Job-shop scheduling with limited buffer capacities

Dissertation

im Fachbereich Mathematik/Informatik der
Universität Osnabrück

Silvia Heitmann

— Osnabrück, März 2007 —

Contents

1 Introduction

Recent developments in manufacturing processes have increased the role of just-in-time production systems. Just-in-time production leads to highly coordinated manufacturing processes, continuous flow of products between work stations, and reduced storage capacities in the shop floor. In particular, the amount of storage facilities plays a significant role to reduce capital binding costs on the one hand and to ensure planning flexibility on the other hand. However, in classical scheduling models it is assumed that unlimited buffer space exists where intermediate products may be stored. Thus, considering the storage of intermediate products in buffers of limited capacity in connection with classical scheduling models provides a more realistic model which may be used to solve modern practical problems.

One of the most popular scheduling problems which has a wide range of applications is the job-shop model. In this model jobs consisting of a chain of different operations have to be planned on certain machines in such a way that a given objective function is minimized. Each machine may process at most one operation at a time and operations belonging to the same job cannot be processed simultaneously.

In the classical job-shop problem it is assumed that unlimited buffer space exists where jobs may be stored in non-processing periods. In this work we generalize this model by considering buffers of limited capacities. 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. Thus, the processing of subsequent jobs on this machine is delayed. Besides a general buffer model, also specific configurations are investigated.

The key issue to develop fast heuristics for the job-shop problem with buffers is to find a compact representation of solutions. 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 this work, we propose two solution representations for the job-shop problem with buffers. Furthermore, we investigate whether the given solution representations 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. Based on the given solution representations we develop local search heuristics in the second part of this work. Therefore, the well-known block approach for the classical job-shop problem is generalized to the job-shop problem with specific buffer configurations.

This work is organized as follows. In Section 2 we give a formulation of the job-shop problem with general buffers which is the most general model we consider. We introduce basic notations and describe several different types of buffers which will be investigated in this work. Previous research on job-shop scheduling with buffers is reviewed.

In Section 3, we consider the job-shop problem with blocking operations which constitutes a basic model for the job-shop problem with general buffers. An operation is called blocking if after its processing the corresponding job remains on the machine and blocks it for other jobs as long as the succeeding operation in the job chain starts processing. In a job-shop problem with blocking operations, it is specified for each operation whether it is blocking or non-blocking. We show that the job-shop problem with blocking operations can be represented by an alternative graph. An alternative graph is a generalization of a disjunctive graph which is the common model to represent the classical job-shop problem. As for the classical job-shop problem, a solution of a job-shop problem with blocking operations can be given by the sequences of the jobs on the machines.

In order to develop solution methods like local search heuristics for the job-shop problem with general buffers, compact solution representations for the problem are proposed in Section 4. We 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. 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 a second representation is derived by introducing 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.

In Section 5, we investigate how the solution representation for the job-shop problem with general buffers specializes in case of specific buffer models. We show that for all special buffer situations input and output buffer sequences can be derived in polynomial time, if the machine sequences are given. For the general buffer configuration it is shown that finding a feasible schedule with minimal makespan respecting given machine sequences is $\mathcal{NP}$-hard in the strong sense.

In the next sections of this thesis our objective is to develop local search heuristics which deal with the job-shop problem with different buffer models. A local search method starts with an initial solution and iteratively searches for good solutions in some neighborhood of the current solution. For the classical job-shop problem, neighborhood structures are often defined with the help of a block approach which is used to identify necessary properties of moves which may improve a given solution. In Section 6, we present block approaches for the job-shop problem with buffers considering several special buffer configurations. Based on the given block approaches, neighborhood structures are proposed which are used in combination with tabu search algorithms. Tabu search is a local search heuristic which exploits information from the recent search process in order to decide in which direction the search procedure will be continued. Tabu search approaches for the job-shop problem with different buffer models are described in Section 7. Implementational details such as the efficient calculation of a best schedule for given machine sequences and the specification of an initial solution are discussed and computational results for

the developed algorithms are given. We conclude this work with some final remarks including topics for further research.

2 The job-shop problem with limited buffer capacities

In this section we describe the job-shop problem with limited buffer capacities where jobs which leave a machine must be stored in some buffer of limited capacity if the next machine is not available. Thus, it is a generalization of the classical job-shop problem (cf. Brucker [5]). After formulating the problem we describe several different types of buffers which will be considered in this work. Related models for scheduling problems with buffers which have been proposed in the literature as well as complexity issues concerning job-shop problems with buffers are discussed in Subsection 2.3.

2.1 Problem formulation

At first we consider the classical job-shop problem which may be formulated as follows:

Given are m machines $M_1, \ldots, M_m$ and n jobs $j = 1, \ldots, n$. A 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 $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{n_j j}$. In this context, $O_{ij} \rightarrow O_{i+1,j}$ means that operation O_{ij} has to be finished completely before $O_{i+1,j}$ starts processing. 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 $i = 1, \ldots, n_j - 1$, i.e. consecutive operations of the same job are assigned to different machines. Each machine can only process one operation at a time. The objective is to find a feasible schedule which minimizes the makespan $C_{\max} = \max_{j=1}^{n} C_j$, where C_j is the finishing time $C_{n_j j}$ of the last operation $O_{n_j j}$ of job j.

In this basic version of the job-shop problem it is assumed that sufficient buffer space to store jobs in non-processing periods is available. However, in practical applications the buffer space is often limited. In the following we will generalize the classical job-shop problem by additionally considering **buffers of limited capacity**. For each transition of a job from one machine to the next machine we specify one buffer from the given set of buffers which may be used as storage place. In our general buffer model, we assume that jobs may enter different buffers on their routes through the machines and that the assignment of buffers to jobs is part of the problem input.

Furthermore, as in the classical job-shop problem, we assume that each job leaves the system directly after the finishing of its last operation, i.e. that sufficient buffer space is available to store all jobs after their last operation.

We suppose that q buffers B_i with a capacity of b_i units are given ($i = 1, \ldots, q$). Associated with operation O_{ij} is a dedicated buffer $\beta_{ij} \in \{B_1, \ldots, B_q\}$ which can

be used as possible storage place ($i = 1, \ldots, n_j - 1$). 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}. 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 $\mu_{i+1,j}$ leaves the machine. 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 objective of this **job-shop problem with general buffers** is to find a feasible schedule which minimizes the makespan.

Other objective functions which are interesting especially for applications depend on due dates d_j and weights w_j associated with each job j. Probably, the most important bottleneck objective besides the makespan is the maximum lateness $L_{\max} = \max_{j=1}^{n} L_j$, where $L_j = C_j - d_j$ is the lateness of job j. The maximum weighted lateness $\max_{j=1}^{n} w_j L_j$ is also widely considered. In this work, we restrict on the makespan as objective function. Nevertheless, the models which are derived in the following may form the basis in order to develop solution methods for the job-shop problem with general buffers and other objective functions.

To simplify notation in most parts of this work, for each operation i we denote by $J(i)$ the job to which i belongs and by $\sigma(i)$ the job successor operation of i, if it exists. 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.

Besides the general buffer model we also investigate different special types of buffers which will be described next.

2.2 Specific types of buffers

Depending on the buffer assignment $\beta(i)$ one can distinguish different buffer models:

- We call a buffer model **general buffer model** if any assignment $\beta(i)$ of operations to buffers is possible.
- If the assignment $\beta(i)$ depends on the job $J(i)$, i.e. if each job has an own buffer, we speak of **job-dependent buffers**.
- If the assignment $\beta(i)$ depends on the machines on which i and $\sigma(i)$ 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 $\mu(i) = M_k$ and $\mu(\sigma(i)) = M_l$, operation i is assigned to buffer B_{kl}.
- If the assignment $\beta(i)$ depends on the machine on which i 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.
- Symmetrically, if the assignment $\beta(i)$ depends on the machine on which $\sigma(i)$ 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.

Besides the job-shop problem with the above described different types of buffers we will investigate in particular the **flow-shop problem with intermediate buffers**. The flow-shop problem is a special case of the job-shop problem in which each job consists of m operations and operation O_{ij} has to be processed on machine M_i. In contrast to a general job-shop environment where the routes of the jobs through the machines depend on the problem input and may differ from each other, the routes are the same for all jobs in a flow-shop problem. Therefore, in connection with the flow-shop situation it is usually used a pairwise or intermediate buffer model. Each job has to use buffer B_k when moving from machine M_k to M_{k+1} and machine M_{k+1} is still occupied. This buffer model is identical with all special buffer models (except the job-dependent buffer model) specialized to the flow-shop situation.

Another basic model is the **job-shop problem with blocking operations** where each operation i has its own buffer B_i with capacity $b_i \in \{0, 1\}$. If no buffer space to store job $J(i)$ after finishing on $\mu(i)$ is available ($b_i = 0$), we call operation i **blocking**. In this case, job $J(i)$ blocks machine $\mu(i)$ if the next machine is occupied by another job. Otherwise (i.e. $b_i = 1$), operation i 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**. Also the problem with job-dependent buffers is a special case of the job-shop problem with blocking operations because we may assume that the buffer capacities of all operations belonging to the same job are either 1 or 0.

To illustrate the function of buffers and to show up a blocking situation in the following, we present an example of a job-shop problem with input buffers.

Example 2.1 : We consider an instance with three machines and input buffers B_1, B_2 and B_3 of capacities $b_1 = 0$, $b_2 = 0$ and $b_3 = 1$. On the machines, five jobs have to be processed where jobs 3 and 4 consist of three operations each and jobs 1, 2 and 5 consist of two operations each. In Table 2.1, for each operation O_{ij} its processing time p_{ij} and the machine μ_{ij} on which O_{ij} must be processed are given.

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

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

Table 2.1: Instance of a job-shop problem with input buffers

Figure 2.1 shows a schedule for the given instance where the jobs on machine M_1, M_2 and M_3 are sequenced in the order $\pi^1 = (1, 4, 5, 3)$, $\pi^2 = (2, 3, 5, 4)$ and $\pi^3 = (4, 2, 1, 3)$, respectively. After finishing processing on machine M_2, job 2 enters the input buffer B_3 at time 1, since the next machine M_3 is occupied by job 4. Thus, job 3 can start processing on M_2 at time 1. At its finishing time 2, buffer B_3 is not available. Consequently, job 3 remains on M_2 and blocks it for further jobs. On machine M_3, job 4 finishes processing at time 3. Since the next machine M_1 is occupied by job 1 at time 3 and the input buffer B_1 of M_1 has capacity 0, job 4 remains on M_3 and blocks it. This blocking situation on M_3 ends when job 1 finishes processing on M_1 at time 4. At this time, a simultaneous swap of the jobs 1, 2 and 4 is performed: Job 4 can be moved from machine M_3 to M_1 when job 1 is moved simultaneously from M_1 into buffer B_3 and job 2 from B_3 to M_3. After job 2 has finished processing on M_3 at time 5, job 1 changes from B_3 on M_3 and job 3 from M_2 into B_3. Thus, job 5 can start processing on M_2 at $t = 5$. At its finishing time 6, machine M_1 is not available. Therefore, a further blocking situation occurs on M_2. This blocking time ends when job 4 finishes processing on M_1 at time 7. At this time, jobs 4 and 5 swap their machines: Job 4 changes from M_1 on M_2 and job 5 from M_2 on M_1. Finally, job 3 is scheduled on M_3 and M_1 starting at time 6. The makespan of the schedule is $C_{\max} = 12$.

Considering the same instance and the same sequences π^1, π^2 and π^3 in the case of a classical job-shop problem, no blocking times on M_2 and M_3 occur, since sufficient buffer capacities to store all jobs are available. Thus, jobs 2, 1 and 3 start one time unit earlier on M_3 and job 3 finishes at $t = 11$ on M_1. The makespan of this schedule is one time unit smaller than the makespan in the case of the corresponding job-shop problem with input buffers. Obviously, in general, the optimal makespan

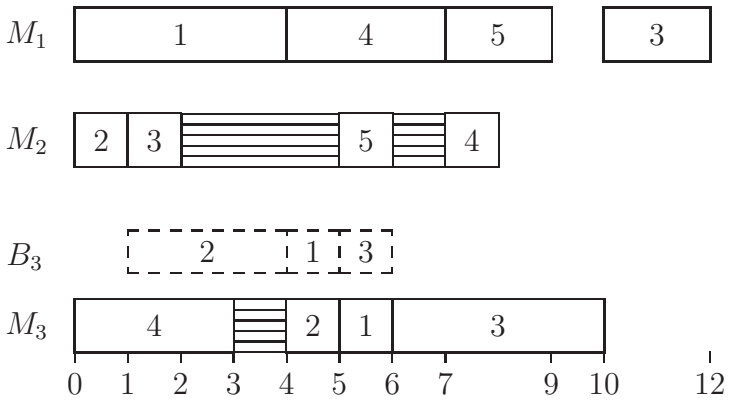

Figure 2.1: Schedule for a job-shop problem with input buffers

of a classical job-shop problem is a lower bound for the optimal makespan of the corresponding job-shop problem with input buffers.

On the other hand, let us consider the blocking job-shop problem for the same instance. In this case, no buffers are available and all operations are blocking. For the given sequences π^1, π^2 and π^3, we have a deadlock situation: Job 4 cannot start on M_1 before job 1 is started on M_3. Job 1 cannot start on M_3 until one time unit after job 4 is started on M_1 because job 2 must be processed on M_3 before the start of job 1 on M_3. This leads to the contradiction $S_{21} \leq S_{24} < S_{24}+1 \leq S_{21}$. Thus, in the case of the blocking job-shop problem, no feasible schedule exists for the sequences π^1, π^2 and π^3. □

In the next subsection, we give an overview of complexity and research results related to the job-shop problem with limited buffer capacities.

2.3 Complexity and previous research

The job-shop problem is one of the most popular scheduling problems and has received much attention in the literature (for a recent overview see e.g. Jain & Meeran [17]). Especially in recent years, a lot of articles investigate job-shop models where additional constraints occurring in applications are taken into account. A collection of a few extensions is given in Brucker & Knust [9]. However, only some results on special cases of the job-shop problem with limited buffer capacities can be found in journal articles published so far. These results mostly concern the blocking job-shop problem or the flow-shop problem with limited buffer capacities.

An overview of complexity results for the blocking job-shop problem is given in the survey of Hall & Sriskandarajah [15]. In particular, they show that the blocking job-shop problem with two machines as well as the blocking flow-shop problem with

three machines are strongly $\mathcal{NP}$-hard. If two machines are given, the blocking flow-shop problem can be reduced to a special case of the traveling salesman problem (see Reddi & Ramamoorthy [31]) which can be solved in polynomial time by the algorithm of Gilmore & Gomory [11]. In Mascis & Pacciarelli [24] heuristics and a branch and bound algorithm for the blocking job-shop problem are presented. Their approach is based on an alternative graph formulation for the problem. They distinguish two cases of blocking: In a blocking problem with swap allowed, jobs can move simultaneously between machines and buffers whereas in a blocking problem with no swap jobs can only move strictly after the subsequent resource has become available. For the special case of the blocking flow-shop problem Ronconi [32] and [33] developped constructive heuristics and a branch and bound algorithm by exploiting specific characteristics of the problem. Recently, Martinez et al. [23] investigated the complexity of flow-shop problems with a different kind of blocking constraint. In this kind, a machine is blocked by a job as long as the successor operation leaves its machine. Thornton & Hunsucker [36] developped a heuristic for minimizing the makespan in a blocking flow-shop problem with identical multiple processors.

Related to the blocking constraint is the no-wait constraint. It does not allow jobs to wait on machines but forces jobs to continue directly on the next machine. Hence, the no-wait constraint is more restrictive for flow-shop problems with three or more machines. In the two-machine case, each solution of a blocking flow-shop problem can be transformed into a solution of the corresponding no-wait problem without changing the completion time of the jobs. For an overview on flow-shop and job-shop problems with no-wait constraints see Hall & Sriskandarajah [15].

Considering the flow-shop problem with limited buffer capacities, already the two-machine case is $\mathcal{NP}$-hard. It has been shown by Papadimitriou & Kanellakis [30] that the two-machine flow-shop problem with a limited buffer capacity of at least one is strongly $\mathcal{NP}$-hard. If the buffer capacity is zero (blocking or no-wait flow-shop), the problem can be solved polynomially (Reddi & Ramamoorthy [31]). On the other hand, the two-machine problem with unlimited buffer capacity (classical flow-shop) can be solved in $O(n \log\ n)$ (Johnson [18]). The more general two-machine job-shop problem with unlimited buffer capacities (classical job-shop) is strongly $\mathcal{NP}$-hard if the processing times are arbitrary (see Lenstra & Rinnooy Kan [22]).

The complexity results show that in order to solve a general job-shop problem with limited buffer capacities in reasonable time, heuristics have to be applied. In the literature mainly 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 [21] presents some priority based heuristics for the permutation flow-shop situation as well as for the general flow-shop situation with buffers. Later, Smutnicki [34] and Nowicki [27] developed tabu search approaches for the permutation flow-shop problem with two and arbitrary many machines, respectively. They use a representation of solutions by the disjunctive graph model where additionally so-called buffer arcs are introduced. If

for each machine M_k, $k = 1, \ldots, m$, a sequence π^k of all operations to be processed on M_k is specified, an optimal schedule respecting the sequences $\pi^1, \ldots, \pi^m$ on the machines can be found by longest path calculations. Thus, the solution space can be represented by the set of vectors $(\pi^1, \ldots, \pi^m)$ of permutations which provide a feasible schedule.

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 [24]) 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.

3.1 Blocking operations and alternative graphs

Assume that there is no 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 3.1(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.

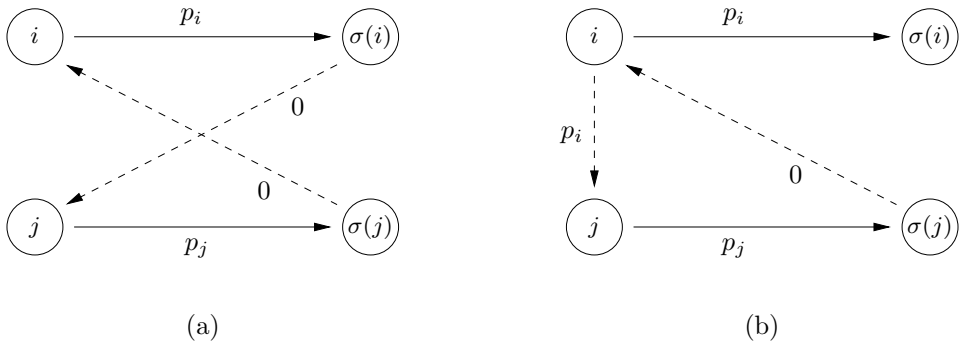

Figure 3.1: A pair of alternative arcs

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 3.1(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 [24]). 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. $V = \{O_{ij} \mid i = 1, \ldots, n_j; j = 1, \ldots, n\} \cup \{\circ, *\}$). 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 $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{n_j j}$ between the operations of each job $j = 1, \ldots, n$. The arc $O_{ij} \rightarrow O_{i+1,j}$ is weighted by the processing time p_{ij} (for $i = 1, \ldots, n_j - 1$). Furthermore, in F we have arcs $\circ \rightarrow O_{1j}$ and $O_{n_j j} \rightarrow *$ 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.

In the classical disjunctive graph model, usually the nodes represent the processing of the operations. Therefore, a node i is weighted with the processing time p_i of operation i. An arc (i, j) between operations i and j induces that operation j cannot start before operation i is finished, i.e. it represents the finish-start relation $S_i + p_i \leq S_j$ between i and j. In the case of the job-shop problem with blocking operations, additionally also start-start relations of the form $S_k \leq S_l$ between two operations k and l have to be modelled. Expanding the classical node-weighted model, this could be achieved by introducing an arc (k, l) with weight $-p_k$. In this way, a job-shop problem with blocking operations could be represented by a graph with node and arc weights.

The graph G we introduced is an equivalent representation where all node weights are omitted and each arc (i, j) representing a finish-start relation is weighted by p_i and each arc (i, j) representing a start-start relation is weighted by 0. This simplifies further presentations and is the common way to represent alternative graphs.

In the following, we consider an example for a job-shop problem with blocking operations and show up the corresponding alternative graph.

Example 3.1 : We consider an instance with 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.

Figure 3.2 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. □

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.

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

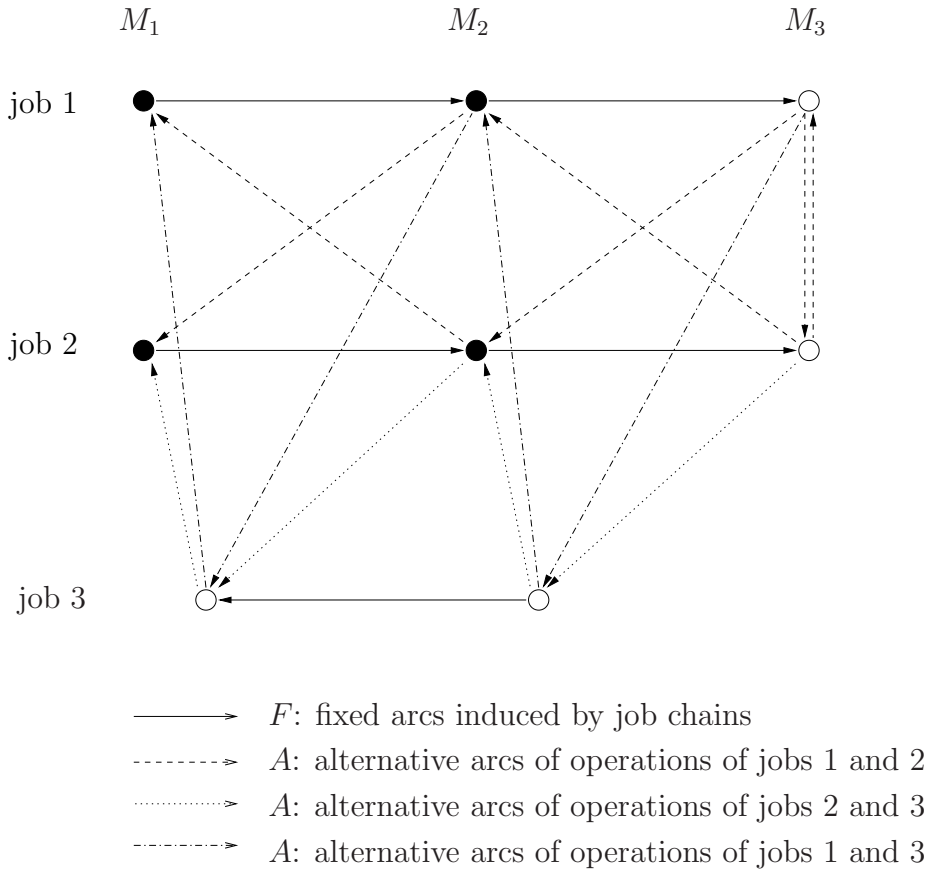

Figure 3.2: An alternative graph $G = (V, A, F)$

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 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.

Example 3.2 : Assume that the jobs in Example 3.1 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 3.3. By longest path calculations in $G(S)$, the schedule $P(S)$ of Figure 3.4 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.

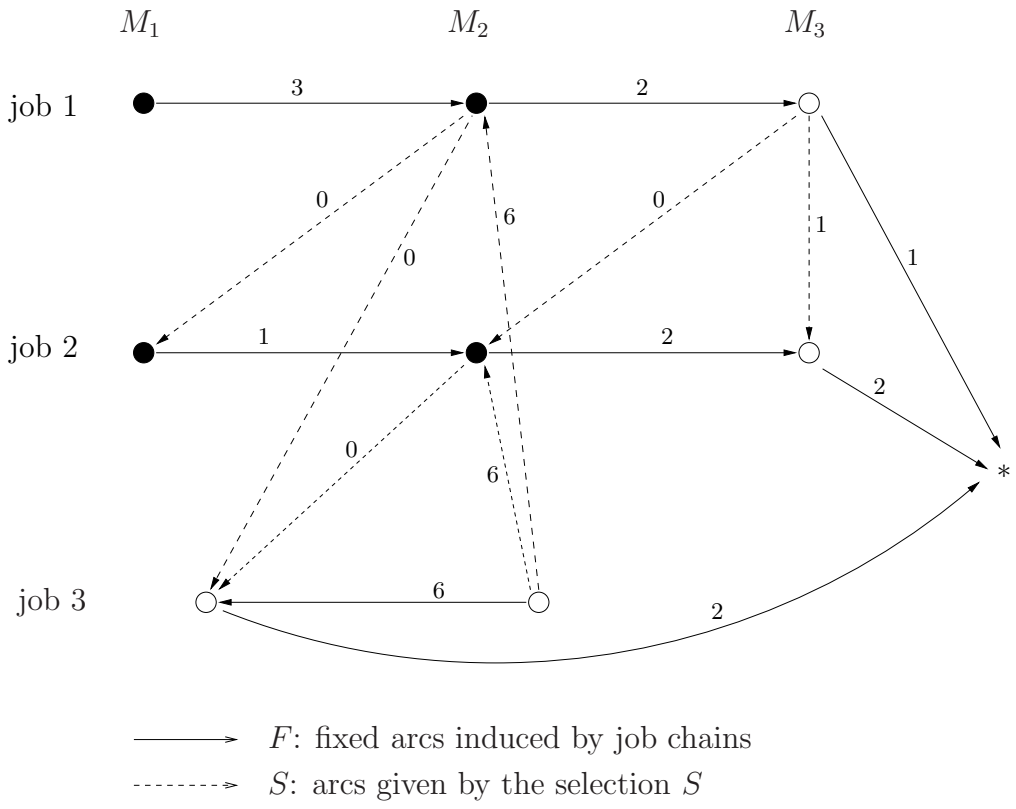

Figure 3.3: The graph $G(S) = (V, F \cup S)$

□

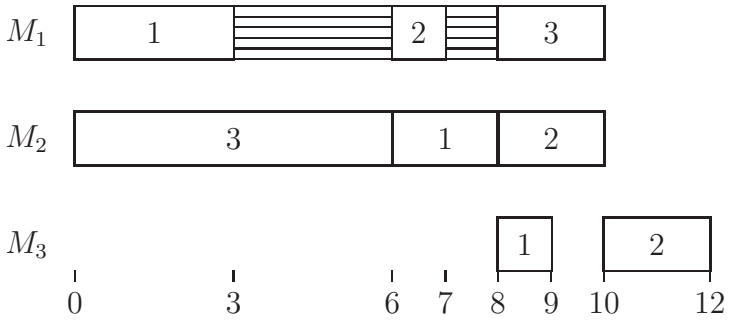

Figure 3.4: Schedule $P(S)$

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 $(j = 1, \ldots, n)$ 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 Example 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 $\boldsymbol{B^1}, \boldsymbol{B^2}, \ldots,$ $\boldsymbol{B^b}$**, 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. Input and output sequences are abstract codings which can be used in order to calculate a buffer slot assignment. Associated buffer slot sequences result from the order in which the jobs assigned to the same buffer slot are sequenced in π_{in} and π_{out}. Thus, π_{in} and π_{out} can be considered as a merging of buffer slot sequences together with all jobs not using buffer B.

We will show that the machine sequences for all machines and the buffer input and output sequences for all buffers can be used as representatives for a dominant set of schedules. Here, a dominant set of schedules is a subset of all schedules containing at least one optimal schedule.

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 4.1 : 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, $i + b - 1 - r$ other jobs have entered buffer B and $i - 1 - r$ jobs have left it, where r is the number of jobs which did not use the buffer. 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, $(i + b + k - 1) - (i - 1) = b + k$ 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. □

Note, that the condition in (a) is sufficient to characterize jobs not entering buffer B, but not necessary. This means dependent on the processing times of a given instance also other jobs than those characterized in (a) may not go through buffer B. Theorem 4.1 characterizes jobs not entering buffer B only on the basis of π_{in} and π_{out}.

Furthermore, in the case that the buffer capacity b is equal to 0, the condition in (b) is equivalent to $\pi_{out}(i) = \pi_{in}(i)$ for all positions i. Obviously, if $b = 0$, this is valid for an input and output sequence corresponding to a feasible schedule. Nevertheless, in the case of a buffer B with $b = 0$ we need not specify π_{in} and π_{out} for the solution representation. Clearly, an operation assigned to a buffer B with $b = 0$ is a blocking operation. As we have seen in Subsection 3.1, a solution of a job-shop problem with blocking operations can be represented only by the sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines.

From Theorem 4.1 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 Condition (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 π_{in} is not empty DO BEGIN
2. Let j be the first job in π_{in};
3. IF $j = \pi_{in}(i+b) = \pi_{out}(i)$ THEN
4. Put j on the next machine and delete j both from π_{in} and π_{out};
   ELSE
5. Put j in the first free buffer slot and delete j from π_{in};
6. WHILE the job k in the first position of π_{out} is in the buffer DO
7. Delete k both from the buffer and from π_{out};
   END

The following example shows how this algorithm works.

Example 4.1 : 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 job 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. Finally, jobs 6 and 1 are deleted from the buffer and from π_{out}. 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}.

Let us assume that k is in a position less than $i + b$ in the input sequence π_{in}, i.e. $k = \pi_{in}(j)$ with $j < i + b$. Then, at the time job k leaves its machine, $j - 1 - l$ other jobs have entered the buffer and $i - 1 - l$ jobs have left it, where l is the number of jobs which did not use the buffer. Thus, $j - 1 - l - (i - 1 - l) = j - i$ jobs are in the buffer at this time. Since $j - i < b$, the buffer would not be full in this case. Therefore, 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 Example 4.1, the buffer slot sequence of B^2 is $(2, 4, 6)$.

Let now P be an arbitrary feasible schedule for a job-shop problem with general buffers. P 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. We get π^B_{in} from P by ordering all jobs assigned to buffer B according to the time at which they leave their machine to enter buffer B or to move directly to the next machine. Similarly, we get π^B_{out} from P by ordering these jobs according to the time at which they start processing on their succeeding machine. If two jobs leave their machine at the same time or start processing on their succeeding machine at the same time, their order is arbitrary in π^B_{in} and π^B_{out}, respectively. If we apply to π^B_{in} and π^B_{out} the buffer slot assignment procedure, we may get an assignment for buffer B which is different to the assignment in the initial schedule P. This is due to the assignment policy of Algorithm Buffer Slot Assignment (see line 5): If several buffer slots are empty a job is assigned to the buffer slot which became free first (or which was not yet occupied). This policy is called **first free buffer slot rule** and ensures that the buffer slot of B which was assigned to a job j is not occupied at the time when j leaves its machine to enter buffer B. This can be seen as follows:

Assume that in the schedule P job j leaves its machine at time t to enter buffer B and that a buffer slot B^* was assigned to job j by the buffer slot assignment procedure. Furthermore, assume that at time t, the buffer slot predecessor i of job j has not yet left B^*. Since job j is assigned to B^* and jobs are always assigned to the first free buffer slot by the procedure, all other buffer slots of B are occupied at time t and will be emptied after job i leaves B^* (i.e. when job i starts processing on its succeeding machine in P). The $b-1$ jobs occupying these buffer slots must leave their machines to enter buffer B until time t in P and do not start on their succeeding machines in P before job i continues processing on its succeeding machine. Thus, in P during time t and the time when job i starts processing on its succeeding machine there are $b-1$ jobs in buffer B apart from jobs i and j. Since this exceeds the buffer capacity of B, this is a contradiction to the feasibility of schedule P. Therefore, buffer slot B^* must be empty at time t.

Thus, with the assignment obtained by the Algorithm Buffer Slot Assignment and

the same machine sequences as in P we get a feasible schedule where its makespan is as long as that of P. Since the assignment obtained by the Algorithm Buffer Slot Assignment may be even more balanced than that of P, with the computed assignment the jobs may start processing on the machines even earlier than they do in P. 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 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 $\mu(i)$ and is called **direct blocking arc**. This situation is depicted in Figure 4.1.

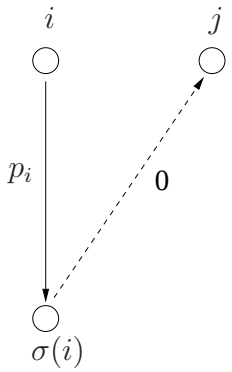

Figure 4.1: Situation where $J(i)$ is not assigned to the buffer

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 4.2 (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. 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.

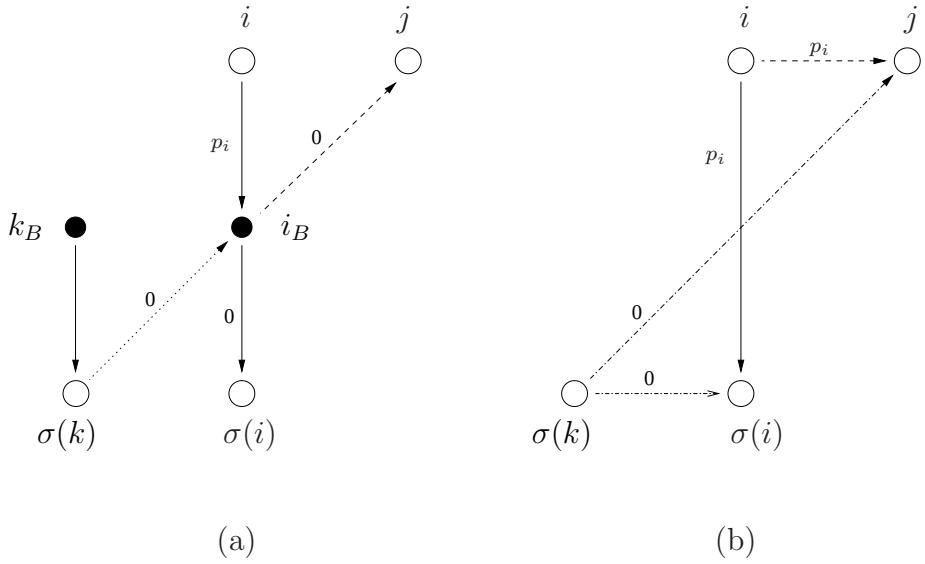

Figure 4.2: (a) Buffer slot operation vertex i_B with its incoming and outgoing arcs (b) Simplification of (a) by deleting i_B

- 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 4.2 (b). We call the simplified graph $\bar{\boldsymbol{G}}(\boldsymbol{\Pi})$ **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. [3]). 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 Specific buffer models

In the following, we discuss specific buffer models introduced in Section 2.2. We will show how the solution representation given in the previous section for the job-shop problem with general buffers specializes in connection with specific buffer models. For each specific buffer model we investigate whether it is possible to compute an optimal schedule respecting given sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines in polynomial time. In the first two subsections, we consider the flow-shop problem with intermediate buffers and the job-shop problem with pairwise buffers. In both cases the buffer input and output sequences are given by the machine sequences. For the output buffer model, we present a polynomial procedure to compute an optimal schedule respecting given sequences $\pi^1, \ldots, \pi^m$. In Subsection 5.4, the job-shop problem with input buffers is reduced to the output buffer model. Finally, we show that for general buffers the problem of finding an optimal schedule respecting given sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines is $\mathcal{NP}$-hard in the strong sense.

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} $(i = 1, \ldots, m)$ 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 $k = 1, \ldots, m-1$. If π^i is the sequence of the jobs on machine M_i $(i = 1, \ldots, m)$, 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 $\pi^1, \ldots, \pi^m$ are sufficient to represent a solution in the case of a flow-shop problem with intermediate buffers.

5.1.1 Characterization of feasible solutions

In the classical flow-shop situation each arbitrary combination of permutations of the jobs on the machines leads to a feasible schedule. However, if limited buffers are given, not all combinations of job permutations yield a feasible schedule. Due to Theorem 4.1, we conclude that if we have a feasible schedule then for each $k \in \{1, \ldots, m-1\}$ the machine sequences π^k and π^{k+1} must satisfy the conditions

$$\pi^{k+1}(i) \in \{\pi^k(1), \ldots, \pi^k(i + b_k)\} \quad \text{for all positions } i. \tag{5.1}$$

Conversely, if Condition (5.1) holds, the Algorithm Buffer Slot Assignment yields an allocation of the buffer such that the buffer capacity will not be exceeded at any time point. Thus, feasible solutions can be characterized by the following Lemma.

Lemma 5.1 : The permutations $\pi^1, \ldots, \pi^m$ correspond to a feasible schedule for the flow-shop problem with intermediate buffers of capacities $b_1, \ldots, b_{m-1}$ iff

$$\pi^{k+1}(i) \in \{\pi^k(1), \ldots, \pi^k(i + b_k)\} \qquad \text{for } k = 1, \ldots, m-1;\ i = 1, \ldots, n - b_k \qquad \square$$

Note, that in the case $b_k = 0$, Condition (5.1) is equivalent to $\pi^{k+1}(i) = \pi^k(i)$ for $i = 1, \ldots, n$, i.e. that the jobs must be processed in the same sequence on M_k and M_{k+1}. Thus, Lemma 5.1 is also valid if $b_k = 0$ holds for any $k = 1, \ldots, m-1$.

If the conditions of Lemma 5.1 are fulfilled we call the machine sequences $\boldsymbol{\pi^1, \ldots, \pi^m}$ **compatible with the buffer capacities** $\boldsymbol{b_1, \ldots, b_{m-1}}$. Based on the result of Lemma 5.1, it is possible to determine the number of feasible solutions for a flow-shop problem with intermediate buffers.

Theorem 5.1 : For the problem of scheduling n jobs in an m machine flow-shop problem with buffer capacities $b_1, \ldots, b_{m-1}$, where $0 \leq b_k \leq n$ holds for $k = 1, \ldots, m-1$, the number of different feasible solutions N_{buf} is given by

$$N_{buf} = n! \prod_{k=1}^{m-1} b_k\,!\ (b_k + 1)^{n-b_k}.$$

Proof: Obviously, the jobs can be scheduled in $n!$ different orders on M_1. For each fixed permutation π^1, according to Lemma 5.1, we get for π^2 the following restrictions:

$$\pi^2(i) \in \{\pi^1(1), \ldots, \pi^1(b_1 + i)\} \backslash \{\pi^2(1), \ldots, \pi^2(i-1)\} \quad \text{for} \quad i = 1, \ldots, n - b_1$$

and

$$\pi^2(i) \in \{1, \ldots, n\} \backslash \{\pi^2(1), \ldots, \pi^2(i-1)\} \quad \text{for} \quad i = n - b_1 + 1, \ldots, n.$$

Thus, for $i = 1, \ldots, n-b_1$ we have $b_1 + i - (i-1) = b_1 + 1$ different possibilities for job $\pi^2(i)$ and for $i = n - b_1 + 1, \ldots, n$ we have $n - i + 1$ different possibilities for job $\pi^2(i)$. Summarizing, this yields $(b_1 + 1)^{n-b_1} \cdot b_1 \cdot (b_1 - 1) \cdot \ldots \cdot 1 = (b_1 + 1)^{n-b_1} \cdot b_1!$ possible permutations π^2 for each fixed permutation π^1. If we continue this argumentation for each following machine, the result of the theorem follows. $\square$

From the result and the proof of Theorem 5.1 we immediately get

Corollary 5.1 :

1. If $b_k = n-1$ or $b_k = n$ holds for all $k = 1, \ldots, m-1$, we get $N_{buf} = (n!)^m$ and the flow-shop problem with buffers reduces to a classical flow-shop problem.
2. If $b_k = 0$ holds for all $k = 1, \ldots, m-1$, we get $N_{buf} = n!$ and the flow-shop problem with buffers reduces to a permutation flow-shop problem where all operations are blocking. □

Thus, the buffer capacities somehow determine how much the structure of a solution may differ from a permutation solution.

5.1.2 A simplified graph model

In the following, we assume that a solution Π defined by machine sequences $\pi^1, \ldots, \pi^m$ which are compatible with the buffer capacities $b_1, \ldots, b_{m-1}$ is given. For the case of the job-shop problem with general buffers, we proposed in Section 4.3 a graph $G(\Pi)$ by which a corresponding schedule respecting the sequences $\pi^1, \ldots, \pi^m$ can be achieved using longest paths calculations. In this section, we discuss how the graph $G(\Pi)$ and the solution graph $\bar{G}(\Pi)$ specializes in the case of a flow-shop problem with buffers. Furthermore, we show that the feasibility conditions derived in Lemma 5.1 guarantee that the solution graph has no cycles.

Example 5.1 : Figure 5.1 shows an example of the graph $G(\Pi)$ for a flow-shop problem with three machines, five jobs and two buffers which have a capacity of $b_1 = 1$ and $b_2 = 2$ units. A corresponding solution Π is given by machine sequences $\pi^1 = (1, 2, 3, 4, 5)$, $\pi^2 = (2, 1, 3, 5, 4)$ and $\pi^3 = (3, 1, 4, 5, 2)$ which obviously fulfill the feasibility condition (5.1). 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 are shown vertically, where the positions of the black circles also indicate the corresponding buffer slot assignment.

Figure 5.2 shows the resulting solution graph $\bar{G}(\Pi)$ after the buffer slot vertices have been eliminated. Obviously, in $\bar{G}(\Pi)$ almost all operations processed consecutively on the same machine are connected by so-called **machine arcs**. However, we have no arc from operation O_{12} to operation O_{13} on M_1 and from O_{23} to O_{25} on M_2 though these operations are processed consecutively on M_1 and M_2, respectively. In fact, these two machine arcs are transitive. As job 2 goes directly from M_1 to M_2, in $G(\Pi)$ (and also in $\bar{G}(\Pi)$) there is a job arc from O_{12} to O_{22} and a direct blocking arc from O_{22} to O_{13}, the direct machine successor of job 2 on M_1. Thus, a machine arc from O_{12} to O_{13} is transitive. Similarly, in $G(\Pi)$ (and also in $\bar{G}(\Pi)$) we have a job arc from O_{23} to O_{33} and a direct blocking arc from O_{33} to O_{25}. Thus, a machine arc from O_{23} to O_{25} is transitive. Furthermore, by deleting the buffer slot vertex of job 4 in $G(\Pi)$

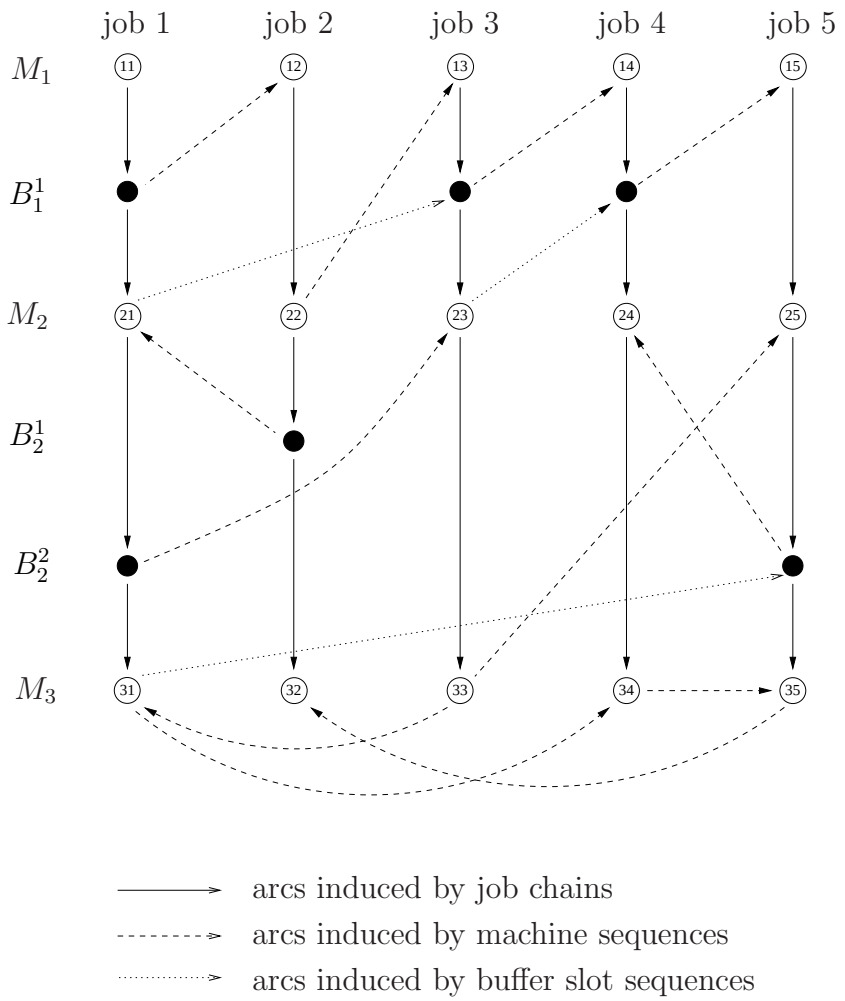

Figure 5.1: An example of the graph $G(\Pi)$

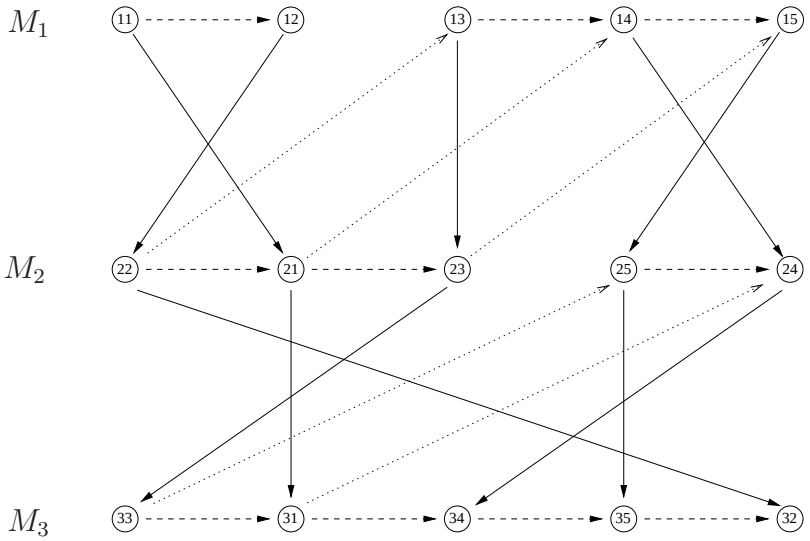

Figure 5.2: Graph $\bar{G}(\Pi)$ after elimination of buffer slot vertices in Figure 5.1

an arc from operation O_{23} to operation O_{24} on M_2 is induced. Similarly, by deleting the buffer slot vertex of job 5 in $G(\Pi)$ an arc from operation O_{31} to operation O_{35} on M_3 is induced. Since both of these arcs are transitive, they are omitted in $\bar{G}(\Pi)$. Note that arcs going from operations on M_{k+1} to operations on M_k show a regular structure. □

In the following, we assume that machine arcs induced by $G(\Pi)$ and connecting operations which are not processed consecutively on the same machine are omitted in $\bar{G}(\Pi)$ (see Example 5.1). We will show later on that these arcs are transitive in $\bar{G}(\Pi)$. Next, we show that buffer slots can always be assigned in such a way that the solution graph $\bar{G}(\Pi)$ consists of the following arcs:

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

Machine arcs and job arcs are weighted by the processing time of the corresponding operations whereas buffer arcs have weight 0. For simplification, we assume that we have machine arcs between all operations processed consecutively on the same machine in $\bar{G}(\Pi)$. As we have seen in Example 5.1, a machine arc connecting an operation which does not use the buffer with its direct machine successor can be omitted.

The buffer arcs can be derived by simplifications from the graph $G(\Pi)$ in the following way: For a buffer B_k with $b_k > 0$, we assume that the jobs are assigned to the buffer slots by the Algorithm Buffer slot assignment (see Section 4.2). The algorithm scans the sequences π^k and π^{k+1} from the first to the last position. In order to get a deeper insight in the assignment policy of the algorithm, look at the following example.

Example 5.2 : We consider two machines M_k and M_{k+1} with an intermediate buffer B_k of capacity $b_k = 2$. The jobs are processed on M_k in the order $\pi^k = (1, 2, 3, 4, 5, 6, 7, 8)$ and on M_{k+1} in the order $\pi^{k+1} = (3, 2, 1, 6, 5, 7, 4, 8)$. Figure 5.3 shows the buffer slot assignment achieved by the Algorithm Buffer slot assignment.

After the first two jobs of π^k, jobs 1 and 2, are assigned to the buffer slots B_k^1 and B_k^2, job 3 goes directly from M_k to M_{k+1}. Both buffer slots are occupied now and the current first elements of π^k and π^{k+1} are job 4 and job 2, respectively. Then, we eliminate job 2 and afterwards job 1 both from the buffer and from π^{k+1}. At this time, both buffer slots are empty and the buffer slot which became free first is the second buffer slot. Therefore, in the next step, job 4 is assigned to buffer slot B_k^2 and thus, job 5 is assigned to buffer slot B_k^1. Then, job 6 goes directly to the next

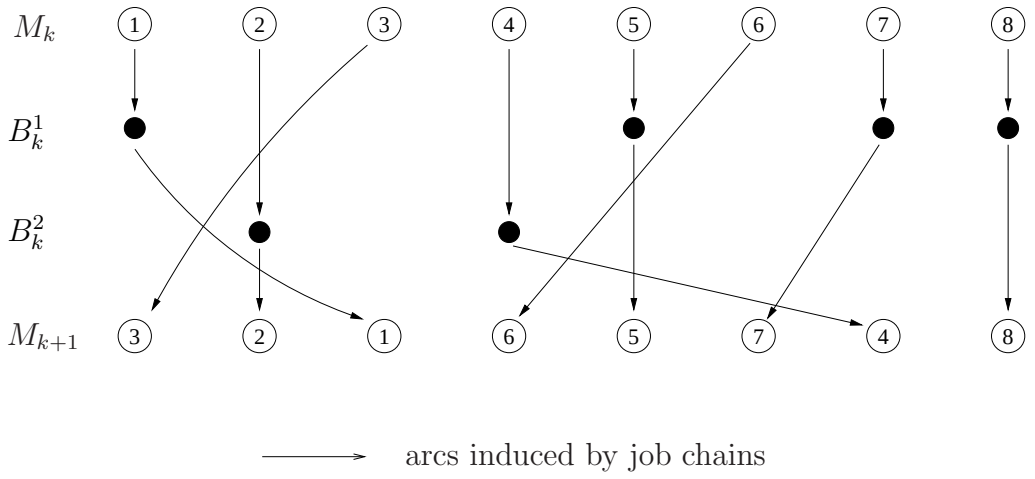

Figure 5.3: Buffer slot assignment

machine. After job 5 is eliminated from B_k^1 and π^{k+1}, job 7 is assigned to the free buffer slot B_k^1. Again, both buffer slots are occupied now. According to the sequence in π^{k+1} we eliminate first job 7 and next job 4 from the buffer and π^{k+1}. Note, that again both buffer slots are empty but the buffer slot which became free first is B_k^1 this time. Therefore, the last job in π^k is assigned to B_k^1. □

The example makes clear that if several buffer slots are empty a job is assigned to the buffer slot which became free first (or which was not yet occupied). Obviously, this **first free buffer slot rule** leads to a specific buffer slot assignment which is important in connection with the definition of the buffer arcs. In order to derive the definition of the buffer arcs now, we differentiate whether a job is assigned to buffer B_k or not. Consider the latter case first:

If $j = \pi^k(i + b_k) = \pi^{k+1}(i)$ holds for a job j, then job j is not assigned to buffer B_k (see Theorem 4.1). Thus, in the graph $G(\Pi)$ we have an arc from the job successor of j on M_k, which is $\pi^{k+1}(i)$, to the operation processed immediately after j on M_k, which is $\pi^k(i + b_k + 1)$ (see Section 4.3). This arc corresponds to the buffer arc $\pi^{k+1}(i) \rightarrow \pi^k(i + b_k + 1)$. (Notice, that if $b_k = 0$ the situation is similar.)

Otherwise, if $\pi^k(i + b_k) \neq \pi^{k+1}(i)$, job $j = \pi^{k+1}(i)$ is assigned to a buffer slot B_k^l of B_k. Consider the situation that $j = \pi^{k+1}(i)$ is the current first element of π^{k+1} and thus, is removed from the buffer slot B_k^l. Denote the time when job j leaves B_k^l to start processing on M_{k+1} by t. In the following, we show that job $\pi^k(i + b_k)$ enters buffer slot B_k^l next. Since jobs are always assigned to the first free buffer slot by the procedure, all buffer slots, which were emptied before buffer slot B_k^l, are reoccupied before B_k^l. Thus, there is no buffer slot different to B_k^l which is empty at time t and which is not reoccupied before B_k^l. Therefore, $b_k - 1$ jobs are in B_k at time t or are inserted into B_k after t but before buffer slot B_k^l is occupied again. Furthermore, $i - 1$ other jobs have finished processing on M_{k+1} at time t since $j = \pi^{k+1}(i)$. Thus,

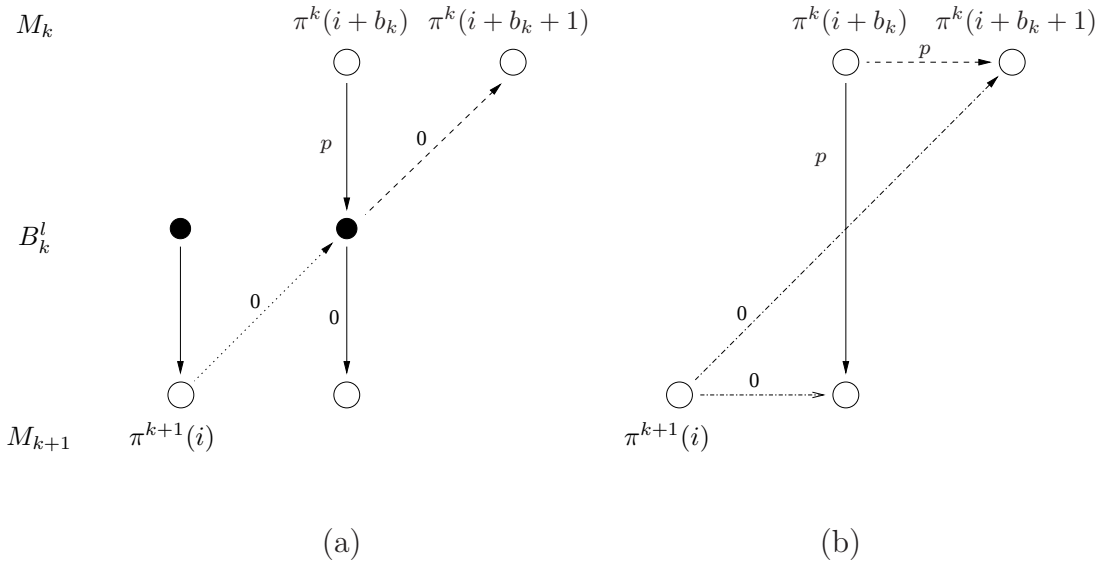

Figure 5.4: (a) Buffer slot operation vertex in $G(\Pi)$ in the case of a flow-shop problem (b) Simplification of (a)

$b_k - 1 + i$ jobs (including job j) are in B_k or on M_{k+1} or have finished on M_{k+1} when B_k^l is reoccupied. Consequently, job $\pi^k(i + b_k)$ must be in the current first position of π^k and is put in buffer slot B_k^l. This means, that job $\pi^k(i + b_k)$ is the buffer slot successor of job $j = \pi^{k+1}(i)$. Consequently, in the graph $G(\Pi)$ we have the arcs as indicated in Figure 5.4 (a). Obviously, by deleting the buffer slot vertices we get the simplification shown in Figure 5.4 (b). In fact, Figure 5.4 (b) shows a job arc, a buffer arc, a machine arc on M_k and a machine arc on M_{k+1} with weight 0 which is transitive in the flow-shop case.

Thus, in the special case of a flow-shop problem with intermediate buffers the solution graph $\bar{G}(\Pi)$ has a relatively easy structure. In the following theorem, we show that $\bar{G}(\Pi)$ is acyclic if and only if the given machine sequences $\pi^1, \ldots, \pi^m$ are compatible with the buffer capacities. Here, $G(\pi^k, \pi^{k+1})$ is the subgraph of $\bar{G}(\Pi)$ which contains all vertices representing operations that have to be processed on machines M_k and M_{k+1} and all arcs connecting these operations.

Theorem 5.2 : The following statements are equivalent:

(a) For each $k = 1, \ldots, m-1$ the permutations π^k and π^{k+1} are compatible with the buffer capacity b_k.

(b) The graph $\bar{G}(\Pi)$ has no cycle.

Proof: First we show that statement (b) follows from (a). Consider an arbitrary buffer arc b and an arbitrary job arc j between machines M_k and M_{k+1} in $\bar{G}(\Pi)$. Let

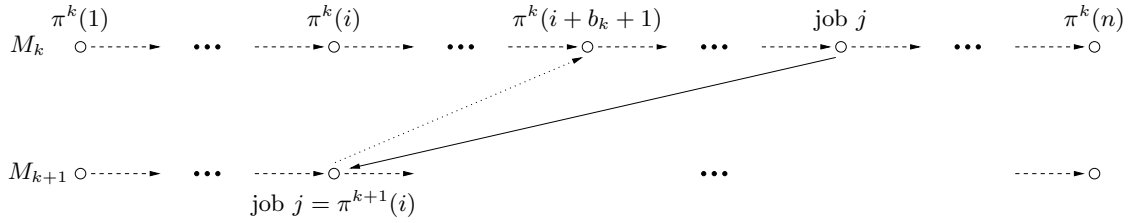

Figure 5.5: Situation, in which π^k and π^{k+1} are not compatible with b_k

$\Delta_b^{k,k+1}$ be the difference between the position of the end vertex of b in permutation π^k and the position of the start vertex of b in permutation π^{k+1}. Since b is defined by $\pi^{k+1}(i) \rightarrow \pi^k(i+b_k+1)$ for $i \in \{1, \ldots, n-b_i-1\}$, it is $\Delta_b^{k,k+1} = b_k+1$. Furthermore, let $\Delta_j^{k,k+1}$ be the difference between the positions of the end vertex of j in permutation π^{k+1} and the start vertex of j in permutation π^k. As the permutations π^k and π^{k+1} are compatible with b_k, it holds $\Delta_j^{k,k+1} \geq -b_k > -\Delta_b^{k,k+1}$ and thus, it is

$$\Delta_j^{k,k+1} + \Delta_b^{k,k+1} > 0 \tag{5.2}$$

For an arbitrary machine arc m of machine M_k in $\bar{G}(\Pi)$ the difference Δ_m between the positions of the end and start vertices of m in π^k is 1. Assume now that $\bar{G}(\Pi)$ contains a cycle C. Obviously, the sum of the Δ-values of all arcs belonging to C must be 0. Furthermore, in C the number of buffer arcs from machine M_{k+1} to M_k is equal to the number of job arcs from M_k to M_{k+1}. Due to (5.2) and since $\Delta_m > 0$ for all machine arcs, the sum of the Δ-values of all arcs belonging to C must be positive which is a contradiction.

To show that statement (a) follows from (b), we assume that the permutations π^k and π^{k+1} are not compatible with b_k. Thus, according to Lemma 5.1 there exists a position $i < n - b_k$ such that job $j = \pi^{k+1}(i)$ is in a position larger than $b_k + i$ in permutation π^k. The job arc corresponding to job j together with the buffer arc from the operation in position i on M_{k+1} to the operation in position $i + b_k + 1$ on M_k and certain machine arcs on M_k form a cycle in $G(\pi^k, \pi^{k+1})$. Such a situation is shown in Figure 5.5. □

Due to Theorem 5.2 the solution graph contains no cycle when the permutations π^k and π^{k+1} are compatible with the buffer capacity b_k for $k = 1, \ldots, m-1$. For each k, the compatibility Condition (5.1) can be checked in $O(n)$ time by applying the Algorithm Buffer Slot Assignment. 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. In Section 7.2.1, we will describe how the longest path calculations can be implemented efficiently.

Leisten [20] has shown that for the inverted problem (i.e. changing the machine sequence from $M_1, \ldots, M_m$ to $M_m, \ldots, M_1$ while keeping the buffer size between each

two consecutive machines the same and inverting any given job sequence $\pi^k(1), \ldots, \pi^k(n)$) into $\pi^k(n), \ldots, \pi^k(1)$) for $k = 1, \ldots, m$) a critical path has the same length as in the original problem.

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} of capacity 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. If the capacity of B_{kl} is equal to 0, sequences π_{in}^{kl} and π_{out}^{kl} can be derived from π^k and π^l in the same way.

Using the subsequences π_{in}^{kl} and π_{out}^{kl} for each buffer we get a simplified graph $\bar{G}_{kl}(\Pi)$ similar to the flow-shop situation (see Figure 5.2). The solution graph $\bar{G}(\Pi)$ for given machine sequences $\pi^1, \ldots, \pi^m$ is a decomposition of all simplified graphs $\bar{G}_{kl}(\Pi)$ where the simplified graphs may be connected in $\bar{G}(\Pi)$ by additional arcs reflecting precedence relations due to machine permutations or blocking situations. In detail, $\bar{G}(\Pi)$ contains job and machine arcs which are defined in the same way as in the case of the flow-shop problem with intermediate buffers. A buffer arc of buffer B_{kl} in $\bar{G}(\Pi)$ is defined by

$$\pi_{out}^{kl}(i) \;\rightarrow\; \rho_\Pi\left(\pi_{in}^{kl}(i + b_{kl})\right) \quad \text{for all positions } i$$

where $\rho_\Pi(j)$ is the direct machine successor of operation j with respect to the solution Π. This buffer arc reflects a blocking situation on machine M_k where after its completion job $\pi_{in}^{kl}(i + b_{kl})$ blocks M_k since buffer B_{kl} and machine M_l are both occupied. At the next time a job assigned to buffer B_{kl} starts processing on M_l, i.e. when job $\pi_{out}^{kl}(i)$ starts processing on M_l, the blocking period on M_k is finished and the direct machine successor of the blocking job, which is $\rho_\Pi\left(\pi_{in}^{kl}(i + b_{kl})\right.$, may start processing on M_k. Note, that during the blocking period b_{kl} jobs are in buffer B_{kl}. Therefore, if the job blocking machine M_k is in position $i + b_{kl}$ of the input sequence π_{in}^{kl}, the next job which starts processing on M_l and is assigned to B_{kl} must be the job in position i of the output sequence π_{out}^{kl}.

Applying the above definition to an intermediate buffer B_k of machines M_k and M_{k+1} yields the definition $\pi^{k+1}(i) \rightarrow \pi^k(i + b_k + 1)$ of a buffer arc of buffer B_k in the case of a flow-shop problem with intermediate buffers. Since in this case the input and output buffer sequences are identical with the machine sequences, the definition of a buffer arc is much simpler than in the more general case of a job-shop problem with pairwise buffers. For the flow-shop problem with intermediate buffers, the machine

successor of (the blocking) job $\pi^k(i + b_k)$ is the job in the next position of π^k, i.e. job $\pi^k(i + b_k + 1)$. For the job-shop problem with pairwise buffers, the buffer input and output sequences are partial sequences of the machine sequences. Therefore, the machine successor of (the blocking) job $\pi_{in}^{kl}(i + b_{kl})$ must not be contained in the buffer input sequence π_{in}^{kl} in general. It is the job which is positioned in the machine sequence π^k directly after job $\pi_{in}^{kl}(i + b_{kl})$ and is denoted by $\rho_\Pi\,(\pi_{in}^{kl}(i + b_{kl}))$.

For the flow-shop problem with intermediate buffers, we showed that the solution graph is acyclic iff the machine sequences $\pi^1, \ldots, \pi^m$ are compatible with the buffer capacities. 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. In the following, we show up an example where conditions (5.1) are fulfilled for all sequences π_{in}^{kl} and π_{out}^{kl}, but the corresponding solution graph $\bar{G}(\Pi)$ contains a positive cycle.

Example 5.3 : We consider an instance of a job-shop problem with three machines where all pairwise buffers B_{12}, B_{13}, B_{21}, B_{23}, B_{31}, and B_{32} have capacity 0. On the machines, three jobs have to be processed where jobs 1 and 2 consist of three operations each and job 3 consists of two operations. Jobs 1 and 2 have to be scheduled first on M_1, then on M_2 and last on M_3, whereas job 3 has to be scheduled first on M_1 and then on M_3. All operations have processing time 1. Figure 5.6 shows the solution graph $\bar{G}(\Pi)$ for the machine sequences $\Pi = (\pi^1, \pi^2, \pi^3)$ where $\pi^1 = (1, 2, 3)$, $\pi^2 = (1, 2)$, and $\pi^3 = (3, 1, 2)$.

Since the jobs only change from M_1 to M_2 or M_3, respectively, and from M_2 to M_3, we have to check conditions (5.1) only for three buffer input and output sequences. It is $\pi_{in}^{12} = (1, 2) = \pi_{out}^{12}$, $\pi_{in}^{13} = (3) = \pi_{out}^{13}$, and $\pi_{in}^{23} = (1, 2) = \pi_{out}^{23}$. Thus, conditions (5.1) are fulfilled and each simplified graph $\bar{G}_{12}(\Pi)$, $\bar{G}_{13}(\Pi)$, and $\bar{G}_{23}(\Pi)$ is cyclefree. Nevertheless, the solution graph $\bar{G}(\Pi)$ contains a positive cycle consisting of the job arc from O_{13} to O_{23}, the machine arc from O_{23} to O_{31} and the arcs from O_{31} to O_{22} as well as from O_{22} to O_{13}. The latter two arcs are induced by blocking situations on M_2 and M_1, respectively. □

The existence of positive cycles in the solution graph is not only due to the buffers since even in the case of the classical job-shop problem the solution graph may contain positive cycles. Furthermore, in contrast to the flow-shop situation, the solution graph may contain cycles of length 0 in the case of the job-shop problem with pairwise buffers. Since in the solution graph job and machine arcs have a positive weight, a cycle of length 0 consists only of arcs ensuring that the blocking restrictions are respected. Such a cycle due to a blocking situation over several machines is called blocking cycle. Obviously, all operations in a blocking cycle have to start at the same time.

The flow-shop problem with intermediate buffers can be seen as a special case of the job-shop problem with pairwise buffers where the jobs have a common route

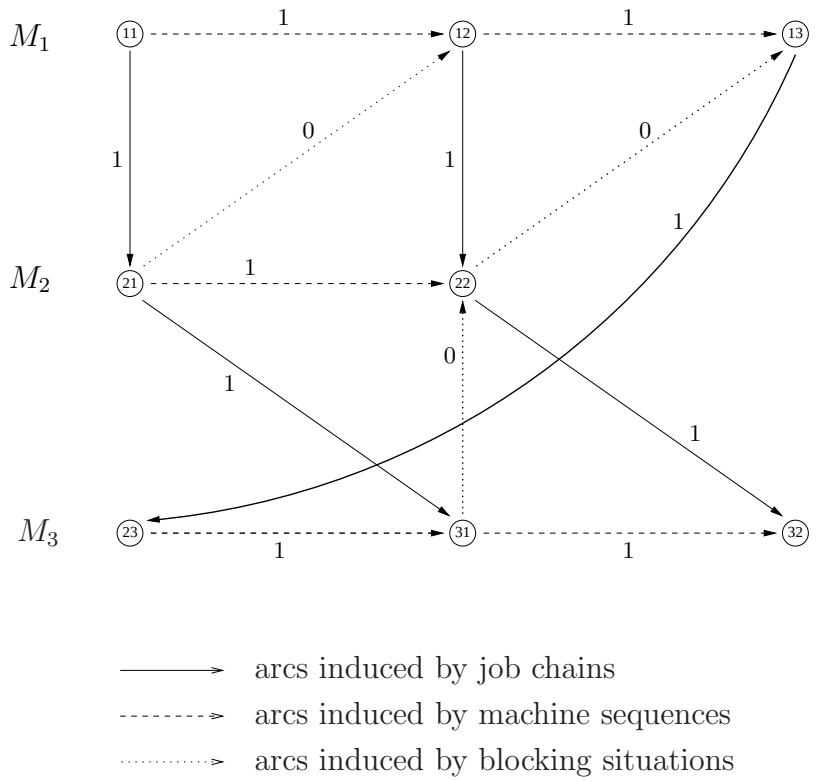

Figure 5.6: A solution graph $\bar{G}(\Pi)$ with a positive cycle

through the machines. Therefore, the solution graph of a flow-shop problem with intermediate buffers has a special structure and does not contain blocking cycles of length 0. Thus, testing feasibility and calculating a schedule for sequences $\pi^1, \ldots, \pi^m$ is less time consuming in this special case. In a job-shop problem with pairwise buffers, 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 [26], 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 $\pi^1_{in}, \ldots, \pi^m_{in}$ and the buffer output sequences $\pi^1_{out}, \ldots, \pi^m_{out}$. Clearly, the buffer input sequence π^k_{in} 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 $\pi^1_{out}, \ldots, \pi^m_{out}$

have to be specified. In the following, we show that buffer output sequences can be derived from the machine sequences $\pi^1, \ldots, \pi^m$. 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 $\nu = 0, \ldots, r-1$,
- 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 π^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. $t := \min\{t_i \mid i \in C; t_i > t\}$;

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 (C, 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 $Z : i_0, \ldots, i_{r-1}$ with $i_\nu \in C$ and $t_{i_\nu} \leq t$ 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 $\nu = 0, \ldots, r-1$ 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 $(\mathbf{C}, \mathbf{t}, \mathbf{Z})$
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_ν from the machine sequence for $\mu(i_\nu)$;
4. Move i_ν into buffer $\beta(i_\nu)$;
END
ELSE BEGIN
5. Eliminate i_ν 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.

Example 5.4 : 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 5.1, for each operation O_{ij} its processing time p_{ij} and the machine μ_{ij} on which O_{ij} must be processed 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 5.1: Instance of a job-shop problem with output buffers

Figure 5.7 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

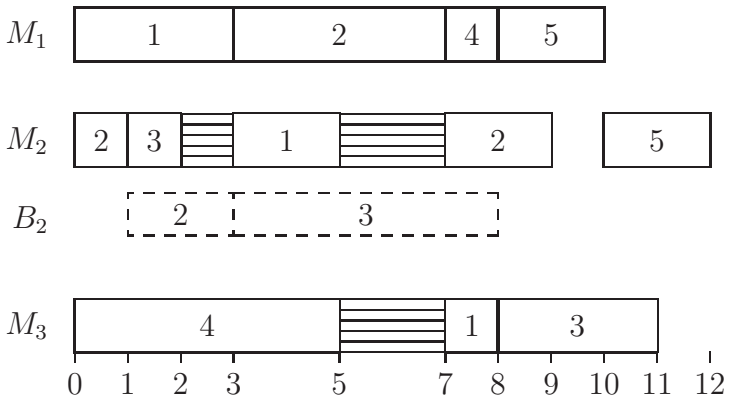

Figure 5.7: Schedule for a job-shop problem with output buffers

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 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 5.2. 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 $\pi^1, \ldots, \pi^m$ 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 the buffer sequences. Considering Example 5.4, only the buffer input sequence π^2_{in} and the buffer output sequence π^2_{out} of B_2 have to be specified since the buffers B_1 and B_3 both have capacity 0. The buffer input sequence π^2_{in} is given by the machine sequence $\pi^2 = (2, 3, 1, 2, 5)$. Since O_{32} as well as O_{25} are the last operation in the job chain of job 2 and job 5, respectively, the buffer input sequence of B_2 is $\pi^2_{in} = (2, 3, 1)$. The buffer output sequence π^2_{out} can be derived from the schedule in Figure 5.7 by ordering jobs 2, 3 and 1 according to the time at which they start processing on their succeeding machines (see Subsection 4.2). Since job 2 starts processing at time 3 on M_1, job 3 at time 8 on M_3, and job 1 at time 7 on M_3, the buffer output sequence of B_2 is $\pi^2_{out} = (2, 1, 3)$. Obviously, π^2_{in} and π^2_{out} fulfill the feasibility condition (4.1).

In contrast to the previous buffer cases, the 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

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 5.2: Output of Algorithm Output Buffers applied to Example 5.4

of the jobs, the buffer ouput sequences and therefore also the optimal assignment of operations to buffer slots may be different though the given machine sequences are equal. This is illustrated by the following example.

Example 5.5 : Given are three machines and output buffers B_1, B_2 and B_3 of capacities $b_1 = 0$, $b_2 = 2$ and $b_3 = 0$. On the machines, six jobs have to be processed where job 3 consists of three operations and all other jobs consist of two operations each. Table 5.3 contains the processing times p_{ij} and the machines μ_{ij} associated with the operations O_{ij}.

Figure 5.8 shows the schedule constructed by the algorithm for the given instance

p_{ij}	j					
i	1	2	3	4	5	6
1	1	2	4	1	2	1
2	2	3	2	2	2	2
3	–	–	6	–	–	–

μ_{ij}	j					
i	1	2	3	4	5	6
1	M_2	M_2	M_3	M_2	M_2	M_2
2	M_1	M_3	M_2	M_3	M_3	M_3
3	–	–	M_1	–	–	–

Table 5.3: Instance of a job-shop problem with output buffers

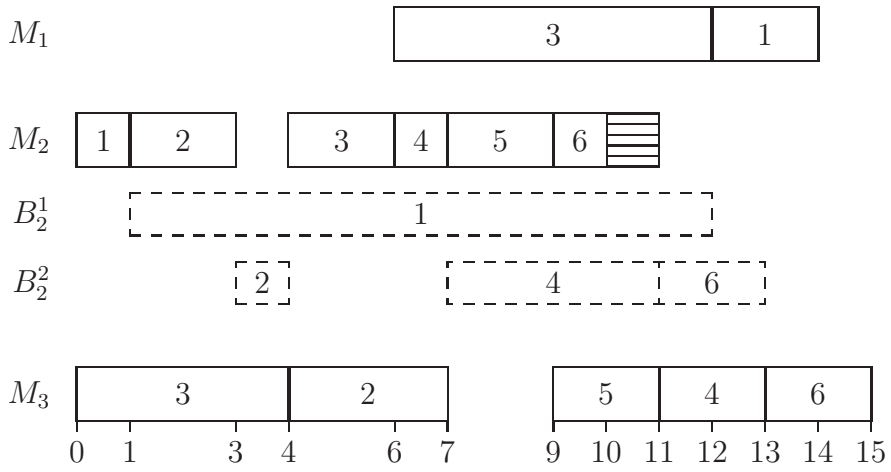

Figure 5.8: Schedule for the given job-shop problem with output buffers

where the jobs on machine M_1, M_2 and M_3 are sequenced in the order $\pi^1 = (3, 1)$, $\pi^2 = (1, 2, 3, 4, 5, 6)$ and $\pi^3 = (3, 2, 5, 4, 6)$, respectively.

If we reduce the processing time of the third operation of job 3 from 6 to 2, we obtain the schedule shown in Figure 5.9. In this schedule, jobs 1 and 6 are assigned to buffer slot B_2^1 and jobs 2 and 4 are assigned to buffer slot B_2^2 whereas in Figure 5.8 only job 1 is assigned to B_2^1 and jobs 2, 4 and 6 are assigned to B_2^2. The buffer input sequence π_{in}^2 of B_2 is equal to the machine sequence $\pi^2 = (1, 2, 3, 4, 5, 6)$ in both cases. If we order these jobs according to the starting time of their second operation on the corresponding machine, we get $\pi_{out}^2 = (2, 3, 5, 4, 1, 6)$ for the schedule of Figure 5.8 and $\pi_{out}^2 = (2, 3, 1, 5, 4, 6)$ for the schedule of Figure 5.9. □

Example 5.5 shows that the output buffer sequences and therefore the buffer slot assignment is dependent on the processing times of an instance. Thus, also the corresponding solution graphs of two instances varying only in the processing times may be different for given machine sequences. 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.

The computational effort of Algorithm Output Buffers is as follows: The set C contains all operations which are staying on a machine or are stored in a buffer at

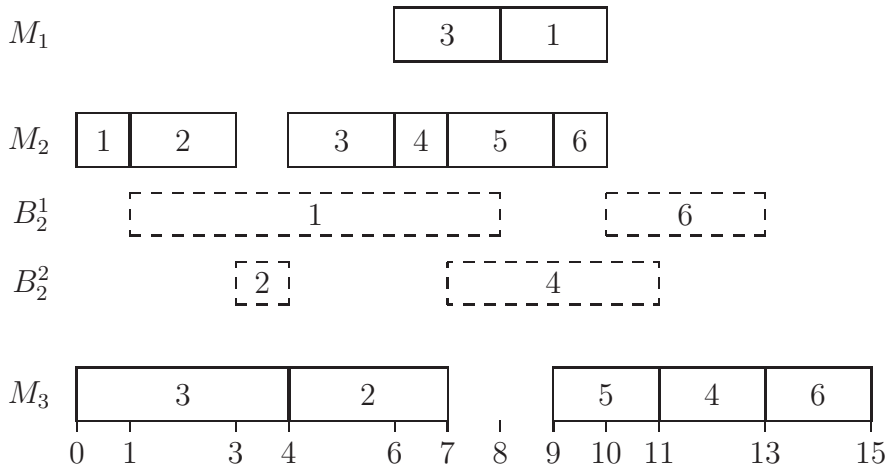

Figure 5.9: Schedule for the instance with $p_{33} = 2$

the current time. These are maximal n operations since at each time between its starting and finishing time any job is either on a machine or in a buffer. If the given sequences $\pi^1, \ldots, \pi^m$ are feasible, each operation is added to C once. Thus, if r denotes the total number of operations, a feasible schedule can be calculated in $O(r \max\{n, m\})$ time. Infeasibility is detected with the same computational effort.

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 $(k = 1, \ldots, m)$. An input buffer B_k stores all jobs that have already finished processing on the previous machine but cannot directly be loaded on machine M_k. In the case of a job-shop problem with input buffers, the output sequence π^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 $O_{1j} \to O_{2j} \to \ldots \to O_{n_j j}$ into $O_{n_j j} \to \ldots \to O_{2j} \to O_{1j}$ 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 5.3 : 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 of capacity 1 and specify sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines:

$n = 12r, \quad m = 8r$

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

$\pi^i = (O_{1,3r+i}, O_{1,i}, O_{2,6r+i})$	$i = 1, \ldots, 3r$
$\pi^i = (O_{1,3r+i}, O_{2,i})$	$i = 3r+1, \ldots, 6r$

$\pi^{6r+i} = (O_{1,9r+i}, O_{1,11r+i}, O_{2,10r+i})$	$i = 1, \ldots, r$
$\pi^{7r+i} = (O_{1,10r+i}, O_{2,9r+i})$	$i = 1, \ldots, r.$

(Since only one buffer is available we do not have to specify β_{ij} 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 5.10.

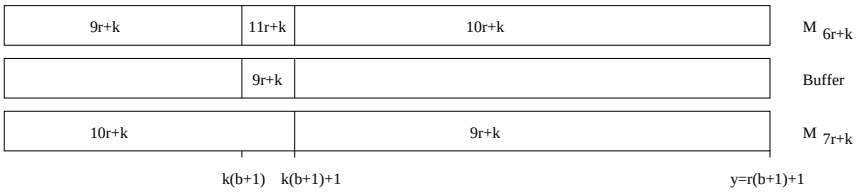

Figure 5.10: Schedule on machines M_{6r+k} and M_{7r+k}

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 5.11 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 5.11. Summarizing, in each feasible schedule with $C_{max} \leq y$ the buffer has exactly r separated intervals of length b left for the jobs $1, \ldots, 9r$.

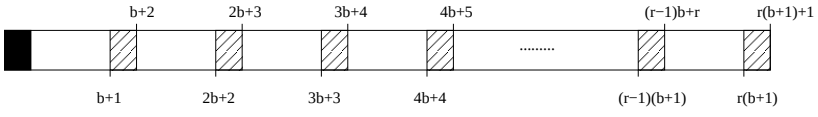

Figure 5.11: Partial schedule of the buffer

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 $6r + k$. 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 to 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 5.12).

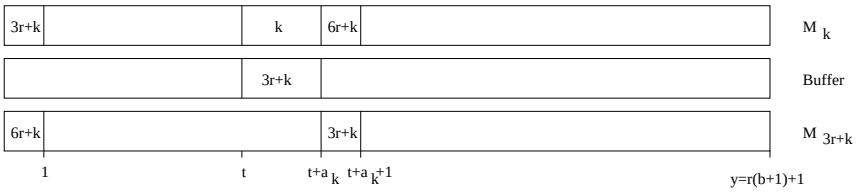

Figure 5.12: Job $3r + k$ occupies the buffer

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 5.13).

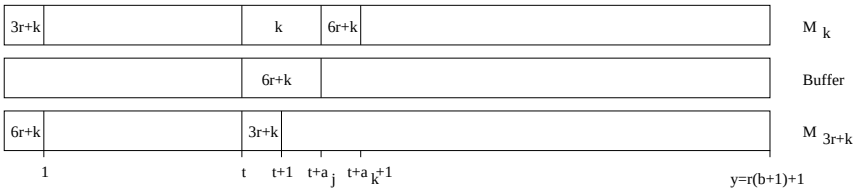

Figure 5.13: Job $6r + k$ occupies the buffer

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 5.12),
- 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 5.12). 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 5.3, 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 of the jobs in the buffer can be used as additional information to fully represent such a solution.

6 Foundations for local search methods

In this section we derive some theoretical foundations for applying local search methods to the job-shop problem with different buffer models. A local search method starts with an initial solution and iteratively searches through the solution space in order to find better solutions. In each step the current solution is replaced by a solution in some neighborhood of the current solution. The definition of an appropriate neighborhood structure has an important impact on the efficiency of local search methods since it determines the way how the method navigates through the search space. Furthermore, the computing time to choose a neighbor from the neighborhood of the current solution depends on the size of the neighborhood. Thus, neighborhoods should be defined in such a way that they help to lead the search process to good solutions and that the computational effort to determine a good neighbor does not get too large.

One possibility to achieve these aims is to incorporate problem-specific properties into the definition of neighborhood structures. For the job-shop problem with buffers, the so-called block approach can be used to identify necessary properties of solutions which may improve a given solution. It was first proposed for the single-machine problem $1 \mid r_j \mid L_{\max}$ (Grabowski et al. [14]). Later it was successfully adapted to some other scheduling problems like the job-shop or flow-shop problem, cf. Grabowski et al. [13], Brucker et al. [8], Nowicki & Smutnicki [28], [29]. In the search process only solutions having the stated necessary properties are considered as neighbored solutions.

The idea of the block approach in connection with local search is explained in detail in Subsection 6.1. In Subsections 6.2, 6.4, and 6.5 we present block approaches for the flow-shop problem with intermediate buffers, the job-shop problem with blocking operations and the job-shop problem with pairwise buffers. Based on the given block approaches also first neighborhood structures are proposed. A main difficulty in defining appropriate neighborhood structures is to ensure that for a given solution sufficient moves exist which provide a feasible neighbor. If the size of the neighborhood would be very small, the diversification of the local search procedure would be weak. Therefore, we develop methods which change infeasible solutions into feasible solutions. For the flow-shop problem with intermediate buffers, such methods are presented in Subsection 6.2. In the case of a job-shop problem with general buffers, methods to repair infeasible solutions are discussed in Subsection 6.3. These methods serve as basis to introduce extended neighborhoods for the flow-shop problem with intermediate buffers, the job-shop problem with blocking operations and the job-shop problem with pairwise buffers. Finally, in Subsection 6.6, we give an example to show the difficulties in deriving a block approach for the job-shop problem with output buffers.

6.1 The idea of the block approach in connection with local search

In Section 4, we have seen that a solution Π of a 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}. The corresponding solution graph $\bar{G}(\Pi)$ contains the following three different kinds of arcs: arcs connecting operations which belong to the same job (called **job arcs**), arcs connecting operations which are processed on the same machine (called **machine arcs**), and all other arcs which take care that blocking times are respected (called **blocking arcs**). Arcs $(\circ, i)$ and $(i, *)$ for an operation i are considered to be job arcs.

If the solution graph $\bar{G}(\Pi)$ does not contain any cycle of positive length, the solution Π is feasible and a critical path $CP(\Pi)$ in $\bar{G}(\Pi)$ can be calculated. Let $L(CP(\Pi)) = C_{\max}(\Pi)$ be the length of $CP(\Pi)$. Consider a maximal sequence $B = (u_1, \ldots, u_k)$ of successive operations on $CP(\Pi)$ to be processed consecutively on the same machine. Let the **incoming arc of B** be the arc on $CP(\Pi)$ which terminates at the vertex representing u_1, and let the **outgoing arc of B** be the arc on $CP(\Pi)$ which emanates from the vertex representing u_k. Obviously, the incoming and outgoing arc of B must be a job arc or a blocking arc.

We call the sequence B a **block** if either

- the sequence contains at least two operations (i.e. $k \geq 2$) or
- the sequence consists of one operation ($k = 1$) and the incoming arc of B is a blocking arc.

An example of a critical path $CP(\Pi)$ and its blocks in the case of a flow-shop problem with intermediate buffers is given in the next subsection.

Assume that $B = (u_1, \ldots, u_k)$ is a block on $CP(\Pi)$. Let the length $L(B)$ of B be the sum of the lengths of the arcs (u_i, u_{i+1}) for $i = 1, \ldots, k-1$ and of the outgoing arc of B. If the outgoing arc of B is a blocking arc, it holds $L(B) = \sum_{i=1}^{k-1} p_{u_i}$, since blocking arcs have length 0. If the outgoing arc of B is a job arc, it is $L(B) = \sum_{i=1}^{k} p_{u_i}$.

Note, that in the case of a classical job-shop problem the solution graph contains no blocking arcs and thus, each block consists of at least two operations. Brucker et al. [8] showed for the job-shop problem that a given feasible solution Π can only be improved when at least one operation of some block B of $\bar{G}(\Pi)$ is moved before the first or after the last operation of B. This result is called a block approach theorem.

In general, a block approach theorem states that only certain changes of a feasible solution Π may reduce the current makespan $C_{\max}(\Pi)$. In the local search process only

such solutions generated by these changes are considered as candidates for neighbored solutions. As basic operators to move from one feasible solution to another one we use shift operators. These operators perform a shift of a job from its current position in a machine sequence to another position in this sequence and leave all other machine sequences unchanged. In the next subsections, we consider different special cases of the job-shop problem with general buffers where a solution can be represented only by the machine sequences.

An important issue in defining neighborhood structures for the given problems is to take into consideration the feasibility of solutions. In the classical flow-shop problem, each arbitrary combination of sequences of the jobs on the machines leads to a feasible schedule. However, the existence of limited buffers as well as the more complicated structure of the job-shop problem can cause infeasibility of machine sequences. Therefore, we derive methods which allow to change an infeasible solution into a feasible solution. For the flow-shop problem with intermediate buffers, we characterize shifts which again lead to feasible solutions in Subsection 6.2. These shifts are used in order to repair an infeasible solution after a promising shift (according to the block approach theorem) was applied. In the case of a job-shop problem with general buffers, it is far more difficult to develop a technique which moves an infeasible solution into a feasible one without changing the structure of the solution too much. In Subsection 6.3, we propose a simple method which can be applied for the case of a job-shop problem with blocking operations and a job-shop problem with pairwise buffers.

In the next subsections, we will generalize the block approach theorem for the classical job-shop problem to the job-shop problem with different buffer models. Since we exploit problem-specific properties to do so, we look at the different buffer models separately.

6.2 The flow-shop problem with intermediate buffers

In the first part of this subsection we will derive a block approach theorem for the flow-shop problem with intermediate buffers. Based on this theorem we define neighborhood structures in the second part of this subsection. We also propose a repair technique which is used to change an infeasible solution into a feasible one.

6.2.1 Block approach

Let a feasible solution $\Pi = (\pi^1, \ldots, \pi^m)$ for the flow-shop problem with intermediate buffers be given. In contrast to the classical flow-shop problem, the solution graph $\bar{G}(\Pi)$ of the flow-shop problem with intermediate buffers may contain blocking arcs. In Section 5.1.2, we showed that the blocking arcs in $\bar{G}(\Pi)$ show a regular structure. They are always directed from operations on machine M_{k+1} to operations on machine

M_k which differ by the same amount of positions in π^{k+1} and π^k (i.e. by $b_k + 1$ positions). In detail, the blocking arcs are defined by $\pi^{k+1}(i) \rightarrow \pi^k(i + b_k + 1)$ for $i = 1, \ldots, n - b_k - 1$ and $k = 1, \ldots, m - 1$. Remember that in this problem type, the blocking arcs are also called **buffer arcs**.

The following example illustrates which consequences buffer arcs may have for the structure of a critical path and its blocks.

Example 6.1 : Consider an instance with three machines and six jobs given by Table 6.1. We assume that the buffers B_1 and B_2 have a capacity of $b_1 = 1$ and $b_2 = 2$ units, respectively. Figure 6.1 shows a schedule for the given instance where the jobs on machine M_1, M_2 and M_3 are sequenced in the order $\pi^1 = (1, 2, 3, 4, 5, 6)$, $\pi^2 = (2, 1, 3, 4, 5, 6)$ and $\pi^3 = (3, 1, 4, 2, 5, 6)$, respectively. Let $\Pi = (\pi^1, \pi^2, \pi^3)$.

p_{ij}	j					
i	1	2	3	4	5	6
1	1	1	1	3	2	1
2	1	3	1	1	2	2
3	1	1	1	1	1	1

Table 6.1: Instance of a flow-shop problem with intermediate buffers

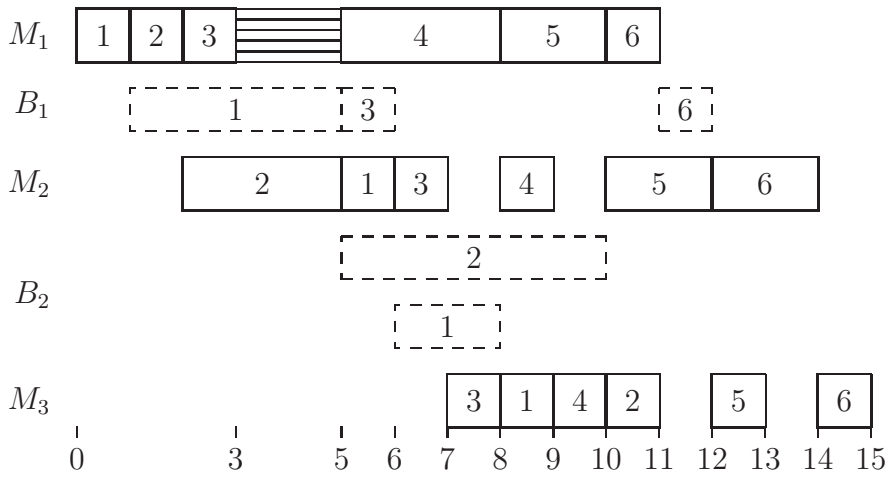

Figure 6.1: Schedule for (π^1, π^2, π^3)

The solution graph $\bar{G}(\Pi)$ is shown in Figure 6.2. Its critical path $CP(\Pi)$ consists of the following operations and arcs (shown with their corresponding weights):

$$\underbrace{O_{11} \xrightarrow{1} O_{12}}_{\text{block } B_1 \text{ on } M_1} \xrightarrow{1} \underbrace{O_{22} \xrightarrow{3} O_{21}}_{\text{block } B_2 \text{ on } M_2} \xrightarrow{0} \underbrace{O_{14} \xrightarrow{3} O_{15}}_{\text{block } B_3 \text{ on } M_1} \xrightarrow{2} \underbrace{O_{25} \xrightarrow{2} O_{26}}_{\text{block } B_4 \text{ on } M_2} \xrightarrow{2} O_{36} \xrightarrow{1} *$$

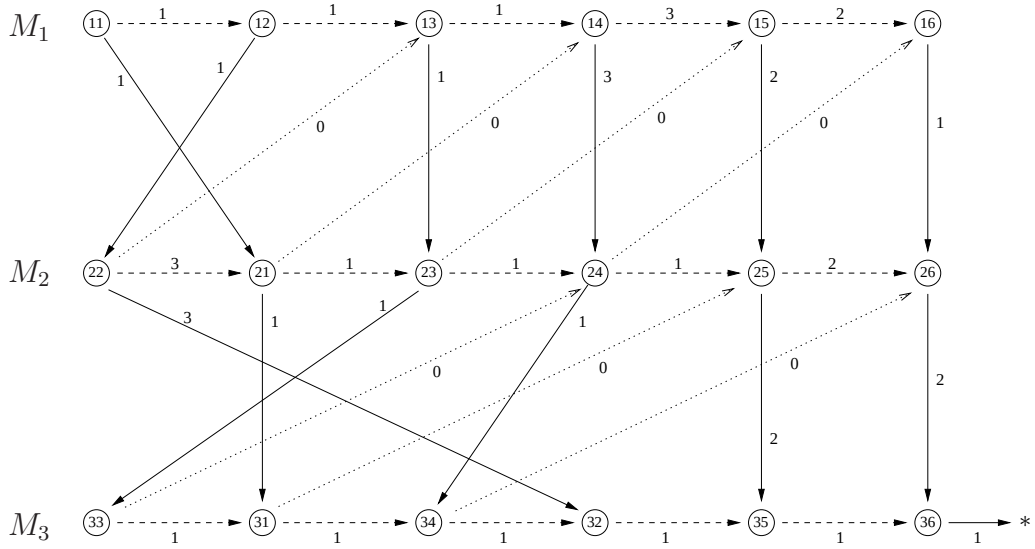

Figure 6.2: Solution graph $\bar{G}(\pi^1, \pi^2, \pi^3)$

On the critical path, we have four blocks B_1, B_2, B_3 and B_4. The outgoing arc $O_{21} \rightarrow O_{14}$ of B_2 is a buffer arc. This is also the incoming arc of B_3. All other incoming and outgoing arcs of the blocks are job arcs.

In the classical flow-shop problem a critical path always starts with the first operation on M_1, passes over machines $M_2, M_3, \ldots, M_{m-1}$ and ends in the last operation on M_m. As $CP(\Pi)$ shows, a critical path in the case of a flow-shop problem with intermediate buffers does not have this staircase structure in general. Obviously, this is due to the buffer arcs which are directed from an operation on machine M_{k+1} to an operation on machine M_k.

Consider now the buffer arc $O_{21} \rightarrow O_{14}$ which takes care that job 4 does not start on M_1 before job 3 has left M_1, i.e. before O_{21} starts processing on M_2. This buffer arc is directed from the operation in the second position of π^2 to the operation in the forth position of π^1. By moving now operations within block B_3 (e.g. shifting a job at the beginning of B_3) the operation in the forth position in π^1 and therefore the end vertex of this buffer arc may change. However, the blocking time on M_1 cannot be reduced by such changes. This means that moving a job of the block to the beginning of the block, as the classical block approach theorem proposes, cannot lead to an improvement of the makespan if the incoming arc of the block is a buffer arc. □

In a flow-shop problem with intermediate buffers, the buffer arcs ensure that all blocking restrictions are respected. A blocking situation is finished when the job blocking machine M_k or a job waiting in buffer B_k can continue processing on M_{k+1}. Thus, the situation that two jobs are connected by a buffer arc depends on the positions of the jobs in the machine permutations π^k and π^{k+1}. Also if we change

the sequence of jobs on M_k and M_{k+1}, we still have buffer arcs connecting the current jobs in the same fixed positions of π^k and π^{k+1} as before the change. This property plays an important role for the proof of the following block approach theorem. The basic idea of the theorem is that in order to get a probably better solution the length of at least one block on a critical path has to be reduced.

Theorem 6.1 : Let $\Pi = (\pi^1, \ldots, \pi^m)$ and $\tilde{\Pi} = (\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ be two arbitrary feasible solutions of the flow-shop problem with intermediate buffers, and let $CP(\Pi)$ be a critical path in the solution graph $\bar{G}(\Pi)$. If $C_{\max}(\tilde{\Pi}) < C_{\max}(\Pi)$ holds, then at least one of the following conditions must be satisfied for some block $B = (u_1, \ldots, u_k)$ of $CP(\Pi)$:
(Assume that block B is processed on machine M_i, such that operation u_1 is processed in π^i at position e and operation u_k at position $e + k - 1$.)

1. If the incoming and the outgoing arc of B are job arcs, then at least one operation from $\{u_2, \ldots, u_k\}$ is processed in $\tilde{\pi}^i$ before operation u_1 or at least one operation from $\{u_1, \ldots, u_{k-1}\}$ is processed in $\tilde{\pi}^i$ after operation u_k.

2. If the incoming and the outgoing arc of B are buffer arcs, then:
$$\sum_{j=e}^{e+k-2} p_{i,\tilde{\pi}^i(j)} < \sum_{j=e}^{e+k-2} p_{i,\pi^i(j)}.$$

3. If the incoming arc of B is a buffer arc and the outgoing arc of B is a job arc, then:
$$\sum_{j=e}^{f} p_{i,\tilde{\pi}^i(j)} < \sum_{j=e}^{e+k-1} p_{i,\pi^i(j)},$$
where operation u_k is processed on M_i at position f in $\tilde{\pi}^i$.

4. If the incoming arc of B is a job arc and the outgoing arc of B is a buffer arc, then:
$$\sum_{j=g}^{e+k-2} p_{i,\tilde{\pi}^i(j)} < \sum_{j=e}^{e+k-2} p_{i,\pi^i(j)},$$
where operation u_1 is processed on M_i at position g in $\tilde{\pi}^i$.

(If $f < e$ in Condition 3 or $e + k - 2 < g$ in Condition 4 holds, we assume the value of the corresponding sum to be 0.)

Proof: Let $CP(\Pi) = (\circ, u_1^1, u_2^1, \ldots, u_{m_1}^1, \ldots, u_1^k, u_2^k, \ldots, u_{m_k}^k, *)$, where the sequence $u_1^j, \ldots, u_{m_j}^j$ $(j = 1, \ldots, k)$ denotes a maximal number of operations to be processed

consecutively on the same machine. This means the sequence $u_1^j, \ldots, u_{m_j}^j$ is a block if either $m_j > 1$ or if $m_j = 1$ and the incoming arc of u_1^j is a buffer arc.

We decompose path $CP(\Pi)$ into subpathes of the form $P_1^j = (u_1^j, \ldots, u_{m_j}^j)$ for $j = 1, \ldots, k$, $P_2^j = (u_{m_j}^j, u_1^{j+1})$ for $j = 1, \ldots, k-1$, and $P_2^k = (u_{m_k}^k, *)$. Using this decomposition, the length of $CP(\Pi)$ can be expressed as follows:

$$L(CP(\Pi)) = L(P_1^1) + L(P_2^1) + \ldots + L(P_1^k) + L(P_2^k) = C_{\max}(\Pi).$$

Now assume, that a feasible solution $\tilde{\Pi}$ with $C_{\max}(\tilde{\Pi}) < C_{\max}(\Pi)$ exists and none of the four conditions given in the theorem is satisfied for any block of $CP(\Pi)$. In the following, we construct a path $\tilde{P} = (\tilde{P}_1^1, \tilde{P}_2^1, \tilde{P}_1^2, \tilde{P}_2^2, \ldots, \tilde{P}_1^k, \tilde{P}_2^k)$ from u_1^1 to $*$ in $\bar{G}(\tilde{\Pi})$ with length $L(\tilde{P}) \geq L(CP(\Pi))$ which is a contradiction to $C_{\max}(\tilde{\Pi}) < C_{\max}(\Pi)$.

The basic idea for the construction of the path $\tilde{P}$ is that $\tilde{P}$ should contain the same job arcs as $CP(\Pi)$ and the buffer arcs, which correspond to the same buffer arcs of $CP(\Pi)$ (i.e. those which connect the same positions in the permutations). In detail, the subpathes $\tilde{P}_2^1, \ldots, \tilde{P}_2^{k-1}$ are defined as follows:

- If $u_{m_j}^j \to u_1^{j+1}$ is a job arc, $\tilde{P}_2^j = (u_{m_j}^j, u_1^{j+1}) = P_2^j$ and, thus, the length of $\tilde{P}_2^j$ in $\bar{G}(\tilde{\Pi})$ is given by $L(\tilde{P}_2^j) = L(P_2^j) = p_{u_{m_j}^j}$.

- If $u_{m_j}^j \to u_1^{j+1}$ is a buffer arc and operation $u_{m_j}^j$ is processed on M_i at position $e + m_j - 1$ in π^i, then $\tilde{P}_2^j = (O_{i,\tilde{\pi}^i(e+m_j-1)}, O_{i-1,\tilde{\pi}^{i-1}(e+m_j+b_{i-1})})$. $\tilde{P}_2^j$ corresponds to the same buffer arc as P_2^j, but in general, it is different to P_2^j since the operations on the corresponding positions will have changed. However, since both subpathes contain only one buffer arc, we have $L(\tilde{P}_2^j) = 0 = L(P_2^j)$.

Subpath $\tilde{P}_2^k$ is defined as a longest path from $u_{m_k}^k$ to $*$ in $\bar{G}(\tilde{\Pi})$. Obviously, we get $L(\tilde{P}_2^k) \geq p_{u_{m_k}^k} = L(P_2^k)$.

The subpaths $\tilde{P}_1^j$ for $j = 1, \ldots, k$ are defined as longest path in $\bar{G}(\tilde{\Pi})$ between the end vertex of $\tilde{P}_2^{j-1}$ and the start vertex of $\tilde{P}_2^j$ (the path $\tilde{P}_1^1$ starts at the vertex u_1^1).

In the following, we will show that path $\tilde{P}_1^j$ is at least as long as path P_1^j for $j = 1, \ldots, k$. If P_1^j consists of a single operation and the arc terminating at u_1^j is a job arc, path P_1^j has length 0, and therefore $L(\tilde{P}_1^j) \geq L(P_1^j)$ holds in this case. Otherwise, P_1^j corresponds to a block $B_j = (u_1^j, \ldots, u_{m_j}^j)$ of $CP(\Pi)$. Depending on whether the incoming and outgoing arc of B_j is a job or a buffer arc, we can conclude for the length of subpath $\tilde{P}_1^j$ the following: (Again, assume that block B_j is processed on machine M_i and that operation u_1^j is processed in π^i at position e and operation $u_{m_j}^j$ at position $e + m_j - 1$.)

(a) If the incoming and outgoing arc of B_j are job arcs (i.e. $m_j \geq 2$ holds), according to Condition 1 no operation of $u_2^j, \ldots, u_{m_j-1}^j$ is processed in $\tilde{\pi}^i$ before

the first operation u_1^j or after the last operation $u_{m_j}^j$ of block B_j. Therefore each operation of $\{u_2^j, \ldots, u_{m_j-1}^j\}$ is processed in $\tilde{\pi}^i$ between operations u_1^j and $u_{m_j}^j$ and, thus, in $\bar{G}(\tilde{\Pi})$ a path from u_1^j to $u_{m_j}^j$ containing all vertices from $\{u_2^j, \ldots, u_{m_j-1}^j\}$ exists. Since u_1^j and $u_{m_j}^j$ are the first and last operation of subpath $\tilde{P}_1^j$ this yields $L(\tilde{P}_1^j) \geq L(P_1^j)$.

(b) If the incoming and outgoing arcs of B_j are buffer arcs (i.e. $m_j \geq 1$ holds), then in $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ they connect the same positions as in $(\pi^1, \ldots, \pi^m)$. Thus, the start vertex of path $\tilde{P}_1^j$ represents operation $O_{i,\tilde{\pi}^i(e)}$ and the end vertex of path $\tilde{P}_1^j$ represents operation $O_{i,\tilde{\pi}^i(e+m_j-1)}$. Since Condition 2 does not hold, we have $L(\tilde{P}_1^j) \geq \sum_{v=e}^{e+m_j-2} p_{i,\tilde{\pi}^i(v)} \geq \sum_{v=e}^{e+m_j-2} p_{i,\pi^i(v)} = L(P_1^j)$.

(c) If the incoming arc of B_j is a buffer arc and the outgoing arc of B_j is a job arc (i.e. $m_j \geq 1$ holds), the start vertex of path $\tilde{P}_1^j$ represents operation $O_{i,\tilde{\pi}^i(e)}$ and the end vertex of path $\tilde{P}_1^j$ represents operation $u_{m_j}^j$. We denote by f the position on which operation $u_{m_j}^j$ is processed on M_i in $\tilde{\pi}^i$. Since Condition 3 is not valid (and $m_j \geq 1$ holds), it is $f \geq e$ and, thus, $u_{m_j}^j$ is processed not before position e in $\tilde{\pi}^i$. Therefore, we have $L(\tilde{P}_1^j) \geq \sum_{v=e}^{f} p_{i,\tilde{\pi}^i(v)} \geq \sum_{v=e}^{e+m_j-1} p_{i,\pi^i(v)} = L(P_1^j)$.

(d) If the incoming arc of B_j is a job arc and the outgoing arc of B_j is a buffer arc (i.e. $m_j \geq 2$ holds), the start vertex of path $\tilde{P}_1^j$ represents operation u_1^j and the end vertex of path $\tilde{P}_1^j$ represents operation $O_{i,\tilde{\pi}^i(e+m_j-1)}$. Denoting by g the position on which operation u_1^j is processed on M_i in $\tilde{\pi}^i$ we have $L(\tilde{P}_1^j) \geq \sum_{v=g}^{e+m_j-2} p_{i,\tilde{\pi}^i(v)} \geq \sum_{v=e}^{e+m_j-2} p_{i,\pi^i(v)} = L(P_1^j)$ since Condition 4 does not hold.

Summarizing, we get $L(\tilde{P}_1^j) \geq L(P_1^j)$ for $j = 1, \ldots, k$, $L(\tilde{P}_2^k) \geq L(P_2^k)$, and $L(\tilde{P}_2^j) = L(P_2^j)$ for $j = 1, \ldots, k-1$. Thus, we have $C_{\max}(\tilde{\Pi}) \geq L(\tilde{P}) \geq L(CP(\Pi)) = C_{\max}(\Pi)$, which is a contradiction. □

Obviously, in the case where the buffer capacity of each buffer is at least $n-1$, the flow-shop problem with intermediate buffers reduces to a classical flow-shop problem and only Condition 1 of Theorem 6.1 is relevant.

Example 6.2 : If we swap jobs 3 and 4 on M_1 in Figure 6.1, we obtain a solution $\tilde{\Pi}$ such that Condition 3 of Theorem 6.1 is fulfilled. As the processing time of job 3 on M_1 is two time units smaller than that of job 4, the length of block B_3 reduces from 5 to 3 by this swap. The schedule corresponding to $\tilde{\Pi}$ is shown in Figure 6.3 and has a makespan of 13. □

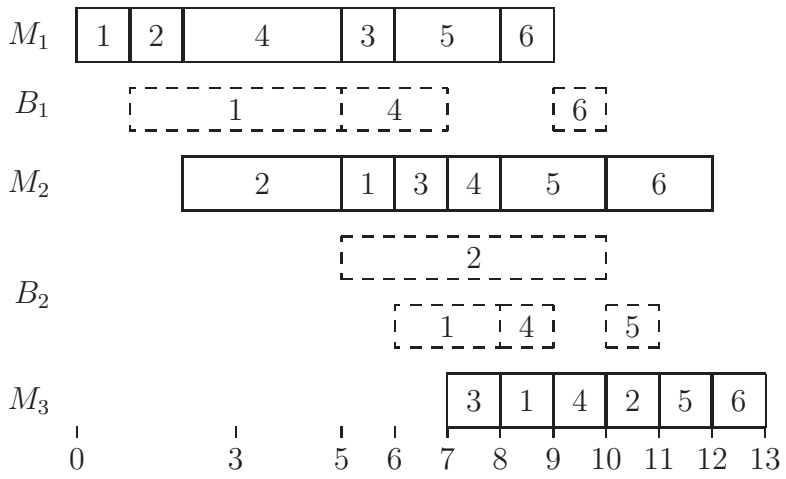

Figure 6.3: Schedule for $\tilde{\Pi}$

6.2.2 Neighborhood structures

Based on Theorem 6.1 which gives necessary conditions for a solution $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ to be better than $(\pi^1, \ldots, \pi^m)$ we will define neighborhood structures for the flow-shop problem with intermediate buffers in this section. Let $\Pi = (\pi^1, \ldots, \pi^m)$ be a feasible solution of the flow-shop problem with intermediate buffers and $B = (u_1, \ldots, u_k)$ be a block on a critical path in the corresponding solution graph $\bar{G}(\Pi)$. Assume that block B is processed on machine M_i, such that operation u_1 is processed in π^i at position e and operation u_k at position $e + k - 1$. Depending on the type of the incoming and outgoing arc of B, Theorem 6.1 yields different possibilities how to change $(\pi^1, \ldots, \pi^m)$ in order to get a possibly better solution $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$:

If the incoming and outgoing arc of B are job arcs, then one operation of block B, different from the first operation in B, must be shifted before block B or one operation of block B, different from the last operation in B, must be shifted after block B. In order to produce not too many neighbors, we only consider shifts of an operation of B directly before the first or directly after the last operation of B.

If the incoming and the outgoing arc of B are buffer arcs, we have to change π^i, such that the sum of processing times of the operations in the positions from e to $e+k-2$ reduces. We consider shifts of operations of B to positions outside the block and shifts of operations outside B into the block. In both cases, we only allow that exactly one operation enters the block and one operation leaves the block. In order to reduce the length of the block, we have to ensure that the processing time of the entering operation is smaller than that of the leaving operation. This condition is fulfilled, if we use one of the following types of shifts:

- Shifting an operation of B with larger processing time than the processing time of the last operation of B (this comes into position $e + k - 2$) after block B on position $e + k - 1$, or

- shifting an operation of B with larger processing time than the processing time of the operation directly before the first operation of B (this comes into position e) before block B on position $e-1$, or
- shifting an operation processed after block B with a processing time less than the processing time of the last but one operation of B (this leaves position $e+k-2$) into block B on position $e+k-2$, or
- shifting an operation processed before block B with a processing time less than the processing time of the first operation of B (this leaves position e) into block B on position e.

We only consider shifts of an operation of block B on the positions $e-1$ and $e+k-1$, respectively, and shifts of an operation outside block B on positions e and $e+k-2$, respectively. The shifts are restricted to these positions in order to produce not too many neighbors.

If the incoming arc of B is a buffer arc and the outgoing arc of B is a job arc or if the incoming arc of B is a job arc and the outgoing arc of B is a buffer arc, the promising shifts correspond to the above discussed shifts. Based on these considerations, we introduce the following neighborhood $\mathcal{N}_F^1$.

Neighborhood $\mathcal{N}_F^1$: Let $\Pi = (\pi^1, \ldots, \pi^m)$ be a feasible solution of the flow-shop problem with intermediate buffers, and let $CP(\Pi)$ be a critical path in $\bar{G}(\Pi)$. Furthermore, let $B = (u_1, \ldots, u_k)$ be an arbitrary block of $CP(\Pi)$ which is processed on machine M_i, such that operation u_1 is processed in π^i at position e and operation u_k at position $e+k-1$. Let v be the operation which is processed in π^i at position $e-1$.
Neighborhood $\mathcal{N}_F^1(\Pi)$ consists of all feasible solutions $(\pi^1, \ldots, \pi^{i-1}, \tilde{\pi}^i, \pi^{i+1}, \ldots, \pi^m)$ which can be constructed by applying the following shifts to all blocks B of the chosen critical path $CP(\Pi)$:

1. If the incoming arc of B is a job arc, then one operation of block B, different from the first operation in B, is shifted at the beginning of block B (i.e. directly before operation u_1).
2. If the incoming arc of B is a buffer arc, then
   (a) one operation $w \neq u_k$ of B with $p_w > p_v$ is shifted directly before operation v, or
   (b) one operation w with $p_w < p_{u_1}$ which is processed in π^i at a position before block B is shifted directly after operation u_1.
3. If the outgoing arc of B is a job arc, then one operation of block B, different from the last operation in B, is shifted at the end of block B (i.e. directly after operation u_k).

4. If the outgoing arc of B is a buffer arc, then
   (a) one operation w of B with $p_w > p_{u_k}$ is shifted directly after operation u_k, or
   (b) one operation w with $p_w < p_{u_{k-1}}$ which is processed in π^i at a position after block B is shifted directly before operation u_{k-1}. □

Formally, the neighborhood $\mathcal{N}_F^1$ can be defined using the following two operator types:

- for $j < k$, operator $rshift(i, j, k)$ shifts job $\pi^i(j)$ directly after job $\pi^i(k)$ in permutation π^i, and
- for $j > k$, operator $lshift(i, j, k)$ shifts job $\pi^i(j)$ directly before job $\pi^i(k)$ in permutation π^i.

First numerical tests indicated that for neighborhood $\mathcal{N}_F^1$ the number of neighbors was often very small, such that the diversification of the search was weak. The reason here is that if a shift is applied to a feasible solution, the new solution may not be compatible with the given buffer capacities (see Lemma 5.1). Such a situation is shown in Figures 6.4 and 6.5, where the corresponding parts of the solution graph before and after applying the operator $rshift(i, j, k)$ are represented. Since the solution graph after applying the right shift contains at least two cycles, the new solution is not compatible with the given buffer capacities (see Theorem 5.2). In fact, the smaller the buffer capacities are, the more moves lead to infeasible neighbors.

To overcome this problem of infeasibility of neighbors, we introduce another neighborhood $\mathcal{N}_F^2$. Before presenting the neighborhood, we characterize the shifts which lead to feasible solutions. Remember, that according to Lemma 5.1 the solution $(\pi^1, \ldots, \pi^m)$ is compatible with the buffer capacities $b_1, \ldots, b_{m-1}$ iff

$$\pi^i(l) \in \{\pi^{i-1}(1), \ldots, \pi^{i-1}(l + b_{i-1})\} \qquad \text{for } i = 2, \ldots, m;\ l = 1, \ldots, n - b_{i-1}$$

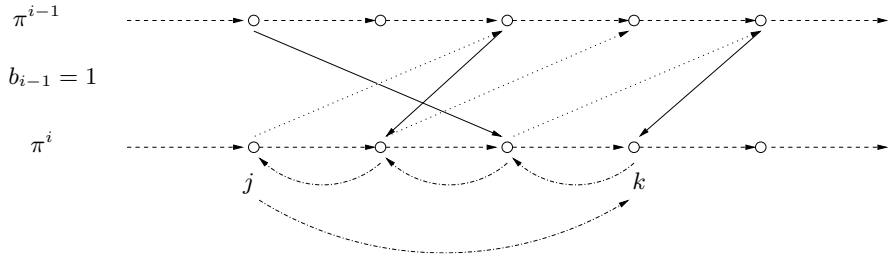

Figure 6.4: Situation before applying the operator $rshift(i, j, k)$

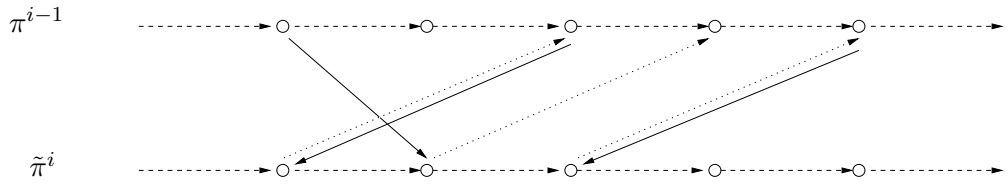

Figure 6.5: Situation after applying the operator $rshift(i, j, k)$

Lemma 6.1 :
When applying the operator $rshift(i, j, k)$ to the feasible solution $(\pi^1, \ldots, \pi^m)$ the resulting solution is feasible if and only if

$$\begin{array}{rcll} \pi^i(l) & \neq & \pi^{i-1}(l + b_{i-1}) & \text{for} \quad l = j+1, \ldots, k \quad \text{and} \\ \pi^{i+1}(j - b_i + l) & \neq & \pi^i(j) & \text{for} \quad l = 0, 1, \ldots, k-j-1 \end{array}$$

When applying the operator $lshift(i, j, k)$ to the feasible solution $(\pi^1, \ldots, \pi^m)$ the resulting solution is feasible if and only if

$$\begin{array}{rcll} \pi^i(j) & \neq & \pi^{i-1}(l) & \text{for} \quad l = k + b_{i-1} + 1, \ldots, j + b_{i-1} \quad \text{and} \\ \pi^{i+1}(l - b_i) & \neq & \pi^i(l) & \text{for} \quad l = k, \ldots, j-1 \end{array}$$

Proof: When applying the operators $rshift(i, j, k)$ or $lshift(i, j, k)$ to the solution $\Pi = (\pi^1, \ldots, \pi^m)$, only permutation π^i is changed. The resulting solution $\tilde{\Pi} = (\pi^1, \ldots, \pi^{i-1}, \tilde{\pi}^i, \pi^{i+1}, \ldots, \pi^m)$ is feasible, iff π^{i-1} and $\tilde{\pi}^i$ are compatible with b_{i-1} and $\tilde{\pi}^i$ and π^{i+1} are compatible with b_i. According to Theorem 5.2 this is valid, iff the subgraphs $G(\pi^{i-1}, \tilde{\pi}^i)$ and $G(\tilde{\pi}^i, \pi^{i+1})$ of $\bar{G}(\tilde{\Pi})$ do not contain any cycle. Recall that $G(\pi^{i-1}, \tilde{\pi}^i)$ is the subgraph of $\bar{G}(\tilde{\Pi})$ which contains all vertices representing operations that are processed on machines M_{i-1} and M_i. Applying the operator $rshift(i, j, k)$ to the feasible solution Π, only job arcs terminating at an operation in position $l \in \{j+1, \ldots, k\}$ in π^i with $\pi^i(l) = \pi^{i-1}(l + b_{i-1})$ can produce a cycle in $G(\pi^{i-1}, \tilde{\pi}^i)$ (see Figures 6.4 and 6.5). A cycle in subgraph $G(\tilde{\pi}^i, \pi^{i+1})$ can only come up, if the job arc emanating from the operation in position j in π^i terminates in π^{i+1} at an operation in the positions $j - b_i, j - b_i + 1, \ldots, k - b_i - 1$ (see Figures 6.6 and 6.7). Thus, $G(\pi^{i-1}, \tilde{\pi}^i)$ and $G(\tilde{\pi}^i, \pi^{i+1})$ are acyclic in this case, iff the first two of the above inequalities are fulfilled. Similarly, conditions are derived for the case of the operator $lshift(i, j, k)$. □

Lemma 6.1 gives a criterion whether applying the operators $rshift(i, j, k)$ or $lshift(i, j, k)$ to a feasible solution $(\pi^1, \ldots, \pi^m)$ leads to a feasible solution again. To face the problem of infeasibility of neighbors we alter $\mathcal{N}_F^1$ in a new neighborhood $\mathcal{N}_F^2$. The idea of $\mathcal{N}_F^2$ is to carry out the promising shift (according to conditions 1.–4. in the definition of neighborhood $\mathcal{N}_F^1$) and to use a so-called repair technique afterwards: If applying an operator $rshift(i, j, k)$ or $lshift(i, j, k)$ results in an infeasible solution, we iteratively repair the permutations $\pi^{i-1}, \pi^{i-2}, \ldots, \pi^1$ as well as

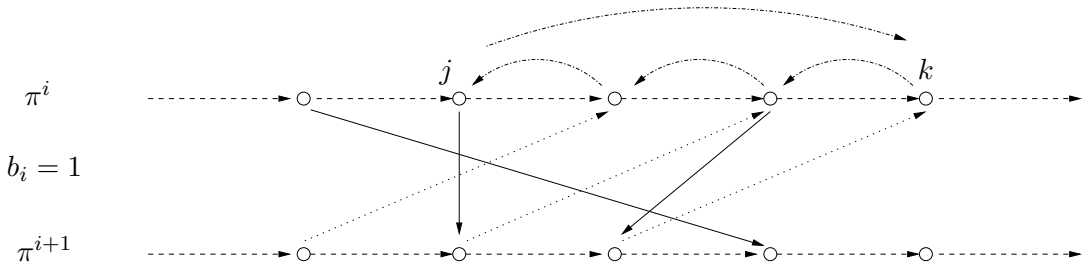

Figure 6.6: Situation before applying the operator $rshift(i, j, k)$

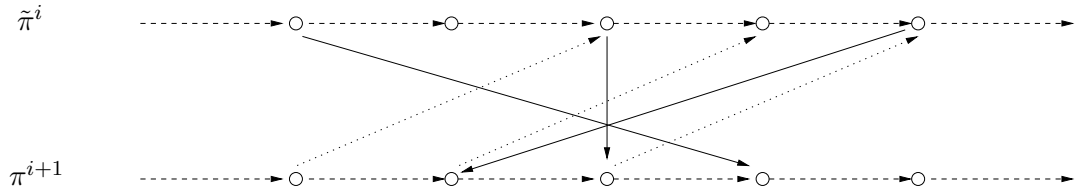

Figure 6.7: Situation after applying the operator $rshift(i, j, k)$

the permutations $\pi^{i+1}, \pi^{i+2}, \ldots, \pi^m$ until we finally get a feasible solution. In the following, we describe this **repair procedure** in detail:

First let us assume that applying the operator $rshift(i, j, k)$ to $(\pi^1, \ldots, \pi^m)$ leads to an infeasible solution $(\pi^1, \ldots, \pi^{i-1}, \tilde{\pi}^i, \pi^{i+1}, \ldots, \pi^m)$. Thus, according to Lemma 6.1 one or both of the following statements a) or b) must be fulfilled:

a) $\pi^i(l) = \pi^{i-1}(l + b_{i-1})$ holds for at least one index $l \in \{j + 1, \ldots, k\}$, i.e. permutations π^{i-1} and $\tilde{\pi}^i$ are not compatible with b_{i-1}.

b) $\pi^{i+1}(j - b_i + l) = \pi^i(j)$ holds for (exactly) one index $l \in \{0, \ldots, k - j - 1\}$, i.e. permutations $\tilde{\pi}^i$ and π^{i+1} are not compatible with b_i.

Consider first case b). To achieve back feasibility we change permutation π^{i+1} into $\tilde{\pi}^{i+1}$ by shifting job $\pi^{i+1}(j - b_i + l)$ to the right on position $k - b_i$. The situation before and after such a shift is illustrated in Figures 6.8 and 6.9. Figure 6.8 shows an example for case b) where $l = 1 = b_i$, i.e. $\pi^{i+1}(j) = \pi^i(j)$ holds. The situation after $rshift(i, j, k)$ and repairing π^{i+1} is represented in Figure 6.9.

After the repair of π^{i+1} job $\pi^i(j) = \pi^{i+1}(j - b_i + l)$ is in position k in $\tilde{\pi}^i$ and in position $k - b_i$ in $\tilde{\pi}^{i+1}$, i.e. its position in $\tilde{\pi}^i$ and $\tilde{\pi}^{i+1}$ differs by b_i places. Thus, the corresponding job arc cannot cause a cycle in $G(\tilde{\pi}^i, \tilde{\pi}^{i+1})$. Furthermore, the jobs in positions $j - b_i + l + 1, \ldots, k - b_i$ in π^{i+1} and the jobs in positions $j + l + 1, \ldots, k$ in π^i are all moved one position to the left. These moves also cannot cause any cycle. Therefore, $G(\tilde{\pi}^i, \tilde{\pi}^{i+1})$ is cyclefree which means that $\tilde{\pi}^i$ and $\tilde{\pi}^{i+1}$ are compatible with

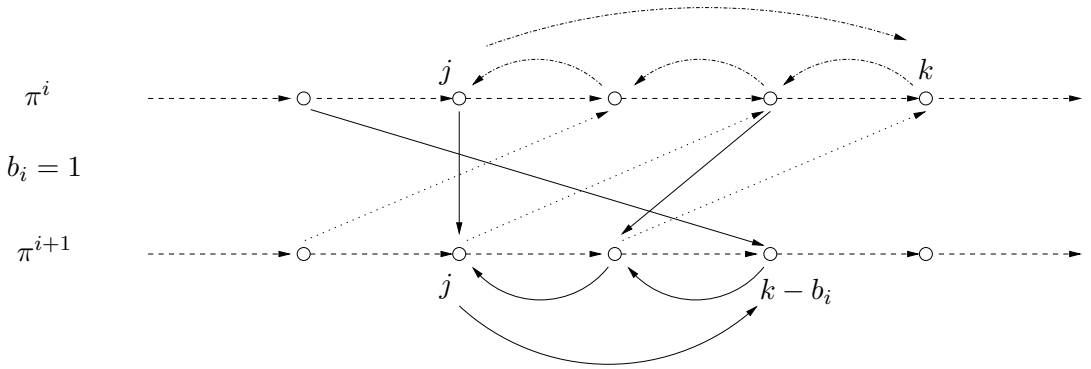

Figure 6.8: An example for case b) where $\pi^i(j) = \pi^{i+1}(j)$ holds

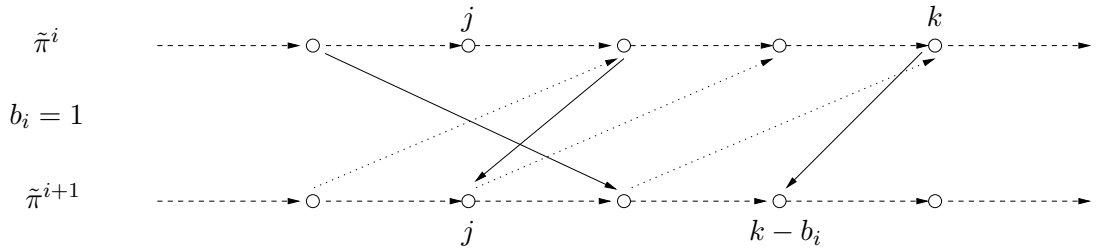

Figure 6.9: Situation after applying $rshift(i, j, k)$ and repairing π^{i+1}

b_i. If after changing π^{i+1} to $\tilde{\pi}^{i+1}$ the permutations $\tilde{\pi}^{i+1}$ and π^{i+2} are not compatible with buffer capacity b_{i+1}, we iteratively repeat the repair procedure.

Now, consider case a). In π^{i-1}, we interchange from left to right all jobs $\pi^{i-1}(l+b_{i-1})$ and $\pi^{i-1}(l + b_{i-1} - 1)$ with indices $l \in \{j + 1, \ldots, k\}$ for which $\pi^i(l) = \pi^{i-1}(l + b_{i-1})$ holds. In this way, we get a permutation $\tilde{\pi}^{i-1}$, such that $G(\tilde{\pi}^{i-1}, \tilde{\pi}^i)$ is cyclefree and thus, $\tilde{\pi}^i$ and $\tilde{\pi}^{i-1}$ are compatible with b_{i-1}. An example for these interchanges is shown in Figures 6.10 and 6.11.

If now permutations $\tilde{\pi}^{i-1}$ and π^{i-2} are not compatible with b_{i-2}, we iteratively repeat this interchange procedure.

Similarly, we can define a corresponding repair procedure if applying the operator $lshift(i, j, k)$ results in an infeasible solution. Summarizing, we introduce the following neighborhood $\mathcal{N}_F^2$.

Neighborhood $\mathcal{N}_F^2$: Let $\Pi = (\pi^1, \ldots, \pi^m)$ be a feasible solution of the flow-shop problem with intermediate buffers. Neighborhood $\mathcal{N}_F^2(\Pi)$ consists of all feasible solutions $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ where $\tilde{\pi}^i$ can be constructed using the criteria of possible improvement (see conditions 1.–4. in the definition of neighborhood $\mathcal{N}_F^1$) and where $\pi^1, \ldots, \pi^{i-1}$ as well as $\pi^{i+1}, \ldots, \pi^m$ are changed (if necessary) into permutations $\tilde{\pi}^1, \ldots, \tilde{\pi}^{i-1}$ and $\tilde{\pi}^{i+1}, \ldots, \tilde{\pi}^m$ with the above described repair technique. □

Neighborhood $\mathcal{N}_F^2$ contains $\mathcal{N}_F^1$ as a subneighborhood (i.e. $\mathcal{N}_F^1(\Pi) \subseteq \mathcal{N}_F^2(\Pi)$ for each

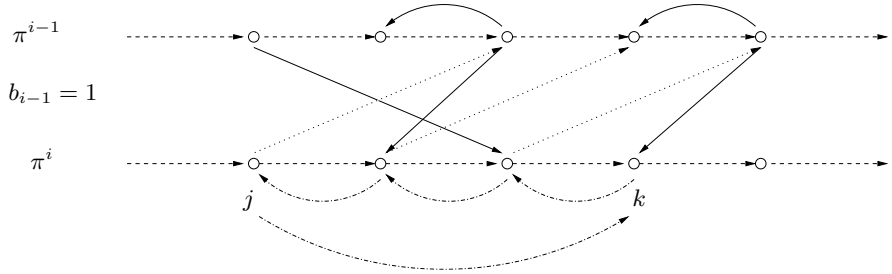

Figure 6.10: An example for case a)

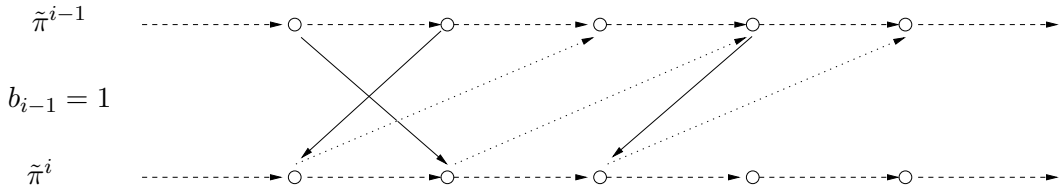

Figure 6.11: Situation after applying $rshift(i, j, k)$ and repairing π^{i-1}

solution $\Pi = (\pi^1, \ldots, \pi^m)$). The size of both neighborhoods is polynomially bounded by mn^2 because the number of shifts on each machine is bounded by n^2.

A nice property of a neighborhood is the opt-connectivity which means that from each feasible solution an optimal solution can be reached by a finite sequence of moves within the neighborhood. It is possible to construct a sophisticated example which shows that the neighborhood $\mathcal{N}_F^2$ is not opt-connected. In this example, we present a solution of the flow-shop problem with intermediate buffers which is not optimal and possesses no neighbor with respect to neighborhood $\mathcal{N}_F^2$. As this example contains many jobs and is rather complex, we do not describe it here. Although the neighborhood $\mathcal{N}_F^2$ is not opt-connected, the computational results for the neighborhood $\mathcal{N}_F^2$ are very satisfying and the constructed counterexample indicates that only in extreme cases disconnected components occur.

6.3 Obtaining feasible neighbors for job-shop problems with buffers

For a job-shop problem with general buffers, the feasibility of solutions cannot be controlled as easy as in the case of a flow-shop problem with intermediate buffers. Due to the special structure of the solution graph in the case of a flow-shop problem with intermediate buffers, already a subgraph $G(\pi^i, \pi^{i+1})$ for $i = 1, \ldots, m-1$ of two successive machine sequences must contain a cycle if there is a cycle in the solution graph. Therefore, the feasibility of a solution can be tested very effectively by the compatibility criteria of Lemma 5.1 and also shifts which lead to feasible solutions again can characterized easily (see Lemma 6.1). However, in the case of a job-shop

problem with general buffers, we do not have this special structure of the solution graph in general. A positive cycle can contain operations which are processed on several machines. Thus, we do not dispose of a direct criteria for feasibility analogous to the case of the flow-shop problem with intermediate buffers. In order to detect a possible positive cycle in the solution graph for the job-shop problem with general buffers, the Floyd-Warshall algorithm can be used (see e.g. Ahuja et al. [3]).

In the following, we suppose a repair technique for the job-shop problem with general buffers which moves an infeasible solution into a feasible one. This repair technique is based on the next Lemma.

Lemma 6.2 : Let Π be a feasible solution of the job-shop problem with general buffers and P be a corresponding feasible schedule. Shifting a job j, $j \in \{1, \ldots, n\}$ at the beginning of P (i.e. shifting all operations of j at the beginning of their machine sequences) yields a feasible schedule again. Analogously, shifting a job j, $j \in \{1, \ldots, n\}$ at the end of P yields a feasible schedule again.

Proof: Consider the partial schedule P' of P where all operations of job j are eliminated from the machines and the buffers. Since all jobs of P' have the same starting times on the machines and the same entering times in the buffers as they have in P, the schedule P' is feasible. Processing all operations of job j successively before the first operation of P' yields a feasible schedule for all jobs again. Analogously, processing all operations of job j successively after the last operation of P' yields a feasible schedule again. □

Unfortunately, by shifting a complete job at the beginning or the end of a given schedule the makespan may increase drastically. Therefore, we try to find a better way of producing a feasible solution. This may be achieved by the following **repair procedure**.

First, let us assume that a right shift $rshift(i, j, k)$ of operation h in permutation π^i from position j to position k does not provide a feasible solution. In this case we iteratively move all job predecessor operations and all job successor operations of h by $k - j$ positions to the right in their machine sequences. If for some operations the distance between the current position and the end of the machine sequence is smaller than $k - j$ positions, the corresponding operation is moved to the end of the machine sequence. The shifts are done in such a way that alternately a predecessor operation and a successor operation is shifted. If after a shift the resulting solution is feasible, the repair process is stopped. If all operations of job $J(h)$ are shifted and the new solution is still infeasible, we repeat moving the operations of job $J(h)$ to the right by one position each. We start with operation h and continue with its job predecessors and successors alternately. The order in which we shift the predecessor and successor operations may be chosen in a different way. However, we chose the

above order since we assume that the infeasibility is somehow close to the originally shifted operation. We repeat this process as long as the new solution is infeasible. According to Lemma 6.2, this process terminates with a feasible solution.

Especially, if the buffer capacities are small, operation h may be shifted far after position k in permutation π^i. Nevertheless, the repair procedure yields a feasible solution where operation h is moved to the right in π^i by at least $k - j$ positions.

Similarly, we can define a symmetric repair procedure if after a left shift the resulting solution is infeasible. The following example illustrates the repair process in the case of a job-shop problem with blocking operations.

Example 6.3 : Consider an instance with two machines and five jobs given by Table 6.2. We assume that operations O_{11} and O_{12} are blocking and all other operations are non-blocking. Figure 6.12 shows a schedule for the instance where the jobs on M_1 and M_2 are sequenced in the order $\pi^1 = (1, 2, 3, 4, 5)$ and $\pi^2 = (3, 1, 2, 4)$, respectively.

p_{ij}	j				
i	1	2	3	4	5
1	2	5	5	4	1
2	3	6	3	2	–

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

Table 6.2: Instance of a job-shop problem with blocking operations

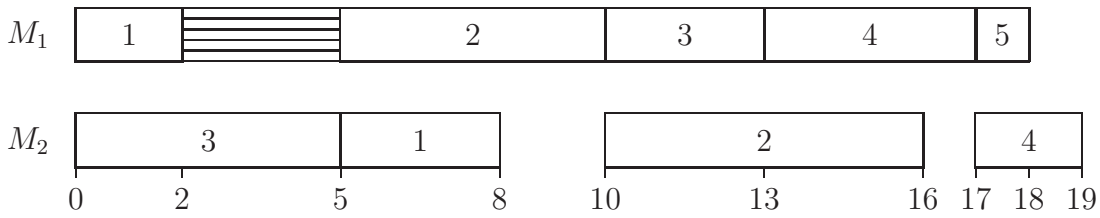

Figure 6.12: Schedule for $\pi^1 = (1, 2, 3, 4, 5)$ and $\pi^2 = (3, 1, 2, 4)$

Let us assume that in solution $\Pi = (\pi^1, \pi^2)$, job 2 should be shifted before job 1 on M_1. (As we will see in the next section, this shift might lead to a reduction of the makespan according to the block approach theorem for the job-shop problem with blocking operations.) This left shift leads to the solution $\tilde{\pi}^1 = (2, 1, 3, 4, 5)$ and $\pi^2 = (3, 1, 2, 4)$ which is infeasible since the corresponding solution graph contains a positive cycle. Thus, in order to find a feasible neighbored solution for (π^1, π^2) the repair procedure is applied to $(\tilde{\pi}^1, \pi^2)$. This means that the second operation of job 2 is also shifted one position to the left on M_2. The resulting solution $\tilde{\pi}^1 = (2, 1, 3, 4, 5)$ and $\tilde{\pi}^2 = (3, 2, 1, 4)$ is feasible and has a makespan of 20. Its schedule is shown in Figure 6.13. □

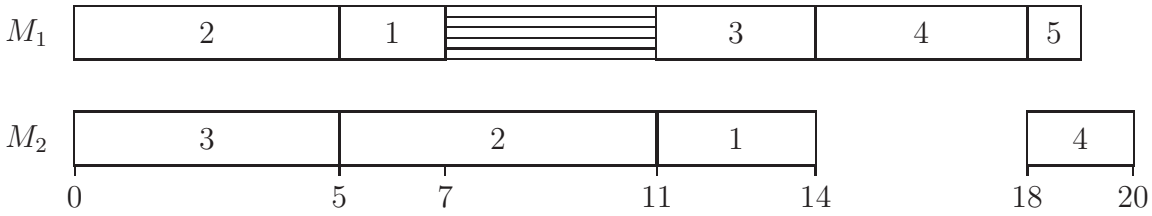

Figure 6.13: Schedule for $\pi^1 = (2, 1, 3, 4, 5)$ and $\pi^2 = (3, 2, 1, 4)$

6.4 The job-shop problem with blocking operations

In the first part of this subsection, we derive a block approach theorem for the job-shop problem with blocking operations which is used to define appropriate neighborhood structures in the second part.

6.4.1 Block approach

In contrast to the flow-shop problem with intermediate buffers where several jobs may share a common buffer, in the job-shop problem with blocking operations each ideal operation has an own buffer. Therefore, ideal operations cannot cause blocking situations. On the other hand, if an operation i is blocking, job $J(i)$ blocks machine $\mu(i)$ as long as the job successor $\sigma(i)$ of i cannot start processing on its machine. This blocking restriction is represented in the solution graph by a blocking arc which is directed from $\sigma(i)$ to the machine successor of operation i.

The following example shows a solution graph in the case of a job-shop problem with blocking operations and illustrates the difference in the definition of the blocking arcs to the definition of the buffer arcs in the case of the flow-shop problem with intermediate buffers.

Example 6.4 : We consider the same instance as in Example 6.3, but assume that only operation O_{11} is blocking. All other operations are non-blocking. Figure 6.14 shows a schedule for the instance where the jobs on machine M_1 and M_2 are sequenced in the order $\pi^1 = (1, 2, 3, 4, 5)$ and $\pi^2 = (3, 1, 2, 4)$, respectively.

The solution graph $\bar{G}(\pi^1, \pi^2)$ is shown in Figure 6.15. Its critical path $C(\pi^1, \pi^2)$ consists of the following operations and arcs (shown with their corresponding weights):

$$\circ \xrightarrow{0} \underbrace{O_{13} \xrightarrow{5} O_{21}}_{\text{block } B_1 \text{ on } M_2} \xrightarrow{0} \underbrace{O_{12} \xrightarrow{5} O_{23} \xrightarrow{3} O_{14}}_{\text{block } B_2 \text{ on } M_1} \xrightarrow{4} O_{24} \xrightarrow{2} *$$

On the critical path, we have two blocks B_1 and B_2. The incoming arc $\circ \rightarrow O_{13}$ of B_1 is a job arc and the outgoing arc $O_{21} \rightarrow O_{12}$ is a blocking arc which is also the incoming arc of B_2. The outgoing arc $O_{14} \rightarrow O_{24}$ of B_2 is a job arc of job 4.

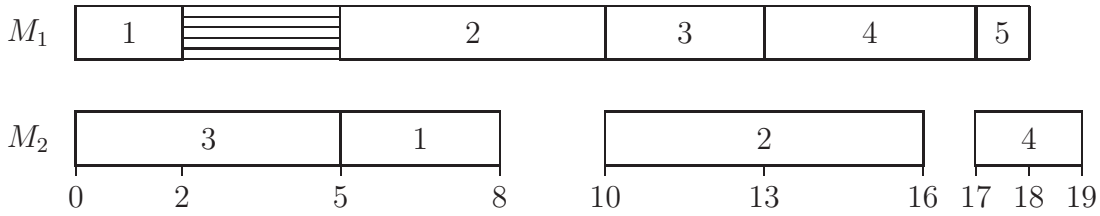

Figure 6.14: Schedule for $\pi^1 = (1, 2, 3, 4, 5)$ and $\pi^2 = (3, 1, 2, 4)$

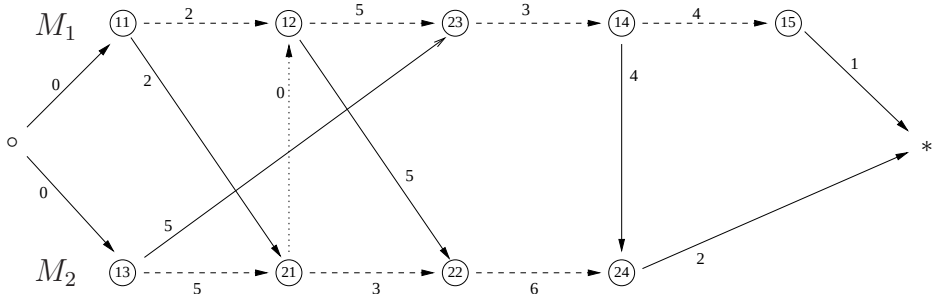

Figure 6.15: Solution graph $\bar{G}(\pi^1, \pi^2)$

Consider now the blocking arc $O_{21} \rightarrow O_{12}$ which takes care that job 2 does not start on M_1 before job 1 has left M_1, i.e. before O_{21} starts processing on M_2. The end vertex of this blocking arc represents the direct machine successor of job 1 on M_1. In contrast to the flow-shop situation, where a buffer arc between operations on M_2 and M_1 is directed from the operation in a fixed position in π^2 to the operation in a fixed position in π^1, a blocking arc in a job-shop problem with blocking operations is mainly defined by the underlying job arc. It is directed from the job successor of the blocking operation, which is O_{21} in this case, to the machine successor of the blocking operation, which is O_{12}. In this connection it does not matter where these operations are positioned in π^1 and π^2.

Therefore, moving a job of B_2 different from job 2 to the second position in π^1 would not reduce the blocking time on M_1, and thus, it would not reduce the makespan of the current solution. This means that in this example moving a job of the block B_2 to the beginning of the block, as the classical block approach theorem proposes, cannot lead to an improvement since the incoming arc of the block is a blocking arc. □

The following theorem generalizes the classical block approach theorem and is the basis for defining suitable neighborhoods on the set of all feasible solutions. In this context, for an operation j we denote by $\nu_\Pi(j)$ the direct machine predecessor operation of j with respect to the solution Π. This means if operation j is processed on machine M_i in position e of π^i, then $\nu_\Pi(j)$ is processed on M_i in position $e-1$ of

π^i. The proof of the theorem follows the same line as the proof of the block approach theorem for the flow-shop problem with intermediate buffers.

Theorem 6.2 : Let $\Pi = (\pi^1, \ldots, \pi^m)$ and $\tilde{\Pi} = (\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ be two arbitrary feasible solutions of the job-shop problem with blocking operations, and let $CP(\Pi)$ be a critical path in the solution graph $\bar{G}(\Pi)$. If $C_{\max}(\tilde{\Pi}) < C_{\max}(\Pi)$ holds, then at least one of the following conditions must be satisfied for some block $B = (u_1, \ldots, u_k)$ of $CP(\Pi)$: (Let M_i be the machine on which block B is processed.)

1. At least one operation from $\{u_1, \ldots, u_{k-1}\}$ is processed in $\tilde{\pi}^i$ after operation u_k.
2. If the incoming arc of block B is a job arc, then at least one operation from $\{u_2, \ldots, u_k\}$ is processed in $\tilde{\pi}^i$ before operation u_1.
3. If the incoming arc of block B is a blocking arc, then at least one operation from $\{u_1, \ldots, u_k\}$ is processed in $\tilde{\pi}^i$ before the predecessor operation $\nu_\Pi(u_1)$ of u_1.

Proof: Let $CP(\Pi) = (\circ, u_1^1, u_2^1, \ldots, u_{m_1}^1, \ldots, u_1^k, u_2^k, \ldots, u_{m_k}^k, *)$, where the sequence $u_1^j, \ldots, u_{m_j}^j$ $(j = 1, \ldots, k)$ denotes a maximal number of operations to be processed consecutively on the same machine. This means the sequence $u_1^j, \ldots, u_{m_j}^j$ is a block if either $m_j > 1$ or if $m_j = 1$ and the incoming arc of u_1^j is a blocking arc.

We decompose path $CP(\Pi)$ into subpathes of the form $P_1^j = (u_1^j, \ldots, u_{m_j}^j)$ for $j = 1, \ldots, k$, $P_2^j = (u_{m_j}^j, u_1^{j+1})$ for $j = 1, \ldots, k-1$, and $P_2^k = (u_{m_k}^k, *)$. Using this decomposition, the length of $CP(\Pi)$ can be expressed as follows:

$$L(CP(\Pi)) = L(P_1^1) + L(P_2^1) + \ldots + L(P_1^k) + L(P_2^k) = C_{\max}(\Pi).$$

Now assume, that a feasible solution $\tilde{\Pi}$ with $C_{\max}(\tilde{\Pi}) < C_{\max}(\Pi)$ exists and none of the three conditions given in the theorem is satisfied for any block of $CP(\Pi)$. In the following, we construct a path $\tilde{P} = (\tilde{P}_1^1, \tilde{P}_2^1, \tilde{P}_1^2, \tilde{P}_2^2, \ldots, \tilde{P}_1^k, \tilde{P}_2^k)$ from u_1^1 to $*$ in $\bar{G}(\tilde{\Pi})$ with length $L(\tilde{P}) \geq L(CP(\Pi))$ which is a contradiction to $C_{\max}(\tilde{\Pi}) < C_{\max}(\Pi)$.

The basic idea for the construction of the path $\tilde{P}$ is that $\tilde{P}$ should contain the same job arcs as $CP(\Pi)$ and the blocking arcs, which start at the same operation as in $CP(\Pi)$. In detail, the subpathes $\tilde{P}_2^1, \ldots, \tilde{P}_2^{k-1}$ are defined as follows:

- If $u_{m_j}^j \rightarrow u_1^{j+1}$ is a job arc, $\tilde{P}_2^j = (u_{m_j}^j, u_1^{j+1}) = P_2^j$ and, thus, the length of $\tilde{P}_2^j$ in $\bar{G}(\tilde{\Pi})$ is given by $L(\tilde{P}_2^j) = L(P_2^j) = p_{u_{m_j}^j}$.

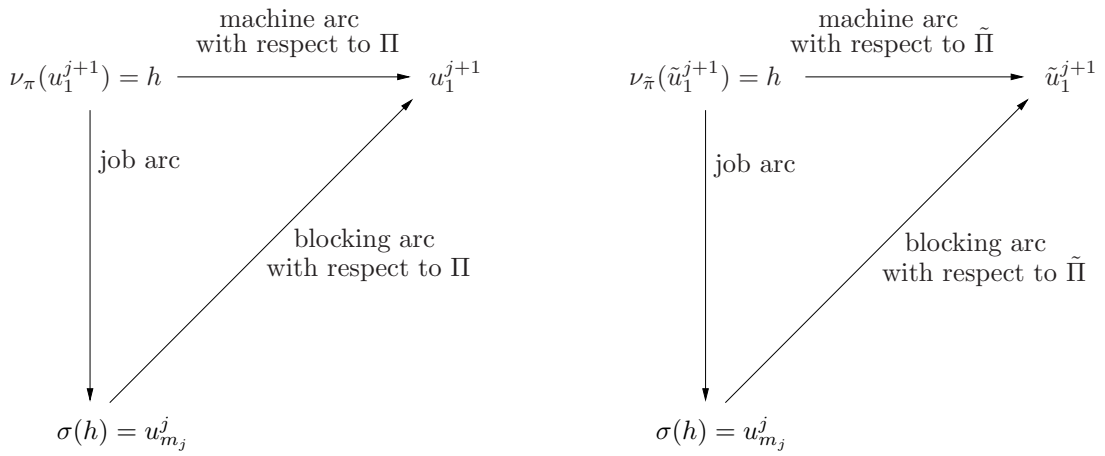

Figure 6.16: (a) A blocking arc in $\bar{G}(\Pi)$
(b) The corresponding blocking arc in $\bar{G}(\tilde{\Pi})$

- If $u_{m_j}^j \to u_1^{j+1}$ is a blocking arc, we have a blocking operation h, where $u_{m_j}^j$ is the job successor of operation h and u_1^{j+1} is the machine successor of operation h (see Figure 6.16 (a)). Subpath $\tilde{P}_2^j$ is then defined by the blocking arc resulting from the same blocking operation h in solution $\tilde{\Pi}$, i.e. $\tilde{P}_2^j = (u_{m_j}^j, \tilde{u}_1^{j+1})$, where $\tilde{u}_1^{j+1}$ is the machine successor operation of h with respect to $\tilde{\Pi}$ (see Figure 6.16 (b)). Note, that the end vertex u_1^{j+1} may have changed. However, since both subpathes contain only one blocking arc, we have $L(\tilde{P}_2^j) = 0 = L(P_2^j)$.

Subpath $\tilde{P}_2^k$ is defined as a longest path from $u_{m_k}^k$ to $*$ in $\bar{G}(\tilde{\Pi})$. Obviously, we get $L(\tilde{P}_2^k) \geq p_{u_{m_k}^k} = L(P_2^k)$.

The subpaths $\tilde{P}_1^j$ for $j = 1, \ldots, k$ are defined as longest path in $\bar{G}(\tilde{\Pi})$ between the end vertex of $\tilde{P}_2^{j-1}$ and the start vertex of $\tilde{P}_2^j$ (the path $\tilde{P}_1^1$ starts at the vertex u_1^1).

In the following, we will show that path $\tilde{P}_1^j$ is at least as long as path P_1^j for $j = 1, \ldots, k$. If P_1^j consists of a single operation and the arc terminating at u_1^j is a job arc, path P_1^j has length 0, and therefore $L(\tilde{P}_1^j) \geq L(P_1^j)$ holds in this case. Otherwise, P_1^j corresponds to a block $B_j = (u_1^j, \ldots, u_{m_j}^j)$ of $CP(\Pi)$. Depending on whether the incoming and outgoing arc of B_j is a job or a blocking arc, we can conclude for the length of subpath $\tilde{P}_1^j$ the following: (Assume that block B_j is processed on machine M_i and that operation u_1^j is processed in π^i at position e.)

(a) If the incoming arc of B_j is a job arc (i.e. $m_j \geq 2$ holds), according to Conditions 1 and 2 no operation of $u_2^j, \ldots, u_{m_j-1}^j$ is processed in $\tilde{\pi}^i$ before the first operation u_1^j or after the last operation $u_{m_j}^j$ of block B_j. Therefore each operation of $\{u_2^j, \ldots, u_{m_j-1}^j\}$ is processed in $\tilde{\pi}^i$ between operations u_1^j and $u_{m_j}^j$ and,

thus, in $\bar{G}(\tilde{\Pi})$ a path from u_1^j to $u_{m_j}^j$ containing all vertices from $\{u_2^j, \ldots, u_{m_j-1}^j\}$ exists. Since u_1^j and $u_{m_j}^j$ are the first and last operation of subpath $\tilde{P}_1^j$ this yields $L(\tilde{P}_1^j) \geq L(P_1^j)$.

(b) If the incoming arc of B_j is a blocking arc (i.e. $m_j \geq 1$ holds), according to Conditions 1 and 3 no operation of $u_1^j, \ldots, u_{m_j-1}^j$ is processed in $\tilde{\pi}^i$ after operation $u_{m_j}^j$ and no operation of $u_1^j, \ldots, u_{m_j}^j$ is processed in $\tilde{\pi}^i$ before operation $\nu_\Pi(u_1^j)$. Therefore each operation of $\{u_1^j, \ldots, u_{m_j-1}^j\}$ is processed in $\tilde{\pi}^i$ between operations $\nu_\Pi(u_1^j) = \pi^i(e-1)$ and u_{m_j}. Since the start vertex of path $\tilde{P}_1^j$ represents the operation in position e of $\tilde{\pi}^i$ and the end vertex of path $\tilde{P}_1^j$ is equal to operation $u_{m_j}^j$, in $\bar{G}(\tilde{\Pi})$ a path containing all vertices from $\{u_1^j, \ldots, u_{m_j-1}^j\}$ and ending in $u_{m_j}^j$ exists. This yields $L(\tilde{P}_1^j) \geq L(P_1^j)$.

Summarizing, we get $L(\tilde{P}_1^j) \geq L(P_1^j)$ for $j = 1, \ldots, k$, $L(\tilde{P}_2^k) \geq L(P_2^k)$, and $L(\tilde{P}_2^j) = L(P_2^j)$ for $j = 1, \ldots, k-1$. Thus, we have $C_{\max}(\tilde{\Pi}) \geq L(\tilde{P}) \geq L(CP(\Pi)) = C_{\max}(\Pi)$, which is a contradiction. □

Obviously, in the case of a classical job-shop problem, Theorem 6.2 corresponds to the classical block approach theorem.

Example 6.5 : Applying Theorem 6.2 to the schedule given in Figure 6.14, condition 3 implies that shifting job 2, 3 or 4 before job 1 on M_1 might lead to a reduction of the makespan. Indeed, swapping jobs 1 and 2 on M_1 leads to a solution without any blocking time since the second operation of job 1 can start directly after its first operation. The corresponding schedule with a makespan of 18 is shown in Figure 6.17.

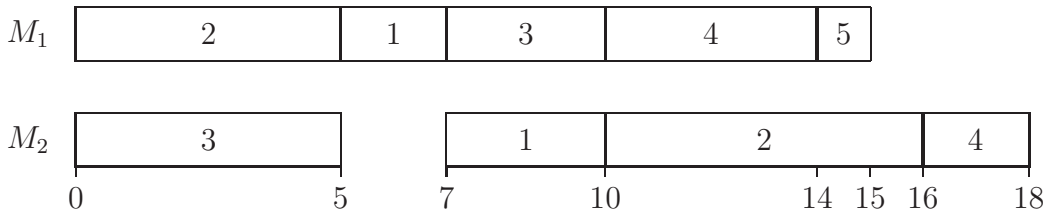

Figure 6.17: Schedule after a swap of jobs 1 and 2 on M_1

□

6.4.2 Neighborhood structures

For the job-shop problem with blocking operations, Theorem 6.2 gives necessary properties of solutions $\tilde{\Pi}$ which may improve a given feasible solution Π. Based on

these characteristics of possibly better solutions, we introduce the following neighborhood $\mathcal{N}_B^1$.

Neighborhood $\mathcal{N}_B^1$: Let $\Pi = (\pi^1, \ldots, \pi^m)$ be a feasible solution of the job-shop problem with blocking operations, and let $CP(\Pi)$ be a critical path in $\bar{G}(\Pi)$. Furthermore, let $B = (u_1, \ldots, u_k)$ be an arbitrary block of $CP(\Pi)$ which is processed on machine M_i.
Neighborhood $\mathcal{N}_B^1(\Pi)$ consists of all feasible solutions $(\pi^1, \ldots, \pi^{i-1}, \tilde{\pi}^i, \pi^{i+1}, \ldots, \pi^m)$ which can be constructed by applying the following shifts to all blocks B of the chosen critical path $CP(\Pi)$:

1. If B contains at least two operations, then one operation of block B, different from the last operation in B, is shifted at the end of block B (i.e. directly after operation u_k).

2. If the incoming arc of B is a job arc, then one operation of block B, different from the first operation in B, is shifted at the beginning of block B (i.e. directly before operation u_1).

3. If the incoming arc of B is a blocking arc, then one operation of block B is shifted directly before the machine predecessor operation $\nu_\Pi(u_1)$ of u_1. □

Note, that the possible operators resulting from blocking arcs are simpler than for the flow-shop situation. This results from the fact that now blocking is a local phenomenon of a single operation, whereas in the flow-shop case it is a more global phenomenon corresponding to a whole machine.

As in the case of the flow-shop problem with intermediate buffers, a lot of solutions are infeasible after applying one of the shifts of neighborhood $\mathcal{N}_B^1$ to a feasible solution Π. In such cases, we apply the repair procedure described in Subsection 6.3 in order to get feasible neighbors of Π.

Neighborhood $\mathcal{N}_B^2$: Let $\Pi = (\pi^1, \ldots, \pi^m)$ be a feasible solution of the job-shop problem with blocking operations. Neighborhood $\mathcal{N}_B^2(\Pi)$ consists of all feasible solutions $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ which can be constructed by applying the shifts described for neighborhood $\mathcal{N}_B^1$ and by repairing the resulting solution into a feasible solution according to the repair procedure described in Subsection 6.3, if necessary. □

As in the case of the neighborhoods for the flow-shop problem with intermediate buffers, neighborhood $\mathcal{N}_B^1$ is a subneighborhood of $\mathcal{N}_B^2$ (i.e. $\mathcal{N}_B^1(\Pi) \subseteq \mathcal{N}_B^2(\Pi)$ for each solution $\Pi = (\pi^1, \ldots, \pi^m)$). If q denotes the maximal number of operations to be processed on one machine, the number of shifts on each machine is bounded by q^2. Thus, the size of both neighborhoods is polynomially bounded by mq^2.

6.5 The 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_{rs} is associated with each pair (M_r, M_s) of machines. Given a feasible solution $\Pi = (\pi^1, \ldots, \pi^m)$, the input sequence π^{rs}_{in} and the output sequence π^{rs}_{out} of buffer B_{rs} are partial sequences of π^r and π^s, respectively. A buffer arc of buffer B_{rs} is defined in a similar way as a buffer arc in the case of a flow-shop problem with intermediate buffers:

$$\pi^{rs}_{out}(i) \;\rightarrow\; \rho_\Pi\left(\pi^{rs}_{in}(i + b_{rs})\right) \quad \text{for all positions } i$$

where $\rho_\Pi(j)$ is the direct machine successor of operation j with respect to the solution Π and b_{rs} is the capacity of buffer B_{rs}. Applying this definition to an intermediate buffer B_r of machines M_r and M_{r+1} yields the definition $\pi^{r+1}(i) \rightarrow \pi^r(i + b_r + 1)$ of a buffer arc of buffer B_r in the case of a flow-shop problem with intermediate buffers. As this definition shows, the situation that two jobs are connected by a buffer arc depends on the positions of the jobs in the machine permutations π^r and π^{r+1}. A similar role as π^r and π^{r+1} in the definition of the buffer arcs play the buffer input sequence π^{rs}_{in} and the buffer output sequence π^{rs}_{out} for the job-shop problem with pairwise buffers. However, these are only partial sequences of the machine sequences π^r and π^s, respectively, such that in this case the situation is more complex. The buffer arc $\pi^{rs}_{out}(i) \;\rightarrow\; \rho_\Pi\left(\pi^{rs}_{in}(i + b_{rs})\right)$ is directed to the machine successor of job $\pi^{rs}_{in}(i + b_{rs})$ in π^r, i.e. for the definition of a buffer arc the sequences π^{rs}_{out}, π^{rs}_{in} and π^r are relevant. If the incoming arc of a block B on M_r is the buffer arc $\pi^{rs}_{out}(i) \;\rightarrow\; \rho_\Pi\left(\pi^{rs}_{in}(i + b_{rs})\right)$, the length of block B can be calculated by determining the position of job $\pi^{rs}_{in}(i + b_{rs})$ in π^r and by summing up the processing times of the jobs in π^r starting with the machine successor of $\pi^{rs}_{in}(i + b_{rs})$ and ending with the last job or the last but one job, respectively, in block B. Obviously, in general also processing times of jobs not belonging to π^{rs}_{in} contribute to the length of B. This has to be taken into account when deriving a block approach theorem and defining neighborhoods for the job-shop problem with pariwise buffers.

If we change the sequence of jobs on M_r and M_{r+1} in the flow-shop problem with intermediate buffers, we still have buffer arcs connecting the current jobs in the same fixed positions of π^r and π^{r+1} as before the change. If we change the sequence of jobs on M_r and M_s in a job-shop problem with pairwise buffers, we have to differentiate whether jobs belonging to π^{rs}_{in} and π^{rs}_{out} have changed their positions or jobs of π^r and π^s not belonging to π^{rs}_{in} and π^{rs}_{out} have changed their positions.

6.5.1 Block approach

A block approach theorem in the case of the job-shop problem with pairwise buffers can be formulated in the following way:

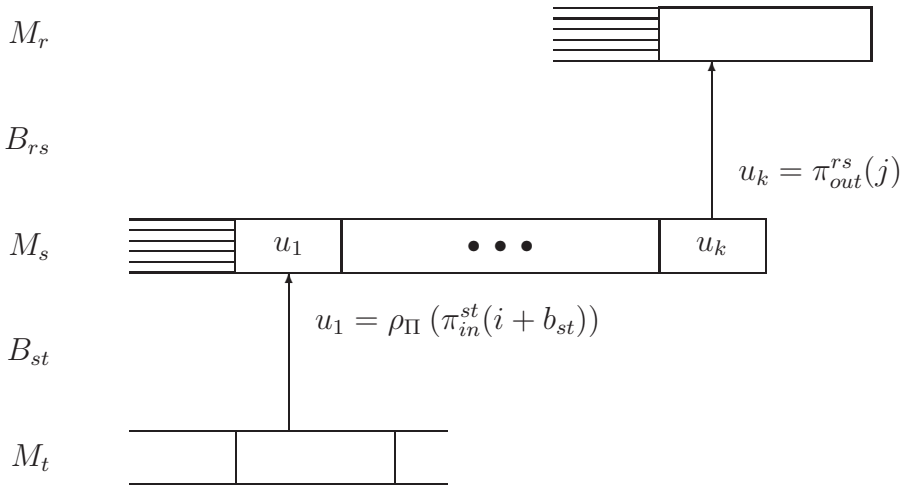

Figure 6.18: Situation when incoming and outgoing arc of B are buffer arcs

Theorem 6.3 : Let $\Pi = (\pi^1, \ldots, \pi^m)$ and $\tilde{\Pi} = (\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ be two arbitrary feasible solutions of the job-shop problem with pairwise buffers, and let $CP(\Pi)$ be a critical path in the solution graph $\bar{G}(\Pi)$. If $C_{\max}(\tilde{\Pi}) < C_{\max}(\Pi)$ holds, then at least one of the following conditions must be satisfied for some block $B = (u_1, \ldots, u_k)$ of $CP(\Pi)$:
(Assume that block B is processed on machine M_s.)

1. If the incoming and the outgoing arc of B are job arcs, then at least one operation from $\{u_2, \ldots, u_k\}$ is processed in $\tilde{\pi}^s$ before operation u_1 or at least one operation from $\{u_1, \ldots, u_{k-1}\}$ is processed in $\tilde{\pi}^s$ after operation u_k.

2. If the incoming and the outgoing arc of B are buffer arcs, we have a situation as shown in Figure 6.18. Assume that the incoming arc of block B is the buffer arc $\pi^{st}_{out}(i) \rightarrow \rho_\Pi\,(\pi^{st}_{in}(i + b_{st})) = u_1$ of buffer B_{st} and the outgoing arc of block B is the buffer arc $u_k = \pi^{rs}_{out}(j) \rightarrow \rho_\Pi\,(\pi^{rs}_{in}(j + b_{rs}))$ of buffer B_{rs}. Furthermore, in solution $\tilde{\Pi}$, let operation $\rho_{\tilde{\Pi}}\,(\tilde{\pi}^{st}_{in}(i + b_{st}))$ be processed on M_s at position $\tilde{i}$ in $\tilde{\pi}^s$ and let operation $\tilde{\pi}^{rs}_{out}(j)$ be processed on M_s at position $\tilde{j}$ in $\tilde{\pi}^s$, i.e. $\tilde{\pi}^s(\tilde{i}) = \rho_{\tilde{\Pi}}\,(\tilde{\pi}^{st}_{in}(i + b_{st}))$ and $\tilde{\pi}^s(\tilde{j}) = \tilde{\pi}^{rs}_{out}(j)$. In this case, it must hold

$$\sum_{l=\tilde{i}}^{\tilde{j}-1} p_{\tilde{\pi}^s(l)} < \sum_{l=1}^{k-1} p_{u_l}.$$

3. If the incoming arc of B is a buffer arc and the outgoing arc of B is a job arc, let the incoming arc of block B be the buffer arc $\pi^{st}_{out}(i) \rightarrow \rho_\Pi\,(\pi^{st}_{in}(i + b_{st})) = u_1$ of buffer B_{st}. Assume, that in solution $\tilde{\Pi}$, operation $\rho_{\tilde{\Pi}}\,(\tilde{\pi}^{st}_{in}(i + b_{st}))$ is processed

on M_s at position $\tilde{i}$ in $\tilde{\pi}^s$ and operation u_k is processed on M_s at position $\tilde{p}$ in $\tilde{\pi}^s$, i.e. $\tilde{\pi}^s(\tilde{i}) = \rho_{\tilde{\Pi}}\left(\tilde{\pi}^{st}_{in}(i + b_{st})\right)$ and $\tilde{\pi}^s(\tilde{p}) = u_k$. In this case, it must hold

$$\sum_{l=\tilde{i}}^{\tilde{p}} p_{\tilde{\pi}^s(l)} < \sum_{l=1}^{k} p_{u_l}.$$

4. If the incoming arc of B is a job arc and the outgoing arc of B is a buffer arc, assume that the outgoing arc of block B is the buffer arc $u_k = \pi^{rs}_{out}(j) \rightarrow \rho_\Pi\left(\pi^{rs}_{in}(j + b_{rs})\right)$ of buffer B_{rs}. In solution $\tilde{\Pi}$, let operation u_1 be processed on M_s at position $\tilde{q}$ in $\tilde{\pi}^s$ and let operation $\tilde{\pi}^{rs}_{out}(j)$ be processed on M_s at position $\tilde{j}$ in $\tilde{\pi}^s$, i.e. $\tilde{\pi}^s(\tilde{q}) = u_1$ and $\tilde{\pi}^s(\tilde{j}) = \tilde{\pi}^{rs}_{out}(j)$. In this case, it must hold

$$\sum_{l=\tilde{q}}^{\tilde{j}-1} p_{\tilde{\pi}^s(l)} < \sum_{l=1}^{k-1} p_{u_l}.$$

(If $\tilde{p} < \tilde{i}$ in Condition 3 or $\tilde{j} - 1 < \tilde{q}$ in Condition 4 holds, we assume the value of the corresponding sum to be 0.) □

The proof of this theorem is similar to that of Theorem 6.1 and can be found in Nieberg [26].

6.5.2 Neighborhood structures

In the case of the job-shop problem with pairwise buffers, necessary properties to possibly improve a given feasible solution $\Pi = (\pi^1, \ldots, \pi^m)$ are given in Theorem 6.3.

Let $B = (u_1, \ldots, u_k)$ be a block on a critical path in the solution graph $\bar{G}(\Pi)$ and assume that block B is processed on machine M_s. If the incoming and/or outgoing arc of B are job arcs, promising shifts can be defined similar as in the case of the flow-shop problem with intermediate buffers: If the incoming arc of B is a job arc, then one operation from $\{u_2, \ldots, u_k\}$ must be shifted before operation u_1. If the outgoing arc of B is a job arc, then one operation from $\{u_1, \ldots, u_{k-1}\}$ must be shifted after operation u_k.

If the incoming and/or outgoing arc of B are buffer arcs, the characterization of promising shifts is more extensive than in the case of the flow-shop problem with intermediate buffers. First, let us assume that the incoming arc of block B is the buffer arc $\pi^{st}_{out}(i) \rightarrow \rho_\Pi\left(\pi^{st}_{in}(i + b_{st})\right) = u_1$ of buffer B_{st}. By $\rho_\Pi(j)$ and $\nu_\Pi(j)$ we again denote the direct machine successor and the direct machine predecessor of operation j with respect to the solution Π, respectively. To get a possible improvement we have to change π^s, such that the length of block B reduces. The length of B composes

of the processing time of the end vertex of the incoming buffer arc as well as the sum of processing times of the operations scheduled between the end vertex of the incoming buffer arc and u_k. In order to determine this length, we have to distinguish whether the shifting operation is also assigned to buffer B_{st}. If we perform one of the following types of shifts, the length of the block may be reduced:

- An operation w of block B, which is not assigned to buffer B_{st}, is shifted directly before operation $\nu_\Pi(u_1)$. Thus, after this shift the incoming buffer arc of block B still terminates at operation u_1 and the length of B is reduced by p_w.
- An operation w of block B, which is assigned to buffer B_{st} and which has a larger processing time than operation $\nu_\Pi(u_1)$, is shifted directly before operation $\nu_\Pi(u_1)$. As w is assigned to buffer B_{st}, the incoming buffer arc terminates at operation $\nu_\Pi(u_1)$ after the shift. Thus, operation $\nu_\Pi(u_1)$ enters block B and operation w leaves block B. Since the processing time of w is larger than that of $\nu_\Pi(u_1)$, the length of B is reduced.
- Operation $\nu_\Pi(u_1)$ is shifted to the right into block B. As insertion positions only positions up to the position of operation $\pi^{st}_{in}(i + b_{st} + 1)$ are considered. After this shift, the entering blocking arc terminates at one operation from $\{u_2, \ldots, u_k\}$ and the length of block B is reduced by at least p_{u_1}.

Now, assume that the outgoing arc of block B is the buffer arc $u_k = \pi^{rs}_{out}(j) \rightarrow \rho_\Pi(\pi^{rs}_{in}(j + b_{rs}))$ of buffer B_{rs}, i.e. the j-th buffer arc of buffer B_{rs}. In this case, we have to change π^s, such that the sum of processing times of the operations scheduled between u_1 and the starting vertex of the j-th buffer arc of B_{rs} reduces. Note, that the processing time of the operation, where the j-th buffer arc of B_{rs} emanates, does not contribute to this sum (see Theorem 6.3). Symmetrically to the above defined shifts in the case of a buffer arc as entering arc, the following shifts might improve the solution Π if the outgoing arc is a buffer arc:

- An operation w of block B, where its direct job predecessor is not assigned to B_{rs}, is shifted directly after u_k. After this shift the outgoing buffer arc of block B still emanates from operation u_k and the length of B is reduced by p_w.
- An operation w of block B, where its direct job predecessor is assigned to B_{rs} and which has a larger processing time than operation u_k, is shifted directly after operation u_k. After this shift, the j-th buffer arc of buffer B_{rs} emanates from operation w and thus, p_w is no longer relevant for the length of block B. Instead, the processing time p_{u_k} contributes to the length of B. Since $p_w > p_{u_k}$ holds, the length of the block is reduced.
- Operation u_k is shifted to the left into block B. As insertion positions only positions up to the position of operation $\pi^{rs}_{out}(j - 1)$ are considered. By this

shift, operation u_k (and therefore also the outgoing buffer arc) is moved at least one position to the left in π^s and the length of block B is reduced by at least $p_{u_{k-1}}$.

Neighborhood $\mathcal{N}_P^1$: Let $\Pi = (\pi^1, \ldots, \pi^m)$ be a feasible solution of the job-shop problem with pairwise buffers, and let $CP(\Pi)$ be a critical path in $\bar{G}(\Pi)$. Furthermore, let $B = (u_1, \ldots, u_k)$ be an arbitrary block of $CP(\Pi)$ which is processed on machine M_s.
Neighborhood $\mathcal{N}_P^1(\Pi)$ consists of all feasible solutions $(\pi^1, \ldots, \pi^{s-1}, \tilde{\pi}^s, \pi^{s+1}, \ldots, \pi^m)$ which can be constructed by applying the following shifts to all blocks B of the chosen critical path $CP(\Pi)$:

1. If the incoming arc of B is a job arc, then one operation of block B, different from the first operation in B, is shifted at the beginning of block B (i.e. directly before operation u_1).

2. If the incoming arc of B is a buffer arc, then

   (a) one operation w of B, which is assigned to a different buffer as operation $\nu_\Pi(u_1)$, is shifted directly before operation $\nu_\Pi(u_1)$, or

   (b) one operation w of B, which is assigned to the same buffer as operation $\nu_\Pi(u_1)$ and where $p_w > p_{\nu_\Pi(u_1)}$ holds, is shifted directly before operation $\nu_\Pi(u_1)$, or

   (c) operation $\nu_\Pi(u_1)$ is shifted to the right into block B at most until directly before the next operation which is assigned to the same buffer as operation $\nu_\Pi(u_1)$.

3. If the outgoing arc of B is a job arc, then one operation of block B, different from the last operation in B, is shifted at the end of block B (i.e. directly after operation u_k).

4. If the outgoing arc of B is a buffer arc, then

   (a) one operation w of B, where its job predecessor is assigned to a different buffer than the job predecessor of operation u_k, is shifted directly after operation u_k, or

   (b) one operation w of B, where its job predecessor is assigned to the same buffer as the job predecessor of operation u_k and where $p_w > p_{u_k}$ holds, is shifted directly after operation u_k, or

   (c) operation u_k is shifted to the left into block B at most until directly after the previous operation whose job predecessor is assigned to the same buffer as the job predecessor of operation u_k. □

As in the case of the job-shop problem with blocking operations, a lot of solutions are infeasible after applying one of the shifts of neighborhood $\mathcal{N}_P^1$ to a feasible solution Π. In such cases, we again apply the repair procedure described in Subsection 6.3 in order to get feasible neighbors of Π.

Neighborhood $\mathcal{N}_P^2$: Let $\Pi = (\pi^1, \ldots, \pi^m)$ be a feasible solution of the job-shop problem with pairwise buffers. Neighborhood $\mathcal{N}_P^2(\Pi)$ consists of all feasible solutions $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ which can be constructed by applying the shifts described for neighborhood $\mathcal{N}_P^1$ and by repairing the resulting solution into a feasible solution according to the repair procedure described in Subsection 6.3, if necessary. □

6.6 The job-shop problem with output buffers

In the job-shop problem with ouput buffers, for each machine M_k $(k = 1, \ldots, m)$ an output buffer B_k is given. In buffer B_k, all jobs are stored which leave machine M_k and cannot directly be processed on the following machine. For feasible sequences $\pi^1, \ldots, \pi^m$ of the jobs on the machines, a polynomial procedure was developped in Section 5.3 which calculates an optimal buffer slot assignment and a corresponding schedule at the same time.

In the following, we show which difficulties arise when trying to derive a block approach for the case of a job-shop problem with output buffers. Example 6.6 shows that even by changing the sequence of jobs on a machine, which is not involved in a critical path, the makespan of the given solution can be reduced.

Example 6.6 : Consider an instance with three machines and six jobs given by Table 6.3. We assume that the output buffers B_1, B_2 and B_3 have a capacity of $b_1 = 2$, $b_2 = 1$ and $b_3 = 0$ units, respectively. Figure 6.19 shows a schedule for the given instance where the jobs on machine M_1, M_2 and M_3 are sequenced in the order $\pi^1 = (1, 2, 3, 4)$, $\pi^2 = (4, 5, 2)$ and $\pi^3 = (6, 1, 3)$, respectively. Let $\Pi = (\pi^1, \pi^2, \pi^3)$.

p_{ij}	j					
i	1	2	3	4	5	6
1	2	2	2	4	5	8
2	2	2	2	7	–	–

μ_{ij}	j					
i	1	2	3	4	5	6
1	M_1	M_1	M_1	M_2	M_2	M_3
2	M_3	M_2	M_3	M_1	–	–

Table 6.3: Instance of a job-shop problem with output buffers

In the schedule of Figure 6.19, jobs 1 and 3 are assigned to buffer slot B_1^1 and job 2 is assigned to buffer slot B_1^2. The buffer input sequences π_{in}^1 and π_{in}^2 are resulting from the machine sequences π^1 and π^2, respectively, and are $\pi_{in}^1 = (1, 2, 3)$ for buffer B_1 and $\pi_{in}^2 = (4)$ for buffer B_2. The buffer output sequence π_{out}^1 can be derived from the

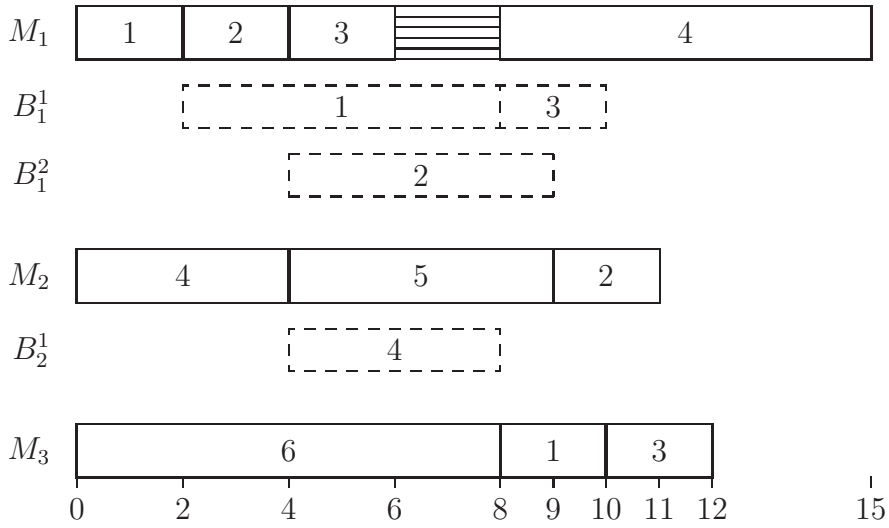

Figure 6.19: Schedule for $\pi^1 = (1, 2, 3, 4)$, $\pi^2 = (4, 5, 2)$ and $\pi^3 = (6, 1, 3)$

schedule in Figure 6.19 by ordering jobs 1, 2 and 3 according to the time at which they start processing on their succeeding machine. Thus, it is $\pi^1_{out} = (1, 2, 3)$ and $\pi^2_{out} = (4)$.

Consider now jobs 1 and 3 which are both assigned to buffer slot B_1^1. Job 3 cannot enter buffer slot B_1^1 before job 1 leaves it to continue on its next machine, i.e. when operation O_{21} starts processing on M_3. At this time, the blocking period on M_1 is finished and job 4 can start processing on M_1. Thus, we have a blocking arc from O_{21} to O_{24} (see also Subsection 4.3 for the definition of blocking arcs). Since there is only one buffer slot occupied by two jobs, the solution graph $\bar{G}(\Pi)$ only contains one buffer arc. The complete solution graph $\bar{G}(\Pi)$ is shown in Figure 6.20.

The critical path of solution Π consists of the following operations and arcs (shown with their corresponding weights):

$$\circ \xrightarrow{0} \underbrace{O_{16} \xrightarrow{8} O_{21}}_{\text{block } B_1 \text{ on } M_3} \xrightarrow{0} \underbrace{O_{24}}_{\text{block } B_2 \text{ on } M_1} \xrightarrow{7} *$$

If we swap now the processing of jobs 5 and 2 on M_2 we obtain the schedule shown in Figure 6.21. In this schedule, job 2 can continue processing on M_2 immediately after finishing on M_1. Thus, job 2 does not enter buffer B_1 and job 3 can occupy buffer slot B_1^2 instead. Therefore, no blocking time on M_1 occurs and the makespan reduces to 13. Note, that by changing the sequence of the jobs on M_2 the buffer slot assignment of the output buffer B_1 has changed and therefore, no blocking time on M_1 occurs. The buffer output sequence of B_1 is $\tilde{\pi}^1_{out} = (2, 1, 3)$ after the swap.

□

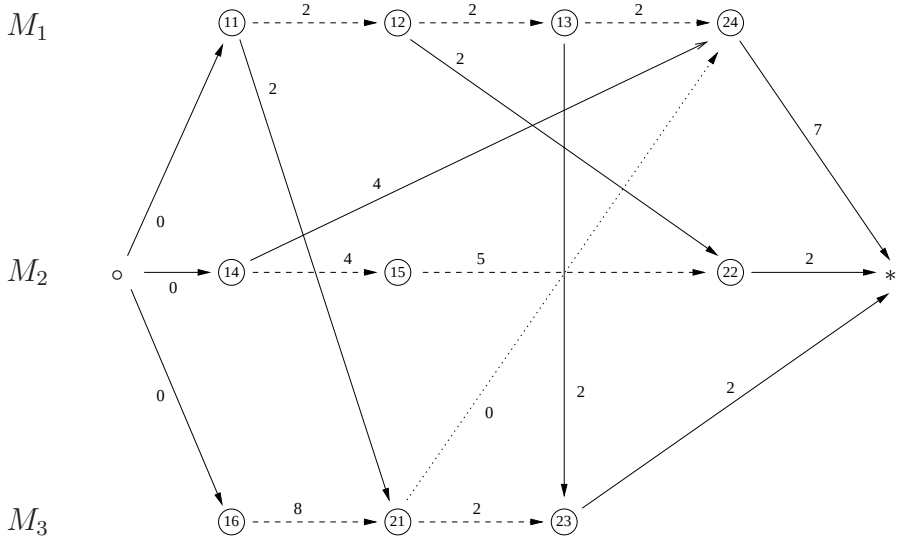

Figure 6.20: Solution graph $\bar{G}(\pi^1, \pi^2, \pi^3)$

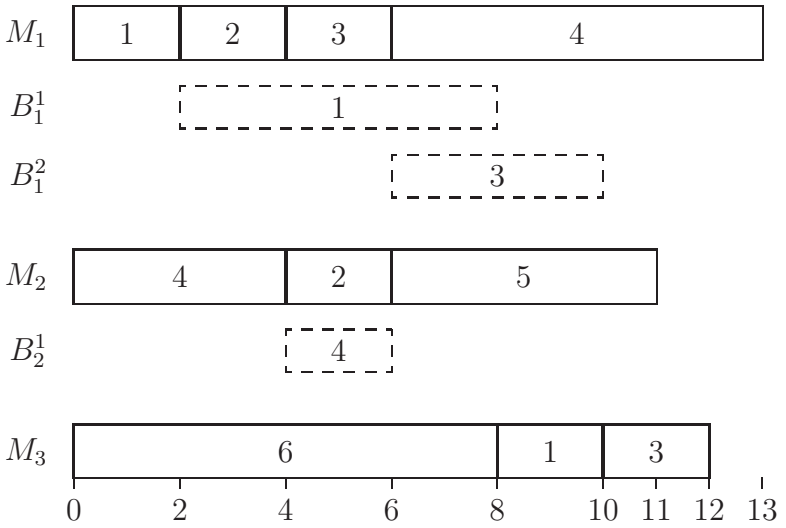

Figure 6.21: Schedule after a swap of jobs 5 and 2 on M_2

In all previously treated buffer models (i.e. the flow-shop problem with intermediate buffers, the job-shop problem with blocking operations and the job-shop problem with pairwise buffers), we can assign to each buffer two fixed surrounding machines. However, for an ouput buffer B_k, we have on the one hand machine M_k which processes all jobs entering B_k but on the other hand there may be several machines to which jobs leaving B_k may proceed. Changing the sequence of the jobs on one of these outgoing machines may result in a different buffer slot assignment and buffer slot sequence of B_k (see Example 6.6). Therefore, a blocking time on machine M_k may be reduced by changing the sequence of the jobs on one of the outgoing machines of B_k which must not be involved in the critical path.

In addition, a blocking time starting at time t on M_k may also be reduced when the sequence of the jobs on an arbitrary machine is changed and thus, also the buffer slot assignment of some buffer before time t is changed. This different buffer slot assignment at an earlier time may cause a different buffer slot assignment of buffer B_k at time t. Therefore, for a job-shop problem with output buffers, it is difficult to formulate an easy condition for possible improvements of a given solution in the case when the incoming arc of some block on a critical path is a blocking arc.

Considering a solution of a job-shop problem with output buffers given by the machine sequences and the resulting buffer slot assignment as a solution of a blocking job-shop problem, we may apply Theorem 6.2 to identify necessary conditions to improve this solution. However, Theorem 6.2 only proposes changes of the sequences of the jobs on the original machines and the blocking machines, which are the buffer slots in the original problem. Nevertheless, as Example 6.6 shows, by changing the assignment of jobs to buffer slots (together with a change of a machine sequence which must not be involved in the critical path) the current solution can be improved.

In the following, we consider the application of local search to the three cases for which we derived block approaches in this section. If a local search approach should be applied to the job-shop problem with output buffers, an appropriate neighborhood structure (independent from a block approach) has to be defined. This could be derived from simple neighborhood structures for the classical job-shop problem.

7 Tabu search approaches

In this section we present tabu search approaches for the job-shop problem with different buffer models which use the neighborhoods we described in the previous section. The tabu search is a local search heuristic which exploits information from the recent search process in order to decide in which direction the search procedure will be continued. In this study, our aim is not to compare the effectiveness of different local search methods (such as e.g. iterative improvement, simulated annealing, threshold acceptance, tabu search) applied to the job-shop problem with limited buffer capacities but to show that the model and the concepts we developped may be applied successfully in a heuristic approach. The decision to chose a tabu search approach resulted from the success of this method for shop problems and also for various extensions of shop problems (see, e.g., Aarts et al. [1], Nowicki & Smutnicki [28], [29]). Surely, also variations of our approach for example in combination with genetic algorithms could be used to solve the job-shop problem with limited buffer capacities.

In Subsection 7.1, the general concept of the tabu search method is described. Afterwards, in Subsection 7.2, we propose tabu search approaches for the flow-shop problem with intermediate buffers and we report computational results for them. Tabu search approaches for the job-shop problem with pairwise buffers and the job-shop problem with blocking operations are presented in Subsections 7.3 and 7.4, respectively. For each problem type, some implementational details such as the generation of an initial solution and the efficient calculation of longest paths are discussed.

7.1 General concept

The **tabu search** method is a metaheuristic approach which was designed by Glover [12]. In each iteration of this local search method the current solution is usually replaced by the best solution in its neighborhood. Contrary to the iterative improvement method, also non-improving solutions are accepted during the search process. Thus, it is possible to leave local minima. On the other hand, solutions may be visited several times and therefore it is possible that cycles occur. One strategy to avoid cycling, is to store all visited solutions in a so-called **tabu list** and to move to a neighbor of the current solution only if it is not contained in this list. Due to memory restrictions in general it is not practicable to store all solutions visited so far. Thus, only solutions which have been visited in the recent iterations are contained in the list. Then only cycles with a length greater than the tabu list length may occur. If the tabu list length is sufficiently large, the probability of cycling becomes very small. But, it requires a high computational effort to store complete solutions in the tabu list and to compare other solutions with the elements of the list. Therefore, only typical properties (attributes) of the solutions or of previous moves are stored in the tabu list and all solutions having one of the properties stored in the tabu list are

declared as tabu. A disadvantage of this concept is that solutions which have never been visited may also be forbidden by the tabu list. To cancel the tabu status on a move so-called **aspiration criteria** are introduced. They allow to accept neighbors even if they are forbidden due to the tabu status. For example, a move should always be accepted if it improves the best solution found so far.

Besides the tabu list, which has the function of a short-term memory, often also a long-term memory is kept which is used for **diversification**. In this long-term memory promising solutions or properties of promising solutions which have been found during the search process are stored. If during the search process the best solution found so far has not been improved for a certain number of iterations, the tabu search is stopped and restarted with a new starting solution (diversification). For such a restart, a promising solution is selected from the long-term memory list or the long-term memory list is used to generate a solution having the promising properties. Thus, the diversification strategy helps to lead the search process away from search regions which have already been well-explored into hopefully good search regions. The whole tabu search procedure terminates after a maximal number of restarts has been finished or if a given time-limit is exceeded.

In the following we will describe how these general concepts are applied in our tabu search algorithms for the job-shop problem with different buffer models. The attributes of a solution $\Pi = (\pi^1, \ldots, \pi^m)$ stored in the tabu list and the tabu conditions will be explained first.

If an operator $rshift(i, j, k)$ or $lshift(i, j, k)$ is applied to the solution Π, besides the objective value $C_{\max}(\Pi)$ we store the pair of jobs $(\pi^i(j), \pi^i(k))$ as an attribute, which characterizes the order of jobs in permutation π^i before applying the operator:

- for the operator $rshift(i, j, k)$ we use the shifted job $\pi^i(j)$ and its new predecessor $\pi^i(k)$ as an attribute, i.e. we store $(\pi^i(j), \pi^i(k))$, and
- for the operator $lshift(i, j, k)$ we use the shifted job $\pi^i(j)$ and its new successor $\pi^i(k)$ as an attribute, i.e. we store $(\pi^i(k), \pi^i(j))$.

A neighbored solution $\tilde{\Pi} = (\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ of the solution Π is defined as tabu if the permutation $\tilde{\pi}^i \neq \pi^i$ results from π^i by reconstructing the order of a pair of jobs belonging to the tabu list. More specifically, $\tilde{\Pi}$ is tabu if

- $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ is constructed from $(\pi^1, \ldots, \pi^m)$ by the operator $rshift(i, j, k)$ and a pair $(\pi^i(l), \pi^i(j))$ with $l \in \{j+1, \ldots, k\}$ is contained in the tabu list, or
- $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ is constructed from $(\pi^1, \ldots, \pi^m)$ by the operator $lshift(i, j, k)$ and a pair $(\pi^i(j), \pi^i(l))$ with $l \in \{k, \ldots, j-1\}$ is contained in the tabu list.

A neighbored solution $\tilde{\Pi}$ satisfies the aspiration criterion if its makespan $C_{\max}(\tilde{\Pi})$ is smaller than the makespan of all solutions with attributes belonging to the tabu list which declare the new solution tabu.

We use a fixed-length tabu list which is emptied if in some iteration of the search a new globally best solution is found. The length of the list is chosen dependent on the number of operations in the current instance.

For selecting a non-tabu neighbor of a given solution, we follow the most commonly used best-fit strategy, i.e. in each iteration we choose a non-tabu neighbor $\tilde{\Pi}$ with minimal $C_{\max}(\tilde{\Pi})$-value.

Our diversification method is based on the following restart technique: The tabu search is stopped if for a certain number of iterations (*#iterations*) the best found solution has not been improved and restarts the search with the second best (third best and so forth) shift from the current best globally found solution. After returning a maximal number (*#restarts*) to the same best solution and restarting the search from it with a shift in a different direction, the whole tabu search procedure is stopped. If in some iteration of the search a new globally best solution is found, the counters for the number of iterations and the number of restarts are reset to 0. Thus, in general, the number of total iterations is much larger than the product of *#iterations* and *#restarts*. This type of diversification has been used also in Nowicki and Smutnicki [28] and in Nowicki [27].

Further details and computational results for our tabu search algorithms can be found in the next three subsections.

7.2 The flow-shop problem with intermediate buffers

In this section, we describe a tabu search approach for the flow-shop problem with intermediate buffers which is based on the neighborhood structures presented in Subsection 6.2.

As initial solution of the tabu search method, we calculate a permutation solution since such solutions are feasible for arbitrary buffer configurations. To do so, we use a relatively fast generalization of an algorithm which was proposed by Nawaz et al. [25] for the permutation flow-shop problem with infinite buffer capacities: The jobs are considered in the order of non-increasing total processing times. We insert the next job at that position in the sequence of already scheduled jobs where the makespan under consideration of limited buffer capacity is minimized. Leisten [21] showed that this constructive algorithm for the flow-shop problem with limited buffers provides the best results for problems with more than two machines (even if it is compared with heuristics where the passing of jobs is allowed).

7.2.1 Efficient calculation of longest paths

In our proposed local search approach the procedure to calculate the best schedule given a fixed m-tuple of permutations plays a central role, as it is used to calculate the objective value of the solutions. In the following, we give a detailed description how the longest path calculation in the solution graph is realized. Although this is a rather technical issue, its use is crucial for getting an efficient local search approach.

In a standard implementation of a longest paths algorithm for an acyclic graph, a topological ordering of the nodes is determined first. This means that the nodes are numbered in that way that $i < j$ for every arc (i, j) of the graph. Given a topological ordering, the longest paths distances $d(i)$ for all nodes i can be calculated iteratively. Suppose nodes $1, \ldots, k-1$ are evaluated. The topological ordering implies that all arcs directed into node k emanate from one of the nodes 1 through $k-1$. Thus, a longest path to node k is composed of a longest path to one of the nodes $i = 1, \ldots, k-1$ together with the arc (i, k). Therefore, to compute the longest path distance of node k we need only select the minimum of $d(i) + w_{ik}$ for all predecessors i of k (where w_{ik} is the weight of (i, k)).

Regarding the solution graph of a flow-shop problem with intermediate buffers each node has at most three predecessors, namely the machine-, job- and buffer arc-predecessor. Thus, given a topological ordering, longest paths can be calculated efficiently. Due to the special structure of the solution graph of a feasible solution (see Subsection 5.1.2), a topological ordering of the nodes can be determined with the following procedure: We number the operations on each machine from the first to the last operation starting with the first operation on M_1. Each time a buffer arc terminates in the current vertex k, follow the buffer arc backwards until a vertex i without a buffer arc predecessor is reached. Starting with i, we number the vertices on the path from i to k. If the last operation on a machine is numbered, continue with the next operation on the succeeding machine which is not yet numbered.

The following example illustrates how this procedure finds a topological ordering in the solution graph.

Example 7.1 : Consider a flow-shop problem with four machines, eight jobs and three intermediate buffers which have a capacity of $b_1 = 1$, $b_2 = 0$ and $b_3 = 2$ units. Figure 7.1 shows a topological ordering of the vertices in the corresponding solution graph (where only the buffer arcs are represented).

At first, operations on M_1 are numbered until the first operation on M_1 has a buffer arc predecessor. This is the case for the third operation on M_1. We follow the buffer arc backwards until the first operation on M_2. Since this does not have a buffer arc predecessor, we number the first operation on M_2 and the third operation on M_1. We continue with the forth operation on M_1 and follow its buffer arc backwards. We reach the second operation on M_2 which also has a buffer arc predecessor. Now,

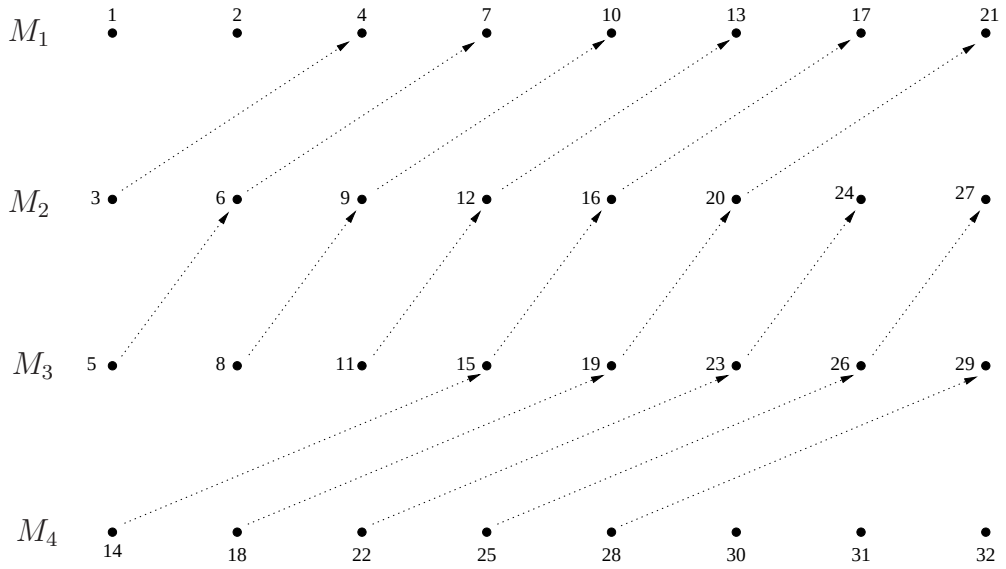

Figure 7.1: Topological ordering of the vertices in a solution graph

operations on M_3, M_2 and M_1 are numbered alternately until the first operation on M_3 has a buffer arc predecessor. Then, operations on M_4, M_3, M_2 and M_1 are numbered alternately until all operations on M_1 are numbered. We continue by numbering operations on M_4, M_3 and M_2 alternately until all operations on M_2 are numbered. Afterwards, an operation on M_4 and the last operation on M_3 are numbered. Finally, the remaining operations on M_4 are numbered. □

It remains to show that the procedure yields a topological ordering, i.e. that $i < j$ for every arc (i, j) in the solution graph. Obviously, for all machine and buffer arcs (i, j) it is $i < j$. Now, assume that there exists a job arc (i, j) from an operation g on machine M_k to an operation h on M_{k+1} such that $i > j$ holds. Since operation h on M_{k+1} was numbered before operation g on M_k, there must be a buffer arc from h to an operation on M_k which has number $j + 1 < i$. This buffer arc together with machine arcs from $j + 1$ to i and the job arc (i, j) would form a cycle in the solution graph and the solution would not be feasible.

Note, that this ordering is only dependent on the given buffer capacities. Thus, given an instance of a flow-shop problem with intermediate buffers we use the same topological ordering for each feasible solution of the instance. Computational tests have shown that using this fixed ordering instead of calculating a topological ordering each time speeds up the longest path calculation considerably. Furthermore, the fixed topological order has another advantage concerning the evaluation of neighbored solutions. Based on the makespan computation of a given solution, the makespan of neighbors of this solution can be computed very effectively. This is done in two steps: In the first step, we determine the first position p in the topological order where different operations in the given and the neighbored solution are scheduled. In the second step, the starting times of the operations until position p are taken

from the given solution and the starting times of the operations on position p and later are recalculated. If a single shift leads to the neighbored solution, the specific position p in the topological order can be determined in constant time. If a repair step is necessary, we have to consider the earliest position on each machine which is concerned by the repair. One of these at most m positions specifies p.

Clearly, the larger the position in the topological order where the solutions differ the first time, the less new calculations are needed. Computational tests have shown that using this evaluation technique for neighbored solutions saves a lot of computing time.

7.2.2 Computational results

In this subsection we report on some computational results achieved with the described tabu search algorithm. We implemented the procedure in C and tested it on a SUN-Enterprise 450 (4× 250 MHz-CPU, 2 GB RAM) with operating system Solaris 2.5.

As no test instances are available for the flow-shop problem with intermediate buffers, we modified $m \times n$ benchmark problems for the classical flow-shop problem, where m denotes the number of machines and n the number of jobs. We used different instances from Taillard [35] with up to 10 machines and 100 jobs. The instances are divided into 8 different classes, where each class contains 10 instances of the same dimension. The processing times of the operations in all instances are uniformly distributed over the interval $[1, 99]$.

To each of the test instances we added different types of buffer configurations. In order to compare our results with the known results for the classical flow-shop problem, we first set the buffer capacity equal to $n - 1$ for all buffers which means the flow-shop problem with buffers reduces to a classical flow-shop problem ($b_i = \infty$). On the other hand we considered the situation where after each machine no buffer is available ($b_i = 0$). In this case we have a blocking flow-shop problem. Furthermore, we considered instances where all buffers have the same capacity of $b_i = 1$ and $b_i = 2$ units for $i = 1, \ldots, m - 1$. We did not consider instances with different buffer sizes since first computational tests with those instances provided no new insight. Thus, altogether we used $8 \cdot 10 \cdot 4 = 320$ test instances.

Each of these instances was solved with the parameter combinations given by Table 7.1. All test results (best found makespan and computational time for each test run) can be found on the web-site
`http://www.mathematik.uni-osnabrueck.de/research/OR/software.html`
The makespan of the initial solution and the best found makespan over all test runs for the different buffer capacities are also documented on this web-site.

In the following, we give an overview of the results. First, we evaluate the quality

$\#iterations$	500			1000			3000		10,000
$\#restarts$	1	3	5	1	3	5	1	3	1

Table 7.1: Different parameter combinations for the tabu search

of the proposed local search approach. Afterwards, we indicate the influence of the buffer capacities on the makespan of the best found solution.

Quality of the tabu search approach

For the flow-shop problem with intermediate buffers, no methods to compute lower bounds for the optimal makespan are available. But, in the case of $b_i = \infty$ the flow-shop problem with buffers reduces to a classical flow-shop problem. Thus, we can compare the computational results for this case with the results from the literature (see Vaessens [37], OR Library).

In the following, we compare the best makespan C_{tabu} found by the tabu search with the best known makespan C_{best} for an instance. For 30 of the 80 instances the solution C_{best} is proven to be optimal. Almost all of the results C_{best} were achieved using branch & bound techniques. Notice that often, the best known solutions for these flow-shop instances do not vary very much from permutation solutions or even are permutation solutions.

In order to estimate the quality of the tabu search procedure we determined the reduction

$$red = 100 \cdot \frac{C_{start} - C_{tabu}}{C_{start} - C_{best}}$$

of the gap between the initial permutation solution C_{start} and the best known solution. In Table 7.2 the average reduction red and the mean computational time CPU (in minutes : seconds) over all 10 test instances of one problem class is shown.

Table 7.2 shows that for the small instances (5×20, 10×20, 5×50) and a parameter configuration of $\#iterations = 3000$ and $\#restarts = 3$ the gap between initial makespan and best known makespan was reduced by approximately 25% to 38%. Further tests showed that these reduction values could be improved to 40% to 50% for the small instances when a configuration $\#iterations = 3000$ and $\#restarts = 10$ was chosen (which took four times longer).

For the larger instances (20×20, 10×50, 20×50, 5×100, 10×100), the reduction values are considerably less than for the small instances: For $\#iterations = 3000$ and $\#restarts = 3$ there was achieved a reduction of 5% to 15%, which was improved to 10% to 20% when the parameter $\#restarts$ was increased to 10 (CPU-time was also approximately four times longer in this case). The minor percentage reductions for the larger instances indicate that for these instances with considerably larger solution spaces more iterations are needed.

Altogether, for 3 of the 80 instances, the best known makespan was reached. The relative deviation $100 \cdot (C_{tabu} - C_{best})/C_{best}$ of the best found makespan from the best

			500			1000			3000		10,000
m	n		1	3	5	1	3	5	1	3	1
5	20	red	9.34	15.86	21.68	12.32	19.50	22.63	29.76	32.54	32.01
		CPU	0:02	0:05	0:09	0:04	0:10	0:19	0:15	0:37	0:47
10	20	red	14.32	17.19	23.19	14.32	17.19	23.19	20.92	25.61	26.78
		CPU	0:03	0:11	0:19	0:07	0:22	0:39	0:32	1:17	1:41
20	20	red	5.76	7.29	10.39	5.76	7.29	10.39	5.76	7.29	5.76
		CPU	0:06	0:17	0:27	0:13	0:35	0:58	0:42	1:48	2:23
5	50	red	22.81	33.40	34.74	27.11	35.50	36.83	27.11	37.87	27.11
		CPU	0:11	0:25	0:38	0:21	0:49	1:15	0:57	2:41	2:56
10	50	red	7.48	9.43	10.15	7.48	9.99	10.52	9.50	14.32	12.54
		CPU	0:17	0:43	1:06	0:33	1:26	2:06	1:51	5:11	6:26
20	50	red	4.75	6.18	7.61	4.75	6.18	7.61	4.79	6.22	4.79
		CPU	0:32	1:24	2:28	1:02	2:41	4:47	3:18	7:56	10:44
5	100	red	3.17	3.44	3.44	3.17	3.44	3.44	4.76	5.89	11.27
		CPU	0:31	1:04	1:35	1:04	2:12	3:15	3:10	6:55	11:28
10	100	red	5.35	5.86	6.05	5.35	5.86	7.06	5.35	6.16	5.35
		CPU	0:58	1:57	3:04	1:51	3:55	6:41	5:33	11:10	18:07

Table 7.2: Mean reduction *red* for the case $b_i = \infty$ and different parameter configurations

known makespan amounts averagely 2.45% over all test instances, where the relative deviation of the starting solution from the best known makespan amounts averagely 3.65%.

Summarizing, for the classical flow-shop problem the tabu search seems to yield quite good solutions in reasonable computational times, especially when considering that the tabu search approach was not designed for the classical flow-shop problem and that the compared values C_{best} were achieved with branch & bound methods.

Next, we evaluate the quality of the tabu search for instances with a buffer capacity equal to $b_i = 0$, 1 and 2. In order to compare the results achieved with or without the restart technique, we consider the tests for the parameter setting $\#iterations = 1000$, $\#restarts = 5$ and $\#iterations = 3000$, $\#restarts = 1$, which took a comparable computational time according to Table 7.2.

Since in the case $b_i = 0$, 1 and 2 no lower bounds for the optimal makespan are available, we determined the relative improvement

$$imp = 100 \cdot \frac{C_{start} - C_{tabu}}{C_{start}}$$

of the makespan of the initial permutation solution C_{start}. The initial solution was

Dimension		$b_i = 0$		$b_i = 1$		$b_i = 2$		$b_i = \infty$	
n	m	mean	max	mean	max	mean	max	mean	max
20	5	1.18	2.75	1.25	2.84	0.85	3.60	1.32	4.21
	10	1.73	3.79	1.23	5.06	1.22	2.52	1.22	2.40
	20	0.92	2.64	0.29	1.23	0.14	0.66	0.13	0.66
50	5	1.61	3.44	0.77	1.40	0.42	1.73	0.36	1.80
	10	2.18	2.99	1.16	2.14	0.69	1.96	0.51	1.32
	20	0.79	1.74	0.87	1.41	0.42	1.34	0.29	1.04
100	5	1.74	3.70	0.39	1.12	0.47	0.91	0.04	0.21
	10	2.10	3.80	1.00	1.47	0.73	1.39	0.13	0.40

Table 7.3: Relative improvements *imp* for $\#iterations = 3,000$, $\#restarts = 1$

Dimension		$b_i = 0$		$b_i = 1$		$b_i = 2$		$b_i = \infty$	
n	m	mean	max	mean	max	mean	max	mean	max
20	5	1.61	3.95	1.43	3.30	1.53	4.21	0.94	3.00
	10	2.14	4.16	1.61	5.26	1.45	2.95	1.34	2.83
	20	1.34	2.64	0.53	1.42	0.66	3.36	0.35	1.51
50	5	2.64	3.90	0.93	1.47	0.63	1.98	0.42	1.80
	10	2.65	4.17	1.52	2.23	1.08	2.32	0.57	1.32
	20	1.05	2.33	0.95	1.50	0.67	1.34	0.45	1.69
100	5	2.44	4.35	0.53	1.31	0.56	1.04	0.02	0.11
	10	2.83	4.54	1.29	1.99	0.89	2.20	0.17	0.40

Table 7.4: Relative improvements *imp* for $\#iterations = 1,000$, $\#restarts = 5$

calculated by a generalization of a heuristic of Nawaz et al. [25]. In Tables 7.3 and 7.4 we report the average and maximal relative improvements *imp* of all 10 test instances of one problem class (for the two above mentioned parameter settings). Table 7.5 shows the mean computational time (in minutes : seconds) of the tabu search for $\#iterations = 3000$ and $\#restarts = 1$. The computational times for the parameter setting $\#iterations = 1000$ and $\#restarts = 5$ lies in the mean 39% over the values in Table 7.5.

- Tables 7.3 and 7.4 show that the relative improvement values achieved with the restart technique are almost always better than the results achieved without this technique. Thus, as for the classical permutation flow-shop problem (see Nowicki & Smutnicki [29]) the restart technique improves the efficiency of the tabu search.

- Comparing the results for different buffer capacities, almost all relative improvement values get better the less the buffer capacities are. This effect may be explained by the way we treat infeasible neighbors: For a capacity of $b_i = \infty$ all neighbors result from a single shift whereas for $b_i = 0$ each neighbor results from a shift together with a repair step. In general, with increasing buffer

Dimension		Buffer Capacity			
n	m	$b_i = 0$	$b_i = 1$	$b_i = 2$	$b_i = \infty$
20	5	0:23	0:19	0:14	0:15
	10	0:53	0:31	0:34	0:32
	20	1:28	0:49	0:47	0:42
50	5	6:01	2:35	1:19	0:57
	10	10:22	4:15	2:59	1:51
	20	19:43	9:52	4:26	3:18
100	5	46:10	11:15	5:44	3:10
	10	132:31	34:16	13:22	5:33

Table 7.5: Mean computational time (in minutes : seconds) for $\#iterations = 3,000$, $\#restarts = 1$

capacities the number of neighbors for which a repair step has to be executed decreases. In one execution of the repair procedure several operations on successive machines may be shifted to another position. As a consequence a solution is changed by one repair step more than by a simple shift. Thus, in a fixed number of iterations, on the average more changes are made for instances with small buffer sizes than for those with large buffer sizes. Obviously, the calculations for one iteration step with repair process take more time than the calculations for one iteration step without repair process. Thus, the computational time for fixed parameter setting will decrease with increasing buffer size. This can be seen in Table 7.5. One may expect that when the computational times for instances with different buffer capacities are similar, the gap between the relative improvement values for different buffer capacities will reduce.

- For the special case of the blocking flow-shop problem ($b_i = 0$), Ronconi [32] proposed two constructive heuristics (called MME and PFE). For each of these heuristics a priority list of jobs is generated which is then used as initial order of the jobs for the algorithm of Nawaz et al. [25]. For generating the priority lists, problem specific characteristics such as minimizing the machine idle time and the blocking time are exploited. Both heuristics are very fast with a computational time of less than a second and improve the heuristic of Nawaz et al. [25] in the mean by approximately 1%. The tabu search procedure achieves improvement values in the mean of approximately 2%, but of course needs much more computational effort. In order to improve the tabu search results in the special case of the blocking flow-shop problem, the MME and PFE heuristics could be used as initial solutions.

Influence of the buffer capacities on the makespan

Besides the quality of the proposed tabu search approach, another interesting aspect is the influence of the buffer capacities on the best makespan found by the tabu

Dimension		$b_i = 0$			$b_i = 1$			$b_i = 2$		
n	m	Δavg	Δmax	Δmin	Δavg	Δmax	Δmin	Δavg	Δmax	Δmin
20	5	14.48	23.22	6.30	1.63	4.71	0.00	0.02	1.06	-0.70
	10	11.64	15.55	5.97	1.55	3.59	-0.61	0.30	1.06	-0.42
	20	4.96	7.40	2.77	-0.13	1.53	-2.72	-0.30	1.53	-3.36
50	5	18.56	24.48	11.36	2.13	4.08	-2.74	0.01	0.34	-0.65
	10	20.95	25.16	17.16	2.78	3.81	1.05	-0.18	1.19	-1.32
	20	17.13	18.96	14.71	1.67	2.54	0.23	0.16	1.57	-0.43
100	5	20.45	22.92	18.04	4.21	5.66	3.28	0.22	0.80	-0.08
	10	25.64	29.91	21.52	4.18	8.18	1.13	0.20	1.59	-1.25

Table 7.6: Influence of the buffer capacity on the best found makespan (relative to the values for $b_i = \infty$)

search. In order to investigate this influence, we determined the relative change (for $j = 0, 1, 2$)

$$\Delta = 100 \cdot \frac{C^j_{tabu} - C^\infty_{tabu}}{C^\infty_{tabu}}$$

of the best found makespan relative to the case of a buffer capacity of $b_i = \infty$ (here, C^j_{tabu} denotes the best found makespan over all test settings for the flow-shop problem with a buffer capacity of $b_i = j$, $j = 0, 1, 2, \infty$). In Table 7.6 the average, maximal and minimal relative changes Δavg, Δmax and Δmin of all 10 test instances of one problem class are shown. A negative Δavg-value indicates that in the mean a relative improvement of the best found makespan was reached when reducing the buffer capacity. A negative Δmin-value indicates that for at least one problem of a problem class a better makespan was found when reducing the buffer capacity from $b_i = \infty$ to $b_i = j$ (although the optimal makespan cannot decrease by reducing the buffer capacity the achieved results of a heuristic may decrease).

The results of Table 7.6 can be summarized as follows: For a buffer capacity of $b_i = 2$, the makespan increases averagely by at most only 0.3% and in each problem class for at least one problem a better makespan was reached. For the case of a buffer capacity of $b_i = 1$, the significance of the buffer size becomes more evident, i.e. the makespan increases in the mean by up to 4.2%. The reduction of the buffer capacity to $b_i = 0$ results in a considerable increase of the makespan. In this case the Δavg-value amounts up to 25.6 %.

There are two main reasons for the increase of the makespan when reducing the buffer capacity: First, due to the buffer restriction, the number of feasible solutions decreases with decreasing buffer capacities. While for the case of the classical flow-shop ($b_i = \infty$) each arbitrary combination of job permutations is allowed, for the case of $b_i = 0$ only permutation solutions are feasible. But as the best known solutions for classical flow-shop problems often do not vary very much from permutation solutions, this restriction seems to have a minor influence on the makespan. A reduction to $b_i = 2$ has sometimes even a positive influence on the heuristic results (see negative

Δ-values in Table 7.6). This may be explained by the fact that the restriction of the buffer size forces the local search to stay 'close' to permutation solutions.

The major influence on the increase of the makespan results from the blocking restriction. If a buffer is filled completely, a job waiting to be inserted in this buffer blocks its machine, such that the processing of the following jobs on that machine is delayed. Obviously, the less the buffer capacities are, the more blocking situations may appear. Table 7.7 quantifies this statement. For each instance the influence of the blocking restriction for the best found solution over all test settings is analysed. Besides the number of occurring blocking situations (#block) on the computed critical path of these solutions, also the length bl of the blocking situations ($bl = i$, $i = 1, 2, > 2$) is given. Here, the length bl of a blocking situation is given by the number of consecutive buffer arcs belonging to this blocking situation. Finally, the row 'mean influence' denotes the relative improvement of the makespan of the best found solution if the blocking restriction is neglected (i.e. if b_i is increased to ∞).

		Dimension ($n \times m$)							
		20			50			100	
	b_i	5	10	20	5	10	20	5	10
#block	0	3.6	3.4	3.6	9.2	8.7	8.4	20.5	18.5
	1	0.4	0.7	0.2	2.1	1.6	2.0	5.0	5.6
	2	0.0	0.0	0.0	0.0	0.6	0.1	0.2	0.5
$bl = 1$	0	1.8	1.4	2.1	3.5	3.5	2.7	8.9	6.9
	1	0.3	0.6	0.2	1.0	1.1	1.4	2.8	3.0
	2	0.0	0.0	0.0	0.0	0.6	0.1	0.1	0.4
$bl = 2$	0	0.9	1.2	0.9	2.8	2.0	2.6	7.0	4.6
	1	0.1	0.1	0.0	0.7	0.2	0.6	1.2	1.5
	2	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0
$bl > 2$	0	0.9	0.8	0.6	2.9	3.2	3.1	4.6	7.0
	1	0.0	0.0	0.0	0.4	0.3	0.0	1.0	1.1
	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
mean influence	0	7.48	6.08	3.46	10.61	10.77	8.26	13.24	13.89
	1	0.20	0.15	0.02	1.27	0.57	0.32	2.34	1.78
	2	0.00	0.00	0.00	0.00	0.02	0.00	0.04	0.02

Table 7.7: Influence of the blocking restriction on the best found makespan

The test results confirm that blocking appears very seldom for a buffer capacity of at least $b_i = 2$. In the case of $b_i = 1$, the influence of blocking on the makespan becomes obvious. When no intermediate buffer space is available, the influence of blocking is considerable. On the one hand, in Table 7.7 the decrease of the makespan for $b_i = 0$ (row 'mean influence') is quite large. On the other hand, since the makespan of the best known permutation solution lies in the mean over all 80 test instances only 0.33% over that of the best known classical solution (see Taillard [35]), the restriction

to permutation solutions has only marginal influence. Therefore, the Δavg-values for $b_i = 0$ in Table 7.6 result mainly from the blocking restriction.

Note, that the processing times of the operations in the benchmark instances from Taillard [35] are uniformly distributed over the interval $[1, 99]$. However, in practical applications, the distribution of the processing times often shows less variance. Using such data in the test instances would probably have some positive consequences with regard to the blocking situations: Since the processing times vary less in the instances from practical applications than in the benchmark instances, the length of the blocking situations and also the number of blocking situations would reduce. Consequently, the makespan and the influence of the blocking-restriction on the makespan would reduce as well. Thus, the extended benchmark instances we used for our tests seem to be more difficult than problem instances from practical applications.

7.3 The job-shop problem with pairwise buffers

In this section, we propose a tabu search algorithm for the job-shop problem with pairwise buffers which applies the neighborhood structures described in Subsection 6.5. The tabu search procedure was implemented in C by Nieberg in the context of his diploma thesis (see Nieberg [26]).

As initial solution of the tabu search method, we use a solution slightly different from the starting solution in the case of the flow-shop problem with intermediate buffers. We modify a greedy algorithm which was proposed by Mascis et al. [24] for the blocking job-shop problem in that way that it always provides a feasible solution for the job-shop problem with pairwise buffers. We consider the operations within a job chain from the first to the last operation. The operations of a job which have not been scheduled yet are denoted by **the residual job**. Initially, the residual job consists of all operations of the corresponding job. In each step of the heuristic, we schedule the first operation i of a residual job with the largest processing time. This means that i is processed at the earliest possible time in the current schedule and that the buffer allocation is updated as follows:

- If the buffer $\beta(i)$ is not completely filled at the finishing time of operation i, put job $J(i)$ into buffer $\beta(i)$.
- If the capacity of buffer $\beta(i)$ is positive and $\beta(i)$ is already completely occupied at the finishing time of operation i, choose a residual job h with the longest processing time of all residual jobs in buffer $\beta(i)$. Replace h by $J(i)$ in buffer $\beta(i)$ and schedule the first operation of h.
- If buffer $\beta(i)$ has capacity 0, schedule the job successor $\sigma(i)$ of operation i.

Thus, in the case when all pairwise buffers have capacity 0, the constructed solution is a permutation solution. (In the case of a job-shop problem, a permutation solution

is given by a permutation π of the job indices and the jobs are scheduled on each machine according to the sequence π.) A permutation solution is always feasible for the job-shop problem with pairwise buffers due to the following reason: If we schedule the jobs one after another by starting the next one at the finishing time of the previous, no job needs buffer space and no blocking occurs. In the general case, when not all pairwise buffers have capacity 0, the constructed solution might not be a permutation solution, but nevertheless it is feasible since no buffer capacity is exceeded and the blocking restrictions are respected.

In the case of the flow-shop problem with intermediate buffers we showed how the longest path calculation in the solution graph can be done efficiently. Due to the special structure of the solution graph of a feasible solution, we could determine an order in which the operations are evaluated and which is only dependent on the given buffer capacities.

For the job-shop problem with pairwise buffers, the solution graph of a feasible solution has a more complex structure than for the flow-shop problem with intermediate buffers. On the one hand this is due to the fact that in general the jobs do not visit the machines in the same sequence in a job-shop environment. On the other hand the solution graph of a feasible solution for a job-shop problem with pairwise buffers may not be acyclic since it may contain blocking cycles of length 0. Thus, for this problem type an evaluation order in the solution graph of a feasible solution cannot be fixed.

In our implementation, we use a Label-Correcting algorithm (see e.g. Ahuja et al. [3]) to either detect a positive cycle or to find a longest path in the solution graph for given sequences of the jobs on the machines. Since each node of the solution graph has at most three outgoing arcs, the number of arcs is bounded by $O(r)$ where r is the total number of operations. Thus, the algorithm has running time $O(r^2)$.

In the following, we present some computational results which we obtained for the job-shop problem with pairwise buffers. The tests were performed on a SUN-Fire V 440 (4× 1281 MHz-CPU, 16 GB RAM) with operating system Solaris 10. Note, that this computer is approximately 5 times faster than the one we used for the computations in the case of the flow-shop problem with intermediate buffers.

As test instances we used ten benchmark instances for the classical job-shop problem by Lawrence [19] (la16 - la20) and by Applegate & Cook [4] (orb01 - orb05). All these instances are problems with 10 jobs and 10 machines where each job is processed on each machine exactly once. The processing times of the operations in all instances are uniformly distributed over the interval $[1, 99]$. Similar to the case of the flow-shop problem with intermediate buffers, we added different types of buffer configurations to each of the test instances. We first set the buffer capacity equal to $n - 1$ for all pairwise buffers. In this case the job-shop problem with pairwise buffers reduces to a classical job-shop problem ($b_{ij} = \infty$). On the other hand, we considered the situation where no buffer space is available between the machines ($b_{ij} = 0$). In this

case we have a blocking job-shop problem. Furthermore, we considered instances where all pairwise buffers have the same capacity of $b_{ij} = 1$ or $b_{ij} = 2$ units.

We solved each of the instances for the three parameter settings $\#iterations = 10,000$, $\#restarts = 1$, and $\#iterations = 1,000$, $\#restarts = 5$, as well as $\#iterations = 3,000$, $\#restarts = 3$. Remember, that if in some iteration of the search a new globally best solution is found, the counters for the number of iterations and the number of restarts during the search process is reset to 0. Therefore, in general, the number of total iterations is much larger than the product of $\#iterations$ and $\#restarts$. In fact, the number of total iterations is the product of $\#iterations$ and $\#restarts$ plus the number of iterations beginning with the first iteration and ending in the iteration in which the makespan was improved last. With an increasing number of restarts we chose $\#iterations$ in that way that the product of $\#iterations$ and $\#restarts$ decreases. By this combination of parameters we expected to get similar computational times for test runs on the same instances.

For the job-shop problem with pairwise buffers, no methods to compute lower bounds for the optimal makespan are available. But in the special cases of the classical job-shop problem and of the blocking job-shop problem, the optimal makespan is known from the literature. Mascis & Pacciarelli [24] solved these problems by a branch-and-bound algorithm. Therefore, to evaluate the quality of the proposed tabu search approach in these two special cases, we compare the best makespan found by the tabu search over all test settings for an instance with the optimal makespan of this instance. In the case of the classical job-shop problem, the relative deviation of the best found makespan from the optimal makespan amounts averagely 0.92% over all test instances where the relative deviation of the starting solution from the optimal makespan amounts averagely 48.39%. In 3 of the 10 instances, the optimal makespan was reached. In the case of the blocking job-shop problem, the relative deviation of the best found makespan from the optimal makespan is in the mean 32.92% and the relative deviation of the starting solution from the optimal makespan is in the mean 273.06%.

Mascis & Pacciarelli [24] proposed three greedy heuristics for different versions of the job-shop problem. For the classical job-shop problem, these heuristics achieved a relative deviation of 6.78% from the optimal makespan in the mean, whereas for the blocking job-shop problem the relative deviation was 72.53%. These results show that the quality of the starting solution of the tabu search procedure is very bad in both cases. However, the heuristics of Mascis & Pacciarelli [24] sometimes do not even find a feasible solution in the case of the blocking job-shop problem. Their computational results show that the feasibility and the quality of a solution are conflicting issues which can be only hardly combined in a heuristic algorithm for the blocking job-shop problem. Since for the tabu search procedure a feasible starting solution is needed, we slightly modified one of their greedy algorithms in order to produce always a feasible solution. This obviously has a strong negative effect on the quality of the starting solution. Furthermore, one has to take into consideration

that our heuristic is designed in that way that it produces feasible starting solutions for the job-shop problem with pairwise buffers of arbitrary buffer capacities.

Considering the performance of the tabu search procedure, the results seem to be very good in the case of the classical job-shop problem. The average deviation of the best found solution from the optimal makespan is less than 1% and the gap between initial makespan and optimal makespan was reduced by more than 95%. The computational times were averagely about 1 minute for a setting of $\#iterations = 1,000$ and $\#restarts = 5$. In the case of the blocking job-shop problem, the gap between initial makespan and optimal makespan was reduced by more than 85%, but the relative deviation of the best found makespan from the optimal makespan is with 32.92% still very large. Moreover, for the setting of $\#iterations = 1,000$ and $\#restarts = 5$, the computations took more than 2 hours in the mean. However, in this case, the heuristics of Mascis & Pacciarelli [24] do not yield solutions of good quality either. This indicates that it is much harder to find heuristic solutions of good quality for the blocking job-shop problem than for the classical job-shop problem.

In order to investigate whether the restart technique improves the efficiency of the tabu search like this was the case for the flow-shop problem with intermediate buffers, we solved each of the instances for the parameter settings $\#iterations = 10,000$, $\#restarts = 1$, and $\#iterations = 1,000$, $\#restarts = 5$, as well as $\#iterations = 3,000$, $\#restarts = 3$. Since in the cases $b_{ij} = 1$ and $b_{ij} = 2$ no lower bounds for the optimal makespan are available, we determined the relative improvement

$$imp = 100 \cdot \frac{C_{start} - C_{tabu}}{C_{start}}$$

of the makespan of the initial solution C_{start} by the best makespan C_{tabu} found applying the tabu search. In Tables 7.8, 7.9, and 7.10, the relative improvement value as well as the required computational time (in minutes : seconds) of each of the test instances are reported for the three different parameter settings. In the last row of each table, the mean relative improvement values and the mean computational times over all 10 test instances are shown.

Tables 7.8, 7.9, and 7.10 show that the average relative improvement values for a fixed type of buffer configuration do not differ considerably. But the mean computational times for the parameter settings $\#iterations = 3,000$, $\#restarts = 3$ and $\#iterations = 10,000$, $\#restarts = 1$, respectively, are about twice and three times, respectively, as long as they are in the case of $\#iterations = 1,000$, $\#restarts = 5$. This indicates that the restart technique improves the efficiency of the tabu search.

As in the case of the flow-shop problem with intermediate buffers, the relative improvement values get better the less the buffer capacities are. On the one hand, this effect seems to have the same reason as in the case of the flow-shop problem with intermediate buffers: For a capacity of $b_{ij} = \infty$ all neighbors result from a single shift whereas for $b_{ij} = 0$ each neighbor results from a shift together with a repair step. Obviously, a solution is changed by one repair step more than by a

	$b_{ij}=0$		$b_{ij}=1$		$b_{ij}=2$		$b_{ij}=\infty$	
Instance	*imp*	CPU	*imp*	CPU	*imp*	CPU	*imp*	CPU
la16	64.41	195:18	33.17	3:30	32.16	2:11	32.78	1:58
la17	64.70	112:49	28.53	2:15	30.66	2:02	30.48	1:29
la18	67.11	492:58	30.04	2:22	24.51	2:32	24.51	2:25
la19	56.35	91:27	33.65	4:26	29.12	1:23	29.12	1:15
la20	67.14	294:40	35.35	2:41	35.40	1:58	34.97	1:24
orb01	51.31	487:31	42.53	6:21	42.83	5:00	37.47	7:52
orb02	61.66	545:59	20.74	2:27	21.19	2:22	21.19	1:56
orb03	40.91	524:59	38.30	10:03	40.04	6:33	39.21	5:53
orb04	66.94	144:34	25.05	6:08	24.91	3:06	24.84	3:01
orb05	67.30	1132:00	41.22	10:09	37.40	3:21	37.74	2:19
mean	60.78	402:14	32.86	5:02	31.82	3:03	31.23	2:57

Table 7.8: Relative improvements *imp* for $\#iterations = 10,000$, $\#restarts = 1$ (CPU-time in minutes : seconds)

	$b_{ij}=0$		$b_{ij}=1$		$b_{ij}=2$		$b_{ij}=\infty$	
Instance	*imp*	CPU	*imp*	CPU	*imp*	CPU	*imp*	CPU
la16	64.41	88:41	31.78	1:08	32.16	0:43	32.78	0:38
la17	64.70	37:25	28.53	1:20	30.75	0:59	30.48	0:31
la18	69.18	178:27	29.88	0:38	24.51	1:15	24.51	1:08
la19	68.82	134:41	33.02	0:39	29.12	0:27	27.95	0:33
la20	66.79	55:33	35.71	1:08	34.97	0:32	35.75	0:37
orb01	50.65	191:28	42.53	4:51	42.94	2:47	36.42	1:33
orb02	66.46	184:07	20.92	1:09	21.10	0:53	21.10	0:52
orb03	40.91	158:46	36.69	3:50	37.15	1:28	37.21	2:26
orb04	66.94	57:56	25.05	1:48	24.84	2:11	24.98	1:04
orb05	67.30	343:09	40.57	1:32	37.67	1:32	37.74	0:43
mean	62.62	143:01	32.47	1:48	31.52	1:17	30.89	1:01

Table 7.9: Relative improvements *imp* for $\#iterations = 1,000$, $\#restarts = 5$ (CPU-time in minutes : seconds)

simple shift. Therefore, in a fixed number of iterations, on the average more changes are made for instances with small buffer sizes than for those with large buffer sizes. Since the calculations for one iteration with repair process take more time than the calculations for one iteration without repair process, the computational time for a fixed parameter setting will decrease with increasing buffer size. Furthermore, the very large initial makespan in the case of $b_{ij} = 0$ causes much larger improvement values for the blocking job-shop problem than for the problems where buffer capacity is available.

Finally, we investigate the influence of the buffer capacities on the best makespan

	$b_{ij}=0$		$b_{ij}=1$		$b_{ij}=2$		$b_{ij}=\infty$	
Instance	*imp*	CPU	*imp*	CPU	*imp*	CPU	*imp*	CPU
la16	64.41	117:18	31.78	1:52	33.75	2:12	32.78	1:14
la17	64.70	71:38	28.53	1:27	30.66	1:29	30.93	2:10
la18	68.12	278:01	30.04	1:15	24.51	1:26	24.51	1:17
la19	67.55	178:28	32.23	1:05	29.12	0:44	29.12	0:48
la20	67.14	209:19	35.07	1:50	35.75	1:37	34.97	1:00
orb01	51.31	315:40	42.53	5:07	43.04	4:01	37.35	4:08
orb02	61.66	335:19	20.74	1:30	21.19	1:42	21.19	1:31
orb03	41.39	221:26	38.53	9:45	38.68	4:36	39.92	6:09
orb04	66.94	89:17	24.69	3:20	24.62	1:37	25.05	2:31
orb05	67.30	653:36	40.57	2:44	37.40	2:25	37.74	1:24
mean	62.05	247:00	32.47	3:00	31.87	2:11	31.36	2:13

Table 7.10: Relative improvements *imp* for $\#iterations = 3,000$, $\#restarts = 3$ (CPU-time in minutes : seconds)

found by the tabu search. Therefore, we determined the relative change

$$\Delta = 100 \cdot \frac{C^k_{tabu} - C^\infty_{tabu}}{C^\infty_{tabu}}$$

(for $k = 0, 1, 2$) of the best found makespan relative to the case of a buffer capacity of $b_{ij} = \infty$ (here, C^k_{tabu} denotes the best found makespan over all test settings for the job-shop problem with a buffer capacity of $b_{ij} = k$, $k = 0, 1, 2, \infty$). In Table 7.11 the relative changes of all test instances are reported.

Instance	Buffer Capacity		
	$b_{ij}=0$	$b_{ij}=1$	$b_{ij}=2$
la16	59.18	-1.13	-1.44
la17	62.50	0.64	0.26
la18	53.58	0.47	0.00
la19	72.68	0.00	0.00
la20	55.10	0.00	0.00
orb01	38.58	1.49	1.12
orb02	50.96	0.34	0.00
orb03	34.15	0.98	-0.20
orb04	48.64	0.00	0.19
orb05	56.38	0.00	0.11
mean	53.18	0.28	0.004

Table 7.11: Influence of the buffer capacity on the best found makespan (relative to the values for $b_{ij} = \infty$)

If the buffer capacity of all pairwise buffers is restricted to $b_{ij} = 2$, the makespan increases averagely by only 0.004 %. For four of the ten instances the same makespan

was reached as without the restriction on the buffer capacities and for two instances the makespan was even reduced (see the negative Δ-values in Table 7.11). For a buffer capacity of $b_{ij} = 1$, the makespan increases slightly by 0.28 % in the mean. If the buffer capacity is reduced to $b_{ij} = 0$, the makespan increases considerably by 53,18 % in the mean.

The results of table 7.11 can be explained as follows: For the test instances, a buffer capacity of two or one seems to be sufficient to store almost all jobs which need a storage capacity during their non-processing periods. The very small increase of the makespan indicates that blocking situations appear only very seldom. Note, that in all test instances 10 jobs and 10 machines are given. It may be expected that the makespan would increase stronger if the ratio of jobs to machines were larger. In this case, more jobs would compete for buffer space at the same time, and thus, blocking situations would increase. For the blocking job-shop problem, the optimal solution value is averagely 15.24 % larger for the ten test instances than the best found solution in the case of the classical job-shop problem. The clearly larger increase of the makespan by 53.18 % in our tests shows again that the local search procedure cannot succeed in finding solutions of good quality in the case of $b_{ij} = 0$.

7.4 The job-shop problem with blocking operations

In this section, we propose a tabu search approach for the job-shop problem with blocking operations which applies the neighborhood structures described in Subsection 6.4. The computational results of this subsection were obtained in the context of the diploma thesis of Hauser [16].

As initial solution of the tabu search procedure, we generate a permutation solution with the help of the algorithm of Nawaz et al. [25] (referred as NEH-heuristic) which we also use for the flow-shop problem with intermediate buffers. Remember that according to this heuristic the jobs are considered in the order of non-increasing total processing times. In each step of the NEH-heuristic, the next job is inserted into the current partial job sequence. Given the partial sequence $1, 2, \ldots, i-1$, the sequences $i, 1, 2, \ldots, i-1$, and $1, i, 2, \ldots, i-1$, and $1, 2, i, 3, \ldots, i-1$, and so forth until $1, \ldots, i-1, i$ are considered. For each of these sequences an earliest start schedule can be constructed iteratively: An operation of the next job is scheduled on the corresponding machine after the preceding job as soon as the machine is available. Here, we take into consideration whether the preceding operation on a machine is blocking or not. We continue the procedure with the partial sequence which provides the minimal makespan.

7.4.1 Efficient calculation of longest paths

In the case of the flow-shop problem with intermediate buffers we showed how the longest path calculation in the solution graph can be done efficiently. Though the

solution graph has a more complex structure in the case of the job-shop problem with blocking operations, a schedule with minimal makespan respecting given sequences of the jobs on the machines can be calculated efficiently by adapting the Algorithm Output Buffers of Subsection 5.3 to the case of blocking operations.

In the job-shop problem with blocking operations we differentiate between blocking and ideal operations. For ideal operations, there is always buffer space available to store them after their processing whereas for blocking operations no buffer space exists. The buffer space for an ideal operation i can be seen as output buffer of capacity one for machine $\mu(i)$. Similarly, if a blocking operation i is processed on $\mu(i)$, the output buffer for $\mu(i)$ has capacity 0. Therefore, we have no output buffers of permanent capacities in this case but dependent on the type of operation finishing on the machine, a buffer place is available or not. Since the Algorithm Output Buffer proceeds in time, this operation dependent buffer capacity of 1 or 0 units can be taken into account when scheduling the operations. Furthermore, as not several operations share one common buffer in the job-shop problem with blocking operations, the Procedure Update as well as the subroutines Move in Buffer, Move out of Buffer and Swap can be simplified. If the given sequences $\pi^1, \dots \pi^m$ are feasible, a schedule can be calculated in $O(r \max\{n, m\})$ time, where r denotes the total number of operations. Infeasibility is detected with the same computational effort.

7.4.2 Computational results

In this subsection we present some computational results which we obtained for the job-shop problem with blocking operations. The tabu search procedures have been implemented in C on a SUN workstation and a 1466 MHz PC.

As test instances we used the five 10×10 instances for the classical job-shop problem by Lawrence [19] (la16 - la20) which we also considered in the case of the job-shop problem with pairwise buffers. By specifying how much percentage of all operations in these instances are ideal we constructed benchmark instances for the job-shop problem with blocking operations.

We chose either all operations belonging to the same job as ideal or all operations processed on the same machine. In the first case, we declare all operations of $1, 2, 3, 4$, or 5 specific jobs as ideal operations and all operations of the remaining jobs as blocking operations. The corresponding problem instances are marked by $J10\%$, $J20\%, \dots,$ $J50\%$. Similarly, instances in which all operations processed on 1 - 5 specific machines are ideal are marked by $M10\%$ - $M50\%$. The jobs composing of ideal operations and the machines processing ideal operations are chosen arbitrarily. Instances for the classical job-shop problem in which all operations are ideal are marked by 100% while instances for the blocking job-shop problem are marked by 0%. The instances $J10\%$, $J20\%, \dots,$ $J50\%$ are also instances for the job-shop problem with job-dependent buffers.

First computational experiments for the job-shop problem with blocking operations showed that a restart technique as in the case of the flow-shop problem with intermediate buffers improves the efficiency of the tabu search. Therefore, the following results were achieved with a tabu search parameter setting of $\#iterations = 3000$ and $\#restarts = 3$.

In Table 7.12 the relative error is documented for the test instances with different storage capacities. The figures in the 0% column represent the relative deviation from the optimal makespan for the blocking job-shop. The optimal makespan has been provided by Mascis & Pacciarelli [24] who solved these problems by a branch-and-bound algorithm. The other columns show the relative deviation from the optimal solution for the classical job-shop problem. The table shows that the solution quality improves with the percentage of available buffers. The difficulty to find solutions if the number of available buffers is small can be also documented by the average computational times for the blocking job-shop instances which are 15 hours for the workstation and 1.5 hours for the PC. Similar results have been achieved for the $M10\% - M50\%$ instances. However for $M10$ % instances the solution quality is in the average 7% better than the corresponding solution quality for the $J10\%$ instances.

instance	0%	J10%	J20%	J30%	J40%	J50%	100%
la16	45.4	41.0	19.1	17.8	13.0	12.5	2.1
la17	81.8	54.9	33.2	20.9	16.0	7.4	0.6
la18	31.1	45.9	42.6	26.7	31.1	10.6	0.6
la19	38.9	61.8	47.0	29.2	26.9	14.2	1.4
la20	36.0	55.8	49.3	27.5	27.1	27.7	0.8
average	46.7	51.8	38.2	24.4	22.8	14.5	1.1

Table 7.12: Relative deviation from the optimal solution value

In Table 7.13, the ratio of all feasible computed solutions to all computed solutions is shown for the case of the blocking job-shop problem (0%) and the classical job-shop problem (100%). While for the blocking job-shop problem in the mean only about 5% of all computed solutions during the tabu search are feasible, for the classical job-shop problem in the mean about 70% of all computed solutions are feasible. Thus, about 30 % of all computed solutions must be infeasible.

	Instance				
	la16	la17	la18	la19	la20
blocking job-shop	4.2	4.8	7.3	5.9	4.7
ideal job-shop	66.9	71.2	70.5	70.3	72.4

Table 7.13: Ratio of all feasible computed solutions to all computed solutions

Further computational results can be found in the diploma thesis of Hauser [16].

8 Concluding remarks

We studied job-shop scheduling problems with limited buffer capacities which generalize the classical job-shop problem. Besides a general buffer model also special buffer configurations, such as pairwise buffers, job-dependent buffers, output buffers and input buffers, were investigated. Furthermore, we considered the job-shop problem with blocking operations which constitutes a basic model for the job-shop problem with general buffers.

First, we tried to find a compact representation of solutions for the job-shop problem with buffers since this is the key issue to develop fast heuristics. We showed that the job-shop problem with general buffers can be reduced to the blocking job-shop problem by dividing each buffer into several buffer slots and assigning the operations to the buffer slots. Since this representation had several disadvantages we proposed a second representation by introducing an input and output sequence for each buffer. The input and output sequence for each buffer together with the sequences of the jobs on the machines can be used to represent a solution of the job-shop problem with general buffers. In order to compute a corresponding schedule, we adapted and extended existing graph models such as the disjunctive graph model and the alternative graph model. For special buffer configurations, such as pairwise buffers, job-dependent buffers, output buffers and input buffers, we showed 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. For the case of general buffers, we showed that machine sequences are not sufficient to represent solutions.

In the second part of this thesis, local search methods based on the given solution representations for the flow-shop problem with intermediate buffers, the job-shop problem with blocking operations and the job-shop problem with pairwise buffers, respectively, were developed. In connection with the local search approaches, we generalized the block approach theorem for the classical job-shop problem to the job-shop problem with blocking operations and the job-shop problem with pairwise buffers. Based on the block approaches we defined neighborhood structures for these problems. Finally, we tested the algorithms on different test data and presented computational results.

There are still many different research topics in the area of job-shop scheduling problems with buffers. Besides the presented local search procedures, the solution representation may also form the base for branch and bound approaches for the general and specific buffer configurations. Furthermore, the presented local search approaches can be adapted in order to develop fast heuristics for the case of job-dependent and ouput buffers. The computational results have shown that problems with tight buffer capacities are much harder than problems with larger buffer capacities. Therefore, additional techniques such as constraint propagation could be developped for problems with tight buffer capacities.

Another interesting aspect concerning the buffer model is to allow variable buffer space requirement of the jobs. Two types of problems could be distinguished: On the one hand, problems where jobs require several not connected buffer places, and on the other hand, problems where jobs require several connected buffer places. In the first case, our modelling could probably partly be extended, whereas for the latter case a different model has to be developped.

References

[1] Aarts, E.H.L., van Laarhoven, P.J.M., Lenstra, J.K., Ulder, N.L.J. (1994) A computational study of local search algorithms for job shop scheduling, ORSA Journal on Computing 6, 118-125.

[2] Adams, J., Balas, E., Zawack, D. (1988) The shifting bottleneck procedure for job shop scheduling, Management Science 34 (3), 391-401.

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

[4] Applegate, D., Cook, W. (1991) A computational study of the job-shop problem, ORSA Journal on Computing 3 (2), 149-156.

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

[6] Brucker, P., Heitmann, S., Hurink, J. (2003) Flow-shop problems with intermediate buffers, OR Spectrum 25, 549-574.

[7] Brucker, P., Heitmann, S., Hurink, J., Nieberg T. (2006) Job-shop scheduling with limited capacity buffers, OR Spectrum 28, 151-176.

[8] Brucker, P., Jurisch, B., Sievers, B. (1994) A fast branch & bound algorithm for the job-shop scheduling problem, Discrete Applied Mathematics 49, 107-127.

[9] Brucker, P., Knust, S. (2006) Complex Scheduling, Springer, Berlin.

[10] Fischer, M.J., Thompson, G.L. (1963) Probablistic learning combinations of local job-shop scheduling rules, in: Industrial Scheduling, Muth, J.F., Thompson, G.L., N.J. Prentice Hall, Englewood Cliffs.

[11] Gilmore, P.C., Gomory, R.E., (1964) Sequencing a one state-variable machine: A solvable case of the traveling salesman problem, Operations Research 12, 655-679.

[12] Glover, F. (1989,1990) Tabu Search, Part I and II, ORSA Journ. Comuputing 1, 190-206, ORSA Journ. Computing 2, 4-32.

[13] Grabowski, J., Nowicki, E., Smutnicki, C. (1988) Block algorithm for scheduling of operations in job shop systems (Polish), Przeglad Statystyczny 35, 67-80.

[14] Grabowski, J., Nowicki, E., Zdrzalka, S. (1986) A block approach for single machine scheduling with release dates and due dates, European Journal of Operational Research 26, 278-285.

[15] Hall, N.G., Sriskandarajah, C. (1996) A survey of machine scheduling problems with blocking and no-wait in process, Operations Research 44, 510-525.

[16] Hauser, T. (2005) Heuristische Verfahren zur Lösung von Job-Shop Problemen mit begrenzten Zwischenspeichern, Master Thesis, University of Osnabrück.

[17] Jain, A.S., Meeran, S. (1999) Deterministic job-shop scheduling: Past, present and future, European Journal of Operational Research 113, 390-434.

[18] Johnson, S.M. (1954) Optimal two-and-three-stage production schedules with set-up times included, Naval Research Logistic Quaterly 1, 61-68.

[19] Lawrence, S. (1984) Supplement to Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques, GSIA, Carnegie Mellon University Pittsburgh, PA.

[20] Leisten, R. (1985) Die Einbeziehung beschränkter Zwischenlager in die Auftragsreihenfolgeplanung bei Reihenfertigung, VDI-Verlag, Düsseldorf 1985.

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

[22] Lenstra, J.K., Rinnooy Kan, A.H.G. (1979) Computational complexity of discrete optimization problems, Annals of Discrete Mathematics 4, 121-140.

[23] Martinez, S., Dauzère-Pérès, S., Guéret, C., Mati, Y., Sauer, N. (2006) Complexity of flowshop scheduling problems with a new blocking constraint, European Journal of Operational Research 169, 855-864.

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

[25] Nawaz, M., Enscore, E. E., Ham, I. (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega 11, 91-95.

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

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

[28] Nowicki, E., Smutnicki, C. (1996) A fast taboo search algorithm for the job shop problem, Management Sciences 42, 797-813.

[29] Nowicki, E., Smutnicki, C. (1996) A fast taboo search algorithm for the permutation flow shop problem, European Journal of Operational Research 91, 160-175.

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

[31] Reddi, S.S., Ramamoorthy, C.V. (1972) On the flowshop sequencing problem with no-wait in process, Operational Research Quarterly 23, 323-330.

[32] Ronconi, D.P. (2004) A note on constructive heuristics for the flowshop problem with blocking, International Journal of Production Economics 87, 39-48.

[33] Ronconi, D.P. (2005) A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking, Annals of Operations Research 138, 53-65.

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

[35] Taillard, E. (1993) Benchmarks for basic scheduling problems, European Journal of Operational Research 64, 278-285. Web Page `http://www.eivd.ch/ina/collaborateurs/etd/problemes.dir/problemes.html`

[36] Thornton, H.W., Hunsucker, J.L. (2004) A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage, European Journal of Operational Research 152, 96-114.

[37] Vaessens, R.J.M. (1996) Operations Research Library of Problems, Management School, Imperial College London. Web Page `http://mscmga.ms.ic.ac.uk/info.html`, File `flowshop2.txt`

Danksagung

Nach einigen Jahren Arbeit mit mehreren kurzen Pausen liegt nun diese Schrift vor mir. Viele Leute haben dazu beigetragen und mir mit Rat und Hilfe zur Seite gestanden. Ich möchte mich ganz herzlich bei ihnen allen bedanken!

Als erstes möchte ich mich bei Herrn Prof. Dr. P. Brucker bedanken. Er hat mir die Chance gegeben, bei ihm als Doktorandin zu arbeiten, und er hat diese Arbeit betreut. Es hat mich mehrfach beeindruckt, wie durch seine Übersicht und Ideen sowie durch seine kritischen Ratschläge und Verbesserungsvorschläge die Zusammenhänge klarer und die Arbeit kompakter und besser wurde. Dankbar bin ich auch dafür, daß er es mir ermöglicht hat, an vielen nationalen und internationalen Workshops und Tagungen teilzunehmen.

Besonders möchte ich mich bei Johann Hurink bedanken. Johann hat sich immer wieder Zeit genommen, Teile der Arbeit zu lesen und wertvolle Tipps und Anregungen zu geben. Es hat Spaß gemacht, bei unseren Treffen gemeinsam über noch offene Probleme zu diskutieren und Lösungen zu erarbeiten.

Für die Hilfe bei den umfangreichen Implementierungsarbeiten und Tests möchte ich mich bei Tim Nieberg und Timo Hauser bedanken.

Weiterhin danke ich Herrn Prof. Dr. R. Leisten für seine wertvollen Anregungen und Hinweise.

Mit Sigrid, Christian und Thomas habe ich während der letzten Jahre nicht nur ein Büro sondern auch viele anstrengende und auch erfolgreiche Stunden geteilt. Für diese Zeit danke ich euch. Außerdem möchte ich mich auch bei den Mitgliedern des Instituts für Mathematik für die gute Zusammenarbeit bedanken.

Neben dem akademischen Leben gibt es auch ein privates Leben. Unsere Familie ist in den letzten sieben Jahren um drei wunderbare Kinder gewachsen. Ohne die Hilfe meiner Oma, meiner Schwiegereltern, meines Bruders und meiner Eltern, die immer wieder liebevoll unsere Kinder betreuen, hätte ich mich nicht so unbeschwert auf meine Arbeit konzentrieren können. Ganz besonders danke ich meinen Eltern und Markus, die mich immer unterstützen und immer da sind, wenn ich sie brauche.

Die letzten sieben Jahre wären ohne die uneingeschränkte Unterstützung meines Mannes nicht möglich gewesen. Er hat die Kinder versorgt und sich mit ihnen beschäftigt, wenn ich nicht da war. Und wenn ich da war, hat er mir immer wieder Kraft und Mut gegeben, weiterzumachen. Vielen Dank Fiddi, Marius, Alina und Laura!

Osnabrück, März 2007
Silvia Heitmann