 1, \dots$. Thus, we have proved the following result.

Theorem 6.1 *(i) If a GPP is feasible then there is a function $\varphi : U^n \to \{1, \dots, n\}$ of the residual time vector, which defines a feasible schedule.*
(ii) A GPP is feasible if and only if $V(u_1, \dots, u_n) = 0$.

It is not clear whether, in general, any function φ defines a periodic schedule. However, Theorem 2.1 describes the procedure how to obtain a periodic solution from a general schedule. Thus, the next natural question is: how can one construct a stationary optimal policy?

This can be done by value iteration, which is described as follows. For any nonpositive Borel function G on X we define the optimality operator T by $TG(x) = \max_{a\in A} T^a G(x)$, $x \in X$, where T^a, $T^a G(x) = r(x,a,f(x,a)) + G(f(x,a))$, $a \in A$.

We consider the sequence V_k, k = 0,1, ... , of nonpositive functions on I defined with $V_0 \equiv 0$ and $V_k = TV_{k-1}$. The meaning of V is the maximal total reward that can be achieved over the infinite horizon and V_k is the maximal total reward that can be achieved over k steps. It is easy to see by standard monotonicity arguments that $V_{k+1}(i) \le V_k(i)$; Strauch [24, pp. 887].

Theorem 6.2 *(i) $V(x) = \lim_{k\to\infty} V_k(x)$ for all $x \in X$; [24, Theorem 9.1(a)].*
(ii) $V = TV$; [24, Theorem 8.2].
(iii) If $T^{\varphi(x)}V(x) = TV(x)$ for all $x \in X$ then the stationary policy φ is optimal; [24, pp. 887].
(iv) there exists a stationary optimal policy; [24, Theorem 9.1(b)].

We notice that in our particular case, all values of $V_k(x)$ are defined recursively and therefore, all values of $V(x) = \lim_{k\to\infty} V_k(x_0)$ are equal to either 0 or -1. The value iteration consists of the following major steps:

Value Iteration Algorithm

1. Calculate $V(x) = \lim_{k\to\infty} V_k(x)$, $x \in X$.
2. If $V(x_0) = -1$, where $x_0 = (u_1, \dots, u_n)$, then there is no feasible schedule.
3. Otherwise (when $V(x_0) = -1$) calculate a stationary policy φ such that $T^{\varphi(x)}V(x) = V(x)$ for all $x \in X$.

A function with only two values 0 and -1 is completely characterized by a subset of its domain on which this function equals 0. For functions V and V_k, $k \in \mathbb{N}$, we denote these subsets by U and U_k respectively; $U, U_k \subseteq U^n$.

For two vectors $s, t \in R^n$, we write $s \prec t$ if $t - s$ is a vector with nonnegative coordinates. The following lemma describes the properties of the sets U_k and U.

Lemma 6.3 *(i) $U_{k+1} \subseteq U_k$, $k \in \mathbb{N}$.*
(ii) $U = \cap_{k=0}^{\infty} U_k$.
(iii) If $x \in U$ (or $x \in U_k$) and $x \prec t$ then $t \in U$ ($t \in U_k$.)

(iv) The GPP has a feasible schedule if and only if $(u_1, \ldots, u_n) \in U$ *which is equivalent to* $U \neq \emptyset$.

Proof Statements (i) and (ii) follow from $V_{k+1}(x) \leq V_k(x)$, $x \in X$, and from Theorem 6.2(i). We observe that $U_0 = U^n$ and therefore (iii) holds for $k = 0$. Let (iii) hold for some k. Then $x \in U_{k+1}$ if and only if $f(x, i) \in U_k$ for some $i = 1, \ldots, n$. If $x \prec t$ then $f(x, i) \prec f(t, i)$. By induction, this implies $f(t, i) \in U_k$. Thus, $t \in U_{k+1}$. For the set U, we apply (ii). Statement (iv) follows from Theorem 6.1 and (iii). □

We remark that it is easier to deal with the sets U_k than with the functions V_k in a computer implementation of step 1 of the above algorithm. Let all the numbers u_i and τ_i, $i = 1, \ldots, n$, be rational as it always holds in computer implementations. Then it is easy to see that the value iteration algorithm converges after a finite number of steps.

Indeed, without loss of generality, we assume that all the numbers u_i and τ_i, $i = 1, \ldots, n$, are integers. If all coordinates of a vector x are integer then $f(x, i)$ is either equal to $\mathbf{g}$ or is a vector with integer coordinates. Therefore U^n can be defined as $\times_{i=1}^{n}\{0, \ldots, u_i\}$ in the definition of X. In this case, all sets U_k and U become finite and we have that $U_k = U_{k+1}$ for some k_0. Now, if we fix any $k \geq k_0$ a feasible schedule exists if and only if $U_k \neq \emptyset$. Thus, we obtained the following result.

Theorem 6.4 *If all the numbers* τ_i *and* u_i, $i = 1, \ldots, n$, *are rational then the value iteration algorithm converges in a finite number of steps.*

Let $u_i = u$, $i = 1, \ldots,$ and all the parameters be integer. Since the state space consists of u^n elements, this algorithm is exponential in n. This was to be expected, since according to Theorem 2.2, the GPP is NP-hard for $\rho = 1$. Hence, any exact algorithm that does not take into an account the value of ρ cannot be efficient. The fact that the size of the state space grows exponentially with the size of the problem in dynamic programming applications is known as Bellman's curse of dimensionality. We were able to implement the value iteration algorithm for $n \leq 6$. However, this is not good enough for the radar sensor management problem when in natural applications n can be up to 12; see [12].

The papers on the PP ($\tau_i \equiv 1$) [3,4,6,7,20,16,17,22] do not contain an example when the PP is not scheduleable when $\rho < 1$. By employing the value iteration algorithm, we have analyzed the following example.

Example 6.5 Consider the PP with $n = 5$ and $u_1 = 2$, $u_2 = u_3 = u_4 = 4$, and $u_5 = 44$. This problem is not feasible and $\rho = 43/45 < 1$. If we consider u_5 as a parameter, this problem is feasible when $u_5 > 443$ and it is not feasible when $u_5 \leq 443$. We observe that $\rho = 1$ when $u_5 = 14$ and $\rho < 1$ when $u_5 > 14$. Verification that this problem is not feasible without a computer is not trivial even for small $u_5 = 15$ or 16.

We conclude this section by commenting on the situation when the server is permitted to idle. It is obvious that if a feasible schedule that keeps a server idle exists then there exists a feasible schedule that never keeps it idle. Indeed, consider a feasible schedule that allows a server to idle at least once. If we remove all of the instances when the server idles, the schedule will remain feasible. Thus, there is no need to utilize the idling option. The remainder of this paper deals with heuristics for the GPP.

7 Simple heuristics

In this section, we consider three simple heuristics: round robin, due date, and critical ratio algorithms.

Due date algorithm. This heuristic always selects a job with the minimal residual time and uses an arbitrary tie breaker rule.

Critical ratio algorithm. The critical ratio is: $\frac{u_i+\tau_i}{x(i)+\tau_i}$, $i = 1, \ldots, n$, where $x(i)$ is the residual time of job i. The algorithm selects a job with the smallest critical ratio and uses an arbitrary tie breaker rule.

Round robin algorithm. The jobs are processed periodically and the period is a natural sequence $(1, 2, \ldots, n)$. Obviously, this schedule is feasible if and only if for all $i = 1, 2, \ldots, n$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} \tau_j \leq u_i. \tag{44}$$

Also, if (44) holds for all $i = 1, \ldots, n$, any permutation of $(1, \ldots, n)$ can be selected as a period.

We remark that

$$\tau_i \leq u_j, \qquad i, j = 1, \ldots, n, \quad i \neq j, \tag{45}$$

is a simple necessary condition for a problem to be feasible. Since this condition is easy, its verification can be added to each of the above heuristics. The verification of $\rho \leq 1$ can be also be added to any heuristic. In fact, the verification of these two conditions is incorporated into our implementations of all the heuristics considered in this paper.

8 Frequency based heuristic

As previously explained, the problem is infeasible if $\rho > 1$. If $\rho \leq 1$, the problem may be feasible and we can try to find a feasible solution. In particular, we set $z_i = (\tau_i + u_i)^{-1}$, $i = 1, \ldots, n$. Utilizing the results from Section 3, we then select the probability q_j to start job $j = 1, \ldots, n$ according to (19) and (18) such that $q_j \geq z_j(\sum_{i=1}^{n} \frac{1}{\tau_i+u_i})^{-1}$ and $\sum_{j=1}^{n} q_j = 1$. Of course, $q_j = z_j(\sum_{i=1}^{n} \frac{1}{\tau_i+u_i})^{-1}$ is the only choice when $\rho = 1$. If $\rho < 1$, it is not clear what is the best selection of q_j, $j = 1, \ldots, n$. For example, it is possible to conduct search in the space of all feasible vectors q. However, in our implementation of the algorithm we selected

$$q_j = \frac{\tau_j}{\tau_j + \alpha u_j}, \tag{46}$$

where $\alpha \leq 1$ is selected in a way that

$$\sum_{i=1}^{n} \frac{\tau_i}{\tau_i + \alpha u_i} = 1. \tag{47}$$

The intuitive justification for such a choice of q is the following. We are trying to find a schedule in which the time between the completion and restart of job i is always limited above by u_i. Instead of this, we are finding a policy in the relaxation for which the expectation of these times are limited by $\bar{u}_i$ where $\bar{u}_i \leq u_i$. In other words, we decrease all u_i to $\bar{u}_i$ with the hope to extract the appropriate sequence from the strategy for which the expectations of these times are decreased. If we decrease all u_i proportionally, we have that $\bar{u}_i = \alpha u_i$ and there is a unique α for which (47) holds.

After q_j, $j = 1, \ldots, n$, are selected, they define a randomized stationary policy for which the expected intervals between consequent instances of job $i = 1, \ldots, n$ are equal to $\bar{u}_i$. Since there is only one state, this randomized stationary policy is very simple: jobs are selected independently and job i is always selected with the probability q_i, $i = 1, \ldots, n$.

Our next step is to find a nonrandomized policy for which these expectations are equal to $\bar{u}_i$. This can be done by using the time sharing approaches of Ross [23] and Altman and Shwartz [2]. Suppose we have a finite schedule $seq_1, \ldots, seq_N$ of length N. Let N_i be the number of times the job $i = 1, \ldots, n$ has been executed, $N = N_1 + \ldots + N_n$. We select $seq_{N+1} = argmax\,\{q_i - (N_i + 1)/(N + 1)\}$ with an arbitrary tie breaker rule. We obtain the sequence $seq_1, seq_2, \ldots, seq_k \ldots$.

Our initial conjecture was that in most cases, the sequence becomes periodic. Computational results showed that this is not the case. We introduce a heuristic whereby we try to cut a feasible interval from the sequence $seq_k, seq_{k+1}, \ldots, seq_{k+j}$ that can be used as a period. In pseudo-code the algorithm is as follows:

Frequency based algorithm

1. Define sum to be the total number of iterations to be simulated.
2. If either condition (45) does not hold or $\sum_{i=1}^{n} \frac{\tau_i}{\tau_i + u_i} > 1$, the problem is infeasible and the algorithm ends.
3. If for each $i = 1, \ldots, n$, the inequality (44) holds, the round robin sequence $1, 2, \ldots, n$, repeated an infinite number of times, forms a feasible sequence and the algorithm ends.
4. Select $z_i \geq \frac{\tau_i}{\tau_i + u_i}$, $i = 1, \ldots, n$, such that $\sum_{i=1}^{n} z_i = 1$ (this can be implemented in a number of ways, e.g. $z_i = \tau_i(\tau_i + u_i)^{-1} / \sum_{j=1}^{n} \frac{\tau_j}{\tau_j + u_i}$ or $z_i = \frac{1}{\tau_i + \alpha u_i}$ with a constant $\alpha \leq 1$).
5. Let $q_i = \frac{z_i}{\sum_{j=1}^{n} z_j}$. $i = 1, \ldots, n$. The value of q_i is the probability that the next selection is job i, $i = 1, \ldots, n$.
6. - Initialize $N_1, \ldots, N_n$ and N to zero. The variable N_i counts the number of times job i was performed. The variable N counts the total number of jobs performed.
   - Define a one-dimensional array called seq. This array will store the sequence of jobs. The size of this array is the maximum length of the generated sequence (sum). In other words, sum is an upper limit for N.

- Define a one-dimensional array called d. This array will store the difference between the theoretical frequency of executing job $j = 1, \ldots, n$ and the actual frequency. The dimension of v is n. Initialize $d_i := 0, i = 1, \ldots, n$.

7. For ($N = 1$ to sum)
   Let $j = \text{argmax}_{i=1,\ldots,n} d_i$. If there is a tie, implement an arbitrary tie breaking rule, e.g select j as the job with the maximal number for which the tie takes place. Set:
   $N_j := N_j + 1$, $d_i := q_i - \frac{N_i}{N}$, $i = 1, \ldots, n$, $seq_N := j$.
   (The result is the array seq will now store the sequence of jobs performed to approximate the probabilities prescribed by q_i, $i = 1, \ldots, n$. The remainder of the algorithm searches the array seq for a feasible fragment.)
8. Initialize variables a and b. Let $a = M$ and $b = a$, where M is a "large" integer which is a small fraction of sum. The variable a is the beginning of the fragment being tested for feasibility and b is the end of this fragment.
9. Define a two-dimensional array called x. This array will be used to store the residual values at each iteration. This array will have dimensions n by sum. Set $x_i^a = u_i$, $i = 1, \ldots, n$.
10. Increase $b := b + 1$ and calculate

$$x_i^b := \begin{cases} u_i, & \text{if } i = seq_b; \\ x_i - \tau_i, & \text{otherwise.} \end{cases} \quad i = 1, \ldots, n.$$

11. If $x_i^b < 0$ at least for one $i = 1, \ldots, n$, go to step 16.
12. If $seq_a \neq seq_b$ then go to step 11.
13. If there is at least one job that has not been performed at least once between iteration a and b inclusive, then go to step 11.
14. Repeat the fragment $l = seq_a, seq_{a+1}, \ldots, seq_{b-1}, seq_b$ and check if the resulted fragment l, l satisfies the condition that the sum of all job durations between any two sequential completion and beginning of each job $i = 1, \ldots, n$ is not greater than u_i. If yes then $l, l, l, \ldots$ is a feasible schedule (see the proof of Theorem 2.1 where the similar idea was used).
15. Increase $a = a + 1$ and set $b = a$.
16. If $a > sum - n$, the algorithm stops without finding a feasible schedule. Otherwise go to step 10.

9 Improved algorithms

The sets I_k introduced in Section 5 consist of the sequences of k jobs $i_1, \ldots, i_k$ such that any two consecutive jobs are different. However, if k is large, some of these sequences cannot be continued to form periodic schedules. For example, if a total length of all jobs in the finite sequence $\bar{i} \in I_k$ is longer than u_j and the job j is absent in this sequence, the sequence $\bar{i}$ cannot be continued as a feasible schedule for the GPP.

First, we shall introduce the subsets $J_k \subseteq I_k$ such that if $\bar{i} \in J_k$ we cannot say immediately that the sequence $\bar{i}$ cannot be continued as a feasible schedule for the GPP. By replacing the set of possible indices I_k in (34 – 37) with J_k we shall obtain tighter constraints that will possess the properties that the constraints for $k + 1$ are tighter that those for k.

Let us consider the following conditions for sequences $i_1, \ldots, i_k$ of integers between 1 and n:

(i) $i_j \neq i_{j+1}$, $j = 1, \ldots, k-1$;
(ii) if $\sum_{\ell=1}^{k} i_\ell > u_j$, $j = 1, \ldots, n$, then $i_\ell = j$ for some $\ell = 1, \ldots, k$;
(iii) For each job $j = 1, \ldots, n$ such that $i_\ell = j$ for some $\ell = 1, \ldots, k$ the following conditions hold:

$$\sum_{m=1}^{\ell_*(j)-1} \tau_m \leq u_j, \qquad \text{where } \ell_*(j) = \min\{\ell = 1, \ldots, k : i_\ell = j\}, \tag{48}$$

$$\sum_{m=\ell^*(j)+1}^{k} \tau_m \leq u_j, \qquad \text{where } \ell^*(j) = \max\{\ell = 1, \ldots, k : i_\ell = j\}, \tag{49}$$

and if there is more than one instance of the job j then for any two consequent instances ℓ and ℓ' of j, where $1 \leq \ell < \ell' \leq k$,

$$\sum_{m=\ell+1}^{\ell'-1} \tau_m \leq u_j. \tag{50}$$

We say that a finite sequence $i_1, \ldots, i_k$ is acceptable if: (a) it satisfies properties (i)–(iii), and (b) there exists $j = 1, \ldots, n$ such that the finite sequence $s = i_1, \ldots, i_k, j$ satisfies these conditions as well for $k := k+1$. A finite sequence $i_1, \ldots, i_k$ is called complete if it is acceptable and for any $j = 1, \ldots, n$ there exists $\ell = 1, \ldots, k$ such that $i_\ell = j$. Of course, any complete sequence contains at least n elements. If we double a finite complete sequence s and the resulting finite sequence s, s is acceptable then the periodic schedule $s, s, \ldots$ is feasible; see the proof of Theorem 2.1.

Let J_k be the set of all acceptable sequences with k elements. For a sequence $i_1, \ldots, i_k \in J_k$ and for $j = 1, \ldots$ we denote by $\ell^*(j)$ the maximal $m = 1, \ldots, k$ such that $i_m = j$. If at least one of the elements of this sequence is equal to j, the value of $\ell^*(j)$ was defined in (49). Otherwise, we set $\ell^*(j) = 0$. We observe that

$$\begin{aligned} J_{k+1} = \{i_1, \ldots, i_k, i \,|\, i_1, \ldots, i_k \in J_k,\ i \neq i_k, i = 1, \ldots, n, \text{ and there exists} \\ j = 1, \ldots, n \text{ for which } \tau_i + \textstyle\sum_{m=\ell^*(j)+1}^{k} \tau_m \leq u_j \text{ and } \tau_i + \tau_j \\ + \textstyle\sum_{m=\ell^*(j')+1}^{k} \tau_m \leq u_{j'} \\ \text{for any } j' = 1, \ldots, n \text{ such that } j' \neq i \text{ and } j' \neq j\}. \end{aligned} \tag{51}$$

Expression (51) defines a recursion to construct J_{k+1} from J_k. Of course, $J_1 = I_1 = \{1, \ldots, n\}$.

Similar to (19), if s is a periodic schedule, the corresponding value of $z_{\bar{i}j}$, where $\bar{i} \in I_k$ for some k and $j = 1, \ldots, n$, is the proportion of time the schedule of jobs $\bar{i}$, j is performed by the sequence s. Therefore, if $\bar{i} \in I_k \setminus J_k$, this sequence $\bar{i}$, j cannot be a part of a feasible schedule and the corresponding variables $z_{\bar{i},j} = 0$. Therefore, if the GPP is feasible, the constraints (34–37) are feasible even if the additional

constraints $z_{\bar{i},j} = 0$ when $\bar{i} \in I_k \setminus J_k$ are imposed. This leads us to a tighter version of the constraints (34 – 37) with the set I_k replaced with the smaller set J_k:

$$\sum_{\substack{j=1\\ j\neq i_k}}^{n} z_{\bar{i}j} = \sum_{\substack{j=1\\ j\neq i_1\\ (j,i_1,\dots,i_{k-1})\in J_k}}^{n} z_{j\bar{i}}, \quad \bar{i} = (i_1,\dots,i_k) \in J_k, \tag{52}$$

$$\sum_{\bar{i}=(i_1,i_2,\dots,i_k)\in J_k} \sum_{\substack{j=1\\ j\neq i_k}}^{n} z_{\bar{i}j}\tau_j = 1, \tag{53}$$

$$\sum_{\substack{\bar{i}=(i_1,i_2,\dots,i_k)\in J_k\\ i_k\neq j}} z_{\bar{i}j} \geq \frac{1}{\bar{u}_j + \tau_j}, \qquad j = 1,\dots,n, \tag{54}$$

$$z_{\bar{i}j} \geq 0, \qquad \bar{i} \in J_k,\ j = 1,\dots,n,\ j \neq i_k. \tag{55}$$

We observe that if $i_1,\dots,i_k \in J_k$ for some $k = 2,3,\dots$ then $i_1,\dots,i_{k-1} \in J_{k-1}$. Therefore, Theorem 5.1 implies the following result.

Theorem 9.1 *If z is a solution of the linear constraints (52 – 55) for some $k \in \mathbb{N}$ then formula (38) defines a solution of the same linear constraints for $k-1$.*

We remark that a statement similar to Theorem 5.2 does not hold because it is possible that the sequences $i_1,\dots,i_k$ and $i_2,\dots,i_{k+1}$ belong to J_k but the sequence $i_1,\dots,i_{k+1}$ does not belong to J_{k+1}. For example, consider the PP with $n = 3$ and $u_i = 2, i = 1,2,3$. Then the sequences $(1,2)$ and $(2,1)$ are acceptable for $k = 2$ but the sequence $(1,2,1)$ is not acceptable. Therefore the constraints (52 – 55) are tighter for higher values of k.

Next, we present the frequency based algorithms for $k \in \mathbb{N}$

Frequency Based Algorithms FBk

1. Define *sum* to be the total number of iterations to be simulated.
2. If either condition (45) does not hold or the system of constraints (52 – 55) is infeasible, the GPP is infeasible and the algorithm ends.
3. If for each $i = 1,\dots,n$ the inequality (44) holds, the round robin sequence $1,2,\dots,n$ repeated an infinite number of times, forms a feasible sequence and the algorithm ends.
4. Select the vector $(\bar{u}_1,\dots,\bar{u}_n)$. This can be done iteratively by using various search techniques. Similar to FB, we have implemented an approach when we select $\bar{u}_i = \alpha u_i$ with $\alpha \leq 1$ being the minimal constant such that the system of constraints (52 – 55) is feasible. The value of α can be found by bisection. Once the vector $(\bar{u}_1,\dots,\bar{u}_n)$ is defined, compute a solution $z_{\bar{i}j}$ of the system of linear constraints (52 – 55). Set $z_{\bar{i}} = \sum_{\substack{j=1\\ j\neq i_k}}^{n} z_{\bar{i}j}$ for all $\bar{i} = (i_1 \dots i_k) \in J_k$.

   Let $\hat{J}_k = \{\bar{i} \in J_k |\ z_{\bar{i}} > 0\}$.
5. Set $q_{\bar{i}j} = z_{\bar{i}j}/z_{\bar{i}}$ for all $\bar{i} \in J_k$ and for all $j = 1,\dots,n$, $j \neq i_k$. If the transition probabilities $q_{\bar{i}j}$ define a Markov chain with more than one recurrent class [18, Algorithm 1], a feasible schedule is not found.

6. • Initialize $N_{\bar{i}j}$, $N_{\bar{i}}$, and N to zero, where $\bar{i} = (i_1 \dots i_k) \in \hat{J}_k$, $j = 1, \dots, n$, and $j \neq i_k$. The variable $N_{\bar{i}}$ and $N_{\bar{i}j}$ respectively count the number of times each of the strings $\bar{i}$ and $(i_1 \dots i_k j)$ were observed. The variable N counts the total number of jobs performed.
   • Define a one-dimensional array called seq. This array will store the sequence of jobs. The size of this array is the maximum length of the generated sequence (sum). In other words, sum is an upper limit for N. Initialize $N = k$ and set $seq = \bar{i}$ for some $\bar{i} = (i_1 \dots i_k) \in \hat{J}_k$. Let M be the number of elements in $\hat{J}_k$.
   • Define a two-dimensional array called d. This array will store the difference between the theoretical frequency of executing job $j = 1, \dots, n$ at states $\bar{i} \in \hat{J}_k$ and the actual frequency. The dimensions of v are M by n. Initialize all elements of d equal to 0.
   • Define a two-dimensional array called r. This array will be used to store the residual values at each iteration. This array will have dimensions n by sum.
   • Select any $\bar{i} \in \hat{J}_k$. Let $\bar{i} = (i_1 \dots i_k)$.
7. For ($N = 1$ to sum)
   Let $j = \text{argmax}_{\{j=1,\dots,n;\, j \neq i_k\}} d_{\bar{i}j}$. If there is a tie, implement an arbitrary tie breaking rule, e.g select j as the job with the maximal number for which the tie takes place. Set: $N_{\bar{i}j} := N_{\bar{i}j} + 1$, $N_{\bar{i}} := N_{\bar{i}} + 1$, $\bar{i} := (i_2 \dots i_k j)$, $seq_N := j$, $d_{\bar{i}j} := q_{\bar{i}j} - \frac{N_{\bar{i}j}}{N_{\bar{i}}}$, $j = 1, \dots, n$. The remainder of the algorithm searches the array seq for a feasible fragment as is in the frequency based algorithm, Steps 8 – 16, described in Section 8.

We remark that, though the transition probabilities calculated at step 5 theoretically can define a Markov chain with multiple recurrent classes, we have never experienced this in the thousands of calculations performed.

Using the heuristics FBk we can formulate the second exact algorithm as follows (the first exact algorithm is the value iteration algorithm from Section 6).

1. Apply the frequency based algorithm from Section 8. Stop if a feasible schedule is found or the problem is detected to be infeasible.
2. Set $k = 1$, $J_k = \{1, \dots, n\}$, and apply the frequency based algorithm FBk from Section 9. Stop if a feasible schedule is found or the problem is detected to be infeasible.
3. Increase the value of k by 1. Calculate the set J_k by using the recursion (51). If $J_k = \emptyset$ then the problem is infeasible. Otherwise, let C_k be the subset of the complete sequences of J_k. Sequentially double each $\bar{i}$ sequence in C_k. If the sequence $\bar{i}, i$ satisfies the condition that the time between all consecutive instances of job $j = 1, \dots, n$ is not greater than u_j, then $\bar{i}, \bar{i}, \bar{i}, \dots$ is a feasible periodic schedule. If the feasible schedule is not found, go to Step 2.

Since, according to Theorem 2.2, the GPP is *NP*-hard, the latter algorithm is not efficient. Its convergence follows from the following arguments. Leaving apart the possibility that the heuristics at Steps 1 or 2 solve the problem, consider the following two possibilities: (i) the problem is feasible, and (ii) the problem is not feasible. If the problem is feasible and its shortest (in terms of the number of jobs in it) period consists of k jobs, it will be detected at Step 3 when the appropriate element of C_k is doubled unless a feasible schedule has been found earlier. If the

problem is infeasible, it is easy to see from the proof of Theorem 2.1 that the set J_k will become empty for large k.

We remark than among all FBk algorithms with $k > 1$, the case $k = 1$ appears to be the most interesting in terms of its applicability. The reason is that FBk algorithms require solutions of linear constraints with up to $n(n-1)^k$ variables. However, as Corollary 5.3 implies, the constraints (52 – 55) become tighter when the set J_k is smaller than the set I_k. For reasonable n this happens with relatively large values of k. For example, typically $I_2 = J_2$ and FB2 probably should not significantly outperform FB1. This is the reason we implemented FBk heuristics for $k = 0$ and $k = 1$. Our simulations indicated that FB1 solves roughly 5% cases more than FB ($k = 0$).

10 Simulation results

We provide simulation results for one of the two data sets considered in [12] for the radar sensor management problem. We conducted a simulation of the job durations of 0.0587, 0.175, 0.223, and 0.299 that were randomly and independently assigned to each of the jobs with the probability $\frac{1}{4}$. Revisit times of 1, 3, or 4 were also randomly and independently assigned to each of the jobs with the probability $\frac{1}{3}$. Each simulation of 1000 cases has been conducted for $n = 4, 5, \ldots, 12$. The RR, CR, FB, and FB1 were applied to the simulated data for comparison. It was found that the frequency based FB algorithm outperformed significantly both the critical ratio and round robin algorithms. The frequency based algorithm for $k = 1$, FB1, demonstrated even better results than FB. It found a feasible schedule in a higher percentage of cases. In addition, FB1 detected more cases when the problem is infeasible. For the RR, CR, and FB heuristics we used the condition $\rho > 1$ to detect the cases that cannot be feasible. The FB1 heuristic checks the feasibility of the constraints (27 – 30) which is a more powerful criterion. The results of this simulation are shown in table 9.1 below. According to these results, FB1 is better than FB and FB outperforms CR. Of course, RR is the most primitive heuristic.

Table 9.1 Simulation results

#jobs	RR	CR	FB	FB1	RR	CR	FB	FB1	FB	FB1
4	100	100	100	100	92.4	100	100	100	0	0
5	92.4	100	100	100	92.4	100	100	100	0	0
6	55.3	97.2	100	100	55.3	97.2	100	100	0	0
7	22.5	89.1	98.4	99.1	22.5	89.1	98.4	99.3	0	.2
8	10.3	76.7	92.7	95.4	12.9	79.3	95.3	98.6	2.6	3.2
9	3.4	53.6	79.5	83.5	8.8	59.0	84.9	91.0	5.4	7.5
10	2.7	27.3	62.3	64.5	16.2	40.8	75.8	81.2	13.5	16.7
11	2.3	18.2	41.6	44.6	32.4	48.3	71.7	79.8	30.1	35.2
12	2	8.1	22.3	25.3	47.8	53.9	68.1	78.5	45.8	53.2

Percentage of scheduled cases using (RR), (CR), (FB), and (FB1).

*Percentage of cases where a solution could be found using (RR), (CR), (FB), and (FB1).

Percentage of cases that were found to be infeasible using (FB) and (FB1).

*Includes the case when infeasible cases are detected.

11 Conclusions and open questions

This paper studied a Generalized Pinwheel Problem (GPP) which is to find a feasible infinite sequence of jobs from a set of n recurring jobs with durations τ_i, $i = 1, \ldots, n$. Each time job i is completed, it should be restarted within u_i units of time. According to Theorem 2.2, his problem is NP-hard, at least when the density ρ equals 1. When all $\tau_i = 1$, this problem becomes the Pinwheel Problem (PP) studied in [3,4,6,7,20,16,17,22].

The GPP can be applied to many problems that involve repetitive executions of tasks with non-uniform durations and deadlines. The examples of applications include: the radar sensor management, machine refuelling, database support, and inspection scheduling problems.

We have shown that a feasible GPP has a feasible periodic schedule. In this paper we established a link between relaxed versions of the GPP and constrained undiscounted Semi-Markov Decision Process. By using this link, we developed relaxations of the GPP and formulated the so-called frequency based algorithms that outperform other known heuristics. These relaxations become tighter when the parameter k, that is relevant to the memory used by the algorithms, increases. The simplest relaxation for $k = 0$ implies that $\rho \leq 1$ is a necessary condition for a problem to be feasible. Other relaxations, for $k = 1, 2 \ldots$ provide more accurate necessary conditions that may detect that the problem is not feasible in some cases when $\rho \leq 1$. We also provided computational results that illustrate the efficiency of the introduced algorithms.

In general, a feasible schedule, if it exists, can be presented as a function of a vector of residual times. We have developed a value iteration algorithm based on Dynamic Programming. Though this algorithm converges, it is computationally intractable for large n, which is consistent with NP-complexity of the GPP. The implementation of this algorithm allowed us to construct the example of an infeasible PP with $\rho < 1$. We also showed in this paper that, unlike the PP, the GPP can be infeasible when ρ is small.

There are several interesting open questions for the PP and for the GPP. When Holte et al [16] introduced the PP, they stated without the proof that the PP is NP-hard when $\rho = 1$. However their detailed paper [17] did not mention this fact. Chan and Chin [7] provided a polynomial algorithm that always constructs a schedule for a PP with $\rho \leq 0.7$. Nothing is known about the complexity of PP when $\rho > 0.7$. It would be interesting to have the boundaries on ρ when the PP is always feasible, when it is polynomial, and when it is NP-hard. The interesting question is whether the GPP is NP-hard for $\rho < 1$ and whether it is possible to develop a polynomial algorithm for the GPP when $\rho \leq \alpha$ for some $\alpha < 1$. Of course, an interesting question is whether it is possible to develop better heuristics than the frequency based algorithms.

Acknowledgements This research was partially supported by NSF grant DMI-0300121 and by a research grant from NYSTAR (New York State Office of Science, Technology and Academic Research). The authors are grateful to Michael Bender, Daniel Huang, Theodore Koutsoudis, and Joel Bernstein for valuable discussions relevant to this paper.

References

1. Avrachenkov, K.E., Filar, J., Haviv, M.: Singular perturbations of Markov chains and decision processes. In: Feinberg, E.A., Adam Shwartz, (eds.) Handbook of Markov Decision Processes, Kluwer, Boston, pp 113–150, 2002
2. Altman, E., Shwartz, A.: Time-sharing policies for controlled Markov chains. Operations Research **41**, 1116–1124 (1993)
3. Baruah, S.K., Lin, S.-S.: Pfair scheduling of generalized pinwheel task systems. IEEE Transactions on Computers **47**(7), 812–816 (1998)
4. Baruah, S.K., Howell, R.R., Rosier, L.E.: Feasibility problems for recurring tasks on one processor. Theoretical Computer Science **118**, 3–20 (1993)
5. Borkar, V.S., Ejov, V., Filar, J.A.: Directed graphs, Hamiltonicity and doubly stochastic matrices. Random Structures and Algorithms **25**, 376–395 (2004)
6. Chan, M.Y., Chin, F.: General schedulers for the pinwheel problem based on double integer reduction. IEEE Transactions on Computers **41**, 755–768 (1992)
7. Chan, M.Y., Chin, F.: Schedulers for larger classes of pinwheel instances. Algorithmica **9**, 425–462 (1993)
8. Ejov, V., Filar, J.A., Nguyen., M.: Hamiltonian cycles and singularly perturbed Markov chains. Math. Oper. Res. **29**, 114–131 (2004)
9. Feinberg, E.A.: Constrained semi-Markov decision process with average rewards. ZOR Math. Meth. Oper. Res. **39**, 257–288 (1994)
10. Feinberg, E.A.: Total reward criteria. In: Feinberg E.A., Adam Shwartz (eds.) Handbook of Markov Decision Processes, Kluwer, Boston, 175–207, 2002
11. Feinberg, E.A.: Constrained discounted Markov decision processes and Hamiltonian cycles. Math. Oper. Res. **25**, 130–140 (2000)
12. Feinberg, E.A., Bender, M.A., Curry, M.T., Huang, D., Koutsoudis, T., Bernstein, J.L.: Sensor resource management for an airborne early warning radar. Proceedings of the SPIE-The International Society for Optical Engineering, Signal and Data Processing of Small Targets **4728**, 145–156 (2002)
13. Filar, J.A., Guo, X.: Linear program for communicating MDPs with multiple constraints. In: Z. Hou et al. (eds.) Markov Processes and Controlled Markov Chains, Kluwer Academic Publishers, Netherlands, pp 245–254, 2002
14. Filar, J.A., Krass, D.: Hamiltonioan cycles and Markov chains. Math. Oper. Res. **25**, 223–237 (1994)
15. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, San Francisco, 1979
16. Holte, R., Rosier, L., Tulchinsky, I., Varvel, D.: The Pinwheel: A realtime scheduling problem. Proceedings of the Annual Hawaii International Conference on System Sciences, 693–702 (1989)
17. Holte, R., Rosier, L., Tulchinsky, I., Varvel, D.: Pinwheel scheduling with two distinct numbers. Theoretical Computer Science **100**, 105–135 (1992)
18. Kallenberg, L.C.M.: Classification problems in MDPs. In: Hou, Z., Filar, J.A., Chen, A. (eds) Markov Processes and Controlled Markov Chains, Kluwer, Dordrecht, pp 151–165, 2002
19. Leung, J.Y.T.: A new algorithm for scheduling periodioc, real time tasks. Algorithmica **4**, 209–219 (1989)
20. Lin, S.S., Lin, K.J.: A pinwheel scheduler for three distinct numbers with a tight schedulability bound. Algorithmica **19**, 411–426 (1997)
21. Puterman, M.I.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, New York, 1994
22. Romer, T.H., Rosier, L.E.: An algorithm reminiscent of Euclidean-gcd for computing a function related to pinwheel scheduling. Algorithmica **17**, 1–10 (1997)
23. Ross, K.W.: Randomized and past-dependent policies for Markov decision processes with multiple constraints. Operations Research **37**, 474–477 (1989)
24. Strauch, R.E.: Negative dynamic programming. The Annals of Mathematical Statistics **37**, 871–890 (1966)