Analysis of multi-stage open shop processing systems*

Christian E.J. Eggermont† Alexander Schrijver‡
Gerhard J. Woeginger§

Abstract

We study algorithmic problems in multi-stage open shop processing systems that are centered around reachability and deadlock detection questions.

We characterize safe and unsafe system states. We show that it is easy to recognize system states that can be reached from the initial state (where the system is empty), but that in general it is hard to decide whether one given system state is reachable from another given system state. We show that the problem of identifying reachable deadlock states is hard in general open shop systems, but is easy in the special case where no job needs processing on more than two machines (by linear programming and matching theory), and in the special case where all machines have capacity one (by graph-theoretic arguments).

Keywords: Scheduling; resource allocation; deadlock; computational complexity.

*This research has been supported by the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403; by DIAMANT (an NWO mathematics cluster); by the Future and Emerging Technologies unit of the European Community (IST priority), under contract no. FP6-021235-2 (project ARRIVAL); by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

†c.e.j.eggermont@tue.nl. Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands

‡lex@cwi.nl. CWI, 1098 XG Amsterdam, and University of Amsterdam, Netherlands

§gwoegi@win.tue.nl. Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands

1 Introduction

We consider a *multi-stage open shop* processing system with n jobs $J_1, \ldots, J_n$ and m machines $M_1, \ldots, M_m$. Every job J_j ($j = 1, \ldots, n$) requests processing on a certain subset $\mathcal{M}(J_j)$ of the machines; the ordering in which job J_j passes through the machines in $\mathcal{M}(J_j)$ is irrelevant and can be chosen arbitrarily by the scheduler. Every machine M_i ($i = 1, \ldots, m$) has a corresponding *capacity* $\mathrm{cap}(M_i)$, which means that at any moment in time it can simultaneously hold and process up to $\mathrm{cap}(M_i)$ jobs. For more information on multi-stage scheduling systems, the reader is referred to the survey [6].

In this article, we are mainly interested in the performance of *real-time* multi-stage systems, where the processing time $p_{j,i}$ of job J_j on machine M_i is a priori unknown and hard to predict. The Central Control (the scheduling policy) of the system learns the processing time $p_{j,i}$ only when the processing of job J_j on machine M_i is completed. The various jobs move through the system in an unsynchronized fashion. Here is the standard behavior of a job in such a system:

1. In the beginning the job is asleep and is waiting outside the system. For technical reasons, we assume that the job occupies an artificial machine M_0 of unbounded capacity.

2. After a finite amount of time the job wakes up, and starts looking for an available machine M on which it still needs processing. If the job detects such a machine M, it requests permission from the Central Control to move to machine M. If no such machine is available or if the Central Control denies permission, the job falls asleep again (and returns to the beginning of Step 2).

3. If the job receives permission to move, it releases its current machine and starts processing on the new machine M. While the job is being processed and while the job is asleep, it continuously occupies machine M (and blocks one of the $\mathrm{cap}(M)$ available places on M). When the processing of the job on machine M is completed and in case the job still needs processing on another machine, it returns to Step 2.

4. As soon as the processing of the job on all relevant machines is completed, the job informs the Central Control that it is leaving the system. We assume that the job then moves to an artificial final machine M_{m+1} (with unbounded capacity), and disappears.

The described system behavior typically occurs in robotic cells and flexible manufacturing systems. The high level goal of the Central Control is to arrive at the situation where all the jobs have been completed and left the system. Other goals are of course to reach a high system throughput, and to avoid unnecessary waiting times of the jobs. However special care has to be taken to prevent the system from reaching situations of the following type:

Example 1.1 *Consider an open shop system with three machines M_1, M_2, M_3 of capacity 1. There are three jobs that each require processing on all three machines. Suppose that the* Central Control *behaves as follows:*

The first job requests permission to move to machine M_1. Permission granted.

The second job requests permission to move to machine M_2. Permission granted.

The third job requests permission to move to machine M_3. Permission granted.

Once the three jobs have completed their processing on theses machines, they keep blocking their machines and simultaneously keep waiting for the other machines to become idle. The processing never terminates.

Example 1.1 illustrates a so-called *deadlock*, that is, a situation in which the system gets stuck and comes to a halt since no further processing is possible: Every job in the system is waiting for resources that are blocked by other jobs that are also waiting in the system. Resolving a deadlock is usually expensive (with respect to time, energy, and resources), and harmfully diminishes the system performance. In robotic cells resolving a deadlock typically requires human interaction. The scientific literature on deadlocks is vast, and touches many different areas like flexible manufacturing, automated production, operating systems, Petri nets, network routing, etc.

The literature distinguishes two basic types of system states (see for instance Coffman, Elphick & Shoshani [2], Gold [5], or Banaszak & Krogh [1]). A state is called *safe*, if there is at least one possible way of completing all jobs. A state is called *unsafe*, if every possible continuation eventually will get stuck in a deadlock. An example for a safe state is the initial situation where all jobs are outside the system (note that the jobs could move sequentially through the system and complete). Another example for a safe state is the final situation where all jobs have been completed. An example for an unsafe state are the deadlock states.

Summary of considered problems and derived results

In this article we study the behavior of safe and unsafe states in open shop scheduling systems. In particular, we investigate the computational complexity of the four algorithmic questions described in the following paragraphs. First, if one wants to have a smoothly running system, then it is essential to distinguish the safe from the unsafe system states:

Problem: Safe State Recognition

Instance: An open shop scheduling system. A system state s.

Question: Is state s safe?

Section 3 provides a simple characterization of unsafe states, which leads to a (straightforward) polynomial time algorithm for telling safe states from unsafe states. Similar characterizations have already been given a decade ago in the work of Sulistyono & Lawley [9] and Xing, Lin & Hu [10]. Our new argument is extremely short and simple.

One of the most basic problems in analyzing a system consists in characterizing those system states that can be reached while the shop is running.

Problem: Reachable State Recognition

Instance: An open shop scheduling system. A system state s.

Question: Can the system reach state s when starting from the initial situation where all machines are still empty?

In Section 4 we derive a polynomial time algorithm for recognizing reachable system states. The main idea is to reverse the time axis, and to make the system run backward. Then reachable states in the original system translate into safe states in the reversed system, and the results from Section 3 can be applied.

Hence recognizing states that are reachable from the initial situation is easy. What about recognizing states that are reachable from some other given state?

Problem: State-to-State Reachability

Instance: An open shop scheduling system. Two system states s and t.

Question: Can the system reach state t when starting from state s?

Surprisingly, there is a strong and sudden jump in the computational complexity of the reachability problem: Section 5 provides an NP-hardness proof for problem State-to-State Reachability.

Another fundamental question is whether an open shop system can ever fall into a deadlock. In case it cannot, then there are no reachable unsafe states and the Central Control may permit all moves right away and without analyzing them; in other words the system is fool-proof and will run smoothly without supervision.

Problem: Reachable Deadlock

Instance: An open shop scheduling system.

Question: Can the system ever reach a deadlock state when starting from the initial situation?

Section 6 proves problem Reachable Deadlock to be NP-hard, even for the highly restricted special case where the capacity of each machine is at most three and where each job requires processing on at most four machines. In Sections 7 and 8 we exhibit two special cases for which this problem is solvable in polynomial time: The special case where every job needs processing on at most two machines is settled by a linear programming formulation and techniques from matching theory. The special case where every machine has capacity one is solved by analyzing cycles in certain edge-colored graphs.

2 Basic definitions

A *state* of an open shop scheduling system is a snapshot describing a situation that might potentially occur while the system is running. A state s specifies for every job J_j

- the machine $M^s(J_j)$ on which this job is currently waiting or currently being processed,

- and the set $\mathcal{M}^s(J_j) \subseteq \mathcal{M}(J_j) - \{M^s(J_j)\}$ of machines on which the job still needs future processing.

The machines $M^s(J_j)$ implicitly determine

- the set $\mathcal{J}^s(M_i) \subseteq \{J_1, \ldots, J_n\}$ of jobs currently handled by machine M_i.

The *initial state* 0 is the state where all jobs are still waiting for their first processing; in other words in the initial state all jobs J_j satisfy $M^0(J_j) = M_0$ and $\mathcal{M}^0(J_j) = \mathcal{M}(J_j)$. The *final state* f is the state where all jobs have been completed; in other words in the final state all jobs J_j satisfy $M^f(J_j) = M_{m+1}$ and $\mathcal{M}^f(J_j) = \emptyset$.

A state t is called a *successor* of a state s, if it results from s by moving a single job J_j from its current machine $M^s(J_j)$ to some new machine in set $\mathcal{M}^s(J_j)$, or by moving a job J_j with $\mathcal{M}^s(J_j) = \emptyset$ from its current machine to M_{m+1}. In this case we will also say that the system *moves* from s to t. This successor relation is denoted $s \to t$. A state t is said to be *reachable* from state s, if there exists a finite sequence $s = s_0, s_1, \ldots, s_k = t$ of states (with $k \geq 0$) such that $s_{i-1} \to s_i$ holds for $i = 1, \ldots, k$. A state s is called *reachable*, if it is reachable from the initial state 0.

Proposition 2.1 *Any reachable state s can be reached from the initial state through a sequence of at most $n + \sum_{i=1}^{n} |\mathcal{M}(J_j)|$ moves.*

A state is called *safe*, if the final state f is reachable from it; otherwise the state is called *unsafe*. A state is a *deadlock*, if it has no successor states and if it is not the final state f.

3 Analysis of unsafe states

Unsafe states in open shop systems are fairly well-understood, and the literature contains several characterizations for them; see for instance Sulistyono & Lawley [9], Xing, Lin & Hu [10], and Lawley [7]. In this section we provide yet another analysis of unsafe states, which is shorter and (as we think) simpler than the previously published arguments.

A machine M is called *full* in state s, if it is handling exactly $\mathrm{cap}(M)$ jobs. A non-empty subset $\mathcal{B}$ of the machines is called *blocking* for state s,

- if every machine in $\mathcal{B}$ is full, and
- if every job J_j that occupies some machine in $\mathcal{B}$ satisfies $\emptyset \neq \mathcal{M}^s(J_j) \subseteq \mathcal{B}$.

Here is a simple procedure that determines whether a given machine M_i is part of a blocking set in state s: Let $\mathcal{B}_0 = \{M_i\}$. For $k \geq 1$ let $\mathcal{J}_k$ be the union of all job sets $\mathcal{J}^s(M)$ with $M \in \mathcal{B}_{k-1}$, and let $\mathcal{B}_k$ be the union of all machine sets $\mathcal{M}^s(J)$ with $J \in \mathcal{J}_k$. Clearly $\mathcal{B}_0 \subseteq \mathcal{B}_1 \subseteq \cdots \subseteq \mathcal{B}_{m-1} = \mathcal{B}_m$. Furthermore machine M_i belongs to a blocking set, if and only if $\mathcal{B}_m$ is a blocking set, if and only if all machines in $\mathcal{B}_m$ are full. In case $\mathcal{B}_m$ is a blocking set, we denote it by $\mathcal{B}^s_{\min}(M_i)$ and call it the *canonical* blocking set for machine M_i in state s. The canonical blocking set is the smallest blocking set containing M_i:

Lemma 3.1 *If machine M_i belongs to a blocking set $\mathcal{B}$ in state s, then $\mathcal{B}^s_{\min}(M_i) \subseteq \mathcal{B}$.*

The machines in a blocking set $\mathcal{B}$ all operate at full capacity on jobs that in the future only want to move to other machines in $\mathcal{B}$. Since these jobs are permanently blocked from moving, the state s must eventually lead to a deadlock and hence is unsafe. The following theorem shows that actually *every* deadlock is caused by such blocking sets.

Theorem 3.2 *A state s is unsafe if and only if it has a blocking set of machines.*

Proof: The if-statement is obvious. For the only-if-statement, we classify the unsafe states with respect to their distances to deadlock states. The set $\mathcal{U}_0$ contains the deadlock states. For $d \geq 1$, set $\mathcal{U}_d$ contains all states whose successor states are all contained in $\mathcal{U}_{d-1}$. Note that $\mathcal{U}_{d-1} \subseteq \mathcal{U}_d$, and note that every unsafe state occurs in some $\mathcal{U}_d$. We prove by induction on d that every state in $\mathcal{U}_d$ has a blocking set of machines. For $d = 0$ this is trivial.

In the inductive step, assume for the sake of contradiction that some state $s \in \mathcal{U}_d$ is unsafe but does not contain any blocking set. Since every move from s leads to a state in $\mathcal{U}_{d-1}$, all successor states of s must contain blocking sets. Whenever in state s some job J moves to some (non-full) machine M, this machine M must become full and must then be part of any blocking set. Among all possible moves, consider a move that yields a state t with a newly full machine M for which the canonical blocking set $\mathcal{B}^t_{\min}(M)$ is of the smallest possible cardinality.

Note that in state t there exist a machine $M' \in \mathcal{B}^t_{\min}(M)$ and a job $J' \in \mathcal{J}^t(M')$ with $M \in \mathcal{M}^t(J')$; otherwise $\mathcal{B}^t_{\min}(M) - \{M\}$ would be a blocking set for state s. Now consider the successor state u of s that results by moving job J' from machine M to M'. Since $\mathcal{M}^u(J') \subseteq \mathcal{B}^t_{\min}(M)$, a simple inductive argument shows that $\mathcal{B}^u_{\min}(M) \subseteq \mathcal{B}^t_{\min}(M)$. Since job J' has just jumped away from M', this machine cannot be full in state u, and hence $M' \in \mathcal{B}^t_{\min}(M) - \mathcal{B}^u_{\min}(M)$. Consequently the canonical blocking set $\mathcal{B}^u_{\min}(M)$ has smaller cardinality than $\mathcal{B}^t_{\min}(M)$. This contradiction completes the proof. □

Lemma 3.3 *For a given state s, it can be decided in polynomial time whether s has a blocking set of machines. Consequently, problem* Safe State Recognition *can be decided in polynomial time.*

Proof: Create an auxiliary digraph that corresponds to state s: the vertices are the machines $M_1, \ldots, M_m$. Whenever some job J_j occupies a machine M_i, the digraph contains an arc from M_i to every machine in $\mathcal{M}^s(J_j)$. Obviously state s has a blocking set of machines if and only if the auxiliary digraph contains a strongly connected component with the following two properties: (i) All vertices in the component are full. (ii) There are no arcs leaving the component. Since the strongly connected components of a digraph can easily be determined and analyzed in linear time (see for instance [3]), the desired statement follows. □

4 Analysis of reachable states

In this section we discuss the behavior of reachable system states. We say that a state t is *subset-reachable* from state s, if every job J_j satisfies one of the following three conditions:

- $M^t(J_j) = M^s(J_j)$ and $\mathcal{M}^t(J_j) = \mathcal{M}^s(J_j)$, or
- $M^t(J_j) \in \mathcal{M}^s(J_j)$ and $\mathcal{M}^t(J_j) \subseteq \mathcal{M}^s(J_j) - \{M^t(J_j)\}$, or
- $M^t(J_j) = M_{m+1}$ and $\mathcal{M}^t(J_j) = \emptyset$.

Clearly whenever a state t is reachable from some state s, then t is also subset-reachable from s. The following example demonstrates that the reverse implication is not necessarily true. This example also indicates that the algorithmic problem Reachable State Recognition (as formulated in the introduction) is not completely straightforward.

Example 4.1 *Consider an open shop system with two machines M_1, M_2 of capacity 1 and two jobs J_1, J_2 with $\mathcal{M}(J_1) = \mathcal{M}(J_2) = \{M_1, M_2\}$. Consider the state s where J_1 is being processed on M_1 and J_2 is being processed on M_2, and where $\mathcal{M}^s(J_1) = \mathcal{M}^s(J_2) = \emptyset$. It can be seen that s is subset-reachable from the initial state 0, whereas s is not reachable from 0.*

Our next goal is to derive a polynomial time algorithm for recognizing reachable system states. Consider an open shop scheduling system and a fixed system state s. Without loss of generality we assume that s is subset-reachable from the initial state. We define a new (artificial) state t where $M^t(J_j) := M^s(J_j)$ and $\mathcal{M}^t(J_j) := \mathcal{M}(J_j) - \mathcal{M}^s(J_j) - \{M^s(J_j)\}$ for all jobs J_j. Note that in both states s and t every job is sitting on the very same machine, but the work that has already been performed in state s is exactly the work that still needs to be done in state t.

Lemma 4.2 *State s is reachable if and only if state t is safe.*

Proof: First assume that s is reachable, and let $0 = s_0 \to s_1 \to \cdots \to s_k = s$ denote a corresponding witness sequence of moves. Define a new sequence $t = t_k \to t_{k-1} \to \cdots \to t_0 = f$ of moves: Whenever the move $s_\ell \to s_{\ell+1}$ $(0 \le \ell \le k-1)$ results from moving job J_j from machine M_a to machine M_b, then the move $t_{\ell+1} \to t_\ell$ results from moving job J_j from machine M_b to machine M_a. (Note that the artificial machines M_0 and M_{m+1} switch their roles.) Hence t is safe. A symmetric argument shows that if t is safe then s is reachable. □

Hence deciding reachability is algorithmically equivalent to deciding safeness. Together with Lemma 3.3 this yields the following theorem.

Theorem 4.3 Reachable State Recognition *can be decided in polynomial time.* □

The following lemma states a simple sufficient condition that makes a state reachable.

Lemma 4.4 *Let s be a state, and let $\mathcal{K}$ be a subset of machines such that every job that still needs further processing in s satisfies $M^s(J_j) \in \mathcal{K}$ and*

$$\mathcal{M}^s(J_j) \cup \{M^s(J_j)\} \;=\; \mathcal{K} \cap \mathcal{M}(J_j).$$

Then s is a reachable system state.

Proof: By renaming the jobs we assume that the jobs J_j with $1 \leq j \leq k$ have $M^s(J_j) = M_{m+1}$ and the jobs J_j with $k+1 \leq j \leq n$ have $M^s(J_j) \in \mathcal{K}$. We handle the jobs one by one in their natural order: every job moves through all machines in $\mathcal{M}(J_j) - \mathcal{M}^s(J_j)$, and ends up on machine $M^s(J_j)$. Then the next job is handled. □

5 Analysis of state-to-state reachability

We establish NP-hardness of State-to-State Reachability by means of a reduction from the following satisfiability problem; see Garey & Johnson [4].

> Problem: Three-Satisfiability
>
> Input: A set $X = \{x_1, \ldots, x_n\}$ of n logical variables; a set $C = \{c_1, \ldots, c_m\}$ of m clauses over X that each contain three literals.
>
> Question: Is there a truth assignment for X that satisfies all clauses in C?

We start from an instance of Three-Satisfiability, and construct a corresponding instance of State-to-State Reachability for it. Throughout we will use ℓ_i to denote the unnegated literal x_i or the negated literal $\overline{x_i}$ for some fixed variable $x_i \in X$, and we will use ℓ to denote a generic literal over X. Altogether there are $5n + m$ machines:

- For every literal ℓ_i, there are three corresponding machines $S(\ell_i)$, $T(\ell_i)$, and $U(\ell_i)$. Machine $U(\ell_i)$ has capacity 2, whereas machines $S(\ell_i)$ and $T(\ell_i)$ have capacity 1. For every variable $x_i \in X$ the two machines $U(x_i)$ and $U(\overline{x_i})$ coincide, and the corresponding machine will sometimes simply be called $U(i)$.
- For every clause $c_j \in C$, there is a corresponding machine $V(c_j)$ with capacity 3.

Furthermore the scheduling instance contains $4n$ jobs that correspond to literals and $6m$ jobs that correspond to clauses. For every literal ℓ_i there are two corresponding jobs:

- Job $J(\ell_i)$ is sitting on machine $S(\ell_i)$ in state s. In state t it has moved to machine $U(\ell_i)$ without visiting other machines inbetween.
- Job $J'(\ell_i)$ is still waiting outside the system in state s, and has already left the system in state t. Inbetween the job visits machines $S(\ell_i)$, $T(\ell_i)$, $U(\ell_i)$ in arbitrary order.

Consider a clause c_j that consists of three literals ℓ_a, ℓ_b, ℓ_c. Then the following six jobs correspond to clause c_j:

- For $\ell \in \{\ell_a, \ell_b, \ell_c\}$ there is a job $K(c_j, \ell)$ that in state s sits on machine $V(c_j)$, then moves through machines $S(\ell)$ and $T(\ell)$ in arbitrary order, and finally has left the system in state t. Note that in state s these three jobs block machine $V(c_j)$ to full capacity.
- For $\ell \in \{\ell_a, \ell_b, \ell_c\}$ there is another job $K'(c_j, \ell)$ that waits outside the system in state s, then moves through machines $U(\ell)$ and $V(c_j)$ in arbitrary order, and finally has left the system in state t.

In Sections 5.1 and 5.2 we will show that in the constructed scheduling instance state t is reachable from state s if and only if the THREE-SATISFIABILITY instance has a satisfying truth assignment. This then implies the following theorem.

Theorem 5.1 STATE-TO-STATE REACHABILITY *is NP-complete.*

5.1 Proof of the if-statement

We assume that the THREE-SATISFIABILITY instance has a satisfying truth assignment. We describe a sequence of moves that brings the scheduling system from the starting state s into the goal state t.

In a first phase, for every true variable x_i the job $J(x_i)$ moves from machine $S(x_i)$ to machine $U(i)$. Then job $J'(x_i)$ enters the system by moving to $U(i)$, then moves to $T(x_i)$, then to $S(x_i)$, and finally leaves the system. Next job $J'(\overline{x_i})$ enters the system, moves to $U(i)$, and finally sits and waits on $T(\overline{x_i})$. Symmetric moves (with the roles of x_i and $\overline{x_i}$ interchanged) are performed for every false variable x_i. At the end of this phase, for every true literal ℓ_i the two machines $S(\ell_i)$ and $T(\ell_i)$ are empty, and there is an empty spot on machine $U(\ell_i)$.

In the second phase, we consider clauses c_j that consist of three literals ℓ_a, ℓ_b, ℓ_c. We pick one true literal ℓ_i from c_j, and we let the corresponding job $K(c_j, \ell_i)$ jump away from machine $V(c_j)$ to machine $S(\ell_i)$, then to $T(\ell_i)$, and finally make it leave the system. This yields a free spot on machine $V(c_j)$. For every $\ell \in \{\ell_a, \ell_b, \ell_c\}$ we let job $K'(c_j, \ell)$ enter the system, move through machines $U(\ell)$ and $V(c_j)$, and then leave the system. At the end of this phase, four out of the six jobs corresponding to every clause have reached their final destination in state t.

In the third phase, for every true variable x_i the job $J(\overline{x_i})$ moves from machine $S(\overline{x_i})$ to machine $U(i)$. Job $J'(\overline{x_i})$ moves from $T(\overline{x_i})$ to $S(\overline{x_i})$, and then leaves the system. Symmetric moves (with the roles of x_i and $\overline{x_i}$ interchanged) are performed for every false variable x_i. At the end of this phase, all jobs $J(\ell_i)$ and $J'(\ell_i)$ have reached their final destination in state t. All machines $S(\ell_i)$ and $T(\ell_i)$ are empty.

In the fourth phase, we again consider clauses c_j that consist of three literals ℓ_a, ℓ_b, ℓ_c. For the two literals ℓ in c_j that did not get picked in the second phase, we move the corresponding job $K(c_j, \ell)$ from machine $V(c_j)$ to machine $S(\ell)$, then to machine $T(\ell)$, and finally make it leave the system. At the end of this phase all jobs have reached their final destination, and the system has reached the desired goal state t.

5.2 Proof of the only-if-statement

We assume that there is a sequence of moves that brings the scheduling system from state s into state t. We will deduce from this a satisfying truth assignment for the THREE-SATISFIABILITY instance.

We say that variable x_i is *activated* as soon as one of the corresponding jobs $J(x_i)$ and $J(\overline{x_i})$ moves to machine $U(i)$. We say that x_i is *deactivated* at the moment μ_i in time where also the other job $J(x_i)$ or $J(\overline{x_i})$ moves to machine $U(i)$. If job $J(x_i)$ moves first and activates x_i, we set variable x_i to true; if $J(\overline{x_i})$ moves first and activates x_i, we set variable x_i to false. We will show that the resulting truth setting satisfies all clauses.

Lemma 5.2 *If ℓ_i is a false literal, then job $K(c_j, \ell_i)$ can visit machine $T(\ell_i)$ only after the deactivation time μ_i of variable x_i.*

Proof: Till the crucial moment μ_i where variable x_i is deactivated, job $J(\ell_i)$ is permanently blocking machine $S(\ell_i)$. From time μ_i onwards, jobs $J(x_i)$ and $J(\overline{x_i})$ together are permanently blocking the machine $U(i)$ with capacity 2.

Suppose for the sake of contradiction that some job $K(c_j, \ell_i)$ moves to machine $T(\ell_i)$ before moment μ_i. Then at time μ_i, it is waiting for its final processing on machine $S(\ell_i)$ and blocking machine $T(\ell_i)$. We claim that under these circumstances job $J'(\ell_i)$ is causing trouble: In case $J'(\ell_i)$ has not yet entered the system at time μ_i, it can never be processed on machine $U(i)$ which is permanently blocked from time μ_i onwards. In case $J'(\ell_i)$ has already entered the system at time μ_i, then at time μ_i it must be sitting on machine $U(i)$ and thereby prevents job $J(\ell_i)$ from moving there. In either case we reach a contradiction. □

Now let us consider some arbitrary clause c_j that consists of three literals ℓ_a, ℓ_b, ℓ_c, let x_a, x_b, x_c be the three underlying variables in X, and assume without loss of generality that the corresponding moments of deactivation satisfy $\mu_a < \mu_b < \mu_c$.

Lemma 5.3 *At time μ_a job $K'(c_j, \ell_a)$ must either be sitting on machine $V(c_j)$, or must have left the system.*

Proof: If at time μ_a job $K'(c_j, \ell_a)$ is still waiting outside the system, then it will never be processed on machine $U(a)$, which is permanently blocked by jobs $J(x_a)$ and $J(\overline{x_a})$. Hence there is no way of reaching state t, which is a contradiction. If at time μ_a job $K'(c_j, \ell_a)$ is sitting on machine $U(a)$, it thereby prevents variable x_a from being deactivated. That's another contradiction. □

Hence at time μ_a job $K'(c_j, \ell_a)$ must already have visited machine $V(c_j)$. Since in the starting state s the three jobs $K(c_j, \ell_a)$, $K(c_j, \ell_b)$, $K(c_j, \ell_c)$ are blocking $V(c_j)$, one of them must have made space and must have moved away before time μ_a; let this job be $K(c_j, \ell_i)$ where $i \in \{a, b, c\}$. We distinguish two cases. First, assume that $K(c_j, \ell_i)$ has moved to machine $S(\ell_i)$. Since variable x_i is still active, literal ℓ_i must be true in this case. Secondly, assume that $K(c_j, \ell_i)$ has moved to machine $T(\ell_i)$. Since variable x_i

is still active, Lemma 5.2 yields that literal ℓ_i is true. In either case, clause c_j contains the true literal ℓ_i.

We conclude that every clause contains some true literal, and that the defined truth setting satisfies all clauses. This completes the proof of Theorem 5.1.

6 Analysis of reachable deadlocks

In this section we show that Reachable Deadlock is an NP-hard problem. Our reduction is from the following variant of the Three-Dimensional Matching problem; see Garey & Johnson [4, p.221].

> Problem: Three-Dimensional Matching
>
> Instance: An integer n. Three pairwise disjoint sets $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$, and $C = \{c_1, \ldots, c_n\}$. A set $T \subseteq A \times B \times C$ of triples, such that every element occurs in at most three triples in T.
>
> Question: Does there exist a subset $T' \subseteq T$ of n triples, such that every element in $A \cup B \cup C$ occurs in exactly one triple in T'?

We start from an arbitrary instance of Three-Dimensional Matching, and construct the following corresponding instance of Reachable Deadlock for it. There are two types of machines. Note that every machine has capacity at most three.

- There are $n + 2$ so-called *structure machines* $S_0, \ldots, S_{n+1}$, each of capacity 1.
- For every triple $t \in T$, there is a corresponding *triple machine* T_t with capacity 3.

Furthermore there are $4n + 2$ jobs.

- For every element $a_i \in A$ there are two corresponding *A-element jobs* $J^+(a_i)$ and $J^-(a_i)$. Job $J^+(a_i)$ requires processing on structure machine S_i, and on every triple machine T_t with $a_i \in t$. Job $J^-(a_i)$ requires processing on structure machine S_{i-1}, and on every triple machine T_t with $a_i \in t$.
- For every element $b_i \in B$ there is a corresponding *B-element job* $J(b_i)$ that requires processing on structure machine S_{n+1}, and on every triple machine T_t with $b_i \in t$.
- For every element $c_i \in C$ there is a corresponding *C-element job* $J(c_i)$ that requires processing on structure machine S_{n+1}, and on every triple machine T_t with $c_i \in t$.
- Finally there is a dummy job D_0 that needs processing on S_0 and S_{n+1}, and another dummy job D_{n+1} that needs processing on S_n and S_{n+1}.

Since every element of $A \cup B \cup C$ occurs in at most three triples, we note that each job requires processing on at most four machines. For the ease of later reference, we also list for every machine the jobs that need processing on that machine.

- A triple machine T_t with $t = (a_i, b_j, c_k)$ handles the four jobs $J^+(a_i)$, $J^-(a_i)$, $J(b_j)$, and $J(c_k)$.

- Structure machine S_i with $1 \leq i \leq n-1$ handles the jobs $J^+(a_i)$ and $J^-(a_{i+1})$.
  Structure machine S_0 handles the two jobs $J^-(a_1)$ and D_0.
  Structure machine S_n handles the two jobs $J^+(a_n)$ and D_{n+1}.
  Structure machine S_{n+1} handles $2n+2$ jobs: D_0, D_{n+1}, all B-element jobs, and all C-element jobs.

The following theorem contains the main result of this section.

Theorem 6.1 Reachable Deadlock *is NP-complete, even if the capacity of each machine is at most three, and if each job requires processing on at most four machines.*

Indeed, Proposition 2.1 yields an NP-certificate for problem Reachable Deadlock. The hardness argument proves that the constructed scheduling instance has a reachable deadlock if and only if the Three-Dimensional Matching instance has answer YES. The only-if-statement will be proved in Section 6.1, and the if-statement will be proved in Section 6.2.

6.1 Proof of the only-if-statement

We assume that the scheduling instance has a reachable deadlock state s, and we will show that then the Three-Dimensional Matching instance has answer YES.

Let $\mathcal{B}^*$ be a blocking set of minimum cardinality in s, and let $\mathcal{J}^*$ denote the jobs that are currently being processed on machines in $\mathcal{B}^*$. For every triple machine T_t in $\mathcal{B}^*$, the job set $\mathcal{J}^*$ contains all four element jobs that need processing on T_t. (First: Machine T_t must be full and hence must process three jobs. Second: If no other job in $\mathcal{J}^*$ needs processing on T_t, then $\mathcal{B}^* - \{T_t\}$ would yield a smaller blocking set.) Similarly, for every structure machine $S_i \in \mathcal{B}^*$ with $0 \leq i \leq n$, the job set $\mathcal{J}^*$ contains both jobs that need processing on S_i.

Lemma 6.2 *The blocking set $\mathcal{B}^*$ contains at least one of the structure machines S_i with $0 \leq i \leq n$.*

Proof: Suppose otherwise. Then $\mathcal{B}^*$ consists solely of triple machines and perhaps of machine S_{n+1}.

We first claim that every triple machine in $\mathcal{B}^*$ processes exactly one A-element job, one B-element job, and one C-element job. Indeed, there is an A-element job $J \in \mathcal{J}^*$ that corresponds to some element $a_i \in A$ and that is processed on some triple machine $T_t \in \mathcal{B}^*$. The machine set $\mathcal{M}^s(J)$ of this job contains another triple machine $T_u \in \mathcal{B}^*$. Then $a_i \in t$ and $a_i \in u$, and both machines T_t and T_u must be processing one A-element job (that corresponds to element a_i), one B-element job, and one C-element job. This established the claim.

Next fix a B-element job $J(b_i) \in \mathcal{J}^*$ that is processed on some machine T_t in $\mathcal{B}^*$. The machine set $\mathcal{M}^s(J(b_i))$ contains yet another machine from $\mathcal{B}^*$. This cannot be a triple machine $T_v \in \mathcal{B}^*$. (Every such machine T_v is processing another B-element jobs $J(b_j)$ with $j \neq i$, which implies $b_i \notin u$). Hence $J(b_i)$ needs future processing on the structure

machine S_{n+1}, and $S_{n+1} \in \mathcal{B}^*$. Then S_{n+1} must be blocked by some job that needs future processing on some other machine in $\mathcal{B}^*$. But neither D_0, nor D_{n+1}, nor any B-element or C-element job can do that. □

Lemma 6.3 *(i) Let $S_i \in \mathcal{B}^*$ with $1 \leq i \leq n$, and let job $J^+(a_i)$ be running on S_i. Then there exists exactly one triple machine $T_t \in \mathcal{B}^*$ with $a_i \in t$, and this machine is processing job $J^-(a_i)$. Furthermore $S_{i-1} \in \mathcal{B}^*$.*

(ii) Let $S_{i-1} \in \mathcal{B}^*$ with $1 \leq i \leq n$, and let job $J^-(a_i)$ be running on S_{i-1}. Then there exists exactly one triple machine $T_t \in \mathcal{B}^*$ with $a_i \in t$, and this machine is processing job $J^+(a_i)$. Furthermore $S_i \in \mathcal{B}^*$.

Proof: As the statements (i) and (ii) are symmetric, we only discuss (i). Consider job $J^+(a_i)$ on machine S_i. Since $\emptyset \neq \mathcal{M}^s(J^+(a_i)) \subseteq \mathcal{B}^*$, we conclude that $J^+(a_i)$ still needs to be processed on a triple machine $T_t \in \mathcal{B}^*$, say with $t = (a_i, b_j, c_k)$. Since T_t is full, it must be processing the three jobs $J^-(a_i)$, $J(b_j)$, and $J(c_k)$. Then none of the remaining triple machines T_u with $a_i \in u$ can be full, and hence none of them can be in $\mathcal{B}^*$.

The job $J^-(a_i) \in \mathcal{J}^*$ is running on $T_t \in \mathcal{B}^*$ and still needs future processing on another machine in $\mathcal{B}^*$. The only remaining candidate for this machine is S_{i-1}. □

Lemma 6.4 *The blocking set $\mathcal{B}^*$ either contains machine S_0 which is busy with $D_0 \in \mathcal{J}^*$, or machine S_n which is busy with $D_{n+1} \in \mathcal{J}^*$. In either case, the blocking set $\mathcal{B}^*$ contains the structure machine S_{n+1}.*

Proof: Lemma 6.2 yields that $S_r \in \mathcal{B}^*$ for some r with $0 \leq r \leq n$. First assume $1 \leq r \leq n$ and that S_r is busy with $J^+(a_r)$. Then an inductive argument based on Lemma 6.3.(i) yields $S_i \in \mathcal{B}^*$ for $0 \leq i \leq r$. Moreover for $1 \leq i \leq r$ machine S_i is busy with $J^+(a_i)$, and finally $S_0 \in \mathcal{B}^*$ must be busy with D_0. Next assume $0 \leq r \leq n-1$ and that S_r is busy with $J^-(a_{r+1})$. Then a symmetric argument based on Lemma 6.3.(ii) yields that machine $S_n \in \mathcal{B}^*$ is busy with D_{n+1}. This establishes the first part of the lemma.

If $S_0 \in \mathcal{B}^*$ is busy with D_0, then $D_0 \in \mathcal{J}^*$ requires future processing on another machine in $\mathcal{B}^*$, which must be S_{n+1}. If $S_n \in \mathcal{B}^*$ is busy with D_{n+1}, then $D_{n+1} \in \mathcal{J}^*$ requires future processing on another machine in $\mathcal{B}^*$, which must be S_{n+1}. In either case this yields the second part of the lemma. □

From now on we will assume that machine $S_0 \in \mathcal{B}^*$ is busy with D_0. (The case where $S_n \in \mathcal{B}^*$ is busy with D_{n+1} can be settled in a symmetric way.) We distinguish two cases on the job running on S_{n+1}.

(Case 1) Assume that S_{n+1} is busy with $D_{n+1} \in \mathcal{J}^*$. Then D_{n+1} is waiting for another machine in $\mathcal{B}^*$, which must be machine S_n that is busy with $J^+(a_n)$. We claim for $1 \leq i \leq n$ that $J^+(a_i)$ is processed on machine $S_i \in \mathcal{B}^*$, and that job $J^-(a_i)$ is processed on a triple machine $T_t \in \mathcal{B}^*$ with $a_i \in t$. The claim is proved by a simple inductive argument based on Lemma 6.3.(i) starting with $i = n$ and going down to $i = 1$. Then every triple machine in $\mathcal{B}^*$ processes one of the jobs $J^-(a_1), \ldots, J^-(a_n)$, one B-element job, and one C-element job. These n triple machines induce a solution for the Three-Dimensional Matching instance.

(Case 2) Assume that S_{n+1} is busy with a B-element job or a C-element job; without loss of generality we assume that it is busy with a B-element job $J(b_r)$. Then $J(b_r)$ is waiting for another machine in $\mathcal{B}^*$, which must be a full triple machine M_t with $t = (a_j, b_r, c_k)$. Machine M_t is busy with the three jobs $J^-(a_j)$, $J^+(a_j)$, and $J(c_k)$.

(i) Job $J^-(a_j)$ is waiting for a full machine in $\mathcal{B}^*$. If $j \geq 2$, then this must be the structure machine S_{j-1} which is processing job $J^+(a_{j-1})$ (and if $j = 1$, then it is the structure machine S_0 which is processing job D_0). An inductive argument based on Lemma 6.3.(i) yields that for $1 \leq i \leq j-1$ job $J^+(a_i)$ is processed on machine $S_i \in \mathcal{B}^*$, and job $J^-(a_i)$ is processed on a triple machine $T_t \in \mathcal{B}^*$ with $a_i \in t$.

(ii) Also job $J^+(a_j)$ is waiting for a full machine in $\mathcal{B}^*$. If $j \leq n-1$, then this must be the structure machine S_j which is processing job $J^-(a_{j+1})$ (and if $j = n$, then it is the structure machine S_n which is processing job D_{n+1}). An inductive argument based on Lemma 6.3.(ii) yields that for $j+1 \leq i \leq n$ job $J^-(a_i)$ is processed on machine $S_{i-1} \in \mathcal{B}^*$, and job $J^+(a_i)$ is processed on a triple machine $T_t \in \mathcal{B}^*$ with $a_i \in t$.

Now the $j-1$ triple machines in (i), the $n-j$ triple machines in (ii), and the triple machine M_t with $t = (a_j, b_r, c_k)$ together induce a solution for the Three-Dimensional Matching instance. This completes the analysis of Case 2, and it also completes the proof of the only-if-statement.

6.2 Proof of the if-statement

We assume that the Three-Dimensional Matching instance has a solution $T' \subseteq T$, and from this we will derive a reachable deadlock state for the scheduling instance.

Consider the subset $\mathcal{K} = \{T_t : t \in T'\} \cup \{S_i : 0 \leq i \leq n+1\}$ of machines. We construct a state t where every job J has already entered the system, has already been processed on all machines in $\mathcal{M}(J) - \mathcal{K}$, and is currently being processed on its first machine from $\mathcal{M}(J) \cap \mathcal{K}$. Hence the assignment of jobs to machines determines the entire state t. We assign job D_0 to machine S_0, and job D_{n+1} to structure machine S_{n+1}. For every triple $t = (a_i, b_j, c_k) \in T'$, we assign the three jobs $J^-(a_i)$, $J(b_j)$, $J(c_k)$ to triple machine T_t, and we assign job $J^+(a_i)$ to triple machine S_i.

The resulting state t has $\mathcal{K}$ as blocking set and is in deadlock. Furthermore Lemma 4.4 shows that t is a reachable state. All in all, this yields a reachable deadlock state t.

7 Reachable deadlocks if jobs require two machines

Throughout this section we only consider open shop systems where $|\mathcal{M}(J)| = 2$ holds for all jobs J. We introduce for every job J and for every machine $M \in \mathcal{M}(J)$ a corresponding real variable $x(J, M)$, and for every machine M a corresponding real variable

$y(M)$. Our analysis is centered around the following linear program (LP):

$$\begin{array}{lll} \min & \sum_M \ \max\{y(M),\ \mathrm{cap}(M)\} & \\ \text{s.t.} & \sum_{J:M\in\mathcal{M}(J)}\ x(J,M) \ = \ y(M) & \text{for all machines } M \\ & \sum_{M\in\mathcal{M}(J)}\ x(J,M) \ = \ 1 & \text{for all jobs } J \\ & x(J,M) \ \geq \ 0 & \text{for all } J \text{ and } M\in\mathcal{M}(J) \end{array}$$

Although this linear program is totally unimodular, we will mainly deal with its fractional solutions.

Lemma 7.1 *One can compute in polynomial time an optimal solution for the linear program (LP) that additionally satisfies the following property (*) for every job J with $\mathcal{M}(J)=\{M_a,M_b\}$: If $y(M_a)\geq \mathrm{cap}(M_a)$ and $x(J,M_a)>0$, then $y(M_b)\geq \mathrm{cap}(M_b)$.*

Proof: We determine in polynomial time an optimal solution of (LP). Then we perform a polynomial number of post-processing steps on this optimal solution, as long as there exists a job violating property (*). In this case $y(M_a)\geq \mathrm{cap}(M_a)$, $x(J,M_a)>0$, and $y(M_b)<\mathrm{cap}(M_b)$.

The post-processing step decreases the values $x(J,M_a)$ and $y(M_a)$ by some $\varepsilon>0$, and simultaneously increases $x(J,M_b)$ and $y(M_b)$ by the same ε. By picking ε smaller than the minimum of $\mathrm{cap}(M_b)-y(M_b)$ and $x(J,M_a)$ this will yield another feasible solution for (LP). What happens to the objective value? If $y(M_a)>\mathrm{cap}(M_a)$ at the beginning of the step, then the step would decrease the objective value, which contradicts optimality. If $y(M_a)=\mathrm{cap}(M_a)$ at the beginning of the step, then the step leaves the objective value unchanged, and yields another optimal solution with $y(M_a)<\mathrm{cap}(M_a)$ and $y(M_b)<\mathrm{cap}(M_b)$.

To summarize, every post-processing step decreases the number of machines M with $y(M)=\mathrm{cap}(M)$. Hence the entire procedure terminates after at most m steps. □

Let $x^*(J,M)$ and $y^*(M)$ denote an optimal solution of (LP) that satisfies the property (*) in Lemma 7.1. Let $\mathcal{M}^*$ be the set of machines M with $y^*(M)\geq\mathrm{cap}(M)$.

Lemma 7.2 *The open shop system has a reachable deadlock, if and only if $\mathcal{M}^*\neq\emptyset$.*

Proof: (Only if). Consider a reachable deadlock state, let $\mathcal{B}'$ be the corresponding blocking set of machines, and let $\mathcal{J}'$ be the set of jobs waiting on these machines. Every job $J\in\mathcal{J}'$ is sitting on some machine in $\mathcal{B}'$, and is waiting for some other machine in $\mathcal{B}'$. Since $|\mathcal{M}(J)|=2$, this implies $\mathcal{M}(J)\subseteq\mathcal{B}'$ for every job $J\in\mathcal{J}'$. Then

$$\sum_{M\in\mathcal{B}'} y^*(M) \ \geq \ \sum_{J\in\mathcal{J}'}\sum_{M\in\mathcal{M}(J)} x^*(J,M) \ = \ |\mathcal{J}'|.$$

Since furthermore $|\mathcal{J}'|=\sum_{M\in\mathcal{B}'}\mathrm{cap}(M)$, we conclude $y^*(M)\geq\mathrm{cap}(M)$ for at least one machine $M\in\mathcal{B}'$.

(If). Let $\mathcal{J}^*$ be the set of jobs with $x^*(J,M) > 0$ for some $M \in \mathcal{M}^*$. Property (*) in Lemma 7.1 now yields the following for every job J: If $J \in \mathcal{J}^*$, then $\mathcal{M}(J) \subseteq \mathcal{M}^*$. Construct a bipartite graph G between the jobs in $\mathcal{J}^*$ and the machines in $\mathcal{M}^*$, with an edge between J and M if and only if $M \in \mathcal{M}(J)$. For any subset $\mathcal{M}' \subseteq \mathcal{M}^*$, the number of job neighbors in this bipartite graph is at least $\sum_{M\in\mathcal{M}'} y^*(M) \geq \sum_{M\in\mathcal{M}'} \mathrm{cap}(M)$. A variant of Hall's theorem from matching theory [8] now yields that there exists an assignment of some jobs from $\mathcal{J}^*$ to machines in $\mathcal{M}^*$ such that every $M \in \mathcal{M}^*$ receives $\mathrm{cap}(M)$ pairwise distinct jobs.

To reach a deadlock, we first send all non-assigned jobs one by one through the system. They are completed and disappear. Then the assigned jobs enter the system, each moving straightly to the machine to which it has been assigned. Then the system falls into a deadlock with blocking set $\mathcal{M}^*$: All machines in $\mathcal{M}^*$ are full, and all jobs are only waiting for machines in $\mathcal{M}^*$. □

Since jobs J with $|\mathcal{M}(J)| = 1$ are harmless and may be disregarded with respect to deadlocks, we arrive at the following theorem.

Theorem 7.3 *For open shop systems where each job requires processing on at most two machines,* Reachable Deadlock *can be solved in polynomial time.* □

The following example illustrates that the above LP-based approach cannot be carried over to the case where every job requires processing on three machines (since the only-if part of Lemma 7.2 breaks down).

Example 7.4 *Consider a system with two jobs and four machines of unit capacity. Job J_1 needs processing on M_1, M_2, M_3, and job J_2 needs processing on M_1, M_2, M_4. A (reachable) deadlock results if J_1 enters the system on M_3 and then moves to M_1, whereas J_2 simultaneously enters the system on M_4 and then moves to M_2.*

We consider a feasible solution with $x(J,M) \equiv 1/3$ for every J and every $M \in \mathcal{M}(J)$, and $y(M_1) = y(M_2) = 2/3$ and $y(M_3) = y(M_4) = 1/3$. The objective value is 4, and hence this is an optimal solution. The post-processing leaves the solution untouched, and the resulting set $\mathcal{M}^*$ is empty.

8 Reachable deadlocks if machines have unit capacity

Throughout this section we only consider open shop systems with $\mathrm{cap}(M_i) \equiv 1$. For each such system we define a corresponding undirected edge-colored multi-graph $G = (V, E)$: The vertices are the machines $M_1, \ldots, M_m$. Every job J_j induces a clique of edges on the vertex set $\mathcal{M}(J_j)$, and all these edges receive color c_j. Intuitively, if two machines are connected by an edge e of color c_j, then job J_j may move between these machines along edge e.

Lemma 8.1 *For an open shop system with unit machine capacities and its corresponding edge-colored multi-graph the following two statements are equivalent.*

(i) The multi-graph contains a simple cycle whose edges have pairwise distinct colors.

(ii) The system can reach a deadlock.

Proof: Assume that (i) holds, and consider a simple cycle C whose edges have pairwise distinct colors. By renaming jobs and machines we may assume that the vertices in C are the machines $M_1, \ldots, M_k$, and that the edges in C are $[M_j, M_{j+1}]$ with color c_j for $1 \leq j \leq k-1$, and $[M_k, M_1]$ with colors c_k. Consider the following processing order of the jobs:

- In the first phase, the jobs J_j with $k+1 \leq j \leq n$ are processed one by one: Job J_{j+1} only enters the system after job J_j has completed all its processing and has already left the system. At the end of this phase we are left with the jobs $J_1, \ldots, J_k$.
- In the second phase, the jobs $J_1, \ldots, J_k$ are handled one by one. When job J_j is handled, first all operations of J_j on machines M_i with $i \geq k+1$ are processed. Then job J_j moves to machine M_j, and stays there till the end of the second phase. Then the next job is handled.

At the end of the second phase, for $1 \leq i \leq k$ job J_i is blocking machine M_i, and waiting for future processing on some other machine in cycle C. The system has fallen into a deadlock, and hence (i) implies (ii).

Next assume that (ii) holds, and consider a deadlock state. For every waiting job J_j in the deadlock, let M'_j be the machine on which J_j is currently waiting and let M''_j denote one of the machines for which the job is waiting. Consider the sub-graph of G that for every waiting job J_j contains the vertex M'_j together with an edge $[M'_j, M''_j]$ of color c_j. This sub-graph has as many vertices as edges, and hence must contain a simple cycle; hence (ii) implies (i). □

Lemma 8.2 *For the edge-colored multi-graph $G = (V, E)$ corresponding to some open shop system with unit machine capacities, the following three statements are equivalent.*

(i) The multi-graph contains a simple cycle whose edges have pairwise distinct colors.

(ii) The multi-graph contains a 2-vertex-connected component that spans edges of at least two different colors.

(iii) The multi-graph contains a simple cycle whose edges have at least two different colors.

Proof: We show that (i) implies (ii) implies (iii) implies (i). The implication from (i) to (ii) is straightforward.

Assume that (ii) holds, and consider a vertex v in such a 2-vertex-connected component that is incident to two edges with two distinct colors. These two edges can be connected to a simple cycle, and we get (iii).

Assume (iii), and consider the shortest cycle C whose edges have at least two different colors. If two edges $[u, u']$ and $[v, v']$ on C have the same color c_j, then the vertices

u, u', v, v' are all in the machine set $\mathcal{M}(J_j)$ of job J_j. Hence they span a clique in color c_j, and some edges in this clique can be used to construct a shorter cycle with edges of at least two different colors. This contradiction shows that (iii) implies (i). □

Lemmas 8.1 and 8.2 together yield that an open shop system can fall into a deadlock state if and only if the corresponding multi-graph contains a 2-vertex-connected component that spans edges of at least two different colors. Since the 2-vertex-connected components of a graph can easily be determined and analyzed in linear time (see for instance [3]), we arrive at the following theorem.

Theorem 8.3 *For open shop systems with unit machine capacities, problem* Reachable Deadlock *can be solved in polynomial time.* □

References

[1] Z.A. Banaszak and B.H. Krogh (1990). Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Transactions on Robotics and Automation 6*, 724–734.

[2] E.G. Coffman, M.J. Elphick, and A. Shoshani (1971). System Deadlocks. *ACM Computing Surveys 3*, 67–78.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein (2001). *Introduction to Algorithms*. MIT Press.

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

[5] M. Gold (1978). Deadlock prediction: Easy and difficult cases. *SIAM Journal on Computing 7*, 320–336.

[6] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (1993). Sequencing and scheduling: Algorithms and complexity. In: *Handbooks in Operations Research and Management Science, Vol. 4*, North Holland, 445–522.

[7] M. Lawley (1999). Deadlock avoidance for production systems with flexible routing. *IEEE Transactions on Robotics and Automation 15*, 497–510.

[8] L. Lovász and M.D. Plummer (1986). Matching Theory. *Annals of Discrete Mathematics 29*, North-Holland.

[9] W. Sulistyono and M. Lawley (2001). Deadlock avoidance for manufacturing systems with partially ordered process plans. *IEEE Transactions on Robotics and Automation 17*, 819–832.

[10] K. Xing, F. Lin, and B. Hu (2001). An optimal deadlock avoidance policy for manufacturing system with flexible operation sequence and flexible routing. *Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA'2001)*, 3565–3570.