Job-shop scheduling with limited capacity buffers

Peter Brucker, Silvia Heitmann
University of Osnabrück, Department of Mathematics/Informatics
Albrechtstr. 28, D-49069 Osnabrück, Germany
{peter,sheitman}@mathematik.uni-osnabrueck.de

Johann Hurink, Tim Nieberg
University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science
P.O. Box 217, 7500 AE Enschede, The Netherlands
{j.l.hurink,t.nieberg}@utwente.nl

April 2005

Abstract

In this paper we investigate job-shop problems where limited capacity buffers to store jobs in non-processing periods are present. In such a problem setting, after finishing processing on a machine, a job either directly has to be processed on the following machine or it has to be stored in a prespecified buffer. If the buffer is completely occupied the job may wait on its current machine but blocks this machine for other jobs. Besides a general buffer model, also specific configurations are considered.

The aim of this paper is to find a compact representation of solutions for the job-shop problem with buffers. In contrast to the classical job-shop problem, where a solution may be given by the sequences of the jobs on the machines, now also the buffers have to be incorporated in the solution representation. In a first part, two such representations are proposed, one which is achieved by adapting the alternative graph model and a second which is based on the disjunctive graph model. In a second part, it is investigated whether the given solution representation can be simplified for specific buffer configurations. For the general buffer configuration it is shown that an incorporation of the buffers in the solution representation is necessary, whereas for specific buffer configurations possible simplifications are presented.

Keywords: Job-shop problem, Buffer, Disjunctive graph, Alternative graph.

AMS classification: 90B35

1 Introduction

The job-shop problem is one of the most popular scheduling problems. The popularity is based on its interesting combinatorial structure and on its wide range of applications. In the literature most articles investigate a basic version of the job-shop problem contrasting the fact that in most applications additional constraints have to be satisfied. One of these constraints is the fact that jobs which leave a machine to be processed on the next machine must be stored in some buffer if the next machine is still processing another job. Usually, the buffers have a limited capacity. Thus, a job cannot leave a machine if the next machine is occupied and the buffer is full. It must stay on the machine and blocks it until either a job leaves the buffer or the next machine releases its job.

In the classical job-shop problem $J \parallel C_{\max}$ for each job a specific route through the machines is defined. In contrast to the flow-shop situation, where the routes must be the same for all jobs, the routes in a job-shop environment depend on the problem input and may differ from each other. Considering a job-shop problem with buffers jobs may enter different buffers on their routes. Thus, one has to assign a buffer each time a job needs a storage place on its route. In a flow-shop situation this assignment is defined in a natural way: Since all jobs take the same route, we have an intermediate buffer between each pair of successive machines.

In this paper, we assume that a set of buffers of limited capacity is given and that for each operation exactly one of these buffers is specified as a possible storage place for the case that after the processing of the operation storage is needed. Depending on this assignment of operations to buffers, several different types of buffers are possible. If the assignment of operation O_{ij} depends on the machine on which operation O_{ij} has to be processed, this type of buffer is called **output buffer**. An output buffer B_k is directly related to machine M_k and stores all jobs which leave machine M_k and cannot directly be loaded on the following machine. Symmetrically, an **input buffer** B_k is a buffer which is directly related to machine M_k and in which jobs are stored that have already finished processing on the previous machine, but cannot directly be loaded on machine M_k. We also consider the model in which a buffer B_{kl} is associated with each pair (M_k, M_l) of machines M_k and M_l. Each job, which changes from machine M_k to M_l and needs storage, has to use buffer B_{kl}. This model is called **pairwise buffer model**. If the assignment of operations to buffers is job-dependent we speak of **job-dependent buffers**. In this case a dedicated buffer for storing each job is available. If the assignment underlies no special structure, we call this type of buffers **general buffers**.

It has been shown by Papadimitriou & Kanellakis [8] that even the two-machine flow-shop problem with a limited buffer between the first and the second machine (which is a special case of each of the above mentioned buffer models if we exclude job-dependent buffers) is strongly $\mathcal{NP}$-hard. Thus, to solve a job-shop problem with limited buffer capacities in reasonable time, heuristics have to be applied. In the literature only flow-shop problems with buffers of limited capacities are considered. All known results concern flow-shop problems with makespan objective and intermediate buffers between successive machines. Leisten [4] presents some priority based heuristics for the permutation flow-shop situation as well as for the general flow-shop situation with buffers. Recently, Smutnicki [9] and Nowicki [7] developed tabu search approaches for the permutation flow-shop problem with two and arbitrary many machines, respectively. Brucker et al. [3] generalized the approach of Nowicki [7] to the case where different job-sequences on the machines are allowed. The special case, where all buffers have capacity 0, is called the blocking job-shop problem. In Mascis & Pacciarelli [5] heuristics and a branch and bound approach for this problem are presented.

The most successful heuristics for the classical job-shop problem are based on the representation of solutions by the disjunctive graph model. If for each machine M_k, $k = 1, \dots, m$, a sequence π^k of all operations to be processed on M_k is specified, an optimal schedule respecting the sequences $(\pi^1, \dots, \pi^m)$ on the machines can be found by longest path calculations. Thus, the solution space can be represented by the set of vectors $(\pi^1, \dots, \pi^m)$ of permutations which provide a feasible schedule. In Smutnicki [9], Nowicki [7] and Brucker et al. [3] it has been shown that the same solution representation can be used for flow-shop problems with intermediate buffers. By introducing so-called buffer arcs an optimal schedule respecting given sequences $(\pi^1, \dots, \pi^m)$ can be found by longest path calculations.

The objective of this paper is to derive solution representations for the job-shop problem with limited capacity buffers which can be used in connection with local search heuristics. We derive two such representations. For the first representation buffers B with capacity b are represented by b buffer slots. The buffer slots are considered as additional "machines" in a blocking job-shop problem. One has to decide whether jobs use associated buffers or not. If a job uses buffer B a corresponding buffer operation with processing time zero must be assigned to a buffer slot of B. Solutions are represented by assignments for buffer slots as well as by machine and buffer slot sequences. In the second representation we introduce for each buffer B an input sequence and an output sequence. These sequences define the order in which jobs using B enter and leave the buffer. We show that for given input/output sequences optimal buffer slot assignments can be calculated in polynomial time. Furthermore, for all special buffer situations input and output sequences can be derived in polynomial time, if the machine sequences are given.

This paper is organized as follows. After a formal description of the job-shop problem with buffers in the next section, we discuss job-shop problems with blocking operations in Section 3. In Section 4 we describe the two different graph models for the problem. In Section 5, we consider the special buffer types and derive further results for these specialized situations. The last section contains some concluding remarks.

2 Problem formulation

The job-shop problem with general buffers is a generalization of the classical job-shop problem and may be formulated as follows:

Given are m machines $M_1, \ldots, M_m$ and q buffers B_i with a capacity of b_i units $(i = 1, \ldots, q)$. On the machines n jobs $j = 1, \ldots, n$ have to be processed. Each job j consists of n_j operations $O_{1j}, O_{2j}, \ldots, O_{n_j j}$ which must be processed in the given order, i.e. we have precedence constraints $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{n_j j}$. Associated with operation O_{ij} is a dedicated machine $\mu_{ij} \in \{M_1, \ldots, M_m\}$ on which O_{ij} must be processed for $p_{ij} > 0$ time units without preemption. We assume that $\mu_{ij} \neq \mu_{i+1,j}$ for all $j = 1, \ldots, n$ and $i = 1, \ldots, n_j - 1$. Thus, for a job a specific route through the machines is defined. When operation O_{ij} finishes processing on machine μ_{ij}, its successor operation $O_{i+1,j}$ may directly start on the next machine $\mu_{i+1,j}$ if this is not occupied by another job. Otherwise, job j is stored in the buffer β_{ij}, where $\beta_{ij} \in \{B_1, \ldots, B_q\}$ is given. However, it may happen that $\mu_{i+1,j}$ is occupied and the buffer β_{ij} is full. In this case, job j has to stay on μ_{ij} until a job leaves buffer β_{ij} or the job occupying $\mu_{i+1,j}$ moves to another machine. Thus, during this time job j blocks machine μ_{ij} for processing other jobs.

A feasible schedule of the jobs is given by an assignment of starting times S_{ij} (and thus, completion times $C_{ij} = S_{ij} + p_{ij}$) to operations O_{ij} $(i = 1, \ldots, n_j; j = 1, \ldots, n)$ such that

1. the precedence relations within the jobs are respected $(C_{ij} \leq S_{i+1,j})$,
2. during the complete time interval $[S_{1j}, C_{n_j j}]$ job j occupies either a machine or a buffer $(j = 1, \ldots, n)$,
3. at each time any machine is occupied by at most one job and buffer B_i is occupied by at most b_i jobs $(i = 1, \ldots, q)$.

The problem we consider is to find a feasible schedule which minimizes the makespan $C_{\max} = \max\limits_{j=1}^{n} C_j$, where C_j is the finishing time $C_{n_j j}$ of the last operation $O_{n_j j}$ of job j.

To simplify notation in some parts of the paper, for each operation i we denote by $\sigma(i)$ the successor operation of i and by $J(i)$ the job to which i belongs. Furthermore, $\mu(i) \in \{M_1, \ldots, M_m\}$ is the machine on which i must be processed and $\beta(i) \in \{B_1, \ldots, B_q\}$ is the specified buffer associated with i.

Depending on the buffer assignment β_{ij} one can distinguish different buffer models:

- We call a buffer model **general buffer model** if any assignment β_{ij} of operations to buffers is possible.

- If the assignment β_{ij} depends on the job index j, i.e. if each job has an own buffer, we speak of **job-dependent buffers**.

- If the assignment β_{ij} depends on the machines on which O_{ij} and $O_{i+1,j}$ are processed, this buffer model is called **pairwise buffer model**. In this situation a buffer B_{kl} is associated with each pair (M_k, M_l) of machines M_k and M_l. If $\mu_{ij} = M_k$ and $\mu_{i+1,j} = M_l$, operation O_{ij} is assigned to buffer B_{kl}. A pairwise buffer model is usually used in connection with the flow-shop problem. Each job has to use $B_{k,k+1}$ when moving from M_k to M_{k+1} and machine M_{k+1} is still occupied.

- If the assignment β_{ij} depends on the machine on which O_{ij} is processed, this type of buffers is called **output buffer model**. An output buffer B_k for machine M_k stores all jobs which leave machine M_k and cannot directly be loaded on the following machine.

- Similarly, if the assignment β_{ij} depends on the machine on which $O_{i+1,j}$ is processed, this type of buffer model is called **input buffer model**. An input buffer B_k for machine M_k stores all jobs, which have finished on their previous machine but cannot be loaded on M_k directly.

Another basic model is the **job-shop problem with blocking operations** where an operation-dependent buffer B_{ij} for each operation O_{ij} is given. If no buffer space to store job j after finishing on μ_{ij} is available ($b_{ij} = 0$), we call operation O_{ij} **blocking**. In this case, job j blocks machine μ_{ij} if the next machine is occupied by another job. Otherwise (i.e. $b_{ij} = 1$), operation O_{ij} is called **non-blocking** or **ideal**. Since in the classical job-shop problem all operations are non-blocking, the classical job-shop problem is a special case of the job-shop problem with blocking operations. On the other hand, the job-shop problem where all operations are blocking is called **blocking job-shop problem**.

3 The job-shop problem with blocking operations

In this section, we investigate the job-shop problem with blocking operations where for each operation i it is specified whether buffer space to store job $J(i)$ after the processing of operation i is available or not. This problem constitutes a basic model for the job-shop problem with general buffers. In Subsection 3.1 we show how the job-shop problem with blocking operations can be represented by an alternative graph. An alternative graph (see Mascis & Pacciarelli [5]) is a generalization of a disjunctive graph which is the common model used to represent the classical job-shop problem. In Subsection 3.2 we refer to the job-shop problem with job-dependent buffers which is a special case of the job-shop problem with blocking operations.

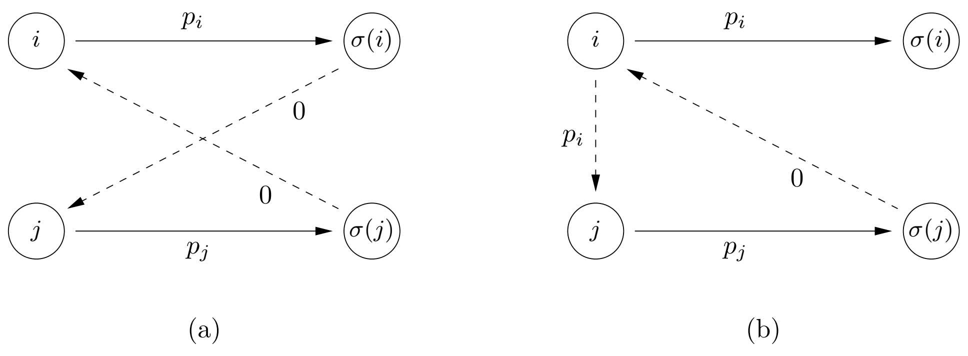

Figure 1: A pair of alternative arcs

3.1 Blocking operations and alternative graphs

Assume that there is not always a buffer to store a job after it has finished on a machine and the next machine is still occupied by another job. Then the job remains on its machine and blocks it until the next machine becomes available. The corresponding operation of this job is a blocking operation. Obviously, blocking operations may delay the start of succeeding operations on the same machine.

Consider two blocking operations i and j which have to be processed on the same machine $\mu(i) = \mu(j)$. If operation i precedes operation j, the successor operation $\sigma(i)$ of operation i must start before operation j can start in order to unblock the machine, i.e. $S_{\sigma(i)} \leq S_j$ must hold. Conversely, if operation j precedes operation i, then operation $\sigma(j)$ must start before operation i can start, i.e. $S_{\sigma(j)} \leq S_i$ must hold.

Thus, there are two mutually exclusive (alternative) relations

$$S_{\sigma(i)} \leq S_j \qquad \text{or} \qquad S_{\sigma(j)} \leq S_i$$

given in connection with i and j. These two mutually exclusive relations can be modelled by a **pair of alternative arcs** $(\sigma(i), j)$ and $(\sigma(j), i)$ as shown in Figure 1(a). The pair of alternative arcs is depicted by dashed lines whereas the solid lines represent precedence constraints within job $J(i)$ and $J(j)$. One has to choose exactly one of the two alternative relations (arcs). Choosing the arc $(\sigma(i), j)$ implies that operation i has to leave the machine before j can start and choosing $(\sigma(j), i)$ implies that j has to leave the machine before i can start.

Next, consider the case where operation i is non-blocking and operation j is blocking and both have to be scheduled on the same machine $\mu(i) = \mu(j)$. If operation i precedes operation j, machine $\mu(i)$ is not blocked after the processing of i. Thus, operation j can start as soon as operation i is finished, i.e. $S_i + p_i \leq S_j$ must hold. On the other hand, if operation j precedes operation i, then operation $\sigma(j)$ must start before operation i, i.e. $S_{\sigma(j)} \leq S_i$ must hold.

Thus, we have the alternative relations

$$S_i + p_i \leq S_j \qquad \text{or} \qquad S_{\sigma(j)} \leq S_i$$

given in connection with i and j. Figure 1(b) shows the corresponding pair of alternative arcs (i, j) and $(\sigma(j), i)$ weighted by p_i and 0, respectively.

Finally, considering two non-blocking operations i and j, which have to be processed on the same machine, leads to the alternative relations

$$S_i + p_i \leq S_j \qquad \text{or} \qquad S_j + p_j \leq S_i$$

These relations can be represented by the alternative arcs (i, j) and (j, i) weighted by p_i and p_j, respectively. This pair of alternative arcs corresponds to a disjunction between operation i and operation j in the classical disjunctive graph model. Choosing one of the two alternative arcs (i, j) or (j, i) is equivalent to directing the disjunction between i and j.

Using this concept, the job-shop problem with blocking operations can be modelled by an alternative graph $G = (V, A, F)$ which is a generalization of a disjunctive graph (see Mascis & Pacciarelli [5]).

The set of vertices V represents the set of all operations. In addition, there is a source node $\circ \in V$ and a sink node $* \in V$ indicating the beginning and the end of a schedule (i.e. $V = \{O_{ij} \,|\, i = 1, \ldots, n_j; j = 1, \ldots, n\} \cup \{\circ, *\}$). The arc set of G consists of a set A of pairs of alternative arcs and a set F of fixed arcs. The fixed arcs reflect the precedence relations $O_{1j} \to O_{2j} \to \ldots \to O_{n_j j}$ between the operations of each job $j = 1, \ldots, n$. The arc $O_{ij} \to O_{i+1,j}$ is weighted by the processing time p_{ij} (for $i = 1, \ldots, n_j - 1$). Furthermore, in F we have arcs $\circ \to O_{1j}$ and $O_{n_j j} \to *$ weighted by 0 and $p_{n_j j}$, respectively. The set A consists of all pairs of alternative arcs for operations i and j which have to be processed on the same machine: If i and j are both blocking, the pair of alternative arcs consists of $(\sigma(i), j)$ and $(\sigma(j), i)$ weighted both by 0. If i is non-blocking and j is blocking, we introduce the pair of alternative arcs (i, j) and $(\sigma(j), i)$ with lengths p_i and 0, respectively. If i and j are both non-blocking, the pair of alternative arcs is (i, j) and (j, i) weighted by p_i and p_j, respectively. In the special case when operation i is the last operation of job $J(i)$, machine $\mu(i)$ is not blocked after the processing of i. Thus, in this case, operation i is always assumed to be non-blocking.

Considering the special case of a classical job-shop problem, all operations are non-blocking. Therefore, each pair of alternative arcs is of the form $\{(i, j), (j, i)\}$ where i and j are operations to be processed on the same machine. The resulting special type of an alternative graph corresponds to a disjunctive graph.

In the following, we consider an example for a job-shop problem with blocking operations and show up the corresponding alternative graph: Given are three machines and three jobs where jobs 1 and 2 consist of three operations each and job 3 consists of two operations. Jobs 1 and 2 have to be processed first on M_1, then on M_2 and last on M_3, whereas job 3 has to be processed first on M_2 and next on M_1. The first two operations of jobs 1 and 2 are assumed to be blocking. All other operations are non-blocking.

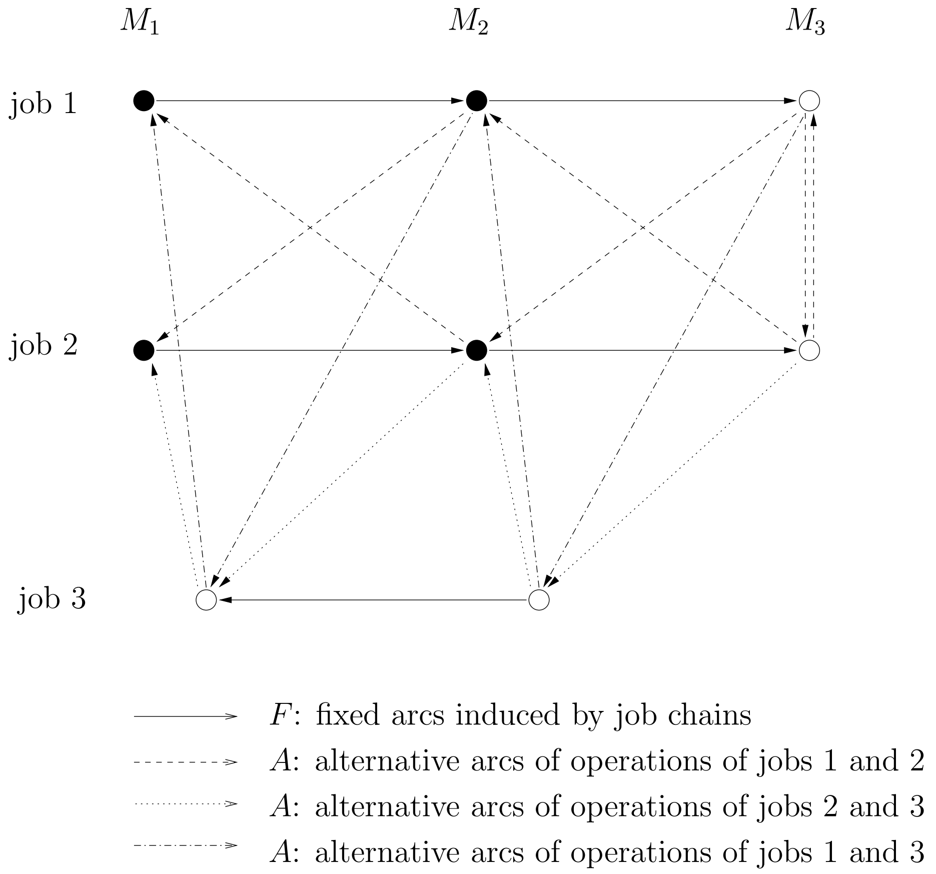
M_1
M_2
M_3
job 1
job 2
job 3
F: fixed arcs induced by job chains
A: alternative arcs of operations of jobs 1 and 2
A: alternative arcs of operations of jobs 2 and 3
A: alternative arcs of operations of jobs 1 and 3

Figure 2: An alternative graph $G = (V, A, F)$

Figure 2 shows the alternative graph $G = (V, A, F)$ for this instance. (The source node and the sink node as well as all arcs emanating from the source and all arcs terminating in the sink are left out.) The job chains of each job are shown horizontally. Black circles represent blocking operations whereas white circles represent non-blocking operations. In order to differentiate pairs of alternative arcs, alternative arcs induced by operations of the same two jobs are depicted in the same line pattern.

Given a job-shop problem with blocking operations, the basic scheduling decision is to define an ordering between the operations to be processed on the same machine. This can be done by choosing at most one arc from each pair of alternative arcs. A **selection S** is a set of arcs obtained from A by choosing at most one arc from each pair of alternative arcs. The selection is called **complete** if exactly one arc from each pair is chosen. Given a selection S, let $G(S)$ indicate the graph $(V, F \cup S)$.

For a graph $G(S)$, we define the length $L(p)$ of a path $p = (i_1, \ldots, i_k)$ with $i_j \in V$ by the sum of the lengths of the arcs (i_{j-1}, i_j) ($j = 2, \ldots, k$). Note that all arc lengths in $G(S)$ are nonnegative and, thus, $L(p) \geq 0$ holds for each path p in $G(S)$.

If a complete selection S is given and $G(S)$ does not contain any cycle of positive

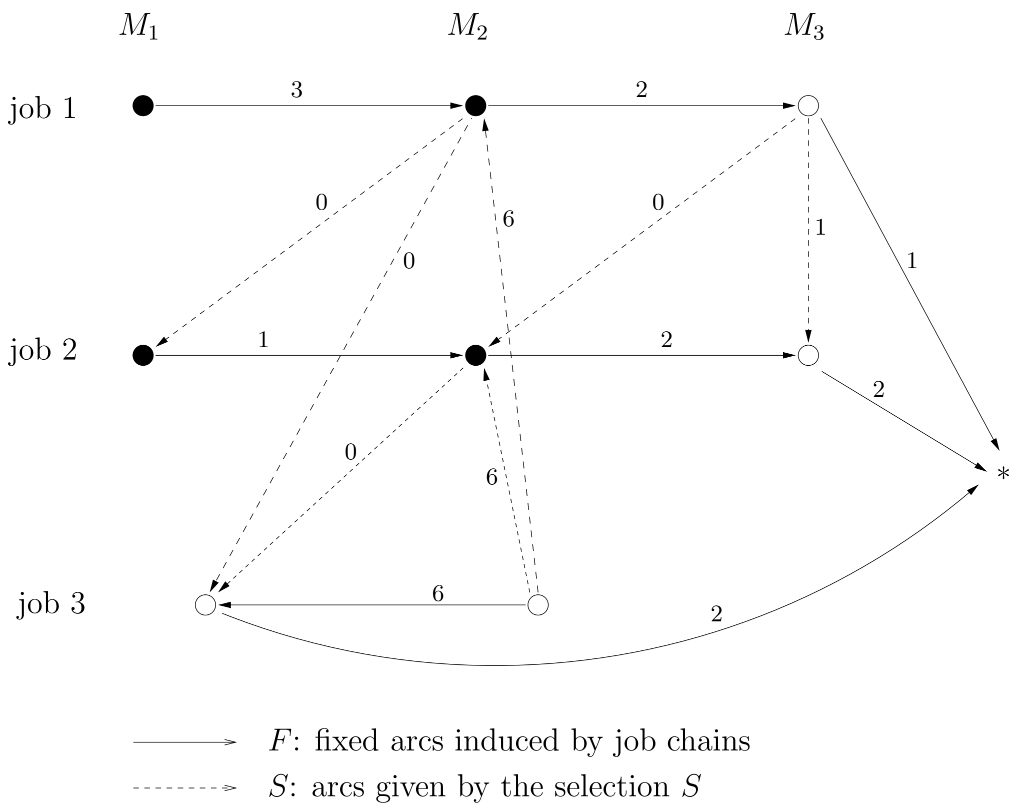
M_1
M_2
M_3
job 1
job 2
job 3
3
2
0
0
6
0
1
1
1
2
2
0
6
6
2
$*$
F: fixed arcs induced by job chains
S: arcs given by the selection S

Figure 3: The graph $G(S) = (V, F \cup S)$

length, let $P(S)$ be the schedule in which the starting time of an operation O_{ij} is equal to the length of a longest path from the source $\circ$ to the vertex representing O_{ij} in $G(S)$. Then, $P(S)$ is a feasible, left-shifted schedule with minimal makespan respecting the ordering given by the selection S. The makespan $C_{\max}(S)$ is equal to the length of a longest $\circ - *$-path in $G(S)$.

In fact, the graph $G(S)$ may contain cycles of length 0. In this case, all operations included in such a cycle start processing at the same time.

As for a classical job-shop problem, a solution for an instance of a job-shop problem with blocking operations can also be given by the sequences $(\pi^1, \ldots, \pi^m)$ of the jobs on the machines, where π^i specifies the order of the jobs on machine M_i ($i = 1, \ldots, m$). A solution $\Pi = (\pi^1, \ldots, \pi^m)$ is called feasible, if there exists a feasible schedule, where the jobs are processed in the sequences $\pi^1, \ldots, \pi^m$ on $M_1, \ldots, M_m$. Obviously, a solution Π induces a complete selection S. The corresponding graph $G(S)$ contains no cycles of positive length if and only if the solution Π is feasible.

Assume that the jobs in the previous example are scheduled in the order $\pi^1 = (1, 2, 3)$ on M_1, $\pi^2 = (3, 1, 2)$ on M_2 and $\pi^3 = (1, 2)$ on M_3. These sequences induce a complete selection S, where the corresponding graph $G(S)$ (with its appropriate arc weights) is shown in Figure 3. By longest path calculations in $G(S)$, the schedule $P(S)$ of Figure 4 can be calculated. The makespan of $P(S)$ is 12. Notice that job 2 cannot start on M_1 before time 6 because job 1 blocks machine M_1 from time 3 to time 6. Similarly, job 2 blocks M_1 from time 7 to time 8.

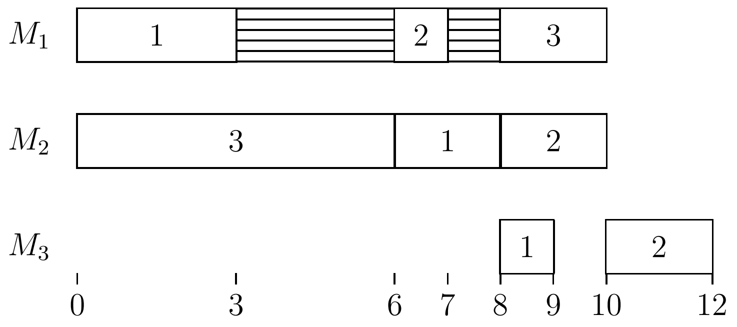

Figure 4: Schedule $P(S)$

3.2 Job-dependent buffers

A special case of the job-shop problem with blocking operations is the job-shop problem with job-dependent buffers. In this model, n buffers B_j $(j = 1, \dots, n)$ are given where B_j may store only operations belonging to job j. Since operations of the same job never require the buffer at the same time, we may restrict the buffer capacity b_j to the values 0 and 1.

Operations belonging to a job with buffer capacity 1 are never blocking since they always can go into the buffer when finishing. On the other hand, all operations of a job with buffer capacity 0 are blocking except its last operation. In the example of Section 3.1, the buffer capacities b_1 and b_2 of jobs 1 and 2 are equal to 0, i.e. jobs 1 and 2 are blocking, whereas job 3 is non-blocking.

4 Solution representation

In the following, we will discuss different ways to represent solutions for the job-shop problem with general buffers. These representations are useful for solution methods like branch-and-bound algorithms and local search heuristics. In later sections, we will show how these representations specialize in connection with specific buffer models. In Subsection 4.1, we show that the job-shop problem with general buffers can be reduced to the blocking job-shop problem. This reduction is based on dividing each buffer into several buffer slots and assigning the operations to the buffer slots. Since this representation has several disadvantages, we propose another representation in Subsection 4.2. Finally, in Subsection 4.3, we present how a corresponding schedule for a given solution can be constructed by longest path calculations in a solution graph model.

4.1 Representation by buffer slot assignments and sequences

In order to apply heuristics to a job-shop problem with general buffers, a suitable representation of solutions is needed. In the case of a job-shop problem with blocking operations, we have seen in the previous section that the solution space can be represented by a set of vectors $(\pi^1, \ldots, \pi^m)$ where π^i specifies an order of the jobs on machine M_i $(i = 1, \ldots, m)$. Given a solution $(\pi^1, \ldots, \pi^m)$, an optimal schedule respecting the sequences $(\pi^1, \ldots, \pi^m)$ can be found by longest path calculation in the graph $G(S)$ where S is the corresponding complete selection. This representation generalizes a representation of solutions for the classical job-shop problem which has been successfully used in connection with local search heuristics. In the following, we will show that the job-shop problem with general buffers can be reduced to the blocking job-shop problem, i.e. to the job-shop problem where all operations are blocking except the last operation of each job.

For this purpose, we differentiate between b storage places within a buffer B of capacity $b > 0$. Thus, the buffer B is divided into b so called **buffer slots $\boldsymbol{B^1, B^2, \ldots, B^b}$**, where a buffer slot B^l represents the l-th storing place of buffer B. Each buffer slot may be interpreted as additional blocking machine on which entering jobs have processing time zero. For each job one has to decide whether it uses a buffer on its route or it goes directly to the next machine. If the job j uses a buffer one has to assign a buffer slot to j. After these decisions and assignments we have to solve a problem which is equivalent to a blocking job-shop problem.

Because of the described reduction, a solution of a job-shop problem with general buffers can be represented by the following three characteristics:

1. sequences of the jobs on the usual machines,
2. a **buffer slot assignment** of each operation to a buffer slot of its corresponding buffer (where an operation may also not use any buffer), and
3. sequences of the jobs on the additional blocking machines (which correspond to buffer slot sequences).

4.2 Representation by sequences

Using the reduction of a job-shop problem with general buffers to a blocking job-shop problem implies that the buffer slot assignment is part of the solution representation. However, this way of solution representation has several disadvantages when designing fast solution procedures for the problem: Obviously, many buffer slot assignments exist which lead to very long schedules. For example, it is not meaningful to assign a large number of jobs to the same buffer slot when other buffer slots remain empty. Also there are many buffer slot assignments which are symmetric to each other. It would be sufficient to choose one of them. Thus, we have the problem to identify balanced buffer slot assignments and to choose one representative buffer slot assignment among classes of symmetric assignments.

To overcome these deficits one may use a different solution representation from which buffer slot assignments can be calculated by a polynomial time algorithm. The basic idea of this approach is to treat the buffer as one object and not as a collection of several slots.

For this purpose, we assign to each buffer B with capacity $b > 0$ two sequences, an input sequence π_{in} and an output sequence π_{out} containing all jobs assigned to buffer B. The input sequence π_{in} is a priority list by which these jobs either enter the buffer or go directly to the next machine. The output sequence π_{out} is a corresponding priority list for the jobs which leave buffer B or go directly to the next machine.

To represent a feasible (deadlock-free) schedule the buffer sequences π_{in} and π_{out} must be compatible with the machine sequences. This means, that two jobs in π_{in} (π_{out}) which come from (go to) the same machine have to be in the same order in the buffer and machine sequence. Additionally, the buffer sequences must be compatible with each other. Necessary conditions for mutual compatibility of π_{in} and π_{out} are given by the next theorem which also describes conditions under which jobs do not use the buffer.

Denote by $\pi_{in}(i)$ and $\pi_{out}(i)$ the job in the i-th position of the sequence π_{in} and π_{out}, respectively.

Theorem 1 : Let B be a buffer with capacity $b > 0$, let π_{in} be an input sequence and π_{out} be an output sequence corresponding with a feasible schedule. Then the following conditions are satisfied:

(a) If $j = \pi_{out}(i) = \pi_{in}(i + b)$ for some position i, then job j does not enter buffer B, i.e. it goes directly to the next machine.

(b) $\pi_{out}(i) \in \{\pi_{in}(1), \ldots, \pi_{in}(i + b)\}$ holds for each position i.

Proof: (a) Let i be a position such that $j = \pi_{out}(i) = \pi_{in}(i + b)$ holds. At the time job j leaves its machine, $i + b - 1$ other jobs have entered buffer B and $i - 1$ jobs have left it. Thus, $(i + b - 1) - (i - 1) = b$ jobs different from j must be in buffer B. Therefore, buffer B is completely filled and job j must go directly to the next machine.

(b) Assume that $j = \pi_{out}(i) = \pi_{in}(i+b+k)$ for some $k \geq 1$. Similar as in (a) we can conclude: At the time job j leaves its machine, $i + b + k - 1$ other jobs have entered buffer B and $i-1$ jobs different from j have left it. Thus, $(i+b+k-1)-(i-1) = b+k$ jobs different from j must be in buffer B. Since this exceeds the buffer capacity, the sequences π_{in} and π_{out} cannot correspond to a feasible schedule. □

From Theorem 1 we conclude that if we have a feasible schedule then for each buffer B the corresponding sequences π_{in} and π_{out} must satisfy the conditions

$$\pi_{out}(i) \in \{\pi_{in}(1), \ldots, \pi_{in}(i+b)\} \quad \text{for all positions } i. \tag{4.1}$$

Conversely, if 4.1 holds then we can find a valid buffer slot assignment by the following procedure which scans both sequences π_{in} and π_{out} from the first to the last position.

Algorithm Buffer Slot Assignment

```
1.  WHILE $\pi_{in}$ is not empty DO BEGIN
2.      Let $j$ be the first job in $\pi_{in}$;
3.      IF $j = \pi_{in}(i+b) = \pi_{out}(i)$ THEN
4.          Put $j$ on the next machine and delete $j$ both from $\pi_{in}$ and $\pi_{out}$;
        ELSE
5.          Put $j$ in the first free buffer slot and delete $j$ from $\pi_{in}$;
6.      WHILE the job $k$ in the first position of $\pi_{out}$ is in the buffer DO
7.          Delete $k$ both from the buffer and from $\pi_{out}$;
    END
```

The following example shows how this algorithm works. Consider the input sequence $\pi_{in} = (1,2,3,4,5,6)$ and the output sequence $\pi_{out} = (3,2,5,4,6,1)$ in connection with a buffer B of capacity $b = 2$. These sequences obviously satisfy Condition (4.1).

We scan π_{in} from the first to the last position. Jobs 1 and 2 are assigned to the buffer slots B^1 and B^2, respectively, and both are deleted from π_{in}. Now, both buffer slots are occupied. Then, we put job 3 on the next machine and delete this job from π_{in} and π_{out}. The new first element of π_{out}, which is job 2, is in the buffer. We eliminate 2 both from the buffer and from π_{out}. Next, we assign job 4 to buffer slot B^2 and delete it from π_{in}. Again, both buffer slots are occupied now. Then, job 5 goes directly to the next machine and we delete it both from π_{in} and π_{out}. Afterwards, we move the first element of π_{out}, which is job 4, from the buffer to the next machine and delete it from π_{out}. Now, we assign the last element 6 of π_{in} to buffer slot B^2 and make π_{in} empty. Thus, by the algorithm job 1 is assigned to buffer slot B^1 and jobs 2, 4 and 6 are assigned to buffer slot B^2.

To prove that the algorithm is correct one has to show that there will be no overflow in the buffer. The only possibility to get such an overflow is when

- the buffer is full, and
- the first element $k = \pi_{out}(i)$ of π_{out} is not in the buffer and not in position $i+b$ of π_{in}.

Then, job k must be in a position greater than $i+b$ in the input sequence π_{in}. Thus, Condition (4.1) is not satisfied.

The buffer slot assignment procedure not only assigns jobs to buffer slots. It also defines a sequence of all jobs assigned to the same buffer slot. This buffer slot sequence is given by the order in which the jobs are assigned to the buffer slot. This order is induced by the buffer input sequence. In the previous example, the buffer slot sequence of B^2 is $(2, 4, 6)$.

Let now Π be an arbitrary feasible solution for a job-shop problem with general buffers. Π defines sequences $\pi^1, \ldots, \pi^m$ for the machines $M_1, \ldots, M_m$ as well as sequences π^B_{in} and π^B_{out} for all buffers B. If we apply to the buffer input and output sequences the buffer slot assignment procedure we get a blocking job-shop problem (where the buffer slots function as additional blocking machines) for which Π is also a feasible solution. This shows, that we do not loose if we represent solutions of the job-shop problem with general buffers by machine sequences $\pi^1, \ldots, \pi^m$ and buffer sequences π^B_{in} and π^B_{out} for all buffers B.

4.3 Calculation of a schedule

We have seen that a solution Π of the job-shop problem with general buffers can be represented by machine sequences $\pi^1, \ldots, \pi^m$ and for each buffer B with $b > 0$ an input sequence π^B_{in} and an output sequence π^B_{out}. A corresponding schedule can be identified by longest path calculations in a directed graph $G(\Pi)$ which is constructed in the following way:

- The set of vertices consists of a vertex for each operation i as well as a source node $\circ$ and a sink node $*$. In addition, for each operation i and each buffer B with $\beta(i) = B$ we have a **buffer-slot operation vertex** $\boldsymbol{i_B}$ if job $J(i)$ is assigned to the buffer by the buffer slot assignment procedure of the previous section.

- We have the following arcs for each operation i where i is not the last operation of job $J(i)$ and i is not the last operation on machine $\mu(i)$: Associated with i and buffer B with $\beta(i) = B$ there is a direct arc $i \to \sigma(i)$ weighted by p_i if $J(i)$ is not assigned to the buffer. Furthermore, we have an arc $\sigma(i) \to j$ with weight 0 where j denotes the operation to be processed immediately after operation i on $\mu(i)$. This arc ensures that operation j cannot start on $\mu(i)$ before the machine predecessor i has left $\mu(i)$.
  If job $J(i)$ is assigned to buffer B, we introduce arcs connected with i and the buffer-slot operation vertex i_B as indicated in Figure 5 (a). In this figure, j again denotes the operation to be processed immediately after operation i on $\mu(i)$. The buffer-slot operation k_B denotes the buffer-slot predecessor of i_B. If there is no such predecessor, the vertex i_B possesses only one incoming arc.

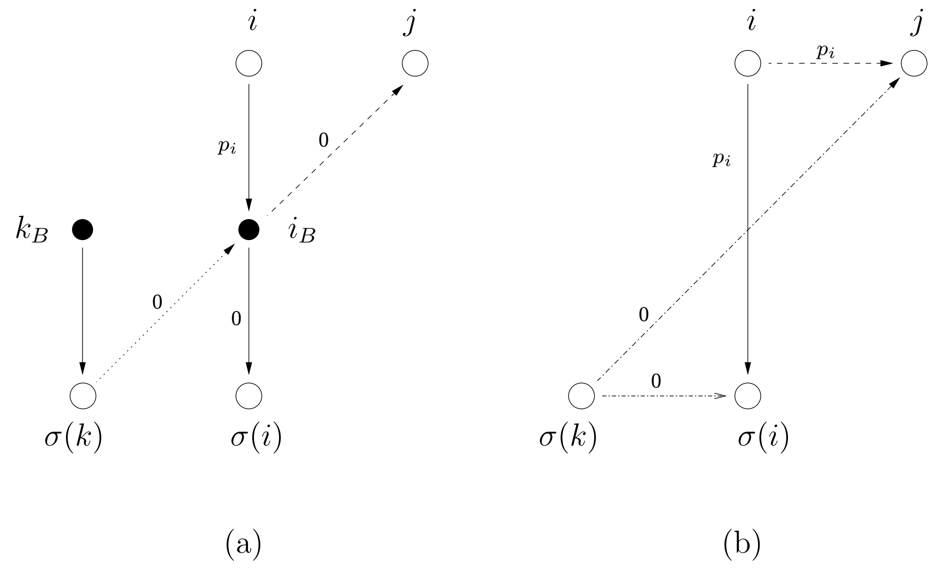

Figure 5: (a) Buffer slot operation vertex i_B with its incoming and outgoing arcs
(b) Simplification of (a) by deleting i_B

The dotted arcs are called **blocking arcs**. The blocking arc $i_B \to j$ ensures that operation j cannot start on $\mu(i)$ before operation i has left $\mu(i)$ and the blocking arc $\sigma(k) \to i_B$ takes care that job $J(i)$ cannot enter the buffer slot before its buffer slot predecessor, which is job $J(k)$, has left the buffer slot.

- We have an arc $\circ \to i$ for each first operation i of a job and an arc $i \to *$ for each last operation i of a job. The arcs $\circ \to i$ and $i \to *$ are weighted by 0 and p_i, respectively. Furthermore, if i is the last operation of job $J(i)$ but not the last operation on machine $\mu(i)$, there is an arc $i \to j$ weighted by p_i where j denotes the operation to be processed immediately after i on $\mu(i)$.

This graph corresponds to the graph introduced in Section 3.1 for the job-shop problem with blocking operations where transitive arcs are left out.

If the graph $G(\Pi)$ does not contain any cycle of positive length, let S_ν be the length of a longest path from $\circ$ to the vertex ν in $G(\Pi)$. Then the times S_ν describe a feasible schedule where S_i is the starting time of operation i and S_{i_B} is the time at which operation i is moved into buffer B.

If we are only interested in the starting times of operations and not the insertion times into buffers, we can simplify $G(\Pi)$ by eliminating buffer-slot operations as indicated in Figure 5 (b). We call the simplified graph $\bar{G}(\Pi)$ **solution graph**. In order to detect a positive cycle in the solution graph, if one exists, and to compute longest paths, the Floyd-Warshall algorithm can be used (see e.g. Ahuja et al. [1]). It has running time $O(r^3)$ where r is the number of vertices, i.e. the total number of operations. An example of the graph $G(\Pi)$ and the solution graph $\bar{G}(\Pi)$ in the case of a flow-shop problem with intermediate buffers will be given in the next section.

5 Special types of buffers

In this section, we consider different special types of buffers and show which simplifications (if any) can be derived in these specialized situations. For each special buffer model, we discuss the question whether it is possible to compute an optimal schedule respecting given sequences $\pi^1, \dots, \pi^m$ of the jobs on the machines in polynomial time.

5.1 Flow-shop problem with intermediate buffers

The flow-shop problem is a special case of the job-shop problem in which each job j consists of m operations O_{ij} $(i = 1, \dots, m)$ and operation O_{ij} has to be processed on machine M_i. This means each job is processed first on M_1, then on M_2, then on M_3, etc. The natural way to define buffers in connection with the flow-shop problem is to introduce an intermediate buffer B_k between succeeding machines M_k and M_{k+1} for $k = 1, \dots, m-1$. If π^i is the sequence of the jobs on machine M_i $(i = 1, \dots, m)$, then obviously the input sequence for B_k is given by π^k and the output sequence must be π^{k+1}. Thus, the sequences $\pi^1, \dots, \pi^m$ are sufficient to represent a solution in the case of a flow-shop problem with intermediate buffers.

Figure 6 shows an example of the graph $G(\Pi)$ for a problem with three machines, five jobs and two buffers which have a capacity of $b_1 = 1$ and $b_2 = 2$ units. The solution Π is given by the machine sequences $\pi^1 = (1, 2, 3, 4, 5)$, $\pi^2 = (2, 1, 3, 5, 4)$ and $\pi^3 = (3, 1, 4, 5, 2)$. The numbers in the white circles denote the indices of the corresponding operations, whereas the black circles represent buffer slot operation vertices. The job chains of each job are shown vertically, where the positions of the black circles also indicate the corresponding buffer slot assignment. Figure 7 shows the resulting simplification after the buffer-slot vertices have been eliminated.

It can be shown that buffer-slots can always be assigned in such a way that the simplified graph consists of the following arcs:

- machine arcs $\pi^k(1) \rightarrow \pi^k(2) \rightarrow \dots \rightarrow \pi^k(n)$ for $k = 1, \dots, m$,
- job arcs $O_{1j} \rightarrow O_{2j} \rightarrow O_{n_j j}$ for $j = 1, \dots, n$, and
- buffer arcs $\pi^{k+1}(i) \rightarrow \pi^k(i+b_k+1)$ for $i = 1, \dots, n-b_k-1$ and $k = 1, \dots, m-1$

(see Brucker et al. [3]).

Due to Condition (4.1) the machine sequences $\pi^1, \dots, \pi^m$ are compatible if and only if

$$\pi^{k+1}(i) \in \{\pi^k(1), \dots, \pi^k(i+b_k)\} \qquad \begin{array}{l} \text{for } k = 1, \dots, m-1 \\ \text{and each position } i = 1, \dots, n-b_k \end{array} \tag{5.1}$$

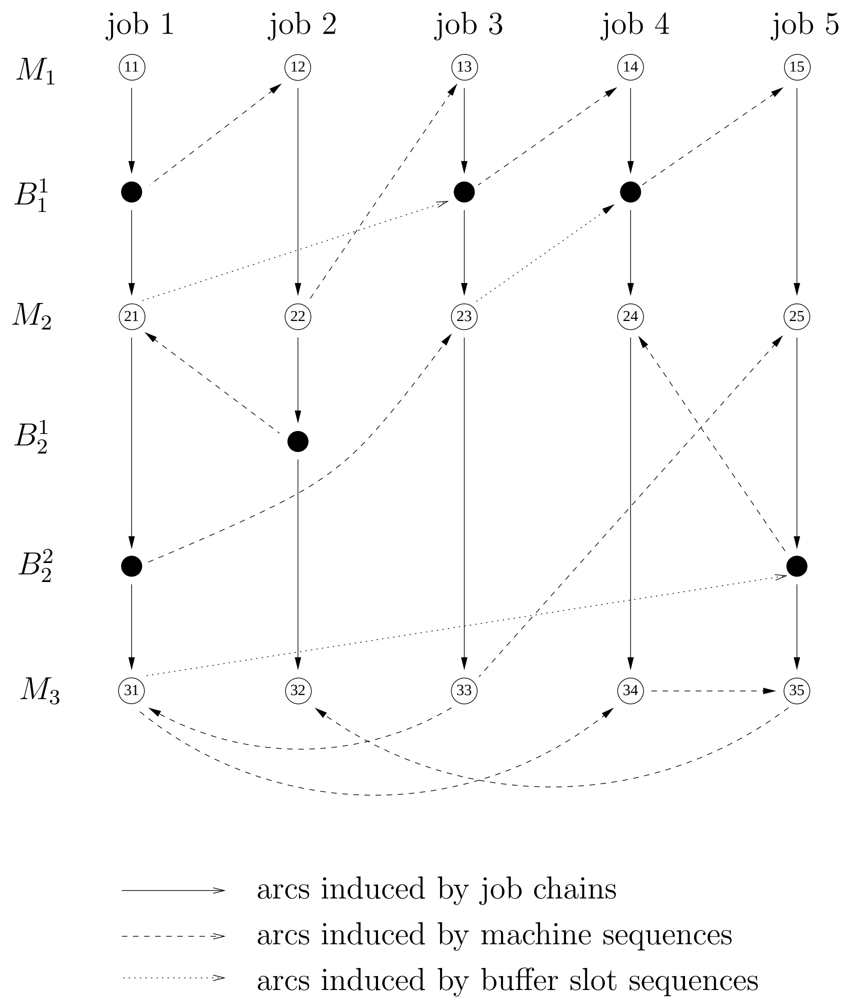
job 1
job 2
job 3
job 4
job 5
M_1
B_1^1
M_2
B_2^1
B_2^2
M_3
11
12
13
14
15
21
22
23
24
25
31
32
33
34
35
arcs induced by job chains
arcs induced by machine sequences
arcs induced by buffer slot sequences

Figure 6: An example of the graph $G(\Pi)$

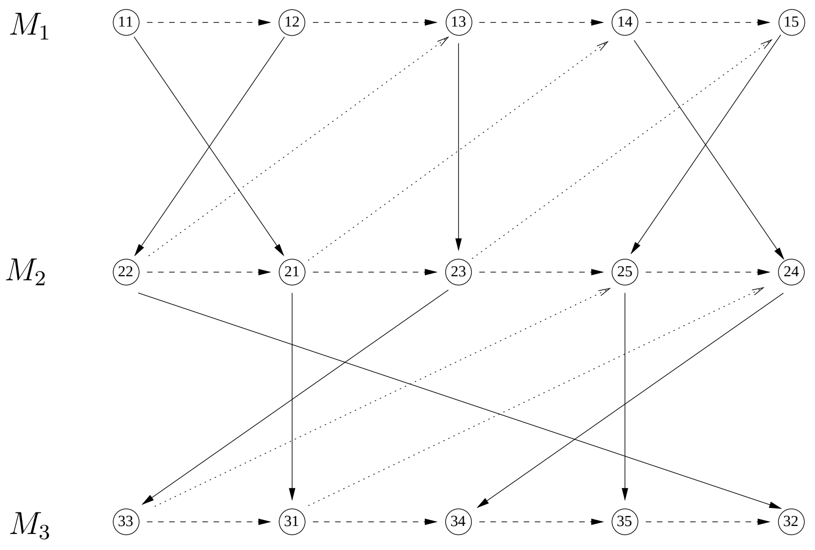

Figure 7: Simplified graph after elimination of buffer-slot vertices in Figure 6

This is equivalent to the condition that the simplified graph contains no cycle. For each k Condition (5.1) can be checked in $O(n)$ time. Thus, we can check in $O(nm)$ time whether the simplified graph contains no cycle. In this case all $\circ - i$ longest path lengths (i.e. a corresponding earliest start schedule) can be calculated in $O(nm)$ time because the simplified graph contains at most $O(nm)$ arcs.

5.2 Job-shop problem with pairwise buffers

For the job-shop problem with pairwise buffers, the situation is very similar to the situation for flow-shop problems with intermediate buffers. In this buffer model, a buffer B_{kl} is associated with each pair (M_k, M_l) of machines. Each job, which changes from M_k to M_l and needs storage, has to use buffer B_{kl}. The input sequence π_{in}^{kl} of buffer B_{kl} contains all jobs in π^k which move to M_l ordered in the same way as in π^k, i.e. π_{in}^{kl} is a partial sequence of π^k. Similarly, π_{out}^{kl} is the partial sequence of π^l consisting of the same jobs but ordered as in π^l. Using the subsequences π_{in}^{kl} and π_{out}^{kl} for each buffer we get a simplified graph $\bar{G}_{kl}$ (see Figure 7). The solution graph for given machine sequences $\pi^1, \ldots, \pi^m$ is a decomposition of all simplified graphs $\bar{G}_{kl}$. However, for the job-shop problem with pairwise buffers conditions similar to (5.1) are not sufficient to guarantee that the solution graph has no cycles. But this is not due to the buffers since even in the case of the classical job-shop problem the solution graph may contain cycles. Furthermore, the solution graph may contain blocking cycles over several machines. Therefore, testing feasibility and calculating a schedule for sequences $\pi^1, \ldots, \pi^m$ is more time consuming. For longest paths calculations the Floyd-Warshall algorithm can be used. It has running time $O(r^3)$, where r is the total number of operations. In Nieberg [6], a tabu search approach for the job-shop problem with pairwise buffers based on the above considerations is presented.

5.3 Job-shop problem with output buffers

A further special type of buffers is that of output buffers. In this case, jobs leaving machine M_k are stored in a buffer B_k $(k = 1, \ldots, m)$ if the next machine is occupied and B_k is not full.

Let us consider a solution of a job-shop problem with output buffers given by the sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines, the buffer input sequences $\pi_{in}^1, \ldots, \pi_{in}^m$ and the buffer output sequences $\pi_{out}^1, \ldots, \pi_{out}^m$. Clearly, the buffer input sequence π_{in}^k of buffer B_k must be identical with the sequence π^k of the jobs on machine M_k $(k = 1, \ldots, m)$. Thus, for the buffers only the buffer output sequences $\pi_{out}^1, \ldots, \pi_{out}^m$ have to be specified. In the following, we show that it is also not necessary to fix buffer output sequences. For given sequences $\pi^1, \ldots, \pi^m$, a polynomial procedure is developed, which calculates optimal buffer output sequences and a corresponding schedule at the same time.

The idea of this procedure is to proceed in time and schedule operations as soon as possible. At the earliest time t where at least one operation is finishing the following moves are performed if applicable:

- move a job finishing at time t to the next machine and start to process it on the next machine,
- move a job finishing at time t on machine M_k into buffer B_k,
- move a job from a buffer to the next machine and start to process it on this machine,
- identify a sequence of operations $i_0, \ldots, i_{r-1}$ with the following properties
  - each operation stays either finished on a machine or in a buffer,
  - at least one of the operations stays on a machine,
  - $J(i_\nu)$ can move to the place occupied by $J(i_{(\nu+1) \bmod r})$,

  and perform a cyclic move, i.e. replace $J(i_{(\nu+1) \bmod r})$ by $J(i_\nu)$ on its machine or in the corresponding buffer for $\nu = 1, \ldots, r-1$,
- move a job out of the system if its last operation has finished.

To control this dynamic process we keep a set C containing all operations which at the current time t are either staying on a machine or are stored in a buffer. Furthermore, machines and buffers are marked available or nonavailable. A machine is nonavailable if it is occupied by an operation. Otherwise, it is available. A buffer is available if and only if it is not fully occupied. For each operation i starting at time t on machine $\mu(i)$ we store the corresponding finishing time $t_i := t + p_i$. At the beginning we set $t_i = \infty$ for all operations i. A job enters the system if its first operation i can be processed on machine $\mu(i) = M_k$, i.e. if the predecessor of i in the machine sequence π^k has left M_k. At the beginning all jobs whose first operation is the first operation in a corresponding machine sequence enter the system.

If at current time t the set C is not empty and no move is possible then we replace t by $\min \{t_i \mid i \in C; t_i > t\}$ if there is an operation $i \in C$ with $t_i > t$. Otherwise, we have a deadlock situation. In this case, the machine sequences are infeasible. An infeasible situation may also occur when C is empty and there are still unprocessed jobs.

Details are described by the **Algorithm Output Buffers**. In this algorithm **Update** $(\mathbf{C}, \mathbf{t})$ is a procedure which performs one possible move.

Algorithm Output Buffers

```
1. $t := 0$; $C := \emptyset$;
```

```
2.  FOR all operations $i$ DO $t_i := \infty$;
3.  Mark all machines and buffers as available;
4.  FOR all first operations $i$ which are sequenced first on a machine DO BEGIN
5.          Schedule $i$ on $\mu(i)$ starting at time $t = 0$;
6.          $t_i := p_i$;
7.          $C = C \cup \{i\}$;
8.          Mark $\mu(i)$ as nonavailable;
    END
9.  WHILE $C \neq \emptyset$ DO BEGIN
10.         FOR each machine $M_j$ which is available DO BEGIN
11.                 IF the current first element $k$ of the machine sequence $\pi^j$ is the first
                    operation of job $J(k)$ THEN BEGIN
12.                         Schedule $k$ on $M_j$ starting at time $t$;
13.                         $t_k := t + p_k$;
14.                         $C = C \cup \{k\}$;
15.                         Mark $M_j$ as nonavailable;
                    END
            END
16.         IF an operation $i \in C$ with $t_i \leq t$ exists and a move of an operation in $C$
            is possible at time $t$ THEN
17.                 **Update (C,t)**;
            ELSE BEGIN
18.                 IF $t_i \leq t$ for all $i \in C$ THEN HALT; /* solution is infeasible */
19.                 $t := \min\{t_i \,|\, i \in C; t_i > t\}$;
            END
    END
20. IF there is an operation $i$ with $t_i = \infty$ THEN solution is infeasible
```

The updating process is done by the following procedure in which $\beta(i)$ denotes the buffer B_k when $\mu(i) = M_k$.

Procedure Update $(\mathbf{C}, \mathbf{t})$

```
1.  IF there is an operation $i \in C$ with $t_i \leq t$ where $i$ is the last operation of job
    $J(i)$ THEN
2.          **Move out of system (C,t,i)**;
3.  ELSE IF there is an operation $i \in C$ with $t_i \leq t$ on machine $\mu(i)$ and $\sigma(i)$ is the
    current first element of the machine sequence $\pi^{\mu(\sigma(i))}$ and $\mu(\sigma(i))$ is available
    THEN
4.          **Move to machine (C,t,i)**;
5.  ELSE IF there is an operation $i \in C$ with $t_i \leq t$ on machine $\mu(i)$ and buffer $\beta(i)$
    is available THEN
6.          **Move in buffer (C,t,i)**;
7.  ELSE IF there is an operation $i \in C$ with $t_i \leq t$ in a buffer and $\sigma(i)$ is the
```

```
    current first element of the machine sequence $\pi^{\mu(\sigma(i))}$ and $\mu(\sigma(i))$ is available
    THEN
8.        **Move out of buffer (C,t,i)**;
9.  ELSE IF there is a sequence of operations $Z : i_0, \dots, i_{r-1}$ with $i_\nu \in C$ and
    $t_{i_\nu} \le t$ such that $\sigma(i_\nu)$ is on the second position in the machine sequence for
    $\mu(i_{(\nu+1) \bmod r})$ or $i_{(\nu+1) \bmod r}$ is in buffer $\beta(i_\nu)$ for $\nu = 0, \dots, r-1$
    and at least one operation of $Z$ is on its machine THEN
10.       **Swap (C,t,Z)**;
    END
```

During the updating process one of the following five different types of moves is performed.

```
**Move out of system** $(\mathbf{C}, \mathbf{t}, \mathbf{i})$
1.  Eliminate $i$ from the machine sequence $\pi^{\mu(i)}$;
2.  Mark machine $\mu(i)$ as available;
3.  $C := C \setminus \{i\}$;
```

```
**Move to machine** $(\mathbf{C}, \mathbf{t}, \mathbf{i})$
1.  Eliminate $i$ from the machine sequence $\pi^{\mu(i)}$;
2.  Mark $\mu(i)$ as available;
3.  Schedule $\sigma(i)$ on $\mu(\sigma(i))$ starting at time $t$;
4.  Mark $\mu(\sigma(i))$ as nonavailable;
5.  $t_{\sigma(i)} := t + p_{\sigma(i)}$;
6.  $C := C \setminus \{i\} \cup \{\sigma(i)\}$;
```

```
**Move in buffer** $(\mathbf{C}, \mathbf{t}, \mathbf{i})$
1.  Move $i$ from machine $\mu(i)$ into buffer $\beta(i)$;
2.  Eliminate $i$ from the machine sequence $\pi^{\mu(i)}$;
3.  Mark $\mu(i)$ as available;
4.  IF buffer $\beta(i)$ is now fully occupied THEN
5.        Mark $\beta(i)$ as nonavailable;
```

```
**Move out of buffer** $(\mathbf{C}, \mathbf{t}, \mathbf{i})$
1.  Eliminate $i$ from the buffer $\beta(i)$;
2.  Mark buffer $\beta(i)$ as available;
3.  Schedule $\sigma(i)$ on $\mu(\sigma(i))$ starting at time $t$;
4.  Mark $\mu(\sigma(i))$ as nonavailable;
5.  $t_{\sigma(i)} := t + p_{\sigma(i)}$;
6.  $C := C \setminus \{i\} \cup \{\sigma(i)\}$;
```

```
Swap (C, t, Z)
1. FOR $\nu := 0$ TO $r-1$ DO BEGIN
2.     IF $i_{(\nu+1) \bmod r}$ is in buffer $\beta(i_\nu)$ THEN BEGIN
3.         Eliminate $i_\nu$ from the machine sequence for $\mu(i_\nu)$;
4.         Move $i_\nu$ into buffer $\beta(i_\nu)$;
       END
       ELSE BEGIN
5.         Eliminate $i_\nu$ from the machine sequence for $\mu(i_\nu)$ or from its buffer;
6.         Schedule $\sigma(i_\nu)$ on $\mu(\sigma(i_\nu))$ starting at time $t$;
7.         $t_{\sigma(i_\nu)} := t + p_{\sigma(i_\nu)}$;
8.         $C := C \setminus \{i_\nu\} \cup \{\sigma(i_\nu)\}$;
       END
```

To show how the Algorithm Output Buffers works, we apply it to the following example. We consider an instance with three machines and output buffers B_1, B_2 and B_3 of capacities $b_1 = 0$, $b_2 = 1$ and $b_3 = 0$. On the machines, five jobs have to be processed where jobs 1 and 2 consist of three operations each and jobs 3,4 and 5 consist of two operations each. In Table 1, for each operation O_{ij} its processing time p_{ij} and the machine μ_{ij} are given.

p_{ij}	j				
i	1	2	3	4	5
1	3	1	1	5	2
2	2	4	3	1	2
3	1	2	–	–	–

μ_{ij}	j				
i	1	2	3	4	5
1	M_1	M_2	M_2	M_3	M_1
2	M_2	M_1	M_3	M_1	M_2
3	M_3	M_2	–	–	–

Table 1: Instance of a job-shop problem with output buffers

Figure 8 shows a schedule for the given instance where the jobs on machine M_1, M_2 and M_3 are sequenced in the order $\pi^1 = (1, 2, 4, 5)$, $\pi^2 = (2, 3, 1, 2, 5)$ and $\pi^3 = (4, 1, 3)$, respectively.

The Algorithm Output Buffers constructs this schedule as follows: We initialize at $t = 0$ by adding the first operations of jobs 1, 2 and 4 to C and set $t_{11} = 3$, $t_{12} = 1$ and $t_{14} = 5$. Since no move is possible at $t = 0$, we increase t to 1. At this time, a move of job 2 into buffer B_2 is performed. Job 2 is eliminated from the first position of π^2, machine M_2 is marked available and B_2 is marked nonavailable. Next, the first operation of job 3 is scheduled on M_2. We add O_{13} to C and set $t_{13} = 2$. Since no further move is possible at $t = 1$ and no move is possible at $t = 2$, the next relevant time is $t = 3$. At this time, a simultaneous swap of the jobs 1, 2 and 3 is performed: Job 1 can be moved from M_1 to M_2 when job 3 is moved simultaneously from M_2 into buffer B_2 and job 2 from B_2 to M_1. Therefore, we eliminate the first operations O_{11} and O_{12} of jobs 1 and 2 from C and add the second operations O_{21} and O_{22} of

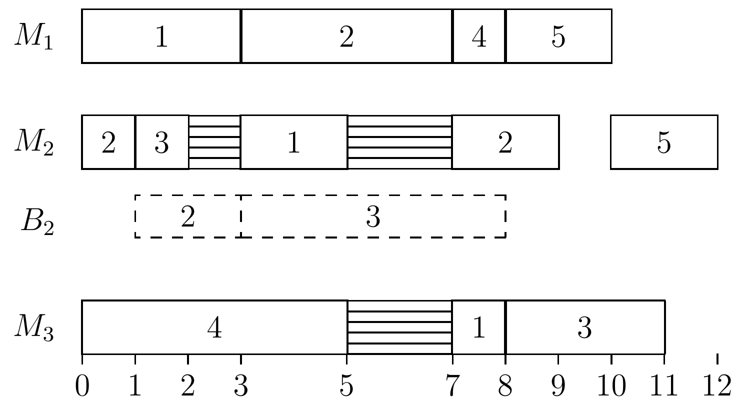

Figure 8: Schedule for a job-shop problem with output buffers

jobs 1 and 2 to C. The first operation O_{13} of job 3 is still contained in C since job 3 only changes from machine M_2 into the buffer B_2. We set $t_{22} = 7$ and $t_{21} = 5$ and eliminate job 1 from the first position of π^1 and job 3 from the first position of π^2. Note, that still M_1, M_2 and B_2 are marked nonavailable. The further steps of Algorithm Output Buffers are shown in Table 2. In the columns M_1, M_2, B_2 and M_3, we set the mark "a" if the corresponding machine or buffer is available.

For given sequences $\pi^1, \dots, \pi^m$ of the jobs on the machines Algorithm Output Buffers provides an optimal solution since each operation is scheduled as early as possible. Postponing the start of an operation i on machine M_j is not advantageous when the sequence π^j of machine M_j is fixed and i is the operation to be processed next on M_j. Note, that the machine sequences are compatible if and only if Algorithm Output Buffers schedules all operations.

Furthermore, the schedule constructed by the algorithm also induces buffer output sequences. In contrast to the previous buffer cases, these buffer output sequences are dependent on the processing times of the given instance. This means, for two instances of a job-shop problem with output buffers which only differentiate in the processing times of the jobs, the optimal assignment of operations to buffer slots may be different though the given machine sequences are equal. Thus, also the corresponding solution graphs of such instances for given machine sequences may be different. Consequently, in the output buffer case, the constructed solution graph is not only dependent on the sequences of the jobs on the machines as in the preceding types of buffers but it is also based on the processing times of the given instance.

5.4 Job-shop problem with input buffers

Similar to an output buffer, an input buffer B_k is a buffer which is directly related to machine M_k ($k = 1, \dots, m$). An input buffer B_k stores all jobs that have already finished processing on the previous machine but cannot directly be loaded on machine

t	action	C	t_i	M_1	M_2	B_2	M_3
0		$\emptyset$		a	a	a	a
	schedule first operations of jobs 1, 2 and 4	O_{11}, O_{12}, O_{14}	$t_{11} = 3$, $t_{12} = 1$, $t_{14} = 5$	–	–	a	–
1	move job 2 in B_2;			–	a	–	–
	schedule first operation of job 3	$O_{11}, O_{12}, O_{13}, O_{14}$	$t_{13} = 2$	–	–	–	–
2	no move is possible			–	–	–	–
3	swap of jobs 1, 3 and 2	$O_{21}, O_{22}, O_{13}, O_{14}$	$t_{21} = 5$, $t_{22} = 7$	–	–	–	–
5	no move is possible			–	–	–	–
7	swap of jobs 1, 4 and 2	$O_{31}, O_{32}, O_{13}, O_{24}$	$t_{31} = 8$, $t_{32} = 9$, $t_{24} = 8$	–	–	–	–
8	eliminate last operations of jobs 1 and 4;	O_{32}, O_{13}		a	–	–	a
	schedule first operation of job 5;	O_{32}, O_{13}, O_{15}	$t_{15} = 10$	–	–	–	a
	move job 3 out of B_2 on M_3	O_{32}, O_{23}, O_{15}	$t_{23} = 11$	–	–	a	–
9	eliminate last operation of job 2	O_{23}, O_{15}		–	a	a	–
10	move job 5 from M_1 to M_2	O_{23}, O_{25}	$t_{25} = 12$	a	–	a	–
11	eliminate last operation of job 3	O_{25}		a	–	a	a
12	eliminate last operation of job 5	$\emptyset$		a	a	a	a

Table 2: Output of Algorithm Output Buffers

M_k. In the case of a job-shop problem with input buffers, the output sequence β^k_{out} of buffer B_k is equal to the sequence π^k.

The job-shop problem with input buffers can be seen as a symmetric counterpart to the problem with output buffers in the following sense: A given instance of a job-shop problem with input buffers can be reversed to an instance of a job-shop problem with output buffers by inverting any job chain $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{n_j j}$ into $O_{n_j j} \rightarrow \ldots \rightarrow O_{2j} \rightarrow O_{1j}$ and by changing the input buffer B_k related to M_k into an output buffer ($k = 1, \ldots, m$). Both problems have the same optimal makespan $C_{\max}$. Therefore, we can solve the corresponding ouput buffer problem going from right to left. The earliest starting time S_i of operations i in an optimal solution of the output buffer problem provide latest finishing times $C_{\max} - S_i$ of operations i in a makespan minimizing solution of the input buffer problem. Clearly, a schedule with finishing times $C_{\max} - S_i$ for the input buffer problem is in general not leftshifted since blocking times and machine waiting times occur before the processing of an operation instead after its processing.

5.5 Job-shop problem with general buffers

In the previous sections we have shown that for all considered special types of buffers an efficient calculation of an optimal schedule respecting given sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines is possible. If we consider general buffers, the easiest type of buffers, which does not belong to the special types, is that of a single buffer with capacity one for all jobs. In the following we show that for this case, the problem of finding an optimal schedule respecting given sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines is already $\mathcal{NP}$-hard in the strong sense.

Theorem 2 : For given sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines in a job-shop problem with a single buffer of capacity one for all jobs, the problem of finding a feasible schedule with minimal makespan respecting these sequences is $\mathcal{NP}$-hard in the strong sense.

Proof: We show that the strongly $\mathcal{NP}$-complete problem 3-PARTITION (3-PART) is polynomially reducible to the decision version of the considered problem.

3-PART: Given $3r$ positive numbers $a_1, \ldots, a_{3r}$ with $\sum_{k=1}^{3r} a_k = rb$ and $b/4 < a_k < b/2$ for $k = 1, \ldots, 3r$, does there exist a partition $I_1, \ldots, I_r$ of $I = \{1, \ldots, 3r\}$ such that $|I_j| = 3$ and $\sum_{k \in I_j} a_k = b$ for $j = 1, \ldots, r$?

Given an arbitrary instance of 3-PART, we construct the following instance of the job-shop problem with a single buffer and specify sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines:

$$n = 12r, \quad m = 8r, \quad q = 1, \quad b_1 = 1$$

$$\begin{array}{llll}
n_j = 1 & p_{1j} = a_j & & j = 1, \ldots, 3r \\
 & \mu_{1j} = j & & j = 1, \ldots, 3r \\
n_j = 2 & p_{1j} = 1 & p_{2j} = 1 & j = 3r+1, \ldots, 6r \\
 & \mu_{1j} = j - 3r & \mu_{2j} = j & j = 3r+1, \ldots, 6r \\
n_j = 2 & p_{1j} = 1 & p_{2j} = 1 & j = 6r+1, \ldots, 9r \\
 & \mu_{1j} = j - 3r & \mu_{2j} = j - 6r & j = 6r+1, \ldots, 9r \\
\hline
n_j = 2 & p_{1j} = (j-9r)(b+1) & p_{2j} = (10r-j)(b+1) & j = 9r+1, \ldots, 10r \\
 & \mu_{1j} = j - 3r & \mu_{2j} = j - 2r & j = 9r+1, \ldots, 10r \\
n_j = 2 & p_{1j} = (j-10r)(b+1)+1 & p_{2j} = (11r-j)(b+1) & j = 10r+1, \ldots, 11r \\
 & \mu_{1j} = j - 3r & \mu_{2j} = j - 4r & j = 10r+1, \ldots, 11r \\
n_j = 1 & p_{1j} = 1 & & j = 11r+1, \ldots, 12r \\
 & \mu_{1j} = j - 5r & & j = 11r+1, \ldots, 12r
\end{array}$$

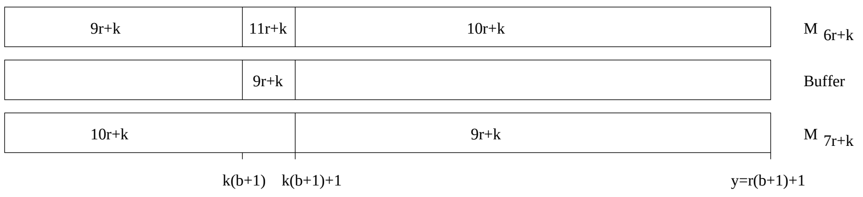

Figure 9: Schedule on machines M_{6r+k} and M_{7r+k}

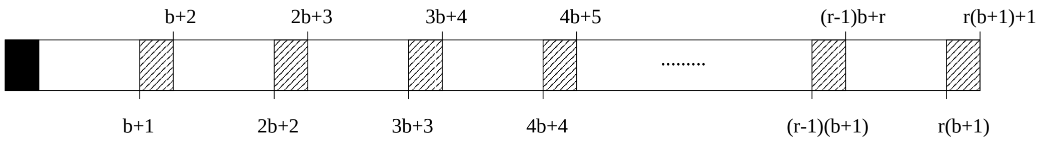

Figure 10: Partial schedule of the buffer

$$\begin{array}{ll}
\pi^i = (O_{1,3r+i}, O_{1,i}, O_{2,6r+i}) & i = 1, \ldots, 3r \\
\pi^i = (O_{1,3r+i}, O_{2,i}) & i = 3r+1, \ldots, 6r \\
\hline
\pi^{6r+i} = (O_{1,9r+i}, O_{1,11r+i}, O_{2,10r+i}) & i = 1, \ldots, r \\
\pi^{7r+i} = (O_{1,10r+i}, O_{2,9r+i}) & i = 1, \ldots, r.
\end{array}$$

(Since only one buffer is available we do not have to specify the $b(i, j)$ values.)

The problem is to find a feasible schedule respecting the sequences $\pi^1, \ldots, \pi^m$ with makespan $C_{max} \leq y = r(b+1)+1$. We show that such a schedule exists if and only if 3-PART has a solution.

First, for a feasible schedule with $C_{max} \leq y$, we determine the structure of the schedule on machines $M_{6r+1}, \ldots, M_{8r}$ and the resulting consequences for the buffer.

Since the sum of the processing times of the operations to be processed on machine M_{6r+k} for $k = 1, \ldots, 2r$ is equal to y, machine M_{6r+k} contains no idle time in each schedule with $C_{max} \leq y$ The corresponding schedules on machines M_{6r+k} and M_{7r+k} are shown in Figure 9.

Thus, job $11r + k$ has to be processed during time interval $[k(b+1), k(b+1)+1]$ and during this time period job $9r + k$ has to wait in the buffer. Consequently, in each feasible schedule with $C_{max} \leq y$, the jobs $9r+1, \ldots, 10r$ occupy the buffer as indicated in Figure 10 by the hatched intervals. Furthermore, since all processing times of the operations are at least 1, the buffer is not occupied in time interval $[0, 1]$ which is marked by the filled area in Figure 10. Summarizing, in each feasible schedule with $C_{max} \leq y$ the buffer has exactly r separated intervals of length b left for the jobs $1, \ldots, 9r$.

Next, we consider the machines $M_1, \ldots, M_{6r}$. Job k for $k = 1, \ldots, 3r$, has to be processed on machine M_k for p_{1k} time units between the processing of the first operation of job $3r + k$ and the processing of the second operation of job $6r + k$.

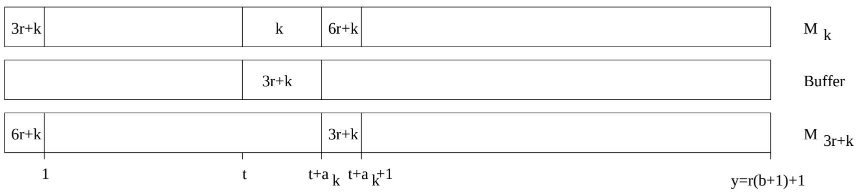

Figure 11: Job $3r + k$ occupies the buffer

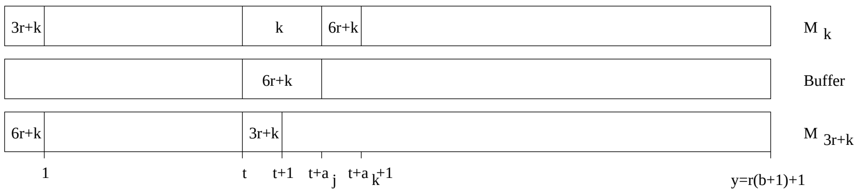

Figure 12: Job $6r + k$ occupies the buffer

Before the processing of job k on M_k can start, job $3r + k$ has to leave machine M_k. It either has to be inserted in the buffer or it has to move on machine M_{3r+k}. In the first case, job $3r + k$ may leave the buffer at the time job $6r + k$ moves from machine M_{3r+k} to machine M_k. In this case job $3r + k$ occupies the buffer for at least $p_{1k} = a_k$ time units (see Figure 11). In the second case, job $6r + k$ must have left machine M_{3r+k} before job $3r + k$ can move on this machine. Since on machine M_k job k has to be processed, job $6r + k$ has to be inserted into the buffer and it has to stay in the buffer until job k leaves machine M_k; i.e. in this case job $6r + k$ occupies the buffer for at least $p_{1k} = a_k$ time units (see Figure 12). Summarizing, one of the two jobs $3r + k$ or $6r + k$ has to be inserted into the buffer for at least p_{1k} time units in each feasible schedule.

Now, let us assume that 3-PART has a solution $I_1, \ldots, I_r$. We get a corresponding feasible schedule with $C_{max} = y$ by scheduling

- all first operations of jobs $3r + 1, \ldots, 9r$ within time interval $[0, 1]$,
- the jobs corresponding to the elements in I_j without overlap within time interval $[(j - 1)(b + 1) + 1, j(b + 1)]$,
- all second operations of the jobs $3r + k$ and $6r + k$, $k = 1, \ldots, 3r$ directly after the completion of job k (see Figure 11),
- the jobs $9r + 1, \ldots, 12r$ in the above sketched only possible way within a schedule with $C_{max} \leq y$.

During the time a job k corresponding to an element in I_j is scheduled, the job $3r + k$ enters the buffer (see Figure 11). Since $\sum_{k \in I_j} a_k = b$, these jobs exactly fit in the corresponding free interval of the buffer. Thus, the resulting schedule is feasible and has $C_{max} = y$.

On the other hand, lets assume that a schedule with $C_{max} \leq y$ exists. As we have argued above, in each schedule with $C_{max} \leq y$ the length of all time intervals, where the buffer is not occupied by jobs $9r+1, \ldots, 12r$, is equal to rb and the minimal time jobs from $3r+1, \ldots, 9r$ have to be in the buffer is at least $\sum_{k=1}^{3r} p_{1k} = \sum_{k=1}^{3r} a_k = rb$. Thus, within $[1, y]$ the buffer has to be occupied all the time and from each pair of jobs $\{3r+k, 6r+k\}$ one job has to be inserted into the buffer for exactly p_{1k} time units. Now consider the jobs which are inserted within the time interval $[(j-1)(b+1)+1, j(b+1)]$ in the buffer. If we choose I_j as the set of elements corresponding to the jobs which force the insertion of these jobs into the buffer, we have: $\sum_{k \in I_j} a_k = j(b+1) - ((j-1)(b+1)+1) = b$. Thus, we get a solution of 3-PART. □

Clearly, as a consequence of Theorem 2, the search space in the case of a job-shop problem with a single buffer has to consist of information for the buffer besides the sequences of the jobs on the machines. In Section 4, we showed that an input and an output sequence for the buffer can be used as additional information to fully represent such a solution.

6 Concluding remarks

We have presented a compact representation of solutions for the job-shop problem with buffers. Existing graph models have been adapted and extended in order to compute a corresponding schedule. For special buffer configurations, such as pairwise buffers, job-dependent buffers, output buffers and input buffers, we have shown that it is sufficient to represent solutions only by the sequences of the jobs on the machines. This is the case since corresponding optimal buffer assignments can be calculated efficiently. In Brucker et al. [3] and Nieberg [6], local search methods based on these representations for the flow-shop problem with intermediate buffers and the job-shop problem with pairwise buffers, respectively, have been developed. These approaches can be adapted in order to develop fast heuristics for the case of job-dependent and output buffers. In the general case, we have shown that machine sequences are not sufficient to represent solutions. Thus, the solution representation has to be enlarged by, e.g., an input and an output sequence for each buffer.

The presented solution representation may form the base for local search methods as well as branch and bound approaches for the general and specific buffer configurations. Important next steps would be to develop and test local search heuristics for the job-shop problem with blocking operations and for the job-shop problem with output (or input) buffers.

References

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B. (1993) Network Flows, Prentice Hall, Englewood Cliffs.

[2] Brucker, P. (2004) Scheduling algorithms, 4th edition, Springer, Berlin.

[3] Brucker, P., Heitmann, S., Hurink, J. (2003) Flow-Shop Problems with Intermediate Buffers, OR Spectrum 25, 549-574.

[4] Leisten, R. (1990) Flowshop sequencing problems with limited buffer storage, International Journal of Production Research 28, 2085-2100.

[5] Mascis, A., Pacciarelli, D. (2002) Job-shop scheduling with blocking and no-wait constraints, European Journal of Operational Research 143, 498-517.

[6] Nieberg, T. (2002) Tabusuche für Flow-Shop und Job-Shop Probleme mit begrenztem Zwischenspeicher, Master Thesis, University of Osnabrück.

[7] Nowicki, E. (1999) The permutation flow shop with buffers: A tabu search approach, European Journal of Operational Research 116, 205-219.

[8] Papadimitriou, C.H., Kanellakis, P.C. (1980) Flow shop scheduling with limited temporary storage, Journal Association Computing Machine 27, 533-549.

[9] Smutnicki, C. (1998) A two-machine permutation flow shop scheduling problem with buffers, OR Spektrum 20, No. 4, 229-235.