Memorandum No. 1625

Flow-shop problems with intermediate buffers

P. Brucker,[1] S. Heitmann[1]
and J.L. Hurink

May, 2002

ISSN 0169-2690

[1]Department of Mathematics/Informatics, University of Osnabrück

Flow-Shop Problems with Intermediate Buffers

Peter Brucker
Silvia Heitmann
Department of Mathematics/Informatics
University of Osnabrück

Johann Hurink
Faculty of Mathematical Sciences
University of Twente

May 2002

Abstract

In this paper the following extension of the classical flow-shop problem is considered: Between each two successive machines a buffer of limited capacity is given in which jobs can be stored. 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 the buffer between these machines. If the buffer is completely occupied the job may wait on its current machine but blocks this machine for other jobs. The objective is to determine a feasible schedule minimizing the makespan.

To model such a problem setting, the classical disjunctive graph model for shop problems is extended. A tabu search procedure is described where neighborhoods based on an extension of the classical block approach theorem are used. Computational results for extended flow-shop benchmark instances are presented.

Keywords: Flow-Shop Problem, Buffer, Tabu Search.

AMS classification: 90B35

1 Introduction

Classical shop scheduling problems considered in the literature mainly deal with the task of scheduling the jobs on the machines taking into account restrictions given for the jobs and/or machines. However, in practice often further aspects (e.g. transportation of jobs between machines or storing them between processing on different machines) have to be taken into account to model the situation in a shop.

In this paper we focus on the storing aspect in a flow-shop environment. Since in a flow-shop all jobs visit the machines in the same order, we have to consider only storage facilities between consecutive machines in this order. We assume that between each pair of consecutive machines a buffer of limited capacity is available. After processing of a job on a machine, this job either directly has to be processed on the next machine or it has to be stored in the buffer between the two machines. If neither of these two alternatives are possible (the buffer is full and the next machine is occupied) the job has to wait on its current machine and blocks the machine. This blocking will be finished if the job, which momentarily is processed on the next machine, leaves that machine. At that time a job from the buffer or the job which blocks its machine may start processing on the next machine. We call the resulting problem a **flow-shop problem with intermediate buffers**.

In the literature (contrary to classical flow-shop problems) only few results on flow-shop problems with intermediate buffers can be found. All known results concern the makespan objective. Papadimitriou and Kanellakis [1980] showed that the flow-shop problem with intermediate buffers is strongly $\mathcal{NP}$-hard even for two machines only. Leisten [1990] presents some priority based heuristics, both, for the permutation and for the general flow-shop problem with intermediate buffers. Recently, Smutnicki [1998] and Nowicki [1999] developed tabu search approaches for the permutation flow-shop problem: Smutnicki considered the two machine case with buffers, whereas Nowicki generalizes the approach to an arbitrary number of machines.

In this study we generalize the approach of Nowicki [1999] to the case where different job-sequences on the machines are allowed. The base of the presented tabu search approach is a polynomial procedure which calculates for a given sequence of job permutations a feasible, left-shifted schedule. Using this procedure we consider the set of all m permutations specifying the sequences in which the jobs are scheduled on the machines as search space.

The paper is organized as follows. After giving in the next section a formal definition of the flow-shop problem with intermediate buffers, in Section 3 we present the polynomial procedure for calculating the best schedule respecting given job orders on the machines. Afterwards, in Section 4 the main issues of the developed tabu search approach are discussed and in Section 5 some technical remarks and computational results are presented. The last section contains some concluding remarks.

2 Problem Formulation

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

Given are m machines $M_1, \ldots, M_m$ and $m-1$ buffers B_i between machines M_i and M_{i+1} with a capacity of b_i units ($i = 1, \ldots, m-1$). On these machines n jobs $j = 1, \ldots, n$ have to be processed. 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 without preemption for $p_{ij} \geq 0$ time units. Between the operations of each job there are precedence relations in form of a chain $O_{1j} \to O_{2j} \to \ldots \to O_{mj}$.

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} ($i = 1, \ldots, m; j = 1, \ldots, n$) such that

1. the precedence relations within the jobs are respected ($C_{ij} \leq S_{i+1,j}$),
2. each machine processes only one job at any time ($[S_{ij}, C_{ij}[\cap [S_{ik}, C_{ik}[= \emptyset$ for $j \neq k$),
3. in each buffer B_i at most b_i jobs may be stored at the same time, and
4. the *blocking-restrictions* are respected. (The last two restrictions are explained in more detail after the next paragraph).

In this paper we restrict ourselves to the objective of minimizing the makespan, i.e. we want to determine a feasible schedule minimizing the makespan $C_{\max} = \max\limits_{j=1}^{n} C_j$, where C_j is the finishing time C_{mj} of the last operation O_{mj} of job j.

It remains to describe the buffer and blocking restrictions for feasible schedules in more detail: If a job j finishes processing on machine M_i before it can start on M_{i+1} (i.e. $C_{ij} < S_{i+1,j}$) then during time period $[C_{ij}, S_{i+1,j}]$ this job either has to wait on machine M_i or it has to be inserted into the intermediate buffer B_i. If buffer B_i is filled completely by other jobs and machine M_{i+1} is busy, job j has to wait on machine M_i. While waiting on M_i, job j blocks this machine, so that no other job can be processed on M_i during this time period. Since all the jobs which are in buffer B_i have to be processed next on machine M_{i+1}, jobs may leave the buffer B_i only when machine M_{i+1} becomes available. A job leaving the buffer may be replaced by job j. Alternatively, all jobs may stay in the buffer and job j may move to machine M_{i+1}. Thus, the blocking of machine M_i by job j ends when the next job starts its processing on machine M_{i+1}.

In the next section we will show how these **buffer and blocking restrictions** may be handled if the job sequences on the machines are given.

3 Construction of a feasible schedule for given job sequences on the machines

In this section we characterize all possible job sequences on the machines that are compatible with the given buffer capacities. After that, we propose a polynomial procedure for solving the following problem: Given compatible permutations of the jobs on the machines, find a left-shifted schedule respecting the blocking-restriction. This problem is solved by longest path calculations in a suitable graph which we call solution graph.

3.1 Characterization of the set 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. In the following, we characterize the set of all m-tuples of job permutations leading to a feasible schedule in the case of limited buffers.

Let $(\pi^1, \ldots, \pi^m)$ be an m-tupel of permutations of n jobs, where the permutation $\pi^i = (\pi^i(1), \pi^i(2), \ldots, \pi^i(n))$ specifies the order of the jobs on machine M_i. Subsequently, we call such an m-tupel **solution**.

We say that **the permutations $\boldsymbol{\pi^i}$ and $\boldsymbol{\pi^{i+1}}$ are compatible with the buffer capacity $\boldsymbol{b_i}$**, if a feasible schedule with job orders π^i on M_i and π^{i+1} on M_{i+1} exists, which respects the buffer capacity b_i. If π^i and π^{i+1} are compatible with b_i for each $i = 1, \ldots, m-1$ then we call $(\pi^1, \ldots, \pi^m)$ a **feasible solution**.

For the further considerations we assume w.l.o.g. that on machine M_i the jobs are scheduled in the order $1, \ldots, n$, i.e. $\pi^i = (1, 2, \ldots, n)$ and that $b_i \leq n$ for $i = 1, \ldots, m-1$.

Next, we show that a job which is processed in the k-th position on M_{i+1} has to be processed at or before position $b_i + k$ on M_i.

Lemma 1 : The permutations $\pi^i = (1, 2, \ldots, n)$ and $\pi^{i+1} = (j_1, j_2, \ldots, j_n)$ are compatible with the buffer capacity b_i, iff

$$j_k \in \{1, \ldots, b_i + k\} \quad \text{for all} \quad k \in \{1, \ldots, n\}.$$

Proof: First, assume that the permutation π^{i+1} is not of the above form, i.e. there exists a k with $j_k > b_i + k$. Thus, on machine M_i at least the jobs $1, 2, \ldots, b_i + k$ have been processed before job j_k is scheduled. On the other hand, on machine M_{i+1} only the jobs $j_1, j_2, \ldots, j_{k-1}$ have been finished before job j_k. Thus, there are at least $b_i + k - (k - 1) = b_i + 1$ jobs waiting in buffer B_i at the same time. Consequently, the buffer capacity of B_i will be exceeded and π^i and π^{i+1} are not compatible with b_i.

Now, consider an arbitrary permutation π^{i+1} which is of the above form. We show that the buffer capacity will not be exceeded at any time point.
Let $k < n - b_i$. Since $j_k \leq b_i + k$ holds, at most $b_i + k - 1$ jobs have been scheduled before job j_k on machine M_i. Therefore, at most $b_i + k - 1 - (k - 1) = b_i$ jobs are waiting in buffer B_i when job j_k starts processing on M_{i+1}. Consequently, the buffer capacity is not violated for all time points when jobs $j_1, \ldots, j_{n-b_i-1}$ start processing on M_{i+1}.
Clearly, in the case $k \geq n - b_i$ the buffer capacity of b_i units is sufficient for the last $b_i + 1$ jobs $j_{n-b_i}, \ldots, j_n$. □

Based on the result of Lemma 1, it is possible to determine the number of feasible solutions for a flow-shop problem with buffers.

Theorem 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 $0 \leq b_i \leq n$ holds for $i = 1, \ldots, m - 1$, the number of different feasible solutions N_{buf} is given by

$$N_{buf} = n! \prod_{i=1}^{m-1} b_i\,! \; (b_i + 1)^{n-b_i}.$$

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

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

and

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

Thus, we have $b_1 + j - (j-1) = b_1 + 1$ different possibilities for each job $\pi^2(j)$ with $j = 1, \ldots, n - b_1$ and $n - j + 1$ different possibilities for each job $\pi^2(j)$ with $j = n - b_1 + 1, \ldots, n$. 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 1 we immediately get

Corollary 1 :

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

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

3.2 The solution graph model

In the following we consider a situation where the orders of the jobs on the machines are fixed. We present a polynomial procedure, which calculates a feasible left-shifted schedule respecting the given orders. For classical flow-shop problems the calculation of such a schedule is straightforward, since each operation may start directly after its job and machine predecessor operations are finished. For the problem with intermediate buffers the situation becomes more complicated since also the blocking restrictions have to be taken into account. The presented solution approach extends approaches of Smutnicki [1998] and Nowicki [1999], who consider a permutation flow-shop problem with two machines and one intermediate buffer and a permutation flow-shop problem with an arbitrary number of machines and intermediate buffers. The approaches of Smutnicki and Nowicki are based on a construction of a schedule for a given sequence of jobs using longest paths calculations in a corresponding directed graph.

Let $(\pi^1, \ldots, \pi^m)$ be the given feasible solution. Providing such a feasible solution, one can always find a feasible schedule respecting the given job orders. We describe how to calculate a schedule which processes the jobs according to the sequences $\pi^1, \ldots, \pi^m$ on machines $M_1, \ldots, M_m$, which meets the blocking-restrictions, and in which the jobs are finished as early as possible (left-shifted schedule). As the finishing times C_j of all jobs j are minimal, we get a feasible schedule with minimal makespan for the given job sequences.

Let us consider an operation O_{ij}, which has to be processed at the k-th position on machine M_i (i.e. $\pi^i(k) = j$). Operation O_{ij} has to start after the previous operation of its job, $O_{i-1,j}$, and after its machine predecessor $O_{i,\pi^i(k-1)}$ have been finished. This is the case iff the condition

$$S_{ij} \geq \max\{C_{i-1,j}, C_{i,\pi^i(k-1)}\} \tag{3.1}$$

is respected.

Furthermore, we have to make sure that machine M_i is not blocked by the machine predecessor $O_{i,\pi^i(k-1)}$. As already mentioned in Section 2, operation $O_{i,\pi^i(k-1)}$ blocks machine M_i if b_i jobs are in the buffer B_i and machine M_{i+1} is still busy. This blocking ends at the time where a new job starts on M_{i+1}. This new job which starts is the job at position $k-1-b_i$ on M_{i+1}. Therefore, if $k-1-b_i > 0$, we also have to guarantee that operation O_{ij} does not start before the job at position $k-1-b_i$ on machine M_{i+1} starts, i.e.

$$S_{i,\pi^i(k)} \geq S_{i+1,\pi^{i+1}(k-1-b_i)} \quad \text{if} \quad k-1-b_i > 0 \tag{3.2}$$

Obviously, conditions (3.1) and (3.2) on the starting times guarantee that all restrictions for scheduling the jobs are satisfied (see also Leisten [1990]).

To construct a schedule, where each operation starts as early as possible and respects the above conditions, we make use of a solution graph (see Smutnicki [1998] and Nowicki[1999]). For the solution $(\pi^1, \ldots, \pi^m)$, the solution graph $G(\pi^1, \ldots, \pi^m) = (V, A_1 \cup A_2 \cup A_3)$ is defined as follows: The set of vertices V represents the set of all operations of all jobs. In addition, there is a source node $\circ \in V$ and a sink node $* \in V$ indicating the beginning and the end of the schedule (i.e. $V = \{O_{ij} \,|\, i = 1, \ldots, m; j = 1, \ldots, n\} \cup \{\circ, *\}$). The vertices representing operations are weighted with the processing times of the corresponding operation, whereas the two dummy vertices $\circ$ and $*$ get weight 0. The arc set of the graph represents the above mentioned timing restrictions between the operations, i.e. an arc $r \to s$ between two operations r and s induces that operation s cannot start before operation r is finished (finish-start relations) or that operation s cannot start before operation r has been started (start-start relations). The graph contains three types of arcs:

- The **job arcs** $(r, s) \in A_1$ reflect the precedence relations $O_{1j} \to O_{2j} \to \ldots \to O_{mj}$ between the operations of each job $j = 1, \ldots, n$. They represent finish-start relations.

- The **machine arcs** $(r, s) \in A_2$ are of the form $\circ \to O_{i,\pi^i(1)} \to O_{i,\pi^i(2)} \to \ldots \to O_{i,\pi^i(n)} \to *$ for $i = 1, \ldots, m$. They ensure that jobs are processed in the order π^i on machine M_i. They represent finish-start relations.

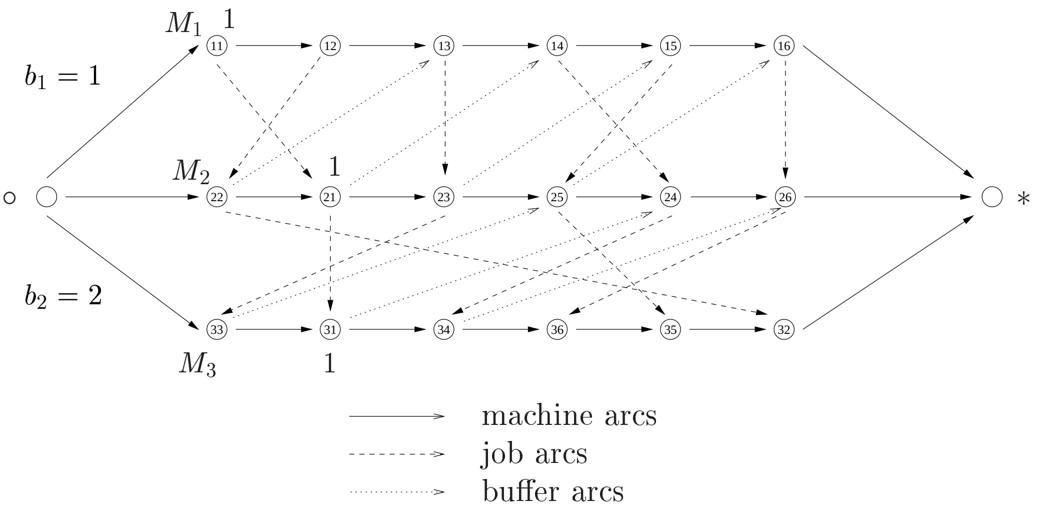

Figure 1: An example of the graph $G(\pi^1, \dots, \pi^m)$

- The **buffer arcs** $(r, s) \in A_3$ guarantee that the blocking-restrictions are respected. They are defined by

$$O_{i+1,\pi^{i+1}(j)} \quad \rightarrow \quad O_{i,\pi^i(j+1+b_i)} \quad \begin{array}{l}(i = 1, \dots, m-1;\\ j = 1, \dots, n-1-b_i)\end{array}$$

  The corresponding relations are start-start relations.

Figure 1 shows an example of the graph $G(\pi^1, \dots, \pi^m)$ for a problem with three machines, six jobs and buffer capacities $b_1 = 1$ and $b_2 = 2$. The numbers in the circles denote the indices of the corresponding operations. In each row the operations that have to be processed on the same machine and the corresponding machine arcs are represented. Additionally, all buffer arcs and all job arcs are shown.

In the following Theorem, we show that the solution graph $G(\pi^1, \dots, \pi^m)$ is acyclic if and only if the solution $(\pi^1, \dots, \pi^m)$ is feasible, i.e. if it is compatible with the buffer capacities. Here, $G(\pi^i, \pi^{i+1})$ is the subgraph of $G(\pi^1, \dots, \pi^m)$ which contains all vertices representing operations that have to be processed on machines M_i and M_{i+1} and all arcs connecting these operations.

Theorem 2 : The following statements are equivalent:

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

(b) For each $i = 1, \dots, m-1$ the graph $G(\pi^i, \pi^{i+1})$ has no cycle.

(c) The graph $G(\pi^1, \dots, \pi^m)$ has no cycle.

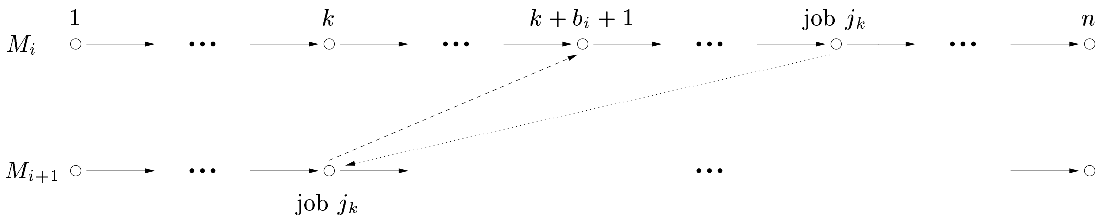

Figure 2: Situation, in which π^i and π^{i+1} are not compatible with b_i

Proof: The graph $G(\pi^1, \dots, \pi^m)$ contains for each machine i a chain of machine arcs. All other arcs (buffer and job arcs) occur between successive machines. This special structure directly induces that (b) and (c) are equivalent.

Next, we show that (a) and (b) are equivalent. First assume that (b) does not hold. It is straightforward to see that in this case $G(\pi^i, \pi^{i+1})$ must contain a cycle with precisely one job arc and one buffer arc. Since the buffer arc connects operations from machine $i+1$ to machine i which differ by $b_i + 1$ positions, we only get a cycle if the job arc connects operations which differ at least the same number of positions in the opposite direction. This contradicts Lemma 1.

To show that statement (a) follows from (b), we assume that the permutations π^i and π^{i+1} are not compatible with b_i. Thus, according to Lemma 1 there exists a job $k < n - b_i$ such that $j_k > b_i + k$, where $\pi^i = (1, \dots, n)$ and $\pi^{i+1} = (j_1, \dots, j_n)$. The job arc corresponding to job j_k together with the buffer arc from the operation in position k on M_{i+1} to the operation in position $k + b_i + 1$ on M_i and certain machine arcs on M_i form a cycle in $G(\pi^i, \pi^{i+1})$. Such a situation is shown in Figure 2. □

To get a unique meaning of the arcs, we may interpret a buffer arc between two operations $O_{i+1,j}$ and O_{ik} (which is a start-start relation) as a finish-start relation with length $-p_{i+1,j}$. By using these length for the buffer arcs and by defining the length of the remaining arcs as 0, a feasible, left-shifted schedule respecting the job orders $\pi^1, \dots, \pi^m$ can be achieved by longest path calculations in the solution graph $G(\pi^1, \dots, \pi^m)$: We define the length $L(P)$ of a path $P = (i_1, \dots, i_k)$ with $i_j \in V$ by the sum of the lengths of the arcs (i_{j-1}, i_j) ($j = 2, \dots, k$) plus the sum of the processing times of the operations corresponding to the vertices $i_1, \dots, i_{k-1}$ on the path (note that i_k is excluded). Let $S(\pi^1, \dots, \pi^m)$ be the schedule in which the starting time of an operation i is equal to the length of the longest path from the source $\circ$ to the vertex representing i in $G(\pi^1, \dots, \pi^m)$. Then, $S(\pi^1, \dots, \pi^m)$ is a feasible, left-shifted schedule with minimal makespan respecting the given job orders $\pi^1, \dots, \pi^m$. The makespan $C_{\max}(\pi^1, \dots, \pi^m)$ of $S(\pi^1, \dots, \pi^m)$ is equal to the length of a longest $\circ - *$-path in $G(\pi^1, \dots, \pi^m)$. Such paths are called **critical paths**.

Because the acyclic graph $G(\pi^1, \dots, \pi^m)$ contains at most $O(nm)$ arcs, the corresponding schedule can be calculated in $O(nm)$ time. In Section 4.3, we will describe how the longest path calculations can be implemented efficiently.

Based on the results of this section, we may solve the flow-shop problem with intermediate buffers by enumerating all feasible solutions and calculating for each feasible solution the makespan of the best schedule respecting the given job orders. However, since the number of feasible solutions grows exponentially, this approach would in reasonable time lead to a solution only for small instances. Taking into account that the problem is $\mathcal{NP}$-hard, in the remaining part of the paper we will present a local search approach with the set of all feasible solutions as search space.

4 A Local Search Approach

In this section we describe a local search approach for the flow-shop problem with intermediate buffers. As indicated in the previous section we restrict the search space to all m-tuples of permutations which are compatible with the buffer capacities. In connection with local search methods the definition of an appropriate neighborhood structure on the search space is an important issue. In Subsection 4.1 we describe suitable neighborhood structures for the flow-shop problem with intermediate buffers. These neighborhoods incorporate structural properties of the problem. Afterwards, in Subsection 4.2 we present a tabu search approach using these neighborhoods. The decision to chose a tabu search approach resulted from the success of this method for shop problems (see, e.g., Aarts et al. [1994], Nowicki & Smutnicki [1996a,1996b]).

4.1 Neighborhood Structures

Neighborhood structures define the way in which a local search approach may navigate through the search space. Furthermore, for a tabu search approach the computing time to chose 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 set of neighbors of a solution does not get too large.

To derive such neighborhood structures, we first will state a theorem which gives necessary properties of solutions which may improve a given solution. This result is based on a so-called block approach. Such an approach was first proposed for the single-machine problem $1 \mid r_j \mid L_{\max}$ (Grabowski et al. [1986]). Later it was successfully adapted to some other scheduling problems (like the job-shop or flow-shop problem, cf. Brucker et al. [1994], Nowicki & Smutnicki [1996a,1996b]). In the search process only solutions having the stated necessary properties will be considered as neighbored solutions.

Let a feasible solution $(\pi^1, \dots, \pi^m)$ be given and let $P(\pi^1, \dots, \pi^m)$ be a critical path in the solution graph $G(\pi^1, \dots, \pi^m)$. A sequence $B = (u_1, \dots, u_k)$ of successive operations on $P(\pi^1, \dots, \pi^m)$ is called a **block** if the following two properties are satisfied:

- The sequence contains at least two operations (i.e. $k \geq 2$).
- The sequence represents a maximal number of operations to be processed consecutively on the same machine.

Let the **incoming arc of block $\boldsymbol{B = (u_1, \dots, u_k)}$** be the arc on $P(\pi^1, \dots, \pi^m)$ which terminates at the vertex representing u_1, and let the **outgoing arc of block $\boldsymbol{B}$** be the arc on $P(\pi^1, \dots, \pi^m)$ which emanates from the vertex representing u_k. Obviously, the incoming and outgoing arc of each block must be a job arc or a buffer arc (in this terminology we will consider arcs $(\circ, i)$ and arcs $(i, *)$ as job arcs).

The following theorem is the basis for defining suitable neighborhoods on the set of all feasible solutions.

Theorem 3 : Let $(\pi^1, \dots, \pi^m)$ and $(\tilde{\pi}^1, \dots, \tilde{\pi}^m)$ be two arbitrary feasible solutions, and let $P(\pi^1, \dots, \pi^m)$ be a critical path in the solution graph $G(\pi^1, \dots, \pi^m)$. If $C_{\max}(\tilde{\pi}^1, \dots, \tilde{\pi}^m) < C_{\max}(\pi^1, \dots, \pi^m)$ holds, then at least one of the following conditions must be satisfied for some block $B = (u_1, \dots, u_k)$ of $P(\pi^1, \dots, \pi^m)$:
(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 one operation from $\{u_2, \dots, u_k\}$ is processed in $\tilde{\pi}^i$ before operation u_1 or one operation from $\{u_1, \dots, u_{k-1}\}$ 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:
$$\sum_{j=e}^{f-1} p_{i,\tilde{\pi}^i(j)} < \sum_{j=e}^{e+k-2} p_{i,\pi^i(j)},$$
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 $f-1<e$ 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 $P(\pi^1,\ldots,\pi^m) = (\circ, u_1^1, u_2^1, \ldots, u_{m_1}^1, \ldots, u_1^k, u_2^k, \ldots, u_{m_k}^k, *)$, where $u_1^j, \ldots, u_{m_j}^j$ $(j=1,\ldots,k)$ denotes a maximal number of operations to be processed consecutively on the same machine (i.e. a block if $m_j>1$).

We decompose path $P(\pi^1,\ldots,\pi^m)$ 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 $P(\pi^1,\ldots,\pi^m)$ can be expressed as follows:
$$L(P(\pi^1,\ldots,\pi^m)) = L(P_1^1) + L(P_2^1) + \ldots + L(P_1^k) + L(P_2^k) = C_{\max}(\pi^1,\ldots,\pi^m).$$

Now assume, that a feasible solution $(\tilde{\pi}^1,\ldots,\tilde{\pi}^m)$ with $C_{\max}(\tilde{\pi}^1,\ldots,\tilde{\pi}^m) < C_{\max}(\pi^1,\ldots,\pi^m)$ exists and none of the four conditions given in the theorem is satisfied for any block of $P(\pi^1,\ldots,\pi^m)$. 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 $G(\tilde{\pi}^1,\ldots,\tilde{\pi}^m)$ with length $L(\tilde{P}) \geq L(P(\pi^1,\ldots,\pi^m))$ which is a contradiction to $C_{\max}(\tilde{\pi}^1,\ldots,\tilde{\pi}^m) < C_{\max}(\pi^1,\ldots,\pi^m)$.

For the construction of a path $\tilde{P}$, the basic idea is that $\tilde{P}$ should contain the same job arcs as $P(\pi^1,\ldots,\pi^m)$ and the buffer arcs, which correspond to the same buffer arcs of $P(\pi^1,\ldots,\pi^m)$ (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 $G(\tilde{\pi}^1,\ldots,\tilde{\pi}^m)$ 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 $G(\tilde{\pi}^1,\ldots,\tilde{\pi}^m)$. Obviously, we get $L(\tilde{P}_2^k) \geq p_{u_{m_k}^k} = L(P_2^k)$.

The subpaths $\tilde{P}_1^j$; $j = 1, \ldots, k$ are defined as longest path in $G(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ 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 $j = 2, \ldots, k-1$. If P_1^j consists of a single operation, 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 $P(\pi^1, \ldots, \pi^m)$. 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, 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 $G(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ 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, 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 $L(\tilde{P}_1^j) \geq \sum\limits_{v=e}^{e+m_j-2} p_{i,\tilde{\pi}^i(v)} \geq \sum\limits_{v=e}^{e+m_j-2} p_{i,\pi^i(v)} = L(P_1^j)$.

(c) If the incoming arc of B_j is a buffer arc and the outgoing arc of B_j is a job arc, 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 $u_{m_j}^j$. Denoting by f the position on which operation $u_{m_j}^j$ is processed on M_i in $\tilde{\pi}^i$ we have $L(\tilde{P}_1^j) \geq \sum\limits_{v=e}^{f-1} p_{i,\tilde{\pi}^i(v)} \geq \sum\limits_{v=e}^{e+m_j-2} p_{i,\pi^i(v)} = L(P_1^j)$ since Condition 3 does not hold.

(d) If the incoming arc of B_j is a job arc and the outgoing arc of B_j is a buffer arc, 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 $L(\tilde{P}_1^j) \geq \sum\limits_{v=g}^{e+m_j-2} p_{i,\tilde{\pi}^i(v)} \geq \sum\limits_{v=e}^{e+m_j-2} p_{i,\pi^i(v)} = L(P_1^j)$ since Condition 4 does not hold.

Summarizing, we get $L(\tilde{P}_1^j) \geq L(P_1^j)$ for $j = 2, \ldots, k-1$, $L(\tilde{P}_2^k) \geq L(P_2^k)$, and $L(\tilde{P}_2^j) = L(P_2^j)$ for $j = 1, \ldots, k-1$. Thus, we have $C_{\max}(\tilde{\pi}^1, \ldots, \tilde{\pi}^m) \geq L(\tilde{P}) \geq L(P(\pi^1, \ldots, \pi^m)) = C_{\max}(\pi^1, \ldots, \pi^m)$, which is a contradiction. □

Based on this theorem which gives necessary conditions for a solution $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ to be better than $(\pi^1, \ldots, \pi^m)$ in the following we will define neighborhood structures for the given problem. As basic operators to move from one feasible solution to another one we will use shift operators. They perform a shift of a job from its position in one of the permutations to another position in the same permutation and leave all other permutations unchanged.

Let $(\pi^1, \ldots, \pi^m)$ be a feasible solution and $B = (u_1, \ldots, u_k)$ be a block on a critical path in the corresponding solution graph. Assume again 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 3 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 $e + k - 2$ will reduce. We consider shifts of operations of B to positions outside the block and shifts of operations outside B into the block. In both cases, 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. Again, we only consider shifts directly before or after the relevant operation.

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 compose of the above discussed shifts. Based on these considerations, we introduce the following neighborhood $\mathcal{N}_1$.

Neighborhood $\mathcal{N}_1$: Let $(\pi^1, \ldots, \pi^m)$ be a feasible solution, and let $P(\pi^1, \ldots, \pi^m)$ be a critical path in $G(\pi^1, \ldots, \pi^m)$. Furthermore, let $B = (u_1, \ldots, u_k)$ be an arbitrary block of $P(\pi^1, \ldots, \pi^m)$ 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 $\mathcal{N}_1(\pi^1, \dots, \pi^m)$ consists of all feasible solutions $(\pi^1, \dots, \pi^{i-1}, \tilde{\pi}^i, \pi^{i+1}, \dots, \pi^m)$ which can be constructed by one of the following shifts:

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}.

These operators are applied to every block B on the chosen critical path P.

Formally, the neighborhood $\mathcal{N}_1$ can be defined by the following two operators:

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

Here, $i \in \{1, \dots, m\}$ is equal to the machine where the block is processed. Furthermore, block B determines the relevant indices $j, k \in \{1, \dots, n\}$.

First numerical tests indicated that for neighborhood $\mathcal{N}_1$ the number of neighbors was often very small, such that a diversification of the search was week. 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. Such a situation is shown in Figures 3 and 4, where the corresponding parts of the solution graph before and after applying the operator $rshift(i, j, k)$ are represented. In fact, the smaller the buffer capacities are, the more moves lead to infeasible neighbors.

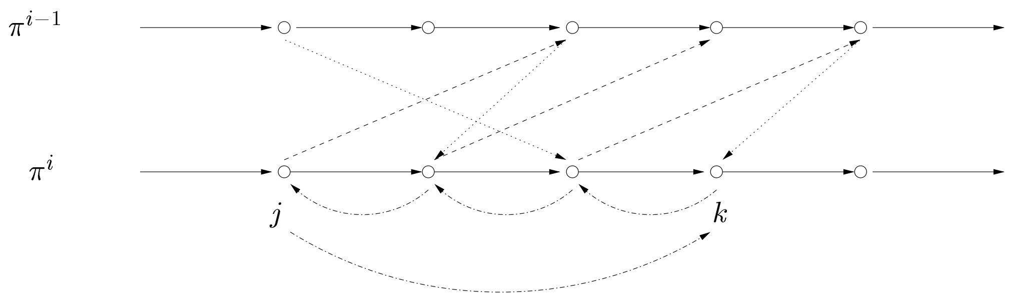

Figure 3: Situation before applying the operator $rshift(i,j,k)$

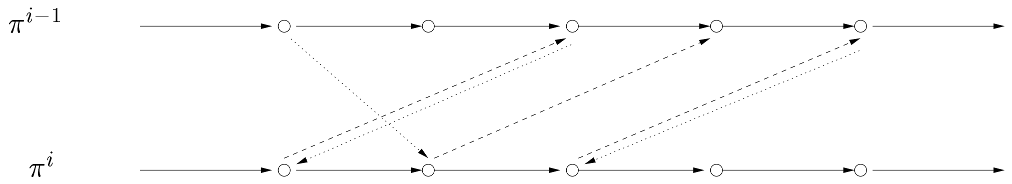

Figure 4: Situation after applying the operator $rshift(i,j,k)$

To overcome this problem of infeasibility of neighbors, we introduce another neighborhood $\mathcal{N}_2$. Before presenting the neighborhood, we characterize the shifts which lead to feasible solutions again.

Lemma 2 :
When applying the operator $rshift(i,j,k)$ to the feasible solution $(\pi^1,\dots,\pi^m)$ the resulting solution is feasible if and only if

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

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

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

Proof: When applying the operators $rshift(i,j,k)$ or $lshift(i,j,k)$ to the feasible solution $(\pi^1,\dots,\pi^m)$, permutation π^i is changed into $\tilde{\pi}^i$. The resulting solution $(\pi^1,\dots,\pi^{i-1},\tilde{\pi}^i,\pi^{i+1},\dots,\pi^m)$ 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 2 this is valid, iff the subgraphs $G(\pi^{i-1},\tilde{\pi}^i)$ and $G(\tilde{\pi}^i,\pi^{i+1})$ does not contain any cycle. Applying the operator $rshift(i,j,k)$ to the feasible solution $(\pi^1,\dots,\pi^m)$, 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 $G(\pi^{i-1}, \tilde{\pi}^i)$. 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 $j - b_i, j - b_i + 1, \ldots, k - b_i - 1$. Thus, $G(\pi^{i-1}, \tilde{\pi}^i)$ and $G(\tilde{\pi}^i, \pi^{i+1})$ are acyclic in this case, iff the first of the above inequalities are fulfilled. Similarly, conditions are derived for the case of the operator $lshift(i, j, k)$ as well. □

Lemma 2 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 $\mathcal{N}_1$ in a new neighborhood $\mathcal{N}_2$. The idea of $\mathcal{N}_2$ is to carry out the promising shift (according to conditions 1.–4. in the definition of neighborhood $\mathcal{N}_1$) 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 the permutations $\pi^{i+1}, \pi^{i+2}, \ldots, \pi^m$ until we finally get a feasible solution. In the following, we will describe this **correction 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 2 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 case b) first. In this case 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 $k - b_i$. It is easy to prove that now the permutations $\tilde{\pi}^i$ and $\tilde{\pi}^{i+1}$ are compatible with b_i. (The relevant jobs in permutations π^i and π^{i+1} at positions $j+l+1, \ldots, k$ and $j-b_i+l+1, \ldots, k-b_i$ are all moved one position to the left.) 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 correction 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 $\tilde{\pi}^i$ and $\tilde{\pi}^{i-1}$ are compatible with b_{i-1}. 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 correction procedure if applying the operator $lshift(i, j, k)$ results in an infeasible solution. Summarizing, we introduce the following neighborhood $\mathcal{N}_2$.

Neighborhood $\mathcal{N}_2$: Let $(\pi^1, \ldots, \pi^m)$ be a feasible solution. Neighborhood $\mathcal{N}_2(\pi^1, \ldots, \pi^m)$ consists of all feasible solutions $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ where $\tilde{\pi}^i$ was constructed using the criteria of possible improvement (see conditions 1.–4. in the definition of neighborhood $\mathcal{N}_1$) and where $\pi^1, \ldots, \pi^{i-1}$ as well as $\pi^{i+1}, \ldots, \pi^m$ were 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 correction technique.

Neighborhood $\mathcal{N}_2$ contains $\mathcal{N}_1$ as a subneighborhood (i.e. $\mathcal{N}_1(\pi^1, \ldots, \pi^m) \subseteq \mathcal{N}_2(\pi^1, \ldots, \pi^m)$ for each solution $(\pi^1, \ldots, \pi^m)$). The size of both neighborhoods is polynomially bounded by mn^2 because the number of shifts on each machine is bounded by n^2.

An important 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. We have constructed a sophisticated example which shows that the neighborhood $\mathcal{N}_2$ 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 $\mathcal{N}_2$. As this example contains many jobs and is rather complex, we do not describe it here. Although the neighborhood $\mathcal{N}_2$ is not opt-connected, we use it in our local search approach, since the computational results for the neighborhood $\mathcal{N}_2$ are very satisfying and the constructed counterexample indicates that only in extreme cases disconnected components occur.

4.2 A Tabu Search Approach

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 the previous section.

The **tabu search** method is a metaheuristic approach which was designed by Glover [1989,1990]. 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. In order to avoid cycling, a so-called **tabu list** is used which stores attributes characterizing solutions that should not be considered again for a certain length of time. All moves to solutions characterized by these attributes are forbidden (tabu). A disadvantage of this procedure 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 the acceptance of 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 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 solution from this long-term memory (diversification). The whole tabu search procedure terminates after a maximal number of restarts has been finished.

In the following we will describe how these general concepts are applied in our tabu search algorithm for the flow-shop problem with intermediate buffers. The attributes of a solution $(\pi^1, \dots, \pi^m)$ 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 a sequence of m permutations $(\pi^1, \dots, \pi^m)$, besides the objective value $C_{\max}(\pi^1, \dots, \pi^m)$ 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 $(\tilde{\pi}^1, \dots, \tilde{\pi}^m)$ of a solution $(\pi^1, \dots, \pi^m)$ 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, $(\tilde{\pi}^1, \dots, \tilde{\pi}^m)$ is tabu if

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

A neighbored solution $(\tilde{\pi}^1, \dots, \tilde{\pi}^m)$ satisfies the aspiration criterion if its makespan $C_{\max}(\tilde{\pi}^1, \dots, \tilde{\pi}^m)$ 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 $(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$ with minimal $C_{\max}(\tilde{\pi}^1, \ldots, \tilde{\pi}^m)$-value.

Our diversification method is based on the following restart technique: The tabu search is stopped if for a certain number of iterations ($maxWorse$) 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 (*maxReturns*) 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.

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. [1983] 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 [1990] 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).

4.3 Efficient calculation of longest paths

In our proposed local search approach the procedure to calculate the best schedule given a fixed m-tupel of permutations plays a central role, as it is used to calculate the objective value of the solutions. Therefore, an efficient implementation of the longest path calculation in the solution graph is essential for an efficient local search approach.

Regarding the solution graph the only point, where some computational effort can be saved, concerns the order in which the operations are evaluated. In a standard implementation one always has to do some book-keeping to detect vertices for which already all predecessors have been evaluated and, thus, may be evaluated next. However, due to the special structure of the solution graph of a feasible solution, we can determine an order in which the vertices may be evaluated and which is independent of the given feasible solution. Clearly, a vertex cannot be evaluated before its machine-, job- and bufferarc-predecessor are evaluated. The order we use is characterized by the following iterative strategy which we call the **Lowest Machineindex-Rule**: In each step, we consider the set M which contains the operations where the machine- and bufferarc-predecessor (if they exist) have already been evaluated. Initially, M contains all first operations of all machines. From the set M the operation is evaluated next which is scheduled on the machine with the lowest index.

According to this rule, all machine and buffer arcs are respected explicitly. That the order determined by this rule also respects the precedence relations given by the job arcs can be seen as follows: Assume that there exists a job arc from operation j on M_i to operation k on M_{i+1}, i.e. operation j has to be evaluated before operation k. Due to the lowest machineindex-rule, operation k on M_{i+1} only would precede operation j on M_i, if a path consisting of a buffer arc and machine arcs from k to j would exist. Thus, together with the job arc from j to k this path would form a cycle in the solution graph and the solution would not be feasible.

Next, we explain more precisely in which order the operations are evaluated according to the lowest machineindex-rule: At first, operations on M_1 are evaluated until the first operation on M_1 has a bufferarc-predecessor. This is the case on position $b_1 + 2$. Now, the first operation of M_2 is evaluated before the operation on position $b_1 + 2$ on M_1 is evaluated. Then, operations on M_2 and M_1 are evaluated alternately until the first operation on M_2 has a bufferarc-predecessor. Then, operations on M_3, M_2 and M_1 are evaluated alternately and so forth.

For the example in Figure 1 the lowest machineindex-rule leads to the following evaluation order of the operations:

$$11, 12, 22, 13, 21, 14, 23, 15, 33, 25, 16, 31, 24, 34, 26, 36, 35, 32, t.$$

Note, that this order is only dependent on the given buffer capacities. Thus, given an instance of a flow-shop problem with intermediate buffers we use the same evaluation order for each feasible solution of the instance. Computational tests have shown that using this fixed order instead of book-keeping the number of not evaluated predecessors speeds up the longest path calculation considerably.

Furthermore, the fixed evaluation 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 evaluation 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 evaluation order can be determined in constant time. If a correction step is necessary, we have to consider the earliest position on each machine which is concerned by the correction. One of these at most m positions specifies p.

Clearly, the larger the position in the evaluation 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 computational time.

5 Computational Results

In this section 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 ($4\times$ 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 [1993] 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 from 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 permutation flow-shop problem with blocking-restrictions. 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 tested with the following parameter combinations: $maxWorse = 500$, $maxReturn = 1, 3, 5$ and $maxWorse = 1000$, $maxReturn = 1, 3, 5$ and $maxWorse = 3000$, $maxReturn = 1, 3$ and $maxWorse = 10,000$, $maxReturn = 1$. 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. In Section 5.1 we evaluate the quality of the proposed local search approach. In Section 5.2 we indicate the influence of the buffer capacities on the makespan of the best found solution.

5.1 Quality of the local 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 [1996], 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 1 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 1 shows that for the small instances (5×20, 10×20, 5×50) and a parameter configuration of $maxWorse = 3000$ and $maxReturns = 3$ 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 $maxWorse = 3000$ and $maxReturn = 10$ was chosen (which took four times longer).

For the larger instances, the reduction values are considerably less than for the small instances: For $maxWorse = 3000$ and $maxReturns = 3$ there was achieved a reduction of 5% to 15%, which was improved to 10% to 20% when the parameter $maxReturns$ was increased to 10 (CPU-time was also approximately four times longer in this case). This is mostly due to the higher makespan values compared to those of the smaller instances.

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 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

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

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 $maxWorse = 1000$, $maxReturns = 5$ and $maxWorse = 3000$, $maxReturns = 1$, which took a comparable computational time according to Table 1.

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 calculated by a generalization of a heuristic of Nawaz et al. [1983] (see Section 4.2). In Tables 2 and 3 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 4 shows the mean computational time (in minutes : seconds)

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 2: Relative improvements imp for $maxWorse = 3,000$, $maxReturns = 1$

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 3: Relative improvements imp for $maxWorse = 1,000$, $maxReturns = 5$

of the tabu search for $maxWorse = 3000$ and $maxReturns = 1$. The computational times for the parameter setting $maxWorse = 1000$ and $maxReturns = 5$ lies in the mean 39% over the values in Table 4.

- Tables 2 and 3 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 [1996b]) 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 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.

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 4: Mean computational time (in minutes : seconds) for $maxWorse = 3,000$, $maxReturns = 1$

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 4. 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.

5.2 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 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 5 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

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

least one problem of a problem class a better makespan was found when reducing the buffer capacity from $b_i = \infty$ to $b_i = j$.

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. The latter effect can again be explained by the way we treat infeasible neighbors. 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 at most 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. The major influence results of 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. 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, especially as 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.

6 Conclusions

We have presented a tabu search approach for the flow-shop problem with intermediate buffers where different job sequences on the machines are allowed. Conditions to characterize feasible solutions have been proposed. The classical disjunctive graph model for shop problems has been adapted to model flow-shop problems with buffers and the classical block-approach for the flow-shop problem has been extended to the problem with buffers.

The provided numerical results seem to be very satisfactory. To get a deeper insight into the quality of the solutions it would be of interest to calculate good lower bounds for the problem with small buffer capacities. Another interesting question is whether the proposed concepts can be generalized and applied to other shop problems with buffers.

Acknowledgment: The authors are grateful to Tim Nieberg for implementing the tabu search procedure proposed in this paper.

References

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

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

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

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

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

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

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

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

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

C.H. Papadimitriou, P.C. Kanellakis [1980] Flow shop scheduling with limited temporary storage, J. Assoc. Comput. Mach. 27, 533-549.

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

E. Taillard [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`

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