Cyclic Scheduling Problems

Dissertation
von
Thomas Kampmeyer

März 2006

Fachbereich Mathematik/Informatik
Universität Osnabrück

Danksagung

Diese letzten Zeilen meiner Dissertation, die ich nun schreibe, möchte ich nun nutzen, um denjenigen zu danken, die mich in den letzten drei Jahren begleitet, unterstützt und gefördert haben.

Besonders bedanken möchte ich mich bei meinem Betreuer Prof. Dr. Peter Brucker, ohne ihn wäre diese Arbeit wohl nie zustande gekommen. Für seine Impulse und Gedanken, die gemeinsamen Artikel und die Zeit, die er immer für mich hatte, Vielen Dank! Ebenfalls gilt Prof. Ph.D. Eugene Levner Dank für die engagierten Ideen, die zur besseren Verständlichkeit dieser Arbeit beitrugen.

Weiterhin sei das Cusanuswerk genannt, das mit seiner finanziellen Förderung mir diese Dissertation ermöglichte. Ebenfalls trug diese Förderung viel dazu bei mir einen Auslandsaufenthalt und den Besuch verschiedener Kongresse zu ermöglichen. Mich wird es freuen einen Teil dieser Unterstützung als Altcusaner zurückzugeben.

Außerdem bedanke ich mich bei allen Mitarbeitern im Institut für Mathematik, besonders bei Silvia Heitmann und Christian Strotmann, mit denen ich viele anregende Gespräche, spannende Kongresse und informelle Zeiten teilen konnte. Weiter möchte ich mich auch bei Tim Nieberg bedanken, der mir bei einigen sprachlichen Problemen immer helfen konnte.

Ein großes Dankeschön richtet sich auch an meine Familie. Mit ihrem Zuspruch trugen sie maßgeblich zu dem Entschluß bei, diese Dissertation zu realisieren.

Zum Schluß möchte ich meiner Freundin Kerstin Polster herzlich danken für ihre Unterstützung und ihren Zuspruch während der Zeit dieser Arbeit. Dank ihr fiel es mir manchmal leichter, diese Arbeit zu vollenden.

Contents

1 Introduction

For classical non-cyclic scheduling problems, we are given a set of operations, each of which has to be processed exactly once. The aim is to minimize or maximize a given objective function such as makespan or sum of all (weighted) completion times for a given set of constraints.

The set of constraints is usually given by precedence constraints between the operations. In contrast to these problems, for cyclic scheduling problems we are given a set of operations, each of which has to be processed infinitely often. Such types of scheduling problems arise in different application areas like compiler design, manufacturing, digital signal processing, railway scheduling, timetabling, etc.

The problem is to find a periodic schedule which minimizes a given objective function. There exist two objective functions which are important in this area of cyclic scheduling. The objective which is considered throughout this work is to minimize the time difference between two succeeding occurrences of one operation for a given set of constraints. This time difference is called cycle time. This objective can be generalized to K-periodic problems. Here the new objective is to minimize the time difference between the l-th occurrence and the (l + K)-th occurrence of an operation, where K occurrences instead of one occurrence of an operation are now processed in one period. The other objective, which is not considered in this work, is to minimize the flow time of a job (see Roundy [62]). The flow time describes the time which is needed to produce one occurrence of one job that consists of several operations.

The main applications which are considered in the cyclic scheduling literature are robotic cell problems (see e.g. Crama et al. [20], Kats and Levner [39], Matsuo et al. [51]). Here a robot transports the jobs from one machine to another machine. Lee and Posner [43] and Hall et al. [29] investigate complexity issues for cyclic job-shop problems with machine chains repetition. Another area in which cyclic scheduling plays an important rule is computer pipelining (see e.g. [3, 59]).

In this thesis, we develop a general framework to model and to describe cyclic scheduling problems with resource constraints. The basis for this thesis is done by Hanen [30]. The underlying subproblem is called Basic Cyclic Scheduling Problem (BCSP) and it is polynomial solvable. We generalize the BCSP and propose a fast algorithm in practise to solve the generalized problem. This algorithm is based on Howard s Algorithm (see e.g. Dasdan et al. [23]). We also present a linear programming formulation for the K-periodic problems. Furthermore, we extend the model by Hanen by adapting the alternative graph model developed by Mascis and Pacciarelli [50] to model blocking constraints for cyclic

scheduling problems. Afterwards, we generalize the new blocking model to describe more general blocking situations. These arise, e.g., in the area of software pipelining.

In order to show that we can formulate a great variety of different cyclic scheduling problems with our framework, we analyse different models from the literature and show how to model these problems within our proposed framework. We present applications both with and without blocking, and both with and without transportation robots. With our proposed model, we can thus gain new insights into the structural properties of each of these applications.

Since we derive a corresponding mixed integer linear programming (MILP) formulation for each application, we also present a way to solve these problems. Unfortunately, this approach can solve problems with only a few operations and machines. In order to deal with this problem, we develop a local search approach that is based on this MILP formulation. This local search approach is the first, and quite promising solution method for solving cyclic scheduling problems.

In Section 2, we present the General Basic Cyclic Scheduling Problem (GBCSP) which generalizes the Basic Cyclic Scheduling Problem. We also adapt an algorithm to solve the GBCSP. The algorithm is a generalization of Howard's Algorithm which is proposed to solve the BCSP.

In Section 3, we present the model proposed by Hanen [30] together with some extensions which are needed to describe the several applications considered later on.

In Section 4, we extend the model proposed in Section 3 by blocking constraints. This is done by adapting the alternative graph model for the non-cyclic scheduling problem of Mascis and Pacciarelli [50]. We also develop a mixed integer linear programming formulation. This formulation is quite similar to the formulation presented in the previous section.

In Section 5, we consider a different type of precedence constraint for cyclic scheduling problems, namely linear precedence constraints. We develop a new mixed integer linear programming formulation for the cyclic problem with linear precedence constraints and resource constraints. This formulation is based on a result by Hanen and Munier-Kordon [32] for the problem without resource constraints.

In Section 6, we present several different cyclic scheduling problems which are proposed in the cyclic scheduling literature. For all these applications, we describe how to model these within our framework. We also consider problems from the area of software pipelining. To model a special problem in this area, the so-called register allocation problem, we extend the alternative arc model described in Section 4 to an alternative arc set model.

In Section 7, two methods for solving some of the considered applications are presented. First, we solve the problems with a mixed integer linear programming solver. As these results are not very promising, we develop a local search approach. We derive several new neighborhoods for cyclic scheduling problems with blocking.

In Section 8, implementation details and computational results for our local search approach are presented.

Finally, we conclude this thesis with some remarks and directions for possible further research in 9.

2 The General Basic Cyclic Scheduling Problem

In this section, we present the General Basic Cyclic Scheduling Problem, or short GBCSP, which is an extension of the Basic Cyclic Scheduling Problem (BCSP). It has been studied by many different scientists (see e.g. [12, 16, 19, 31, 60, 61]).

This section consists of four parts. In the first part we describe the problem definition for the GBCSP and derive conditions under we can find a periodic schedule. In the second part we present an in practice very fast algorithm which can be applied first to check conditions for the existence of a periodic schedule and second to compute the optimal cycle time if the graph described by the precedence constraints is a strongly connected graph. In the third part we consider a general graph, which consists of several strongly connected components, and propose a way to compute the cycle time for this general graph. In the last part we consider K-periodic problems and show that the 1-periodic problems provide better solutions than K-periodic problems.

2.1 The Problem Definition

As already explained in the introduction, the GBCSP is the basis for solving the general cyclic machine scheduling problem and its extensions, which are presented in Sections 3 and 4.

The GBCSP can be described as follows: Let $T = \{1, \ldots, n\}$ be a set of generic operations. Operation i has a processing time $p_i > 0$ and must be performed infinitely often. We denote by $< i; k >$ the k-th occurrence of the generic operation i. A **schedule** assigns a starting time $t(i; k)$ to each occurrence $< i; k >$. A schedule is called **periodic** with cycle time α if

$$t(i; k) = t(i; 0) + \alpha k \text{ for all } i \in T,\ k \in \mathbb{Z}. \tag{2.1}$$

We define $t_i := t(i; 0) \geq 0$ for all $i \in T$. A periodic schedule is defined by the vector $(t_i)_{i \in T}$ and the cycle time $\alpha \geq 0$. We also postulate that the $k + 1$-th occurrence of operation i can only start if the k-th occurrence is finished. Thus, we get the following constraint

$$t(i; k + 1) \geq t(i; k) + p_i.$$

Furthermore, there is given a graph $G = (T, E)$ with vertex set T and arc set E. Each arc $(i, j) \in E$ is supplied by two values L_{ij} and H_{ij}. L_{ij} is called (start-start) **delay** and H_{ij}

is called the **height** (or **distance**). The delays are assumed to be rational numbers and the heights are assumed to be arbitrary integers.

This graph leads to the following uniform precedence constraints

$$t(i;k) + L_{ij} \leq t(j;k + H_{ij}).$$

The basic cyclic scheduling problem (BCSP) is the special case of the GBCSP in which the delays are restricted to be nonnegative rational numbers.

Thus, the problem can be formulated as

$$min \ \alpha \tag{2.2}$$

s.t.

$$t(i;k) = t(i;0) + k\alpha \quad i \in T, k \in \mathbb{Z} \tag{2.3}$$

$$t(i;k) + L_{ij} \leq t(j;k + H_{ij}) \quad (i,j) \in E, k \in \mathbb{Z} \tag{2.4}$$

$$t(i;k) + p_i \leq t(i;k+1) \quad i \in T, k \in \mathbb{Z} \tag{2.5}$$

By substituting equation (2.1) into (2.4) and (2.5) we get

$$min \ \alpha \tag{2.6}$$

s.t.

$$t_i + L_{ij} - \alpha H_{ij} \leq t_j \quad (i,j) \in E \tag{2.7}$$

$$t_i + p_i \leq t_i + \alpha \quad i \in T \tag{2.8}$$

In the following we assume that the constraints (2.8) are included in (2.7) by adding loops (i,i) with $L_{ii} = p_i$ and $H_{ii} = 1$ to E.

The problem given by (2.6) to (2.8) is a special case of the **maximum cost-to-time ratio problem**. In the general maximum cost-to-time ratio problem the delays and heights are assumed to be real numbers. An easy way to solve the cost-to-time ratio problem is to use the simplex method. This way of solving the problem is first mentioned by Dantzig et al. [22]. However, it turns out that using specialised algorithms would solve the problem (2.6) to (2.8) much faster. Chen et al. [15] developed an algorithm for the problem with complexity $O(n^6)$. A faster algorithm is introduced in Levner and Kats [47]. This algorithm has a complexity of $O(n^4)$.

In the remaining part of this subsection, we present the necessary and sufficient conditions for the existence of a periodic schedule. We also show that the optimal cycle time can be computed by analysing the circuits in the graph G.

Let μ be a circuit in E. Then we denote

$$L(\mu) := \sum_{(i,j)\in\mu} L_{ij}$$

and

$$H(\mu) := \sum_{(i,j)\in\mu} H_{ij}$$

the **delay** and the **height** of μ, respectively.

The following theorem describes the conditions under we can find a solution for the problem (2.6) to (2.8). Note, a very similar theorem is given in e.g. Hanen and Munier [31] and Dasdan et al. [23] for positive delays and heights.

Theorem 2.1 *The GBCSP has a feasible periodic solution with cycle time $\alpha > 0$ if and only if each circuit μ fulfills one of the following three conditions*

1. *The circuit μ has a positive height and arbitrary delay,*
2. *The circuit μ has a negative height and a negative delay,*
3. *The circuit μ has height zero and a non-positive delay,*

and additionally to the three conditions the following inequalities

$$\min\left\{\frac{L(\mu)}{H(\mu)} | \mu \textit{ is a circuit with } H(\mu) < 0\right\} \geq \alpha \geq$$

$$\max\left\{\frac{L(\mu)}{H(\mu)} | \mu \textit{ is a circuit with } H(\mu) > 0\right\} \tag{2.9}$$

hold.

Note that if no circuit μ with $H(\mu) < 0$ exists, then
$\min\left\{\frac{L(\mu)}{H(\mu)} | \mu \text{ is a circuit with } H(\mu) < 0\right\}$ is set to ∞. By the inclusion of loop constraints (2.8) into the set E, there always exist circuits with positive delay and height. Thus, $\max\left\{\frac{L(\mu)}{H(\mu)} | \mu \text{ is a circuit with } H(\mu) > 0\right\} > 0$ holds.

Proof: First we show that if there exists a feasible solution, then Conditions 1 to 3 and the inequalities (2.9) hold. As there exists a feasible solution with cycle time $\alpha > 0$, the inequality

$$L(\mu) - \alpha H(\mu) \leq 0 \tag{2.10}$$

holds which we get by adding the inequalities (2.7) along the circuit μ for every circuit μ from i to i ($i \in T$). Thus, inequalities (2.9) are fulfilled. Now we show that for each circuit μ in graph G one of the Conditions 1 to 3 hold.

If $H(\mu) > 0$, there is nothing to prove.
If $H(\mu) < 0$, then from equation (2.9) we get $\frac{L(\mu)}{H(\mu)} \geq \alpha > 0$. Therefore, $L(\mu) < 0$ and thus Condition 2 is fulfilled.
If $H(\mu) = 0$, then from equation (2.10) we get $L(\mu) \leq 0$. Thus, Condition 3 is fulfilled.

Now we show that if one of the conditions 1 to 3 and the inequality (2.9) are fulfilled, then there exists a feasible schedule with $\alpha > 0$. We choose an arbitrary α from the interval which is described by the inequalities (2.9). Based on the conditions 1 to 3 and inequalities (2.9), we have $L(\mu) - \alpha H(\mu) \leq 0$ for all circuits μ in G. Thus, we can compute the starting times t_i for all $i \in T$ satisfying (2.7) for the chosen cycle time α by longest path calculations. □

A direct conclusion is the following lemma.

Lemma 2.2 *The GBCSP has no feasible periodic solution if one circuit μ fulfills one of the following two conditions*

1. *The circuit μ has a negative height and non-negative delay,*
2. *The circuit μ has height zero and positive delay,*

or the inequalities (2.9) *do not hold.*

Theorem 2.1 leads to the following definition:

Definition 2.3 *A graph G is called **consistent** if each circuit in G fulfills one of the Conditions* (1) *to* (3) *of Theorem 2.1 and the interval described by* (2.9) *is not empty.*

The next theorem describes how to compute the optimal cycle time α. Consider a circuit μ. Then $V(\mu) := L(\mu)/H(\mu)$ is called the **value** of μ. The circuits with the maximum value and positive height are called **critical circuits**.

Theorem 2.4 *Assume that the GBCSP has a feasible periodic solution. Then the optimal cycle time is equal to the value of a critical circuit.*

Proof: The GBCSP can be described by the following linear program.

$$z_P = min \ \alpha \tag{2.11}$$

s.t.

$$t_j - t_i + \alpha H_{ij} \geq L_{ij} \quad (i,j) \in E \tag{2.12}$$

The dual of this linear program is

$$z_{DP} = max \sum_{(i,j)\in E} L_{ij} y_{ij} \tag{2.13}$$

s.t.

$$\sum_{(j,i)\in E} y_{ji} - \sum_{(i,j)\in E} y_{ij} = 0 \quad i \in T \tag{2.14}$$

$$\sum_{(i,j)\in E} H_{ij} y_{ij} = 1 \tag{2.15}$$

$$y_{ij} \geq 0 \quad (i,j) \in E \tag{2.16}$$

Due to the Duality Theorem in Chvatal [17, Theorem 5.1] the optimal value of the linear program (2.11) to (2.12) is equal to the value of the linear program (2.13) to (2.16). Now we show that every circuit provides a feasible solution for (2.13) to (2.16).

Let μ be a circuit in G with $\sum_{(i,j)\in\mu} L_{ij} = L(\mu)$ and $\sum_{(i,j)\in\mu} H_{ij} = H(\mu)$. There are m arcs in the graph G. If an arc $(i,j) \in E$ lies in the circuit μ, then we set y_{ij} to $1/H(\mu)$, otherwise y_{ij} is set to 0. We can easily see that the vector y_{ij} is a feasible solution for (2.13) to (2.16). The objective value is $\frac{L(\mu)}{H(\mu)}$.

Now we show that a solution of the dual problem with the objective value z_{DP} leads to one or more circuits with the value z_{DP}.

The dual problem can be seen as a circulation problem with the additional constraint (2.15). Let y be an optimal circulation for the problem (2.13) to (2.16) with the objective value z_{DP}. In Ahuja et al. [2, Theorem 3.5] it is shown that a circulation y can be decomposed into flows along at most m directed circuits. Consider now a decomposition $\mu_1 \ldots, \mu_r$ with cycle flows $f(\mu_1), \ldots, f(\mu_r) > 0$ of the optimal circulation y with $r \leq m$. We may assume that $H(\mu_j) > 0$ for at least one index j because otherwise the additional constraint (2.15) is violated. For an index i with $i \neq j$ we replace the flow $f(\mu_i)$ by $f(\mu_i) + \epsilon$ and the flow $f(\mu_j)$ by $f(\mu_j) + x$. To keep the additional constraint (2.15)

satisfied, $H(\mu_i)\epsilon + xH(\mu_j) = 0$ or $x = -\frac{H(\mu_i)}{H(\mu_j)}\epsilon$ must hold. Thus, if we replace $f(\mu_i)$ by $f(\mu_i) + \epsilon$ and $f(\mu_j)$ by $f(\mu_j) - \frac{H(\mu_i)}{H(\mu_j)}\epsilon$ with $|\epsilon|$ small such that $f(\mu_i) + \epsilon > 0$ and $f(\mu_j) - \frac{H(\mu_i)}{H(\mu_j)}\epsilon > 0$ hold, we get another feasible solution and the objective function changes by

$$\epsilon L(\mu_i) - \frac{H(\mu_i)}{H(\mu_j)}\epsilon L(\mu_j)$$

or

$$\epsilon(L(\mu_i) - \frac{L(\mu_j)}{H(\mu_j)}H(\mu_i))$$

which implies that

$$L(\mu_i) - \frac{L(\mu_j)}{H(\mu_j)}H(\mu_i) = 0 \tag{2.17}$$

for all $i = 1, \ldots, r$. Otherwise we could increase the objective function by choosing ϵ either positive or negative and with small $|\epsilon|$-value. Thus, the value of each circuit with positive flow is the same.

Equation (2.17) is equivalent to

$$L(\mu_i)f(\mu_i) = \frac{L(\mu_j)}{H(\mu_j)}H(\mu_i)f(\mu_i) \tag{2.18}$$

for all $i = 1, \ldots, r$.

If we add the equations (2.18) for all $i = 1, \ldots, r$, we have

$$\sum_{i=1}^{r} L(\mu_i)f(\mu_i) = \frac{L(\mu_j)}{H(\mu_j)}\sum_{i=1}^{r} H(\mu_i)f(\mu_i) = \frac{L(\mu_j)}{H(\mu_j)}$$

because $\sum_{i=1}^{r} H(\mu_i)f(\mu_i) = 1$ due to the additional constraint (2.15).

Thus, the optimal solution value of (2.13) to (2.16) has the desired form and the value of the optimal solution is equal to the value of a circuit with positive length and height. □

Now we present an example for a GBCSP.

Example 2.5 *The data for the problem is given in Table 2.1.*

Operation	*0*	*1*	*2*	*3*	*4*	*5*
Processing time	*0*	*2*	*3*	*1*	*1*	*0*

Table 2.1: Data for the example 2.5

The graph with the precedence constraints is given in Figure 2.1.

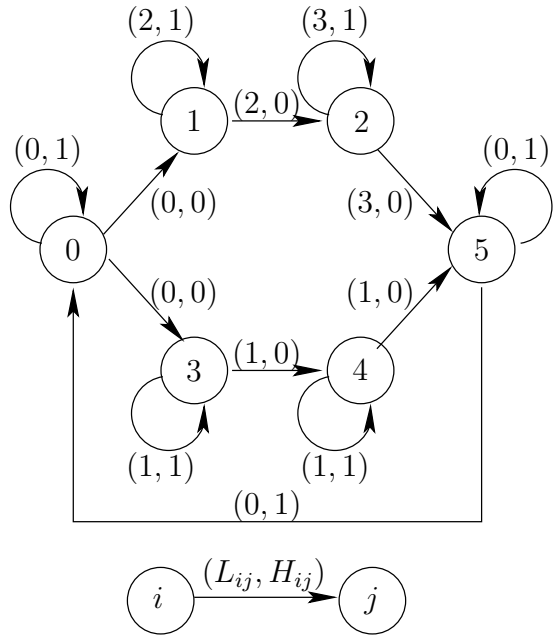

Figure 2.1: The graph G for the example 2.5

We can easily see that the graph is consistent because each circuit in the graph G has a positive delay and positive height. The critical circuit is $(0, 1, 2, 5, 0)$. *The value of this circuit is* $\alpha = \frac{0+2+3+0}{0+0+0+1} = 5$.

In the Examples 3.4 and 6.8 there exist circuits with negative delay and negative height and circuits with negative delay and zero height.

In this subsection, we described the GBCSP and provided a linear programming formulation. Furthermore, we derived conditions under we can show that there exists a feasible periodic solution. Finally, we showed that the value of the optimal solution is equal to the value of a critical circuit.

The next step is now to describe an algorithm which is capable of first to check for the existence of a periodic solution and second to compute the optimal cycle time α. This is done in the next subsection.

2.2 Algorithm to Solve the GBCSP

In this subsection, we present an algorithm to solve the problem (2.6) to (2.8). There exist several algorithms to solve the maximum cost-to-time ratio problem. A good overview on these algorithms can be found in Dasdan et al. [23].

As we want to develop a local search approach for solving cyclic scheduling problems, we need a very fast algorithm to evaluate a solution because in a local search approach many different solutions are considered and must be evaluated. Dasdan et al. [23] compared the running time of several algorithms for the problem in which the height of each arc is fixed to 1. It turns out that the fastest algorithm to solve these kinds of problems is Howard's Algorithm. Howard's Algorithm introduced in Howard [34] is first adapted to these kind of problems in Cochet-Terrasson et al. [18]. Therefore, we choose Howard's Algorithm for evaluating the solutions in our local search approach. Another reason for using Howard's Algorithm in our local search approach is that if the algorithm terminates, the algorithm has found a critical circuit or a circuit that violates the conditions of Theorem 2.1.

In [23] and [18] the height of all circuits has a positive value. As we consider problems in which the height of the circuits can be any arbitrary integer number, we need to generalize Howard's Algorithm. This is done in this subsection. The running time of the generalized algorithm is pseudo-polynomial. The pseudocode of Howard's Algorithm is given in Listing 1.

```
1  for each node u ∈ T do
2      d(u) := p_u;
3      π(u) := u;
4  improved:=TRUE;
5  while (improved==TRUE) do
6      E_π := {(u, π(u)|u ∈ T};
7      Examine all circuits in G_π = (T, E_π);
8      if there exists a circuit μ with H(μ) < 0 and L(μ) > 0 or
         H(μ) == 0 or H(μ) < 0 and L(μ) < 0 then
9          return INFEASIBLE;
10     Let μ be the circuit with maximum cost-to-time ratio in
         G_π with H(μ) > 0;
11     α := L(μ)/H(μ);
12     Select the node s ∈ μ with the smallest index;
13     T_μ := {u ∈ T|there exists a path from u to s in G_π};
14     while (T_μ ≠ T) do
```

```
15      Find node u ∈ T \ T_μ such that there is a v ∈ T_μ and
          (u, v) ∈ E;
16      π(u) := v;
17      T_μ := T_μ + {u};
18  \muompute the longest path d(u) from all u ∈ T to s in G_π;
19  improved=FALSE;
20  for each arc (u, v) ∈ E do
21      δ(u) := d(u) − (d(v) + L_uv − αH_uv);
22      if (δ(u) < 0) then
23          if (δ(u) < −ϵ) then
24              improved=TRUE;
25          d(u) := d(v) + L_uv − αH_uv;
26          π(u) := v;
27  return α;
```

Listing 1: Howard's algorithm

The general idea of Howard's Algorithm is the same as for all other algorithms which are based on the algorithm by Karp and Orlin [38] (see e.g. Ichimori and Soumis [36], Levner and Kats [47], Young et al. [72]): The algorithm starts with a small α and increases, in a very special way, α until some conditions for the optimality are fulfilled.

First, we discuss the structure of Howard's Algorithm. Then we consider an example to show how the algorithm works. Finally, we show that the algorithm finds in finite time either a critical circuit or a circuit that violates the conditions of Theorem 2.1.

The algorithm is divided into three parts. In the first part (line 1 to 4) the initialization of a special subgraph G_π of G given by $G_\pi = (T, E_\pi)$ with $E_\pi = \{u, \pi(u) | u \in T\}$ is done. During the runtime of the algorithm G_π is updated by updating π. G_π is called **policy graph**. G_π has the same node set as G. However, in G_π each node u has exactly one successor $\pi(u)$. After the initialization G_π consists of several disjunctive circuits.

The while-loop (line 5 to 26) can be divided into two parts. When the while-loop starts, the graph G_π always contains of several disjunctive circuits and for all nodes which are not in a circuit there exists a path to a node in a circuit. Then in the first part of the while-loop all disjunctive circuits are examined and it is checked whether the conditions of Theorem 2.1 are fulfilled. If these conditions are fulfilled, the algorithm can determine the circuit μ with the maximum value in G_π. Afterwards (line 14 to 17) the graph G_π is changed in such a way that the circuit μ is the only circuit in G_π and for all other nodes which are not in this circuit there exists a path to a node on the circuit μ. Note, if we delete one arc of the circuit μ, the graph G_π is a tree. Afterwards, labels $d(u)$ in the graph G_π are computed.

In the last part of the while loop the algorithm checks whether we can improve the labels $d(u)$ by changing the arcs in the graph G_π. We show:

- If the labels cannot be improved, then the algorithm leaves the loop and has found the minimal cycle time.
- If the labels are improved, then the changed graph G_π consists again of several disjunctive circuits and for all nodes which are not in a circuit there exists again a path to a node in a circuit.

Then the while-loop starts again with examining all circuits in the graph G_π.

In the algorithm labels $d(u)$ for all $u \in T$ are computed. The labels are an estimation of the value of a longest path from u to a chosen node s (line 12) on a circuit with maximal value in the policy graph.

During the initialization of the algorithm the first policy graph is computed and the labels $d(u)$ for all $u \in T$ are initialized. The loops created during the initialization belong to the graph G and are defined by constraints (2.8). Thus, all circuits in the first policy graph have a positive delay and positive height.

In the second part, all circuits of the policy graph are examined. As after the initialization all circuits have a positive delay and positive height, the algorithm can find a circuit with the maximum value in the policy graph. If the algorithm finds a circuit with zero height (negative height and negative delay), then the Condition 3 (Inequalities (2.9)) of Theorem 2.1 is (are) violated. The reason for this is given after example.

After all circuits in G_π are examined, the circuit μ with the maximal value is chosen and the cycle time α is set to $\frac{L(\mu)}{H(\mu)}$. Note, as the policy graph is a graph in which each node has out-degree one, all circuits in G_π can be examined in linear time. The complexity of this step is $O(m)$. Afterwards, the policy graph is changed so that the circuit μ is the only circuit in the policy graph G_π (line 13 to 17) and for all other nodes which are not in the circuit μ there exists a path to a node of the circuit μ. The complexity of these steps is $O(nm)$.

The third part of the algorithm starts with computing the longest paths from all nodes $u \in T$ to s. During this step the labels $d(u)$ are updated (line 18). Note that if we delete the outgoing arc of the node s in G_π, we get an intree with root s. Thus, we can apply any longest path algorithm to compute the labels $d(u)$ for all $u \in T$. The weight of each arc $(u, \pi(u)) \in G_\pi$ in the longest path computation is defined by $L_{u\,\pi(u)} - \alpha H_{u\,\pi(u)}$. The complexity of these steps is $O(m)$.

After this computation, we check whether the labels $d(u)$ can be increased by inserting an arc $(u, v) \in E$ of the graph G with $\pi(u) \neq v$ into the arc set E_π. This can be done in $O(m)$. Let v be the node for which the equation

$$d(v) + L_{uv} - \alpha H_{uv} = max_{(u,v)\in E}\{d(v) + L_{uv} - \alpha H_{uv}\} \tag{2.19}$$

holds. Then $d(u)$ is updated to $d(u) = d(v)+L_{uv}-\alpha H_{uv}$ and $\pi(u)$ is updated to $\pi(u) = v$ if $d(u) < d(v) + L_{uv} - \alpha H_{uv} - \epsilon$ holds with $\epsilon > 0$. Thus, during the update procedure the labels $d(u)$ for all $u \in T$ cannot decrease.

Before analysing what happens if the policy graph does not change during the update procedure, we want to show by an example how the algorithm works.

Example 2.6 *We reuse the example 2.5. Thus, the data is given in Table 2.1 and the precedence graph is given in Figure 2.1.*

In Figure 2.2 the first policy graph G^1_π after the initialization is shown.

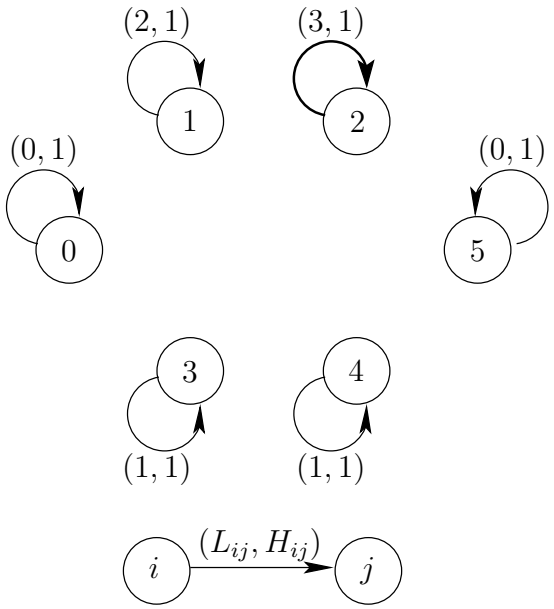

Figure 2.2: The policy graph G^1_π

The circuit μ with the maximum value in the graph G^1_π is $(2, 2)$. The value of this circuit is $\alpha = \frac{3}{1} = 3$. In all figures of this example the arcs of the chosen circuit μ with maximal

value are printed bold. In line 14 to 17 the policy graph is changed. The new policy graph G_π^2 is shown in Figure 2.3.

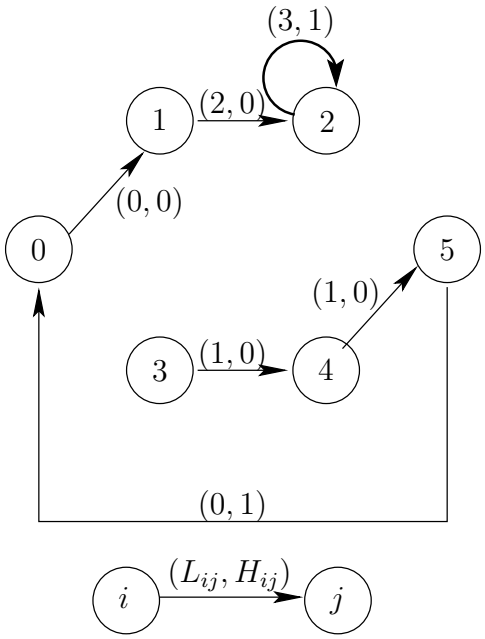

Figure 2.3: The policy graph G_π^2

Then the labels $d(u)$ are computed. These values are given in Table 2.2.

Node	*0*	*1*	*2*	*3*	*4*	*5*
Label	*2*	*2*	*0*	*1*	*0*	*-1*

Table 2.2: The labels $d(u)$ for G_π^2

Finally, in line 18 to 26 the policy graph is updated based on the computed labels. The new policy graph is given in Figure 2.4. This concludes the first iteration of Howard's algorithm.

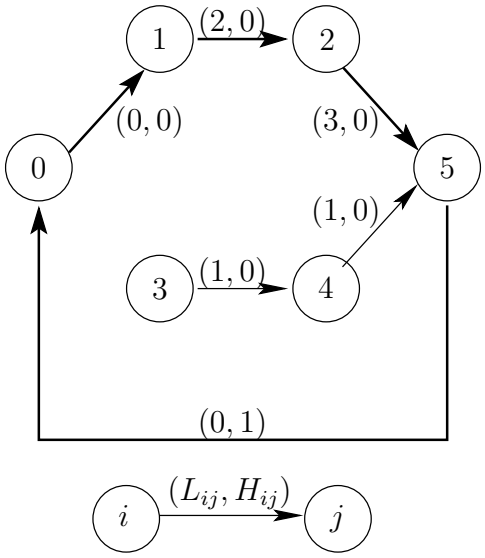

Figure 2.4: The policy graph G_π^3

Now the second iteration of Howard's algorithm starts. Again all circuits in the policy graph are analysed. The circuit μ *with the maximum value is* $(0, 1, 2, 5, 0)$*. The value of this circuit is* $\alpha = 5$*. As there exists only one circuit and for all nodes which are not in the circuit* μ *there exists a path to a node in* μ*, the graph* G_π^3 *is not changed in line 14 to 17. Finally, the labels* $d(u)$ *are updated. The new labels are given in Table 2.3.*

Node	*0*	*1*	*2*	*3*	*4*	*5*
Label	*0*	*0*	*-2*	*-3*	*-4*	*-5*

Table 2.3: The labels $d(u)$ for G_π^3

In line 18 to 26 the algorithm checks if the labels can be increased, but we cannot find an arc which can increase the label $d(u)$ *for a node. Thus, the algorithm has found the critical circuit* $(0, 1, 2, 5, 0)$ *and the optimal cycle time is* $\alpha = 5$*.*

Now we consider the case that the policy graph is not changed after the update procedure. Then the equation

$$d(u) = max_{(u,v)\in E}\{d(v) + L_{uv} - \alpha H_{uv}\} \tag{2.20}$$

holds for all nodes in G. Thus, there exists no circuit μ' with

$$\frac{L(\mu')}{H(\mu')} > \alpha + \frac{\epsilon}{H(\mu')} \text{ and } H(\mu') > 0 \tag{2.21}$$

because this is equivalent to

$$L(\mu') - \alpha H(\mu') > \epsilon$$

or

$$\sum_{(u,v)\in\mu'} (L_{uv} - \alpha H_{uv}) > \epsilon$$

or

$$\sum_{(u,v)\in\mu'} (d(u) - d(v) - L_{uv} + \alpha H_{uv}) < -\epsilon.$$

So, if there exists a circuit μ' with $\frac{L(\mu')}{H(\mu')} > \alpha + \frac{\epsilon}{H(\mu')}$ and $H(\mu') > 0$, then there exists at least one arc $(u, v) \in E$ which violates the equation (2.20).

With the same argument we can show that if there exists a circuit μ' with $\frac{L(\mu')}{H(\mu')} < \alpha + \frac{\epsilon}{H(\mu')}$ and $H(\mu') < 0$, there exists at least one arc in G for which equation (2.20) is not fulfilled.

Summarizing, the complexity of one iteration of the while-loop (line 5 to 26) is $O(nm)$.

These observations lead to the following lemma

Lemma 2.7 *If Howard's Algorithm terminates with a feasible solution, then α is the value of a critical circuit and the chosen circuit μ (line 10 in Listing 1) is a critical circuit.*

Furthermore, if during the third step a new arc is inserted into the policy graph and this arc creates a new circuit with positive height, then the cycle time α increases. This can be seen as follows. As a new arc (u, v) is inserted into the policy graph, the inequality $d(u) - d(v) - L_{uv} + \alpha H_{uv} < -\epsilon$ holds. If this new arc creates a new circuit μ' with positive height, we get

$$\sum_{(u,v)\in\mu'} (d(u) - d(v) - L_{uv} + \alpha H_{uv}) < -\epsilon$$

or

$$\frac{L(\mu')}{H(\mu')} > \alpha + \frac{\epsilon}{H(\mu')},$$

because for all other arcs (u', v') in G_π the equation $d(u) - d(v) - L_{uv} + \alpha H_{uv} = 0$ holds.

This leads to the following lemma

Lemma 2.8 *If in the third step of the algorithm a new circuit with positive height is created, then the cycle time α increases by at least $\frac{\epsilon}{H(\mu^+)}$, where $H(\mu^+)$ is the maximum height of all simple circuits in the graph G.*

Note that we call a circuit *simple* if no node is repeated in the circuit.

Now we can discuss the reasons why the algorithm terminates with an infeasible solution if a circuit μ' with height zero or negative height and negative delay exists. There exists at least one new arc $(u, v) \in \mu'$ which is inserted into the policy graph during the third part of the algorithm because after the initialization there exist only circuits with positive height and positive delay. Thus, $d(u) - d(v) - L_{uv} + \alpha H_{uv} < -\epsilon$ holds for at least one arc (u, v) of the detected circuit μ' during the third part of the algorithm. Otherwise, the algorithm would terminate. So, during the third part of the algorithm the inequality $\sum_{(u,v)\in\mu'}(d(u) - d(v) - L_{uv} + \alpha H_{uv}) < -\epsilon$ holds for the circuit μ' which is equivalent to

$$L(\mu') - \alpha H(\mu') > \epsilon$$

or

$$L(\mu') > \epsilon + \alpha H(\mu') \tag{2.22}$$

Therefore, if a circuit μ' with $H(\mu') = 0$ is found, then due to inequality (2.22)

$$L(\mu') > \epsilon > 0$$

holds. Thus, the circuit μ' has a positive delay and zero height. Otherwise, if a circuit μ' with $H(\mu') < 0$ is found, then, due to inequality (2.22),

$$\frac{L(\mu')}{H(\mu')} < \alpha + \frac{\epsilon}{H(\mu')}$$

holds. Thus, the circuit μ' has a lower value than α because $\frac{\epsilon}{H(\mu')} < 0$ and inequality (2.9) of Theorem 2.1 are violated. So there exists no feasible solution.

These observations lead to the following lemma

Lemma 2.9 *If Howard's Algorithm terminates, then either a feasible solution is found or a circuit which violates the conditions of Theorem 2.1 is found.*

Note that if Howard's Algorithm terminates with an infeasible solution, then the computed circuit μ (see line 8 in Listing 1) is one of the circuits which violates the conditions of Theorem 2.1.

The main advantages of using this algorithm to compute the optimal cycle time α compared to other algorithms are on the one hand the speed of the algorithm and on the other hand the returned circuit is either a critical or a forbidden circuit.

Now the only open question is whether the algorithm terminates in finite time. As we explained in the previous paragraphs, the value α increases or there exists no feasible solution, if a new arc creates a new circuit in the third part of the algorithm. Now the only open question is after how many iterations of the while-loop a new arc creates a new circuit. We show that after at most $n-1$ iterations a new circuit is created. This proof is taken from Dasdan et al. [23].

We have to consider the case where there are inserted new arcs into G_π but these arcs create no new circuit. We show that this can happen only $n-1$ times in a row. Let $G_\pi^1, \ldots, G_\pi^x$ be the sequence of policy graphs in which there is no new circuit with a new arc. All these graphs have exactly one circuit and in every graph, this is the same circuit. This also implies that the chosen node s in Line 12 to which the longest path is computed is always the same. The labels which are computed in the graph G_π^j are denoted with d^j. Each graph G_π^j, $j = 1, \ldots, x$, has exactly one circuit and for every node u there exists exactly one path to s.

We claim that each new path in G_π^k contains of at least k arcs. This implies that after computing G_π^n, either there is not any new arc in G_π^n, in this case the algorithm stops or there is a new arc in G_π^n, which leads to a path with at least n arcs. Thus, as the number of the nodes in the graph G_π equals to n, this path leads to a circuit with a new arc.

We prove our claim by contradiction. Therefore, we assign a number $n(u)$ to each node $u \in T$, which is the number of arcs in the path from u to s.

During the computation of G_π^j it happens the first time that for an arbitrary node y the successor $\pi(y)$ is reassigned from x to z with $n(z) < j-1$. Thus, we get a new path from y to s with $n(y) < j$. Let $d^j(y)$ and $d^j(z)$ be the labels for y and z at the time of the reassignment. We know due to the reassignment that

$$d^j(y) < d^j(z) - \alpha H_{yz} + L_{yz} \tag{2.23}$$

holds. Let $d^{j-1}(y)$ and $d^{j-1}(z)$ be the labels for y and z at the time when the arc (y, z) is considered during the computation of G_π^{j-1}. As our claim is violated during the computation of G_π^j the first time, the path from z to s cannot change during the computation of G_π^{j-1} because $n(z) < j-1$ holds and a new path in G_π^{j-1} must have at least $j-1$ arcs due to our claim.

Thus,

$$d^{j-1}(z) = d^j(z). \tag{2.24}$$

We also know that

$$d^{j-1}(y) \leq d^j(y) \tag{2.25}$$

since the labels cannot decrease during the algorithm. Substituting equation (2.24) into (2.23), we get

$$d^{j-1}(z) - \alpha H_{yz} + L_{yz} > d^j(y)$$

and with (2.25) we get

$$d^{j-1}(z) - \alpha H_{yz} + L_{yz} > d^j(y) \geq d^{j-1}(y),$$

which implies that during the computation of G_π^{j-1} the value $\pi(y)$ would be assigned to z and $d^{j-1}(y)$ would have been assigned to $d^{j-1}(z) - \alpha H_{yz} + L_{yz}$ which is strictly greater than $d^j(y)$. As the labels cannot decrease, this leads to a contradiction.

So now, we can present the main result of this section.

Theorem 2.10 *If there exists no feasible solution, then Howard's Algorithm finds one circuit in the graph G which does not fulfill the conditions of the Theorem 2.1. If there exists a feasible solution, then Howard's Algorithm computes the optimal cycle time α. Furthermore, the cycle time α increases by $\frac{\epsilon}{H(\mu^+)}$ at least every n iterations of the while loop of Howard's Algorithm, where $H(\mu^+) := max\{H(\mu) | \mu$ is a simple circuit in $G\}$.*

In Cochet-Terrasson et al. [18] it is proved for a special type of a GBCSP in which the height of all arcs equals to one, that the algorithm does not compute a policy graph once again. Therefore, the algorithm terminates in finite time and the running time is bounded by the number of different policy graphs. Thus, the complexity of the algorithm is $O(Nm)$, where N is the number of different policy graphs. Due to Theorem 2.10 we can derive the following two complexity results.

The complexity of one iteration of the while-loop is $O(nm)$. As the algorithm finds after at most n iterations of the while loop a new circuit, the while-loop is performed at most $O(nC)$ times, where C describes the number of simple circuits in the graph G. As the cycle time increases by $\frac{\epsilon}{H(\mu^+)}$ at most after n iterations, the while loop is performed at most $O(n\alpha^{opt}\frac{H(\mu^+)}{\epsilon})$. Thus, these results lead to the running time of $O(n^2mC)$ or $O(n^2m\alpha^{opt}\frac{H(\mu^+)}{\epsilon})$ to compute the optimal cycle time with Howard's Algorithm.

After computing the optimal cycle time α^{opt}, the only open question is how to compute the starting time of each occurrence of operation i for all $i \in T$. As the starting time of the k-th occurrence depends only on the starting time of the 0-th occurrence of operation i (see equation (2.1)), it is sufficient to compute the starting time $t_i = t(i;0)\ \forall\ i \in T$.

This is done in the following way. The starting time of the operation s is set to zero. Operation s is the chosen node in Howard's algorithm (see Listing 1 line 12). Then we compute the longest path from s to all other operations. This computation is performed in the graph $G_{\alpha_{opt}}$. The weight of an arc $(i, j) \in E$ in the graph $G_\alpha := (T, E)$ is defined by $a_\alpha := L_{ij} - \alpha H_{ij}$. The arc weight a_α is called **amplitude**. The value of the longest path from s to node i equals to the starting time of the 0-th occurrence of operation i.

Example 2.11 *Now we can compute the starting time for the operation in Example 2.5. The starting times are given in Table 2.4.*

Operation	*0*	*1*	*2*	*3*	*4*	*5*
Starting time	*0*	*0*	*2*	*0*	*1*	*5*

Table 2.4: The starting times for the operations

The schedule for this problem is given in Figure 2.5.

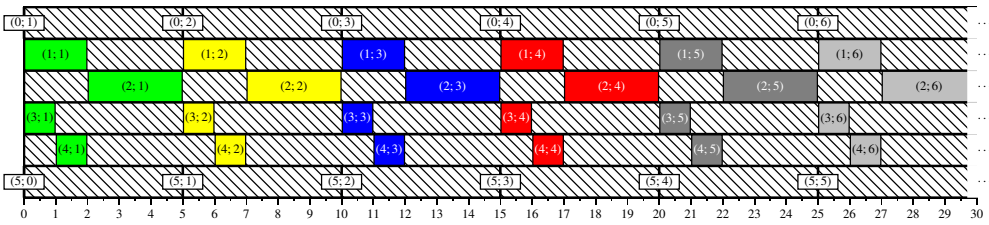

Figure 2.5: Optimal schedule for Example 2.11

In this subsection, we described Howard's Algorithm. As the algorithm can only be applied on strongly connected graphs, we show in the next section how to compute the optimal cycle time for an arbitrary graph G.

2.3 Several Strongly Connected Components

In this subsection, we consider an instance of the GBCSP, where the precedence constraints are given by an arbitrary graph G. Since Howard's algorithm can only be applied

to strongly connected graphs, we need to compute all strongly connected components $G^1, \ldots, G^n$ of the graph G.

If the graph has no strongly connected components, then the graph is acyclic and the cycle time has to be computed in a different way. This can still be done in polynomial time even if there exist resource constraints. Therefore, we assume that there exists at least one strongly connected component.

The computation of the components $G^1, \ldots, G^n$ can be done in $O(V + E)$. An algorithm with this running time is proposed by Tarjan [70].

After this computation we apply Howard's Algorithm on each strongly connected component G^j, $j = 1, \ldots, n$ and checks whether all conditions of Theorem 2.1 are fulfilled.

If the algorithm finds the optimal cycle time α^j for each component G^j, then, as the last step, we must compute the maximal cycle time $\alpha^{max} = max_{j=1}^{n} \alpha^j$ and check whether the cycle time α^{max} is still valid for all strongly connected components G^j, $j = 1, \ldots, n$ because now it can happen that the first inequality in (2.9) is violated.

In the remaining work we assume that the graph G is strongly connected.

2.4 The K-periodic Cyclic Scheduling Problem

In the cyclic scheduling literature we distinguish between the 1-periodic or periodic schedules and K-periodic schedules with $K > 1$ and $K \in \mathbb{N}$. In periodic schedules the difference between the starting time of two succeeding occurrences of operation i equals to the cycle time α. Thus, in any time interval with length α each operation is processed exactly once. In contrast to this, in a K-periodic schedule the difference between the starting times of the l-th occurrence and the $l + K$-th occurrence of operation i equals to the cycle time α_K. Therefore, in any time interval with length α_K each operation is processed exactly K times.

Closely related to K-periodic schedules are K-degree or K-part schedules. These kinds of schedules are usually considered in cyclic scheduling problems with one transportation robot (see e.g. Che et al. [14]). A K-degree schedule can be defined as follows: We must process one job j, which consists of several operations and K occurrences of the job j must start its processing and K occurrences of the job j must be finished in a time interval with length α_K. So, in each time interval each operation of the job j must be processed K times. Thus, if we consider all operations of a GBCSP as one job, the corresponding K-degree and K-periodic schedules describe the same schedule.

If we now compare the value of the objective function $\frac{\alpha_K}{K}$ for K-periodic schedules for all $K \in \mathbb{N}$ (including the value of the objective function $\frac{\alpha}{1}$ for all 1-periodic schedules), we can ask for which value of $K' \in \mathbb{N}$ the objective function $\frac{\alpha_{K'}}{K'}$ is minimal. The schedule for which the value is minimal is called a **dominant** schedule. It is known that the periodic schedules are the dominant schedules for a given instance of the BCSP.

In the following we present a very natural way to transform the uniform precedence constraints for the 1-periodic problem into constraints for the K-periodic problem. Furthermore, we show that if we compare the value α and $\frac{\alpha_K}{K}$, $\alpha \leq \frac{\alpha_K}{K}$ holds for all $K \in \mathbb{N} \setminus \{1\}$ and for any problem instance of the GBCSP. Therefore, the periodic schedules are the dominant schedules for any problem instance of the GBCSP.

Now we present a linear program for computing an optimal K-periodic schedule for a given graph G. In each time interval of length α_K each operation $i \in T$ has to be processed K times. Therefore, the operations $i_0, \ldots, i_{K-1}$ with $p_{i_l} = p_i$, $l = 0, \ldots, K-1$ must be processed once and the starting times of all operations must fulfill the following two constraints:

$$t(i_{l+1}; k) \geq t(i_l; k) \; \forall \; k \in \mathbb{Z}, l = 0, \ldots, K-2$$

and

$$t(i_0; k+1) \geq t(i_{K-1}; k) \; \forall \; k \in \mathbb{Z}.$$

These two constraints can be summarized by one constraint

$$t(i_q; k+p) \geq t(i_l; k) \; \forall \; k \in \mathbb{Z}, l = 0, \ldots, K-1,$$

where the values p and q are defined by the equation $l + 1 = pK + q$ with $p \in \mathbb{Z}$ and $q \in \{0, \ldots, K-1\}$ for all $l = 0, \ldots, K-1$. Thus, $p := \lfloor \frac{l+1}{K} \rfloor$ and $q := (l+1) \; mod \; K$.

Now consider the following precedence constraint between the operations i and j formulated for the 1-periodic case:

$$t(j; k + H_{ij}) \geq t(i; k) + L_{ij} \; \forall \; k \in \mathbb{Z} \tag{2.26}$$

As these constraints describe a relation between the different occurrences of the operations i and j, we have to adapt these constraints for the K-periodic problem because in any time interval with length α_K we process K different occurrences of each operation.

For the K-periodic problem we define the following constraints:

$$t(j_q; k+p) \geq t(i_l; k) + L_{ij} \; \forall \; k \in \mathbb{Z} \text{ and } l \in \{0, \ldots, K-1\}, \tag{2.27}$$

where the values p and q are defined by the equation $l + H_{ij} = pK + q$ with $p \in \mathbb{Z}$ and $q \in \{0, \ldots, K-1\}$ for all $l = 0, \ldots, K-1$ or $p := \left\lfloor \frac{l+H_{ij}}{K} \right\rfloor$ and $q := (l + H_{ij}) \; mod \; K$.

Note, if we set $K = 1$, we have only $l = 0$, and in this case we get $p = H_{ij}$ and $q = 0$. Thus, we have the original precedence constraints.

The linear program for computing the optimal K-periodic problem can be written as:

$$min\ \alpha_K \tag{2.28}$$

s.t.

$$t(i_l;k) = t(i_l;0) + k\alpha_K \quad i \in T, l = 0,\dots,K-1, k \in \mathbb{Z} \tag{2.29}$$

$$t(i_l;k+1) \geq t(i_l;k) + p_i \quad i \in T, l = 0,\dots,K-1, k \in \mathbb{Z} \tag{2.30}$$

$$t(i_q;k+p) \geq t(i_l;k) \quad i \in T, l = 0,\dots,K-1, k \in \mathbb{Z}, \; l+1 = pK+q, p \in \mathbb{Z}, \; q \in \{0,\dots,K-1\} \tag{2.31}$$

$$t(j_q;k+p) \geq t(i_l;k) + L_{ij} \quad (i,j) \in E, l = 0,\dots,K-1, k \in \mathbb{Z}, \; l+H_{ij} = pK+q, p \in \mathbb{Z}, \; q \in \{0,\dots,K-1\} \tag{2.32}$$

Now we describe the transformation of the precedence constraints additionally by an example.

Example 2.12 *Consider three operations i, j and k. The precedence constraints are given by a graph G which is shown in Figure 2.6.*

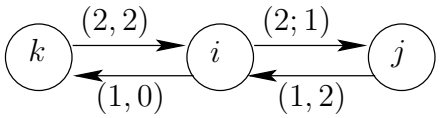

Figure 2.6: The graph G for Example 2.12

Now we want to derive the precedence constraints for a 4-periodic schedule. So we have to process in one period four occurrences of i, i.e. i_0, i_1, i_2 and i_3, four occurrences of j, i.e. j_0, j_1, j_2 and j_3, and four occurrences of k, i.e. k_0, k_1, k_2 and k_3.

Due to the first arc (i,j) with $L_{ij} = 2$ and $H_{ij} = 1$, the 0-th occurrence of operation i must start before the $0+1$-th occurrence of operation j. However, as there are processed four different occurrences of operation 2 in one period, it is clear that we get an arc between i_0 and j_1 with length 2 and height 0. This can also be seen together with Figure 2.7.

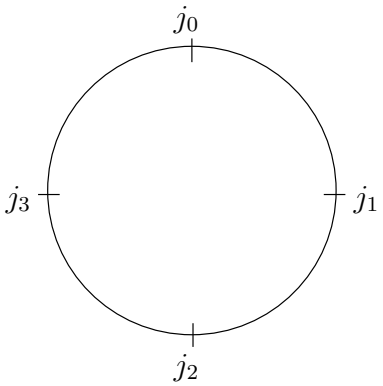

Figure 2.7: Duplications of operation j in a K-periodic schedule

We start at operation j_0 and go 1 step clockwise through the cycle. Thus, we stop at operation j_1. Thus, $q = 1$ and $p = 0$ hold because we do not pass the operation j_0. If we pass the operation j_0, we must increase the value p by one.

Consider now the 3-rd occurrence of operation i. The 3-rd occurrence of operation i must start before the $1+3$-th occurrence of operation j. So we start at operation j_3 in the cycle of Figure 2.7 and go again one step through the cycle. Thus, we stop at operation j_0. Therefore, $q = 0$ and $p = 1$ hold because we pass the operation j_0 once.

Summarizing we get the following arcs:

- (i_0, j_1) *with* $L_{i_0\,j_2} = 2$ *and* $H_{i_0\,j_2} = 0$ *because of* $p = \left\lfloor \frac{1}{4} \right\rfloor = 0$ *and* $q = 1 \; mod \; 4 = 1$,
- (i_1, j_2) *with* $L_{i_1\,j_0} = 2$ *and* $H_{i_1\,j_0} = 0$ *because of* $p = \left\lfloor \frac{1+1}{4} \right\rfloor = 0$ *and* $q = (1 + 1) \; mod \; 4 = 2$,
- (i_2, j_3) *with* $L_{i_2\,j_1} = 2$ *and* $H_{i_2\,j_1} = 0$ *because of* $p = \left\lfloor \frac{2+1}{4} \right\rfloor = 0$ *and* $q = (2 + 1) \; mod \; 4 = 3$,
- (i_3, j_0) *with* $L_{i_2\,j_1} = 2$ *and* $H_{i_2\,j_1} = 1$ *because of* $p = \left\lfloor \frac{3+1}{4} \right\rfloor = 1$ *and* $q = (3 + 1) \; mod \; 4 = 0$,
- *etc.*

The corresponding graph G^4 is shown in Figure 2.8. Note only the height values greater than zero are shown in the graph G^4.

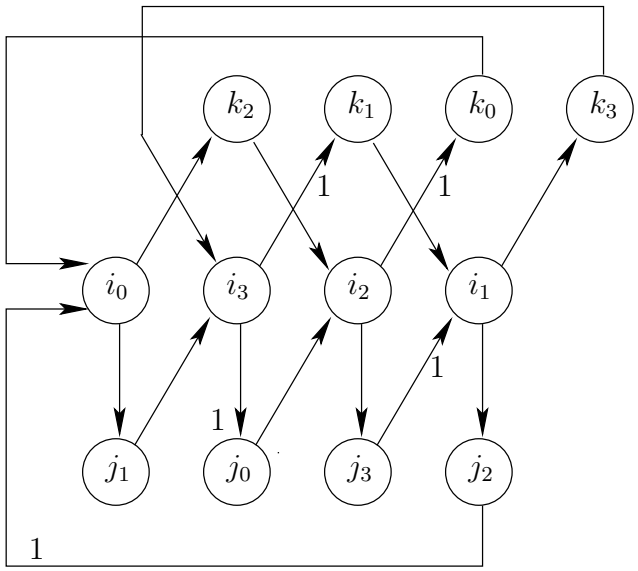

Figure 2.8: The graph G^4 for Example 2.12

Now we analyse the circuits in the graph G^4. There exist three simple circuits. The circuit $(i_0, j_1, i_3, j_0, i_2, j_3, i_1, j_2, i_0)$ is derived from the circuit (i, j, i) in G. The value of this circuit is $\frac{4*L(\mu)}{H(\mu)}$. The circuit $(i_0, k_2, i_2, k_0, i_0)$ and $(i_1, k_3, i_3, k_1, i_1)$ are derived from the circuit (i, k, i). The value of this circuit is $\frac{2*L(\mu)}{H(\mu)/2}$.

If we compare the value α and $\alpha^4/4$, we get

$$\max\{3/3, 3/2\} = \alpha = \alpha^4/4 = \max\{12/3, 6/1\}/4 = \max\{12/12, 6/4\}.$$

In the following we show that each simple circuit in the precedence graph for the 1-periodic problem has a direct counterpart in the graph for the K-periodic problem. We denoted the graph of the K-periodic problem with G^K, the length of a circuit μ in G^K with $L^K(\mu)$ and the height of a circuit μ in G^K with $H^K(\mu)$.

First, we recall some results for the floor and the modulo function.

Lemma 2.13 *For $x \in \mathbb{R}$ and $k \in \mathbb{Z}$ the equation*

$$\lfloor x \pm k \rfloor = \lfloor x \rfloor \pm \lfloor k \rfloor$$

holds with $\lfloor x \rfloor \in \mathbb{Z}$.

Definition 2.14 *The modulo function is defined as follows*

$$a \; mod \; m = a - \left\lfloor \frac{a}{m} \right\rfloor m.$$

Now we can show the following result:

Lemma 2.15 *For a given integer number $n \geq 1$ the equation*

$$((\ldots(((((a_1 \; mod \; m) + a_2) \; mod \; m) + a_3) \; mod \; m) + \ldots + a_n) \; mod \; m)$$
$$= (a_1 + a_2 + \ldots + a_n) \; mod \; m$$

holds for all $a_1, \ldots, a_n \in \mathbb{R}$ and $m \in \mathbb{N}$, $m > 1$.

Proof: In the following we show that the equation holds for $n = 2$.

$$\begin{aligned}
((a_1 \; mod \; m) + a_2) \; mod \; m &= a_1 \; mod \; m + a_2 - \left\lfloor \frac{a_1 \; mod \; m + a_2}{m} \right\rfloor m \\
&= a_1 - \left\lfloor \frac{a_1}{m} \right\rfloor m + a_2 - \left\lfloor \frac{a_1 + a_2 - \left\lfloor \frac{a_1}{m} \right\rfloor m}{m} \right\rfloor m \\
&= a_1 + a_2 - \left\lfloor \frac{a_1}{m} \right\rfloor m - \left\lfloor \frac{a_1 + a_2}{m} \right\rfloor m + \left\lfloor \frac{a_1}{m} \right\rfloor m \\
&= (a_1 + a_2) \; mod \; m
\end{aligned}$$

With induction we can easily show that the equation also holds for any integer number n. □

Now we show that each simple circuit μ in G with height $H(\mu) = mK$ with $m \in \mathbb{Z}$, and length $L(\mu)$ creates K simple circuits in G^K with height m and length $L(\mu)$.

Lemma 2.16 *If there exists a simple circuit $\mu = (i^0, \ldots, i^n, i^0)$ in G with $L(\mu) = \sum_{(i,j)\in\mu} L_{ij}$ and $H(\mu) = \sum_{(i,j)\in\mu} H_{ij} = mK$ with $m \in \mathbb{Z}$, then there exist K simple circuits $\mu_l = (i^0_l, i^1_{q_1}, \ldots, i^n_{q_n}, i^0_{q_0} = i^0_l)$ with $L^K(\mu) = L(\mu)$ in G^K and $H^K(\mu) = m$ for $l = 0, \ldots, K-1$.*

Proof: The circuit μ_l consists of the nodes $(i_l^0, i_{q_1}^1, \ldots, i_{q_n}^n, i_l^0)$ with the arcs
$(i_l^0, i_{q_1}^1)$ with $q_1 = (l + H_{i^0 i^1})\ mod\ K$ and height $p_0 = \left\lfloor \frac{l+H_{i^0 i^1}}{K} \right\rfloor$,
$(i_{q_1}^1, i_{q_2}^2)$ with $q_2 = (q_1 + H_{i^1 i^2})\ mod\ K$ and height $p_1 = \left\lfloor \frac{q_1+H_{i^1 i^2}}{K} \right\rfloor$,
...
and
$(i_{q_n}^n, i_{q_0}^0)$ with $q_0 = (q_n + H_{i^n i^0})\ mod\ K$ and height $p_n = \left\lfloor \frac{q_n+H_{i^n i^0}}{K} \right\rfloor$.

If now $q_0 = l$ holds, we find a circuit μ_l in G^K with $L^K(\mu) = L(\mu)$ and $H^K(\mu) = \sum_{t=0}^{n} p_t$.

As $q_t = (q_{t-1} + H_{i^{t-1} i^t})\ mod\ K$ for $t = 1, \ldots, n$ holds, the equation

$$q_0 = (q_n + H_{i^n i^0})\ mod\ K$$

can be rewritten to

$$q_0 = (((\ldots(((l+H_{i^0 i^1})\ mod\ K)+H_{i^1 i^2})\ mod\ K+\ldots+H_{i^{n-1} i^n})\ mod\ K)+H_{i^n i^0})\ mod\ K$$

or

$$q_0 = (l + H_{i^0 i^1} + H_{i^1 i^2} + \ldots + H_{i^{n-1} i^n} + H_{i^n i^0})\ mod\ K$$

because of Lemma 2.15. Because of Definition 2.14 and $H(\mu) = mK$, this is equivalent to

$$q_0 = (l + H(\mu))\ mod\ K = l + H(\mu) - \left\lfloor \frac{l + H(\mu)}{K} \right\rfloor K = l + H(\mu) - mK = l$$

for each $l = 0, \ldots, K-1$. Thus, we found K circuits in G^K.

As $q_t = (q_{t-1} + H_{i^{t-1} i^t})\ mod\ K$ for $t = 1, \ldots, n$ holds, we can rewrite

$$p_s = \left\lfloor \frac{q_s + H_{i^s i^{s+1}}}{K} \right\rfloor$$

for $s = 1, \ldots, n$ with $i^{n+1} = i^0$ to

$$p_s = \left\lfloor \frac{((q_{s-1} + H_{i^{s-1} i^s})\ mod\ K) + H_{i^s i^{s+1}}}{K} \right\rfloor$$

$$= \left\lfloor \frac{((\ldots(((l + H_{i^0 i^1})\ mod\ K) + H_{i^1 i^2}) + \ldots + H_{i^{s-1} i^s})\ mod\ K) + H_{i^s i^{s+1}}}{K} \right\rfloor.$$

With Lemma 2.15 and Definition 2.14 this equation is equivalent to

$$p_s = \left\lfloor \frac{(l + H_{i^0 i^1} + H_{i^1 i^2} + \ldots + H_{i^{s-1} i^s}) \; mod \; K + H_{i^s i^{s+1}}}{K} \right\rfloor$$

$$= \left\lfloor \frac{l + H_{i^0 i^1} + \ldots + H_{i^s i^{s+1}} - \left\lfloor \frac{l + H_{i^0 i^1} + \ldots + H_{i^{s-1} i^s}}{K} \right\rfloor K}{K} \right\rfloor$$

$$= \left\lfloor \frac{l + H_{i^0 i^1} + \ldots + H_{i^s i^{s+1}}}{K} \right\rfloor - \left\lfloor \frac{l + H_{i^0 i^1} + \ldots + H_{i^{s-1} i^s}}{K} \right\rfloor$$

Thus,

$$\begin{aligned}
H^K(\mu) &= \sum_{t=0}^{n} p_t \\
&= p_0 + p_1 + \ldots + p_n \\
&= \left\lfloor \frac{l + H_{i^0 i^1}}{K} \right\rfloor + \left\lfloor \frac{l + H_{i^0 i^1} + H_{i^1 i^2}}{K} \right\rfloor - \left\lfloor \frac{l + H_{i^0 i^1}}{K} \right\rfloor + \ldots \\
&\quad + \left\lfloor \frac{l + H_{i^0 i^1} + \ldots + H_{i^n i^0}}{K} \right\rfloor - \left\lfloor \frac{l + H_{i^0 i^1} + \ldots + H_{i^{n-1} i^n}}{K} \right\rfloor \\
&= \left\lfloor \frac{l + H_{i^0 i^1} + \ldots + H_{i^n i^0}}{K} \right\rfloor \\
&= \left\lfloor \frac{l + H(\mu)}{K} \right\rfloor \\
&= \frac{H(\mu)}{K} = m
\end{aligned}$$

As the properties hold for all circuits μ_l, $l = 0, \ldots, K-1$, we have shown that there exist K circuits in G^K with $L^K(\mu) = L(\mu)$ and $H^K(\mu) = m$. □

Now we show that all other simple circuits μ in G with height $H(\mu)$ and length $L(\mu)$ create $GCD(|H(\mu)|, K)$ simple circuits in G^K with height $\frac{H(\mu)}{GCD(|H(\mu)|,K)}$ and length $L(\mu) \cdot \frac{K}{GCD(|H(\mu)|,K)}$.

Lemma 2.17 *If there exists a simple circuit $\mu = (i^0, \ldots, i^n, i^0)$ in G with the following properties*

1. $L(\mu) = \sum_{(i,j)\in\mu} L_{ij}$ *and*

2. *there exists no* $m \in \mathbb{Z}$ *with* $H(\mu) = \sum_{(i,j)\in\mu} H_{ij} = mK$,

then there exists a simple circuit $\mu_0 = (i_0^0 = i_{q_0^0}^0, i_{q_1^0}^1, \ldots, i_{q_n^0}^n, i_{q_0^1}^0, i_{q_1^1}^1, \ldots, i_{q_n^{g-1}}^n, i_{q_0^g}^0 = i_0^0)$ *with* $g = \frac{K}{GCD(|H(\mu)|,K)}$. *The length of* μ_0 *is* $L(\mu_0) = \frac{K}{GCD(|H(\mu)|,K)} \cdot L(\mu)$ *and the height is* $H(\mu_0) = \frac{H(\mu)}{GCD(|H(\mu)|,K)}$.

Furthermore, we can find $GCD(|H(\mu)|, K) - 1$ *other simple circuits with the same length and height.*

Proof: Due to the computation of the q-values in the proof of Lemma 2.16, we can conclude that

$$q_0^1 = (H_{i^0 i^1} + \ldots H_{i^n i^0}) \; mod \; K = H(\mu) \; mod \; K$$

holds. As there exists no $m \in \mathbb{Z}$ with $H(\mu) = mK$, $q_0^1 \neq 0$ holds. This result holds for the values q_0^r, $r = 0, \ldots, g-1$, where $g = \frac{K}{GCD(|H(\mu)|,K)}$.

If $r = g$, we get

$$q_0^r = q_0^g = \frac{K}{GCD(|H(\mu)|, K)} \cdot H(\mu) \; mod \; K$$

or

$$q_0^g = \frac{K}{GCD(|H(\mu)|, K)} \cdot H(\mu) - \left\lfloor \frac{K \cdot H(\mu)}{GCD(|H(\mu)|, K) \cdot K} \right\rfloor K = 0.$$

Thus, we found a circuit in G^K. The height of the circuit μ_0 is

$$H(\mu_0) = \left\lfloor \frac{\frac{K}{GCD(|H(\mu)|,K)} \cdot H(\mu)}{K} \right\rfloor = \frac{H(\mu)}{GCD(|H(\mu)|, K)}.$$

Each arc (i, j) in G creates K arcs with the same length in G^K. As $q_0^g =$ holds, in the circuit μ^0 there are g different copies of the node i^0. This result holds also for the other nodes in μ. Thus, g different copies of all arcs in μ are visited in μ^0. Therefore, $L(\mu^0) = gL(\mu) = \frac{K}{GCD(|H(\mu)|,K)} L(\mu)$ holds.

If $\frac{K}{GCD(|H(\mu)|,K)} \neq K$ holds, then not all copies of the node i^0 are in the circuit μ^0. Thus, we can find a copy i_l^0 of the node i^0 which is not in μ^0. So starting with i_l^0 we can find another circuit μ^1 with $L(\mu^1) = \frac{K}{GCD(|H(\mu)|,K)} L(\mu)$ and $H(\mu_0) = \frac{H(\mu)}{GCD(|H(\mu)|,K)}$. If $2\frac{K}{GCD(|H(\mu)|,K)} \neq K$ holds, then we can find again a copy of i^0 which is not in the circuit μ^0 and μ^1. This copy creates again a circuit etc. Thus, we can find $GCD(|H(\mu)|, K)$

different circuits μ^l with $L(\mu_l) = \frac{K}{GCD(|H(\mu)|,K)}L(\mu)$ and $H(\mu_l) = \frac{H(\mu)}{GCD(|H(\mu)|,K)}$ for all $l = 0, \ldots, GCD(|H(\mu)|, K) - 1$. □

Now we can present the main result of this section.

Theorem 2.18 *A critical circuit μ in G with value $\frac{L(\mu)}{H(\mu)} = \alpha$ leads to $GCD(|H(\mu)|, K)$ circuits μ_l with value $\alpha_K = \frac{K \cdot L(\mu)}{H(\mu)}$ for all $l = 0, \ldots, GCD(|H(\mu)|, K) - 1$.*

Thus,

$$\alpha \leq \frac{\alpha_K}{K}.$$

Proof: If $H(\mu) = mK$ with $m \in \mathbb{Z}$ holds, we can apply Lemma 2.16. Thus, the circuit μ creates $K = GCD(|H(\mu)|, K)$ circuits with length $L(\mu) = \frac{K}{GCD(|H(\mu)|,K)}L(\mu)$ and height $m = \frac{H(\mu)}{GCD(|H(\mu)|,K)}$.

If there exists no $m \in \mathbb{Z}$ with $H(\mu) = mK$, we can apply Lemma 2.17. Thus, we get $GCD(|H(\mu)|, K)$ circuits with length $\frac{K}{GCD(|H(\mu)|,K)}L(\mu)$ and height $\frac{H(\mu)}{GCD(|H(\mu)|,K)}$. Therefore, the value of the critical circuit α_K in G^K must be greater or equal to

$$\alpha_K \geq \left(\frac{K}{GCD(|H(\mu)|, K)}L(\mu)\right) \Big/ \left(\frac{H(\mu)}{GCD(|H(\mu)|, K)}\right) = \frac{K \cdot L(\mu)}{H(\mu)}.$$

Thus,

$$\alpha \leq \frac{\alpha_K}{K}$$

holds. □

To sum up, in the section we proved that for the GBCSP the periodic schedules are the dominant schedules.

3 The General Cyclic Machine Scheduling Problem

In this section, first we extend the GBCSP of Section 2 with disjunctive resource constraints. This extension is due to Hanen [30]. This idea generalizes the approach to model classical non-cyclic scheduling problems with disjunctive constraints (see e.g. Balas [4]). In the second part, we briefly discuss cyclic K-periodic scheduling problems with disjunctive resource constraints. This discussion is also due to Hanen [30]. In the third part of this section, we propose several different extensions. In the last part, we present a new complexity result.

3.1 The Basic Cyclic Machine Scheduling Problem

The basic cyclic machine scheduling problem (BCMSP) is an extension of the GBCSP. We get this problem by adding resource constraints. We introduce disjunctive resource constraints as follows:

Associated with each operation $i \in T$, there is a dedicated machine $M(i) \in M = \{1, \ldots, m\}$, on which each occurrence $< i; k >$ of i must be processed. Occurrences of different operations to be processed on the same machine cannot overlap. The set of operations which are processed on machine m is denoted by $T_m \subset T$.

Therefore, for all occurrences k and l with $k, l \in \mathbb{Z}$ of pairs of operations i and j which are processed on the same machine, we have to add the following constraints:

$$t(i;k) + p_i \leq t(j;l) \vee t(j:l) + p_j \leq t(i;k). \tag{3.1}$$

Due to this constraint, we have to define an order between the k-th occurrence of operation i and the l-th occurrence of operation j.

Thus, after extending the linear program for the GBCSP (2.6) to (2.8) by the constraints (3.1) the basic cyclic machine scheduling problem can be written as:

$$min \ \alpha \tag{3.2}$$

s.t.

$$t(i;k) = t(i;0) + \alpha k \qquad i \in T, k \in \mathbb{Z} \tag{3.3}$$

$$t(i;k+1) \leq t(i;k) + p_i \qquad i \in T, k \in \mathbb{Z} \tag{3.4}$$

$$t(i;k) + L_{ij} \leq t(j;k + H_{ij}) \qquad (i,j) \in E, k \in \mathbb{Z} \tag{3.5}$$

$$t(i;k) + p_i \leq t(j;l) \ \vee \ t(j;l) + p_j \leq t(i;k) \quad i, j \in T \text{ with } i \neq j \text{ and } M(i) = M(j), k, l \in \mathbb{Z} \tag{3.6}$$

Hanen [30] showed the following theorem.

Theorem 3.1 *The problem* (3.2) *to* (3.6) *is equivalent to the following mixed integer linear program* (3.7) *to* (3.12).

$$\min \ \alpha \tag{3.7}$$

s.t.

$$t_j - t_i \geq L_{ij} - \alpha H_{ij} \quad (i,j) \in E \tag{3.8}$$

$$t_j - t_i \geq p_i - \alpha H X_{ij} \quad i,j \in T \text{ with } i \neq j \text{ and } M(i) = M(j) \tag{3.9}$$

$$H X_{ij} + H X_{ji} = 1 \quad i,j \in T \text{ with } i \neq j \text{ and } M(i) = M(j) \tag{3.10}$$

$$H X_{ij} \in \mathbb{Z} \quad i,j \in T \text{ with } i \neq j \text{ and } M(i) = M(j) \tag{3.11}$$

$$p_i \leq \alpha \quad i \in T \tag{3.12}$$

If we compare the mixed integer linear program (3.7) to (3.12) with the linear program for the GBCSP (2.6) to (2.8) we can easily see that if we fixe the HX-variables, then the problem to compute the optimal cycle time is reduced to the problem of solving the corresponding instance of the GBCSP.

Furthermore, we can easily derive two lower bounds for this problem. We get the first lower bound α^-_{GBCSP} by relaxing the resource constraints, therefore, the first lower bound equals to the optimal solution of the corresponding GBCSP. The second lower bound α^-_{Mach} is defined by $\alpha^-_{Mach} = max_{m \in M} \sum_{i \in T_m} p_i$. Thus, this bound gives the time which is needed to process all operations once on the bottleneck machine.
Thus, $\alpha^- = \max\{\alpha^-_{GBCSP}, \alpha^-_{Mach}\}$ is a lower bound, too.

Now we want to extend the Example 2.11 with resource constraints.

Example 3.2 *The assignment of the operations to the machines is given in Table 3.1.*

Operation	*0*	*1*	*2*	*3*	*4*	*5*
Machine	*0*	*1*	*2*	*1*	*2*	*3*

Table 3.1: The assignment of the operations to the machines

The graph with the disjunctive and conjunctive arcs is given in Figure 3.1.

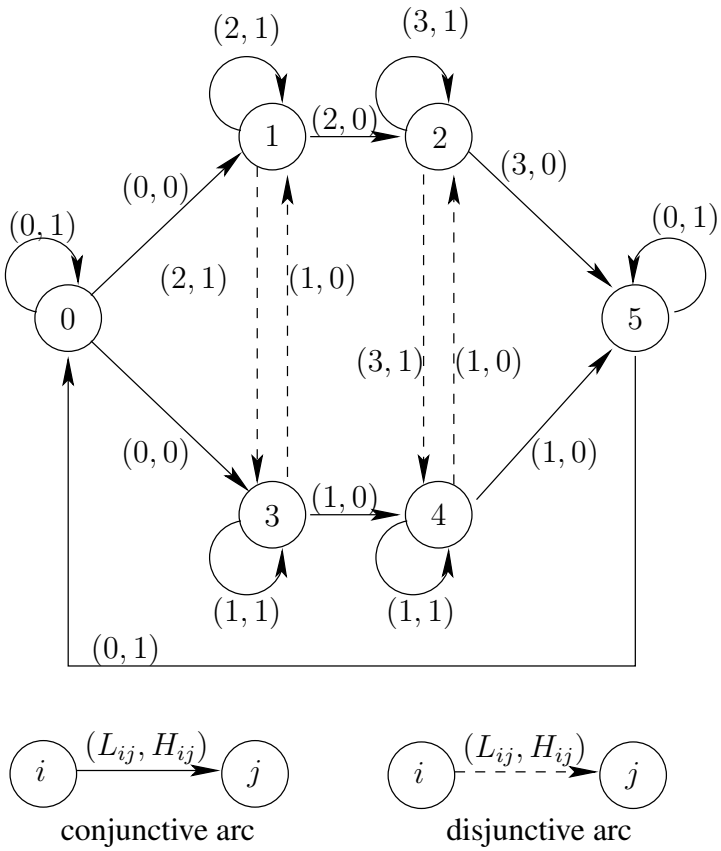

Figure 3.1: The graph G for Example 3.2

The optimal solution with cycle time $\alpha = 6$ is given in Figure 3.2.

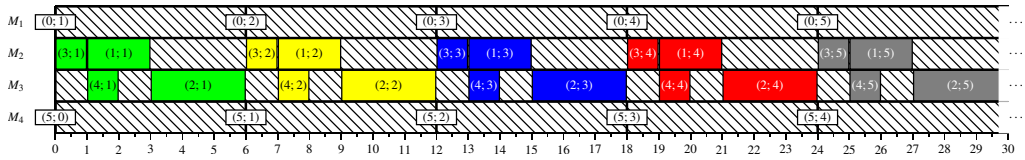

Figure 3.2: Optimal schedule for Example 3.2

3.2 Cyclic K-periodic Scheduling Problems

In Section 2.4 we show that for the GBCSP the periodic schedules are the dominant schedules. The same result cannot be shown for the BCMSP and therefore, for all its extension as the following example shows, which is taken from Hanen [30].

Example 3.3 *Consider the following instance of a BCMSP problem. The instance consists of 4 operations and 3 machines. The data of this problem is given in Table 3.2. The graph with the uniform precedence constraints for the 1-periodic schedule is given in Figure 3.3.*

Operation	*1*	*2*	*3*	*4*
Processing time	*2*	*3*	*2*	*2*
Machine	*1*	*2*	*1*	*3*

Table 3.2: Data for the 1-periodic schedule in Example 3.3

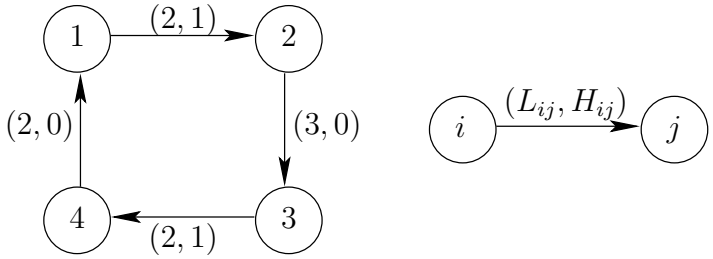

Figure 3.3: Precedence constraints for the Example 3.3

To find the optimal 2-periodic schedule, we have to solve the following periodic problem. The data is given in Table 3.3 and the graph with the precedence constraints is given in Figure 3.4.

Operation	1_1	1_2	2_1	2_2	3_1	3_2	4_1	4_2
Processing time	*2*	*2*	*3*	*3*	*2*	*2*	*2*	*2*
Machine	*1*	*1*	*2*	*2*	*1*	*1*	*3*	*3*

Table 3.3: Data for the 2-periodic schedule in Example 3.3

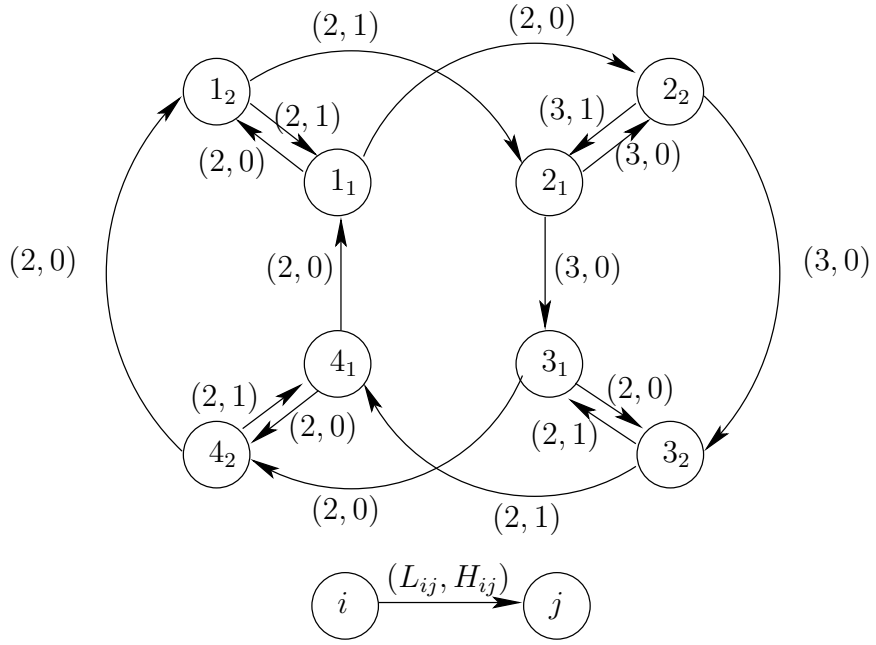

Figure 3.4: Precedence constraints for the Example 3.3

The graph in Figure 3.4 is derived from the graph in Figure 3.3 as follows: First, all operations are duplicated as the graph in Figure 3.4 describes the precedence constraints for a 2-periodic problem. Second, the given precedence constraints are adapted to the 2-periodic schedule as it is described in Section 2.4. For example in a 2-periodic problem the precedence constraint $t(2; k+1) \geq t(1; k) + p_1$ *for the periodic problem leads to the following precedence constraints: For* $l = 0$ *we get* $0 + H_{12} = 1 = 2p + q$ *with* $p := \lfloor \frac{1}{2} \rfloor = 0$ *and* $q := 1 \ mod \ 2 = 1$*. Therefore, we get the precedence constraint* $t(2_q; k+p) = t(2_1; k) \geq t(1; k) + p_1$*. This constraint leads to the following arc* $(1_0, 2_1)$

with $L_{1_0\,2_1} = 2$ *and* $H_{1_0\,2_1} = 0$. *If we consider* $l = 1$, *we get the following arc* $(1_1, 2_0)$ *with* $L_{1_1\,2_0} = 2$ *and* $H_{1_1\,2_0} = 1$.

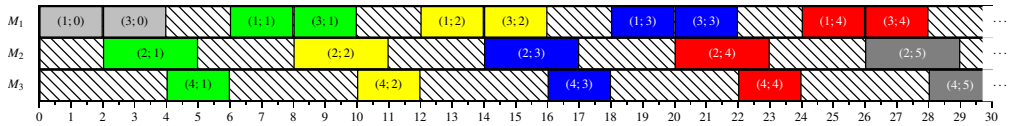

Figure 3.5: Optimal Schedule for the 1-periodic problem with $\alpha = 6$

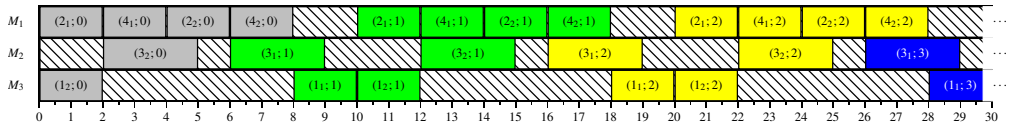

Figure 3.6: Optimal Schedule for the 2-periodic problem with $\frac{\alpha}{2} = \frac{10}{2} = 5$

The optimal schedule for the 1-periodic problem is given in Figure 3.5. The optimal schedule for the 2-periodic schedule is given in Figure 3.6. As we can see, the 2-periodic schedule leads to a better schedule because $\frac{10}{2} < \frac{6}{1}$.

Thus, the K-periodic schedules can lead to better solutions.

In the rest of this work we consider only periodic problems because any K-periodic problem can be reformulated to a periodic problem as described in Example 3.3 and Section 2.4. All proposed solution methods could also be applied to the reformulated periodic problem. The only disadvantage of this reformulation is that the problem size increases by the factor of K.

3.3 Some Extensions

Now we present some minor extensions of the basic cyclic machine scheduling problem.

The first extension concerns the delay between two operations i and j. With the constraints

$$t(j; k + H_{ij}) \geq t(i; k) + L_{ij} \text{ or } t(j; k + H_{ij}) - t(i; k) \geq L_{ij}$$

with $L_{ij} \in \mathbb{Q}$, $H_{ij} \in \mathbb{Z}$ we define a minimal delay of L_{ij} between the starting time of the k-th occurrence of operation i and the $k + H_{ij}$-th occurrence of operation j. Now we

want to present a way to define a maximal delay L_{ij}^{max} with $L_{ij}^{max} - L_{ij} \geq 0$ between the k-th occurrence of operation i and the $k + H_{ij}$-th occurrence of operation j. So we get

$$L_{ij}^{max} \geq t(j; k + H_{ij}) - t(i; k) = t(j; k) - t(i; k - H_{ij})$$

or

$$t(i; k - H_{ij}) - t(j; k) \geq -L_{ij}^{max}.$$

With $H_{ji} := -H_{ij}$ and $L_{ji} := -L_{ij}^{max}$ we get

$$t(i; k + H_{ji}) - t(j; k) \geq L_{ji}.$$

Thus, we need to introduce a directed arc between j and i with $L_{ji} = -L_{ij}^{max} \leq -L_{ij}$ and $H_{ji} = -H_{ij}$. Note that these arcs (i, j) and (j, i) create a circuit with zero height ($H_{ji}+H_{ij} = -H_{ij}+H_{ij} = 0$) and non-positive delay ($L_{ji}+L_{ij} \leq 0$). Due to Theorem 2.1, this circuit is an allowed circuit. Therefore, if we add these constraints to a feasible solution of a general basic cyclic scheduling problem, we can still compute a feasible solution, perhaps with another optimal cycle time α.

The second extension concerns a restriction on the latest starting time of an operation in a period. This can be achieved by using similar constraints as in the first extension. To define the starting time of an operation in a period, we need to define the start and the end of a period. This can be done by introducing two dummy nodes, the source node 0 and the sink node $\star$, which are both processed alone on a dummy machine. The source node is connected with each operation $i \in T$ with some delay and zero height. Furthermore, each operation $i \in T$ is connected with the sink node with height zero. The delay of these arcs is equal to the processing time of the considered operation i. To get again a strongly connected graph, we must connect the sink node with the source node with delay 0 and height n, where n is the number of nodes in the graph. As we show in Section 3.4, this additional arc does not add any new restriction to the problem. Thus, each occurrence of a source node defines the start of a new period, and the end of a period is defined by an occurrence of the sink node. With these two nodes, we can now restrict the starting time of an operation in a period. Assume the minimal starting time of an operation i in a period is r_i, then we must create an arc from the source node 0 to i with delay r_i and height zero because the following constraint must be fulfilled:

$$t(i; k) - t(0; k) \geq r_i.$$

Symmetrically, if the latest starting time of an operation i in a period is d_i with $d_i \geq r_i$, then the constraint

$$t(0; k) + d_i \geq t(i; k) \text{ or } t(0; k) - t(i; k) \geq -d_i$$

must be fulfilled. Thus, this constraint leads to an arc between i and 0 with delay $-d_i$ and height zero. Note that these arcs $(0, i)$ and $(i, 0)$ also create an allowed circuit. In the non-cyclic scheduling the minimal starting time r_i describes a release date and the maximal starting time $d_i + p_i$ describes a due date. Therefore, these concepts from the non-cyclic scheduling can be easily generalized for cyclic scheduling problems.

With the third extension we can model an upper bound $\alpha^+ \geq \alpha$ on the optimal cycle time α. As each operation is repeated every α time units, we can introduce a directed arc from an arbitrary operation i to itself with height $H_{ii} = -1$ and delay $L_{ii} = -\alpha^+$, which leads to

$$t(i; k-1) \geq t(i; k) - \alpha^+ \text{ or } \alpha^+ \geq t(i; k) - t(i; k-1) = \alpha.$$

In the following we consider again the Example 3.2.

Example 3.4 *We want to restrict the time between the start of the processing of the k-th occurrence of operation 1 and the end of the processing of the k-th occurrence of operation 2 to 14. Thus,*

$$t(i; k) + 14 \geq t(2; k) + 3$$

or

$$t(i; k) \geq t(2; k) - 11$$

must hold. Note, if we consider operations 1 and 2 as a job, then this constraint can be seen as a restriction on the total processing time of the job. Therefore, we add an arc $(2, 1)$ with $L_{21} = -11$ and $H_{21} = 0$ to the graph. The new graph is given in Figure 3.7. Additionally to introducing the arc $(2, 1)$, the height of the arc $(5, 0)$ is increased to 2.

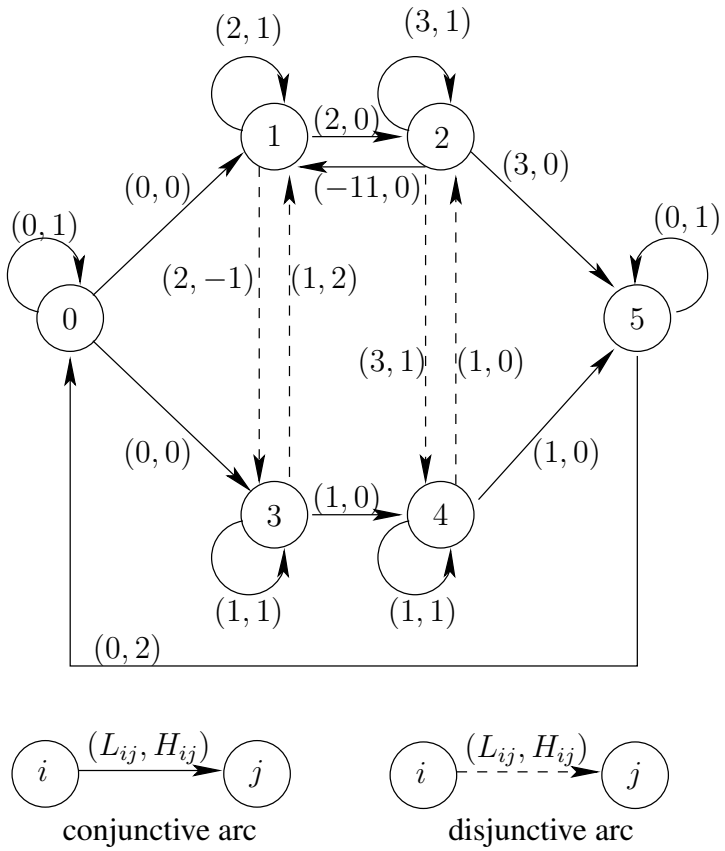

Figure 3.7: Precedence constraints for Example 3.4

We can easily see that all circuits fulfill the conditions of Theorem 2.1. The circuit $(2, 1, 3, 4, 2)$ *has a negative delay and negative height. The circuit* $(1, 2, 1)$ *has a negative delay and zero height. All other circuits have a positive delay and positive height. The critical circuit is* $(1, 3, 4, 2, 5, 0, 1)$*. The cycle time is* $\alpha = 7$*.*

The last extension, which we want to present in this subsection, is the introduction of multiprocessor tasks, or short MPT for cyclic scheduling problems. Up to now we only allow that operations are processed on a dedicated machine.

This restriction is now generalized. We consider a cyclic machine scheduling problem with m machines and n operations. Each operation i requires during its processing time p_i all machines belonging to a predefined subset $\mu_i \subseteq \{M_1, \ldots, M_m\}$. If two operations i

and j require the same machine, then two operations cannot be processed simultaneously. Thus, if the intersection of the machine sets μ_i, μ_j for two operations i and j is non-empty, $\mu_i \cap \mu_j \neq \emptyset$, then both operations cannot be processed simultaneously. So, we can rewrite the disjunctive resource constraint (3.1) to

$$t(i;k) + p_i \leq t(j;k) \vee t(j;l) + p_j \leq t(i;k) \tag{3.13}$$

for all $i, j \in T$ with $\mu_i \cap \mu_j \neq \emptyset$ and $k, l \in \mathbb{Z}$.

We can easily verify that Theorem 3.1 still holds, if we replace (3.6) by (3.13).

In this subsection, we presented some easy extensions, which are needed to model various applications from the literature, which are presented in Section 6.

3.4 Some Complexity Results

In this subsection, we give first a new complexity results for a special case of the general basic cyclic machine scheduling problem and finally, we present a complexity result by Hanen [30] for the general basic cyclic machine scheduling problem.

We can show for an instance of a BCMSP that if the graph G described by the precedence constraints consists only of one circuit μ with $H(\mu) \geq n$, where n is the number of operations in the instance, then the optimal cycle time α is

$$\alpha_{opt} = \max_{j=1}^{m} \sum_{i \in M(j)} p_j, \tag{3.14}$$

if $L_{ij} = p_i$ for all $(i, j) \in E$. The graph G is given in Figure 3.8.

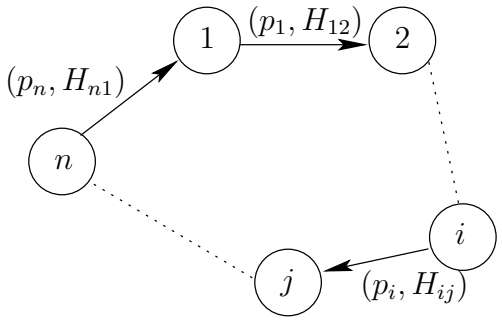

Figure 3.8: The graph G with one circuit μ with $H(\mu) \geq n$

This can be seen in the following way. As α_{opt} equals to the lower bound α_{Mach} of the problem, the solution would be optimal if all precedence constraints are fulfilled. We can assume that the starting time of the 0-th occurrence of operation 1 on machine $M(1)$ equals to 0. All other operations are scheduled once in an arbitrary order on their machines without creating any idle time between the operations. So, we fixed the starting time for one occurrence of each operation. Therefore, the starting time of the other occurrences of each operation can be computed easily.

Now the only open question is if all precedence constraints are fulfilled. To check this, we have to assign to each occurrence an occurrence number. The only operation which has already an occurrence number is operation 1. Furthermore, each occurrence of operation 1 defines the start of a new period with length α. In each period, all operations are processed once. Thus, the start of period P_l equals to $t(1;l)$. Due to the described scheduling of the operations, each operation which is started in period P_l is also finished in period P_l. Therefore, the succeeding operation 2 of operation 1 which is started in period P_1 gets the occurrence number $0 + H_{12}$. With this occurrence number the precedence constraint between both operations is fulfilled. The occurrence number of the succeeding operation 3 of operation 2, which is processed in period P_2, is $0 + H_{12} + H_{23}$. Thus, the precedence constraint between 2 and 3 is also fulfilled. This assignment is repeated until we reaches the predecessor operation n of 1.

The occurrence number of operation n which is performed in the period P_{n-1} is $H(\tau)$, where $H(\tau)$ is the height of the path from 1 to n. So, we have to check if the precedence constraint between n and 1 is fulfilled. The occurrence number of operation 1 in period P_n is n. Thus,

$$t(1;n) \geq t(n;H(\tau)) + p_n$$

holds. As the height of the circuit μ is greater or equal to n, we get

$$t(1;H(\mu)) \geq t(1;n) \geq t(n;H(\tau)) + p_n.$$

So, all precedence constraints between all operations are fulfilled. Thus, we have constructed a feasible schedule with cycle time α_{opt}.

Summarizing, we can say that introducing an arc with height greater or equal to the number of operations leads to no new restriction on the problem.

The following complexity result is shown by Hanen [30].

Theorem 3.5 *If the graph consists of one circuit with arbitrary height h, then the decision problem finding a feasible periodic schedule with cycle time $\alpha \leq \alpha^+$ where α^+ is a given rational number, is $\mathcal{NP}$-complete.*

4 The Cyclic Job Shop Scheduling Problem With Blocking

In this section, we first model the cyclic job shop problem with blocking. This is done by adapting the alternative graph model by Mascis and Pacciarelli [50]. Another approach to model blocking is given in Levner [49] and Pinedo [58]. Afterwards we establish a mixed integer linear program which is similar to the mixed integer linear program (3.7) to (3.12).

In accordance with the definitions in Section 2.1 the cyclic job shop problem can be defined as follows.

Let $J = \{1, \ldots, n\}$ be a set of jobs. Each job $j \in J$ consists of n_j operations $O_{1j}, O_{2j}, \ldots, O_{n_j\, j}$. For $l = 1, \ldots, n_j - 1$ operation $O_{l+1\, j}$ is called the successor of operation O_{lj}. O_{1j} and $O_{n_j\, j}$ are the first and last operations of job j, respectively. The set of all operations is denoted by T. Each operation $i \in T$ has a processing time $p_i > 0$ and belongs to the job $j(i)$. The set of all operations of job j is denoted by $O(j)$. $s(i)$ denotes the **succeeding operation** of i, if it exists. All operations $i \in T$ must be performed infinitely often.

We denoted by $< i; k >$ the k-th occurrence of operation i. A **schedule** assigns a starting time $t(i; k)$ to each occurrence $< i; k >$. It is called **periodic** with cycle time α if

$$t(i; k) = t(i; 0) + \alpha k \text{ for all } i \in T,\ k \in \mathbb{Z}.$$

We define $t_i := t(i; 0) \geq 0$ for all $i \in T$. Then a periodic schedule is defined by the vector $(t_i)_{i \in T}$ and the cycle time $\alpha \geq 0$.

Additionally, we assume that there exist uniform precedence constraints of the form

$$t(i; k) + L_{i\, s(i)} \leq t(s(i); k + H_{i\, s(i)})$$

between operation i and the succeeding operation $s(i)$ of i, if a succeeding operation exists. The **delay** $L_{i\, s(i)}$ is set to $L_{i\, s(i)} = p_i$. The **height** is set to $H_{i\, s(i)} = 0$.

Furthermore, we introduce a source node 0 and a sink node $\star$. We also introduce uniform precedence constraints between 0 and the first operations of all jobs $j \in J$. The delay and the height of these constraints starting at the source node 0 is zero. Additionally, we introduce uniform precedence constraints between the last operations i of all jobs and $\star$. The delay of these arcs is p_i and the height is zero. We add a precedence constraint between $\star$ and 0 with delay 0 and height $H_{\star 0}$.

These uniform precedence constraints define the arc set of the directed graph $G = (T, E)$.

We again postulate that $t(i;k) + p_i \leq t(i;k+1)$ is satisfied for all $i \in T$, $k \in \mathbb{Z}$ and these constraints are also added to the graph G. So, the length of these loops is p_i and the height is 1.

Associated with each operation i there is a dedicated machine $M(i) \in M = \{1, \ldots, m\}$, on which each occurrence $< i; k >$ of operation i must be processed. The set of operations which are processed on machine m is denoted by $T_m \subset T$.

Occurrences of different operations to be processed on the same machine cannot overlap. In the presence of blocking restrictions, two blocking operations i and i' which are both processed on machine $m = M(i) = M(i')$, must stay on machine m until the succeeding operation $s(i)$ and $s(i')$ can start on the machines $M(s(i))$ and $M(s(i'))$, respectively. Assume that the l-th occurrence of operation i is scheduled before the k-th occurrence of operation i' on machine $M(i')$. Then, the k-th occurrence of operation i' can start processing on machine $M(i')$ if the l-th occurrence of the succeeding operation $s(i)$ is started on its machine. So, we get the following inequalities:

$$t(i;l) + p_i \leq t(s(i);l) \leq t(i';k) \tag{4.1}$$

If the k-th occurrence of operation i' is scheduled before the l-th occurrence of i, we get:

$$t(i';k) + p_{i'} \leq t(s(i');k) \leq t(i;l) \tag{4.2}$$

Therefore, we have the following disjunctive constraints:

$$t(s(i);l) \leq t(i';k) \vee t(s(i');k) \leq t(i;l) \tag{4.3}$$

Not all operations must be blocking operations. Therefore, we distinguish between three different cases. In the first case both operations i and i' are blocking operations, then we get the disjunctive constraint (4.3). In the second case only one operation is a blocking operation, assume operation i is this operation, then we have the following disjunctive constraints:

$$t(s(i);l) \leq t(i';k) \vee t(i';k) + p_{i'} \leq t(i;l) \tag{4.4}$$

In the last case both operations are non-blocking operations, then we have the disjunctive constraints

$$t(i;l) + p_i \leq t(i';k) \vee t(i';k) + p_{i'} \leq t(i;l) \tag{4.5}$$

The last operations of jobs, i.e. operations i with $s(i) = \star$ are never blocking operations. We can define the following two variables

$$b(i) := \begin{cases} s(i) & \text{if } i \text{ is a blocking operation and } s(i) \neq \star \\ i & \text{otherwise} \end{cases}$$

and

$$p_i^b := \begin{cases} 0 & \text{if } i \text{ is a blocking operation and } s(i) \neq \star \\ p_i & \text{otherwise.} \end{cases}$$

With these variables we can summarize the disjunctive constraints (4.3), (4.4) and (4.5) to

$$t(b(i);l) + p_i^b \leq t(i';k) \vee t(b(i');k) + p_{i'}^b \leq t(i;l) \tag{4.6}$$

for all $k, l \in \mathbb{Z}$ and for all operation i, i' with $M(i) = M(i')$.

Notice that by definition of $b(i)$ and p_i^b we have

$$t(i;k) + p_i \leq t(b(i);k) + p_i^b. \tag{4.7}$$

Furthermore, if operation i is a blocking operation, then the next occurrence of operation i can only start if the succeeding operation $s(i)$ of operation i is started. If i is a non-blocking operation, then due to the introduced loops the $k+1$-th occurrence of operation i can start right after the k-th occurrence of operation i is finished. Thus, we get

$$t(b(i);k) + p_i^b \leq t(i;k+1) \tag{4.8}$$

for all $i \in T$ and $k \in \mathbb{Z}$.

The cycle time minimization problem for cyclic job shop problems with blocking constraints can be written as:

$$\min \ \alpha \tag{4.9}$$

s.t.

$$t(i;k) = t(i;0) + \alpha k \qquad i \in T, k \in \mathbb{Z} \tag{4.10}$$

$$t(i;k) + p_i \leq t(i';k + H_{ii'}) \qquad (i,i') \in E, k \in \mathbb{Z} \tag{4.11}$$

$$t(b(i);k) + p_i^b \leq t(i;k+1) \qquad i \in T, k \in \mathbb{Z} \tag{4.12}$$

$$\begin{aligned} t(b(i);l) + p_i^b &\leq t(i';k) \\ \vee\, t(b(i');k) + p_{i'}^b &\leq t(i;l) \end{aligned} \qquad \begin{aligned} &k, l \in \mathbb{Z}, i, i' \in T, i \neq i', \\ &\text{with } M(i) = M(i') \end{aligned} \tag{4.13}$$

Theorem 4.1 *The following mixed integer linear program*

$$\min\ \alpha \tag{4.14}$$

s.t.

$$t_{i'} - t_i \geq p_i - \alpha H_{ii'} \qquad (i,i') \in E \tag{4.15}$$
$$t_{i'} - t_{b(i)} \geq p_i^b - \alpha K_{b(i)i'} \qquad i,i' \in T, i \neq i', M(i) = M(i') \tag{4.16}$$
$$K_{b(i)i'} + K_{b(i')i} = 1 \qquad i,i' \in T, i \neq i', M(i) = M(i') \tag{4.17}$$
$$K_{b(i)i'}, K_{b(i')i} \in \mathbb{Z} \qquad i,i' \in T, i \neq i', M(i) = M(i') \tag{4.18}$$

is equivalent to (4.9) *to* (4.13)

Proof: First, we show that the constraints (4.13) are equivalent to constraints (4.16)-(4.18). By substituting (4.10) into (4.13) we get

$$t_{b(i)} + p_i^b \leq t_{i'} + \alpha(k-l)\ \vee\ t_{b(i')} + p_{i'}^b \leq t_i + \alpha(l-k).$$

As

$$t_i + p_i\ \leq\ t_{b(i)} \leq t_{b(i)} + p_i^b$$

and

$$t_{i'} + p_{i'}\ \leq\ t_{b(i')} \leq t_{b(i')} + p_{i'}^b$$

we get

$$t_i + p_i\ \leq t_{b(i)} + p_i^b \leq\ t_{i'} + \alpha(k-l) \vee \tag{4.19}$$
$$t_{i'} + p_{i'}\ \leq t_{b(i')} + p_{i'}^b \leq\ t_i + \alpha(l-k) \tag{4.20}$$

If we multiply the inequalities (4.19) by -1, we have the inequalities

$$-t_i - p_i\ \geq -t_{b(i)} - p_i^b \geq\ -t_{i'} - \alpha(k-l) \vee$$
$$t_{i'} + p_{i'}\ \leq t_{b(i')} + p_{i'}^b \leq\ t_i + \alpha(l-k)$$

which are equivalent to

$$t_{i'} - t_i\ \geq t_{i'} - t_{b(i)} - p_i^b + p_i\ \geq p_i - \alpha(k-l) \vee \tag{4.21}$$
$$t_{i'} - t_i\ \leq t_{b(i')} - t_i + p_{i'}^b - p_{i'}\ \leq -p_{i'} + \alpha(l-k) \tag{4.22}$$

Thus, the difference between $t_{i'}$ and t_i does not lie in the following intervals

$$...,]-p_{i'}-\alpha m, p_i - \alpha m[,]-p_{i'}-\alpha(m-1), p_i-\alpha(m-1)[,...,$$

where $m := k - l$. So $t_{i'} - t_i$ lies in the following intervals

$$..., [p_i - \alpha m, -p_{i'} - \alpha(m-1)], ... \tag{4.23}$$

Note that this interval is not empty because $\alpha > \alpha^- = \max\{\alpha^-_{GBCSP}, \alpha^-_{Mach}\}$. Thus, we get

$$p_i - \alpha m \leq t_{i'} - t_i \leq -p_{i'} - \alpha(m-1).$$

Due to (4.19) and (4.20) we have

$$\begin{aligned} & t_{i'} - t_{b(i)} \geq p^b_i - \alpha m \\ \text{and} \quad & t_i - t_{b(i')} \geq p^b_{i'} + \alpha m \geq p^b_{i'} - \alpha(1-m). \end{aligned}$$

With $K_{b(i)i'} := m$ and $K_{b(i')i} := 1 - K_{b(i)i'}$ we get

$$\begin{aligned} & t_{i'} - t_{b(i)} \geq p^b_i - \alpha K_{b(i)i'} \\ \text{and} \quad & t_i - t_{b(i')} \geq p^b_{i'} - \alpha K_{b(i')i} \end{aligned}$$

Finally, we show that constraints (4.11) are equivalent to constraints (4.15). By substituting (4.10) into (4.11) we get

$$\begin{aligned} t_{i'} + \alpha H_{ii'} &\geq t_i + p_i \text{ or} \\ t_{i'} - t_i &\geq p_i - \alpha H_{ii'} \end{aligned}$$

□

If there exists a pair of disjunctive arcs between $(b(i), i')$ and $(b(i'), i)$, then we call $(b(i'), i)$ the **alternative arc** of $(b(i), i')$ and vice versa.

Again, we want to illustrate the blocking situation with an example.

Example 4.2 *We consider again the Example 3.2. The first job consists of the operations 1 and 2 and the second job consists of the operations 3 and 4.*

The first operations of each job are blocking operations. The graph for the optimal solution is given in Figure 4.1.

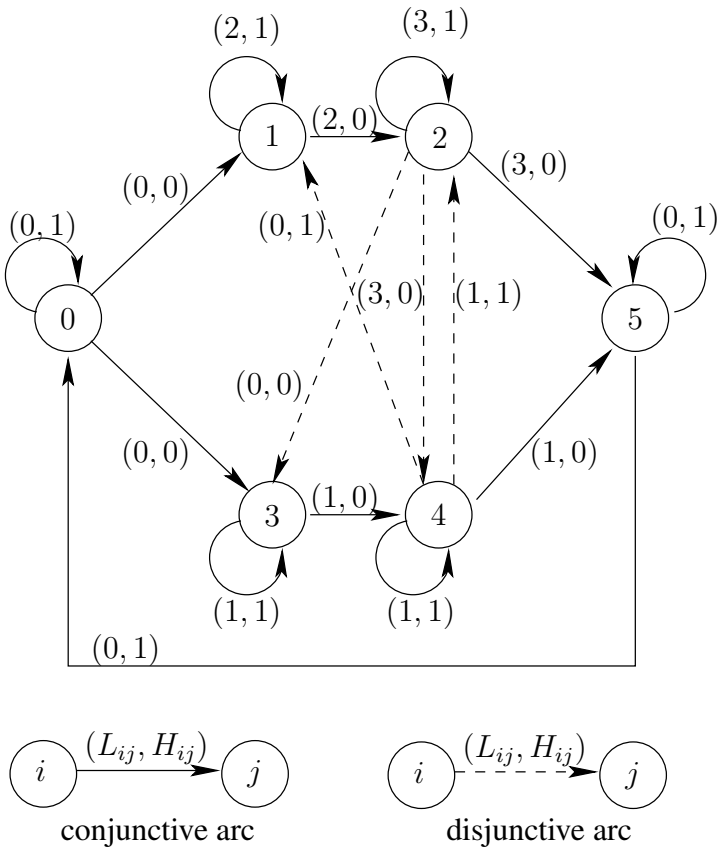

Figure 4.1: The graph for Example 4.2

The optimal solution with cycle time $\alpha = 6$ is given in Figure 4.2.

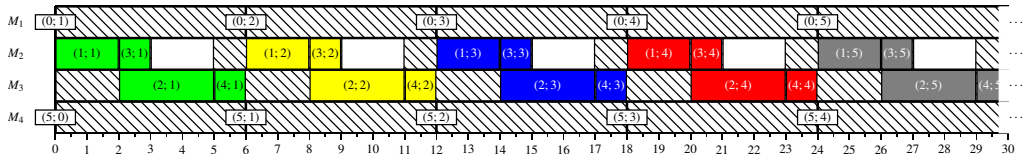

Figure 4.2: Optimal schedule for Example 4.2

5 Cyclic Scheduling Problem With Linear Precedence Constraints

In this section, we introduce a different concept of periodic schedules. Up to now each operation has the same cycle time α. This constraint is now generalized. This concept is first introduced by Munier [54]. This section is divided into two parts. In the first subsection, we describe the linear precedence constraints for problems without any resource constraints. In the second subsection, we extend the results of the first section to problems with resource constraints.

5.1 Basic Cyclic Scheduling Problems With Linear Precedence Constraints

We denote the cycle time of operation i by w_i. We call a schedule **periodic** if

$$t(i;k) = t(i;0) + kw_i \tag{5.1}$$

for all $i \in T$ and all $k \in \mathbb{Z}$.

In the first part of this section, we introduce the problem and show that we can reuse the algorithms for solving the GBCSP (see Section 2) to solve this problem. In the second part, we introduce disjunctive resource constraints and show that we can also derive a mixed integer linear program, which is quite similar to the program (3.7) to (3.12). We also develop a new lower bound for this problem.

Linear precedence constraints of the form

$$t(i; p_{ij}k + q_{ij}) + L_{ij} \leq t(j; p'_{ij}k + q'_{ij}) \tag{5.2}$$

for all $k \in \mathbb{Z}$ may be given for all arcs $(i,j) \in E$ of a directed graph $G = (T, E)$ with vertex set T. L_{ij} is again called (start-start) **delay** and is again a rational number. In addition, the following properties are satisfied: p_{ij}, p'_{ij} are positive integers and q_{ij}, q'_{ij} are integer values.

Note that the linear precedence constraints are a generalization of the uniform precedence constraints because by setting p_{ij} and p'_{ij} to one, we get the uniform precedence constraints introduced in Section 2.

We also assume that the precedence constraints are **unitary**. This means that the graph G is strongly connected and all cycles c of G have the weight $\pi(c) = 1$, whereas the weight of an arc (i,j) is $\pi_{ij} = \frac{p'_{ij}}{p_{ij}}$ and the weight of a path μ is $\pi(\mu) = \Pi_{(i,j)\in\mu}\pi_{ij}$.

The aim is to minimize simultaneously the cycle time w_i for all operations $i \in T$.

In the first part, we show that

$$w_i = \alpha W_i \text{ with } \alpha = \frac{w_1}{\beta}$$

for $i \in T$ with constants W_i and β depending on the values π_{1i} hold, and that (5.2) can be written in the form

$$t_j - t_i \geq L_{ij} - \alpha H_{ij}$$

where the values H_{ij} are integer constants depending on the values π_{1i}.

Thus, the problem of finding a periodic schedule that minimizes the w_i-values subject to the constraints (5.2) can be reduced to the following problem which is called General Basic Cyclic Scheduling Problem (GBCSP)

$$\min \ \alpha \tag{5.3}$$

s.t.

$$t_j - t_i \ \geq \ L_{ij} - \alpha H_{ij} \quad \forall \ (i,j) \in E \tag{5.4}$$

The reduction to the GBCSP is due to [32]. Here we recall the main results of this paper.

As we are looking for a periodic schedule, we can show the following Lemma.

Lemma 5.1 *A periodic schedule meets the linear precedence constraints, if and only if the following inequality holds for any arc $(i,j) \in E$:*

$$t_j - t_i \geq L_{ij} + (w_i p_{ij} - w_j p'_{ij})k + w_i q_{ij} - w_j q'_{ij} \ \forall \ k \in \mathbb{Z}. \tag{5.5}$$

Proof: A schedule fulfills the linear precedence constraints (5.2) if and only if

$$\begin{aligned} t(i;0) + (p_{ij}k + q_{ij})w_i + L_{ij} \leq t(j;0) + (p'_{ij}k + q'_{ij})w_j \text{ or} \\ t_j - t_i \geq L_{ij} + (w_i p_{ij} - w_j p'_{ij})k + w_i q_{ij} - w_j q'_{ij} \end{aligned} \tag{5.6}$$

□

A direct conclusion of this Lemma is that $w_i p_{ij} - w_j p'_{ij} = 0$ because the inequality (5.5) must be true for all $k \in \mathbb{Z}$. Thus, condition (5.5) is equivalent to the following two conditions

$$t_j - t_i \geq L_{ij} + w_i q_{ij} - w_j q'_{ij} \tag{5.7}$$

and

$$w_i p_{ij} - w_j p'_{ij} = 0. \tag{5.8}$$

Note, that (5.8) is equivalent to

$$\pi_{ij} := \frac{w_i}{w_j} = \frac{p'_{ij}}{p_{ij}}. \tag{5.9}$$

Lemma 5.2 *The linear precedence constraints*

$$t_j - t_i \geq L_{ij} + w_i q_{ij} - w_j q'_{ij}$$

for any arc $(i,j) \in E$ can be rewritten to

$$t_j - t_i \geq L_{ij} - \alpha H_{ij}$$

*with integer H_{ij}. H_{ij} is called **height** of the arc (i,j).*

Proof: As G is unitary, all paths $\mu : 1 =: i_1, i_2, \ldots, i_r := i$ from 1 to i have the same weight $\rho_i = \Pi_{l=1}^{r-1} \pi_{i_l i_{l+1}} = \Pi_{l=1}^{r-1} \frac{w_{i_l}}{w_{i_{l+1}}} = \frac{w_1}{w_i} = \frac{p'_{1i}}{p_{1i}}$.

Now we can rewrite inequality (5.7) for any arc $(i,j) \in E$ to

$$\begin{aligned} t_j - t_i &\geq L_{ij} + w_i q_{ij} - w_j q'_{ij} && (5.10)\\ &= L_{ij} + \frac{1}{\rho_i} w_1 q_{ij} - \frac{1}{\rho_j} w_1 q'_{ij} && (5.11)\\ &= L_{ij} - w_1 (\frac{1}{\rho_j} q'_{ij} - \frac{1}{\rho_i} q_{ij}) && (5.12) \end{aligned}$$

If we rewrite the rational numbers $\rho_i = \frac{w_1}{w_i} = \frac{p'_{1i}}{p_{1i}}$ as $\rho_i = \frac{\beta_i}{\gamma_i}$ with $GCD(\beta_i, \gamma_i) = 1$ and define $\beta := LCM(\beta_1, \ldots, \beta_n)$, then

$$w_1 (\frac{1}{\rho_j} q'_{ij} - \frac{1}{\rho_i} q_{ij}) = \frac{w_1}{\beta} \beta (\frac{\gamma_j}{\beta_j} q'_{ij} - \frac{\gamma_i}{\beta_i} q_{ij}) = \alpha H_{ij}$$

with $\alpha := \frac{w_1}{\beta}$ and $H_{ij} := \beta(\frac{\gamma_j}{\beta_j} q'_{ij} - \frac{\gamma_i}{\beta_i} q_{ij}) \in \mathbb{Z}$. □

A consequence of the proof of Lemma 5.2 is that we can rewrite the cycle time w_i for all operations $i \in T$ by

$$w_i = \frac{w_1}{\rho_i} = \alpha \frac{\beta}{\rho_i} = \alpha W_i$$

with $W_i := \frac{\beta}{\rho_i} \in \mathbb{N}$.

So, we get the following value for the height of an arc (i, j):

$$H_{ij} = W_j q'_{ij} - W_i q_{ij} \tag{5.13}$$

Therefore, we can consider only problems with $W_i \in \mathbb{N} \forall\, i \in T$.

Furthermore, by minimizing α we minimize simultaneously the cycle time for all operations $i \in T$ because the values W_i depend on the values p_{1i} and p'_{1i} only.

The BLCP, which is described by (5.3) to (5.4), can be solved by using the same methods that can be used for solving the GBCSP. Furthermore, there exists only a solution for the problem if the graph G is consistent.

5.2 A Cyclic Scheduling Problem With Linear Precedence Constraints and Resource Constraints (CLSP)

In this part of the section, the BLCP is extended by disjunctive resource constraints.

Associated with each operation i there is a dedicated machine $M(i) \in M = \{1, \ldots, m\}$, on which each occurrence $< i; k >$ of i must be processed. Occurrences of different operations to be processed on the same machine cannot overlap.

Therefore, for all occurrences of pairs of operations i and j which are processed on the same machine we have to add the constraints

$$t(i;k) + p_i \leq t(j;l) \vee t(j:l) + p_j \leq t(i;k) \tag{5.14}$$

for all $k, l \in \mathbb{Z}$.

Thus, the general cycle time minimization problem for linear precedence constraints can be written as:

$$min\ \ \alpha \tag{5.15}$$

s.t.

$$t(i;k) = t(i;0) + \alpha k W_i \qquad i \in T, k \in \mathbb{Z} \tag{5.16}$$

$$t(i; p_{ij}k + q_{ij}) + L_{ij} \leq t(j; p'_{ij}k + q'_{ij}) \qquad (i,j) \in E, k \in \mathbb{Z} \tag{5.17}$$

$$t(i;k) + p_i \leq t(j;l) \ \vee\ t(j;l) + p_j \leq t(i;k) \qquad i, j \in T \text{ with } i \neq j \text{ and } M(i) = M(j), k, l \in \mathbb{Z} \tag{5.18}$$

Now we want to show that this problem is equivalent to a mixed integer linear program. In the proof of this equivalence, we use results which are based on the *Extended Euclidean Algorithm*. The Extended Euclidean Algorithm is presented in Theorem 5.3. A proof can be found e.g. in [64].

Theorem 5.3 ***Extended Euclidean Algorithm*** *Let $a \in \mathbb{N}_0, b \in \mathbb{N}$. The greatest common divisor $GCD(a,b)$ can be written as linear combination of a and b.*

$$GCD(a,b) = u \cdot a + v \cdot b, \text{ with } u, v \in \mathbb{Z}$$

Now we describe the main result of the second part:

Theorem 5.4 *The problem* (5.15) *to* (5.18) *is equivalent to the following mixed integer linear program* (5.19) *to* (5.23)*.*

$$\min \ \alpha \tag{5.19}$$

s.t.

$$t_j - t_i \geq L_{ij} - \alpha H_{ij} \qquad (i,j) \in E \tag{5.20}$$

$$t_j - t_i \geq p_i - \alpha K_{ij} \cdot GCD(W_i, W_j) \qquad i,j \in T \text{ with } i \neq j \text{ and } M(i) = M(j) \tag{5.21}$$

$$K_{ij} + K_{ji} = 1 \qquad i,j \in T \text{ with } i \neq j \text{ and } M(i) = M(j) \tag{5.22}$$

$$K_{ij} \in \mathbb{Z} \qquad i,j \in T \text{ with } i \neq j \text{ and } M(i) = M(j) \tag{5.23}$$

Proof: By substituting (5.16) into (5.17) we get

$$t_i + \alpha W_i(p_{ij}k + q_{ij}) + L_{ij} \leq t_j + \alpha W_j(p'_{ij}k + q'_{ij})$$

$$\Leftrightarrow t_j - t_i \geq L_{ij} + k(p_{ij}\alpha W_i - p'_{ij}\alpha W_j) + \alpha(W_i q_{ij} - W_j q'_{ij}).$$

With $\alpha W_i = w_i = w_j \pi_{ij} = w_j \frac{p'_{ij}}{p_{ij}}$ and $\alpha W_j = w_j$ we have $p_{ij}\alpha W_i - p'_{ij}\alpha W_j = 0$ and thus, with (5.13)

$$t_j - t_i \geq L_{ij} + \alpha(W_i q_{ij} - W_j q'_{ij}) = L_{ij} - \alpha H_{ij}.$$

Now consider two tasks $< i;k >$ and $< j;l >$ to be processed on the same machine. Again (5.18) with (5.16) is equivalent to

$$t_i + \alpha k W_i + p_i \leq t_j + \alpha l W_j \vee t_j + \alpha l W_j + p_j \leq t_i + \alpha k W_i$$

or

$$p_i + \alpha(kW_i - lW_j) \leq t_j - t_i \vee t_j - t_i \leq -p_j + \alpha(kW_i - lW_j).$$

We have $\{-W_i k + W_j l | k, l \in \mathbb{Z}\} = \{m * GCD(W_i, W_j) | m \in \mathbb{Z}\}$ because of Theorem 5.3. So, we get

$$p_i - \alpha m * GCD(W_i, W_j) \leq t_j - t_i \vee t_j - t_i \leq -p_j - \alpha m * GCD(W_i, W_j)$$

Therefore, the numbers $t_j - t_i$ cannot be contained in the intervals

$$\ldots,]-p_j - \alpha m * GCD(W_i, W_j), p_i - \alpha m * GCD(W_i, W_j)[,$$

$$]-p_j - \alpha(m-1) * GCD(W_i, W_j), p_i - \alpha(m-1) * GCD(W_i, W_j)[, \ldots$$

Thus, $t_j - t_i$ must be contained in one of the intervals

$$\ldots, [p_i - \alpha m * GCD(W_i, W_j), -p_j - \alpha(m-1) * GCD(W_i, W_j)], \ldots \tag{5.24}$$

which implies that for some integer K_{ij} we must have

$$p_i - \alpha K_{ij} * GCD(W_i, W_j) \leq t_j - t_i \leq -p_j + \alpha(1 - K_{ij}) * GCD(W_i, W_j)$$

With $K_{ji} := 1 - K_{ij}$ conditions (5.21) to (5.23) are satisfied. On the other hand, if (5.20) to (5.23) are satisfied, then conditions (5.16) to (5.18) hold if we set $t(i;k) := t_i + \alpha k W_i$ for $i \in T$ and $k \in \mathbb{Z}$. □

The next theorem shows that the intervals (5.24) cannot be empty.

For the proof of this theorem we need the following lemma, which describes the possible positions of the occurrences of operation i and j in a feasible periodic schedule, if $GCD(W_i, W_j) = 1$. In connection with this lemma intervals I_h are defined as follows. Consider a feasible schedule for a given instance of a CLSP. We choose an operation i. Then

$$I_h := [t(i;0) + h\alpha, t(i;0) + (h+1)\alpha]$$

for each $h \in \mathbb{Z}$.

Lemma 5.5 *Let $z \in \mathbb{Z}$ and $i, j \in T$ with $GCD(W_i, W_j) = 1$. Then there exist $k, l \in \mathbb{Z}$ with $k - l = z$ such that an occurrence of operation i starts in the interval I_k and an occurrence of j starts in the interval I_l.*

Proof: Due to the defined labelling of the intervals in a feasible schedule and according to (5.16), the different occurrences of operation i start at the beginning of each intervals

$I_{W_i \cdot a} \forall\, a \in \mathbb{Z}$. If an occurrence of operation j starts in the c-th interval, then the different occurrences of operation j start in the intervals $I_{c+W_j \cdot b}\ \forall\, b \in \mathbb{Z}$.

Now we need to find integer numbers a' and b' such that $I_k = I_{W_i \cdot a'}$ and $I_l = I_{c+W_j \cdot b'}$ with $k - l = z$. We have

$$\begin{aligned} & k - l && = z \\ \Leftrightarrow \quad & W_i \cdot a' - c - W_j \cdot b' && = z \\ \Leftrightarrow \quad & (-c - z) \cdot 1 && = -W_i \cdot a' + W_j \cdot b' \\ \Leftrightarrow \quad & (-c - z) GCD(W_i, W_j) && = -W_i \cdot a' + W_j \cdot b' \end{aligned}$$

Due to Theorem 5.3, the last equation is equivalent to

$$-W_i(-(-c - z))u + W_j(-c - z)v \ = -W_i \cdot a' + W_j \cdot b' \tag{5.25}$$

with $u, v \in \mathbb{Z}$. Thus, for a' and b' we have to choose the values $a' = -(-c - z)u$ and $b' = (-c - z)v$. □

With the previous lemma, we can now show the following theorem, which shows that the intervals (5.24) cannot be empty.

Theorem 5.6 *Given is a feasible instance of a cyclic machine scheduling problem with linear precedence constraints. Then*

$$\alpha \geq \frac{p_i + p_j}{GCD(W_i, W_j)} \tag{5.26}$$

holds for every $\alpha \geq \alpha^*$ and $i, j \in T$ with $i \neq j$ and $M(i) = M(j)$, whereas α^* is the optimal cycle time for the given instance.

Proof: Without loss of generality we can assume that $GCD(W_i, W_j) = 1$. Otherwise, we can change all computed W_k-values for all $k \in T$ and the cycle time α in the following way:

$$W'_k := \frac{W_k}{GCD(W_i, W_j)}\, \forall\, k \in T$$

and

$$\alpha' := \alpha \cdot GCD(W_i, W_j).$$

Then we get $W'_k \in \mathbb{Q}^+\ \forall\, k \in T \setminus \{i, j\}$ and $W'_i, W'_j \in \mathbb{N}^+$ and $GCD(W'_i, W'_j) = 1$. Note, that the schedule with the new values has the same properties as the original schedule. Therefore, we can assume that $GCD(W_i, W_j) = 1$ holds.

Due to Lemma 5.5, we can find an interval I_ν in which both operations i and j are started. Furthermore, due to the definition of the intervals I_h, job i is started at the beginning of I_ν.

We can also find two succeeding intervals I_μ and $I_{\mu+1}$ such that j is started in I_μ and i is started (again at the beginning) in $I_{\mu+1}$. This implies that j must be completely processed within I_μ (see Figure 5.1b). Because we have a periodic schedule with period length α, both i and j must be completely processed in I_ν (see Figure 5.1a), i.e. $p_i + p_j \leq \alpha$. □

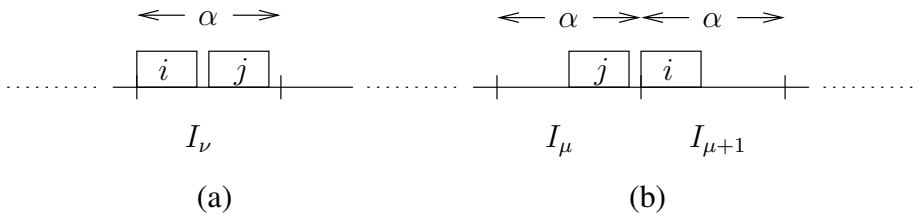

Figure 5.1: Different positions of i and j in a feasible schedule

Now we want to given an example with linear precedence constraints.

Example 5.7 *The graph with the linear precedence constraints is given in Figure 5.2. The computed W-values are given in Table 5.1. As we can see the operations 1,2,3, and 4 have a W-value of 2 and therefore these operations are only processed in every two periods, whereas the begin of a period is defined by the operation 0. In Figure 5.3 the optimal schedule is given. The computed height values for the arcs is given in Table 5.2.*

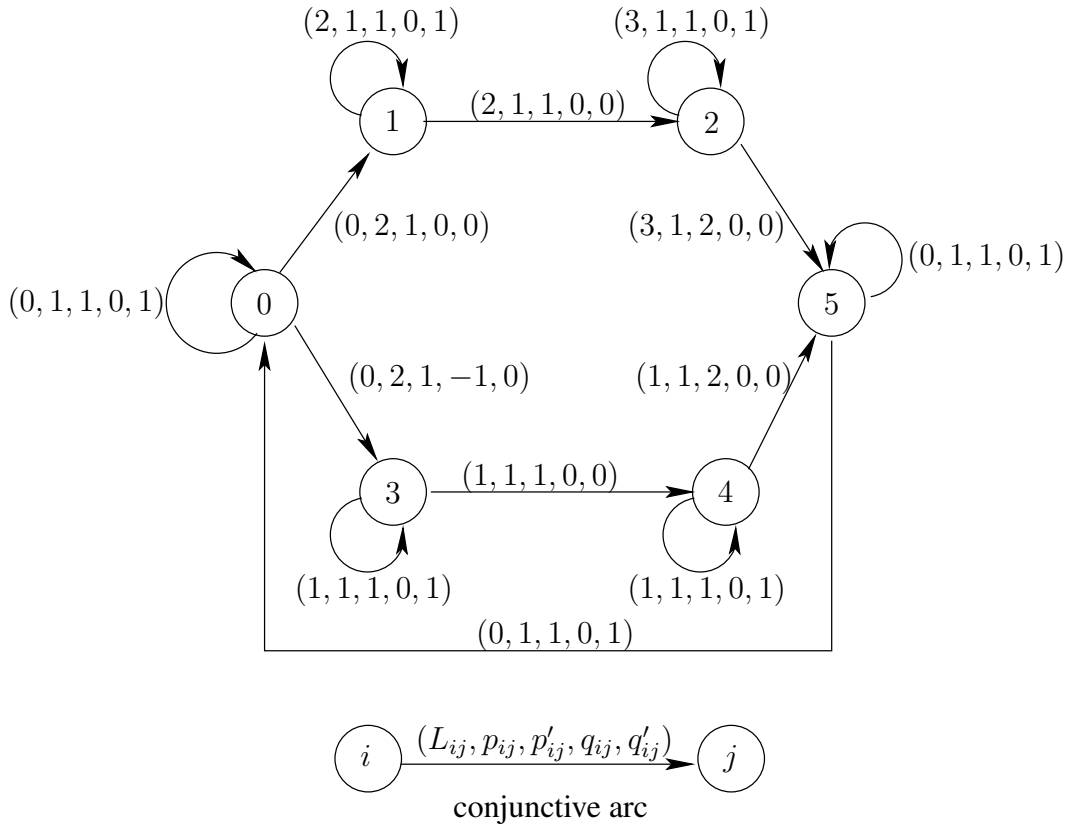

Figure 5.2: Precedence constraints for Example 5.7

Operation	*0*	*1*	*2*	*3*	*4*	*5*
W*-value*	*1*	*2*	*2*	*2*	*2*	*1*

Table 5.1: The computed W-values

Arc	$(0,1)$	$(0,3)$	$(1,2)$	$(2,5)$	$(3,4)$	$(4,5)$	$(5,0)$
Delay	*0*	*0*	*2*	*3*	*1*	*1*	*0*
Height	*0*	*1*	*0*	*0*	*0*	*0*	*1*

Arc	$(0,0)$	$(1,1)$	$(2,2)$	$(3,3)$	$(4,4)$	$(5,5)$
Delay	*0*	*2*	*3*	*1*	*1*	*0*
Height	*1*	*2*	*2*	*2*	*2*	*1*

Table 5.2: The heights of the conjunctive arcs

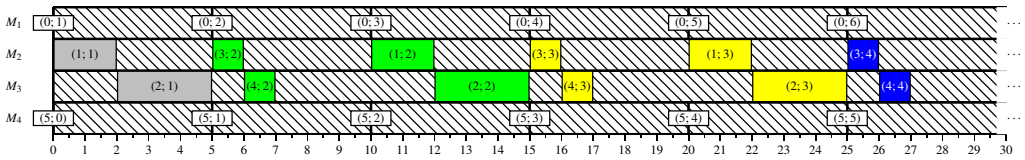

Figure 5.3: The optimal schedule for Example 5.7

In this section, we presented a different view on periodic schedules, in which all operations can have a different cycle time.

6 Applications

In this section, we present important cyclic scheduling problems which are mentioned in the literature and which can be modelled by our framework.

There exists up to now no such general model which can be used for solving all these different cyclic scheduling problems. This is the main advantage of our framework.

This section is structured as follows. For each application, we describe first the constraints and how these constraints fit into our model. Then we give some references to the literature and finally, we present for each application a short example, which includes the graph G described by the precedence constraints and a gantt chart for an optimal solution. Note that all optimal solutions are computed by solving the corresponding mixed integer linear program. As a MIP solver, we used Cplex 8.1 (see [1]).

The type of applications can be distinguished into three different types. As first type, cyclic scheduling problems are considered which are all based on the classical job shop problem. As second type, these applications are extended by transportations robots. The third type describes some problems that are not directly linked to machine scheduling problems. These problems are derived from the area of Software Pipelining.

6.1 Cyclic Job Shop

In this subsection again we describe the cyclic job shop problem, which is already mentioned in Section 4, and also several other extensions of this cyclic job shop problem. The basis for these types of problems is the classical job shop problem.

The classical job shop problem may be defined as follows: We have n jobs ($J = 1, \ldots, n$). Each job $j \in J$ consists of n_j operations $O_{1j}, \ldots, O_{n_j j}$. For $l = 1, \ldots, n_j - 1$ we call operation $O_{l+1\,j}$ the succeeding operation of O_{lj}. In a similar way for $l = 2, \ldots, n_j$ we call operation $O_{l-1\,j}$ the preceding operation of O_{lj}. O_{1j} and $O_{n_j j}$ are the first and last operations of job j, respectively. The set of all operations of all jobs is denoted by T. Each operation $i \in T$ has a processing time p_i and belongs to a job $j(i)$. The set of all operations of job j is denoted with $O(j)$. $s(i)$ denotes the succeeding operation of i, if it exists and $p(i)$ denotes the preceding operation of i, if it exists.

Furthermore, we have precedence constraints of the form $t_{s(i)} \geq t_i + L_{ij}$ between two succeeding operations i and $s(i)$ of the job $j(i)$ with delay $L_{ij} = p_i$, where t_i defines the starting time of operation i in a schedule.

Each operation i has to be processed on a dedicated machine $M(i) \in M = \{1, \ldots, m\}$. Each machine can only process one operation at a time. If several operations are processed on the same machine m, we need to fix an order between all operations that are processed on this machine.

Example 6.1 *In this example we give an instance of a classical job-shop scheduling problem which is extended in the following example in several different ways leading to different cyclic job shop scheduling problems. The example consists of three jobs. Each job has three different operations. The data for this instance is given in Table 6.1. The graph given by the precedence constraint is given in Figure 6.1.*

Job	*1*			*2*			*3*		
Operation	1	2	3	4	5	6	7	8	9
Processing time	*1*	*2*	*2*	*2*	*1*	*1*	*2*	*2*	*2*
Machine	*1*	*2*	*3*	*3*	*2*	*1*	*3*	*2*	*1*

Table 6.1: Data for job-shop instance given in Example 6.1

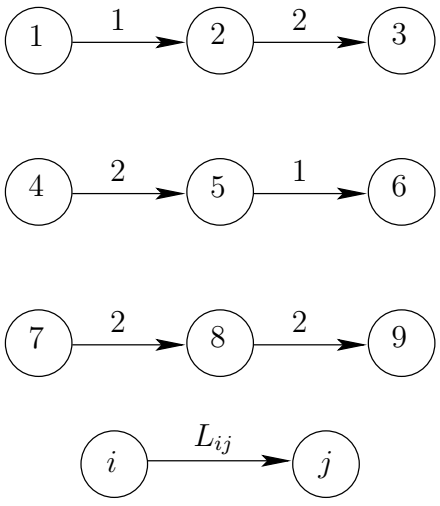

Figure 6.1: Precedence Constraints for the instance in Example 6.1

This classical job shop problem can be generalized to cyclic job shop problems in three

different ways, which are all considered in the literature. What all models have in common is that the height $H_{i\,s(i)}$ of the precedence constraints between the operation i and its succeeding operation $s(i)$, if it exists, is set to $H_{i\,s(i)} = 0$.

The first model is directly derived from the noncyclic job shop problems. In the second model, the job chains described by the precedence constraints between the operations of the same job are repeated. In contrast to this, the machine chains are repeated in the third model.

In the following, we discuss how to model these problems within our framework.

We start with the first model. These types of problems are called **cyclic job shop problems**. We introduce a source node 0 and a sink node $\star$. The source node is connected to the first operation of all jobs $j \in J$. These arcs have both length and height 0. The last operations i of all jobs j are connected to the sink node which has length p_i and height 0. The sink and the source node are connected by an arc with length 0 and variable height $H_{\star 0}$. If the height $H_{\star 0}$ is set equal to 1, then the optimal cycle time of this problem is equal to the optimal makespan of the classical job-shop problem. Note that the only difference between the cyclic job shop problem and the classical job shop is the directed arc in the cyclic job shop between the sink and the source node. This model is used in Hanen [30] and Brucker and Kampmeyer [8].

In the following we present an example for the cyclic job shop problem and also describe the influence of the height $H_{\star 0}$ of the arc $(\star, 0)$.

Example 6.2 *The Example 6.1 is changed so that we get a cyclic job shop problem. The new graph defining the precedence constraints is given in Figure 6.2.*

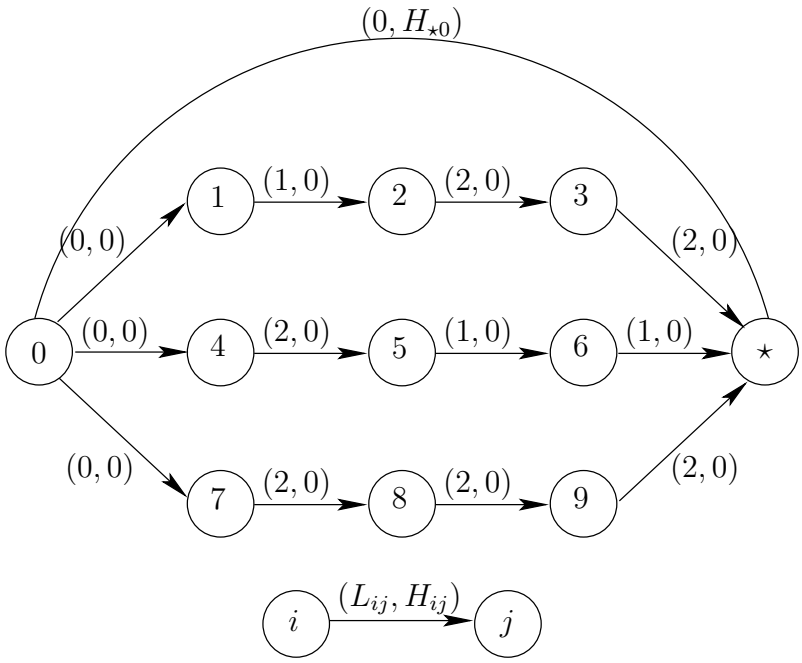

Figure 6.2: Precedence Constraints for the cyclic job shop instance for Example 6.2

If we fix the height of the arc $(\star, 0)$ *to* $H_{\star 0} = 1$*, we get an optimal schedule with cycle time* $\alpha = 8$*. The gantt chart for the optimal schedule is given in Figure 6.3.*

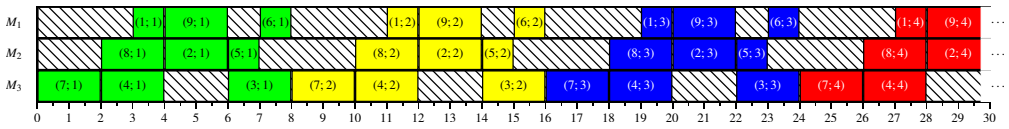

Figure 6.3: Optimal schedule for the Example 6.2 with $H_{\star 0} = 1$

If we increase the height of the arc $(\star, 0)$ *to* $H_{\star 0} = 2$*, the optimal cycle time decreases to* $\alpha = 6$*. The corresponding schedule can be found in Figure 6.4.*

Due to increasing the height of $H_{\star 0}$ *to* 2*, it is now feasible that operations with different occurrence numbers are processed at the same time. This leads to a decrease of the optimal cycle time.*

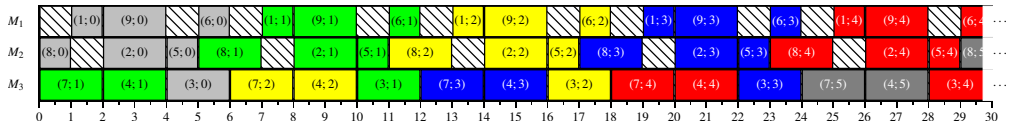

Figure 6.4: Optimal schedule for the Example 6.2 with $H_{\star 0} = 2$

The second model is not widely used. It is introduced in [8] and is called **cyclic job shop problems with job repetition**. To model the job chain repetition, we introduce arcs from the last operations l to the first operation i with length p_l and height H_{Job} for all jobs $j \in J$. These arcs mean that after the k-th occurrence of the last operation of a job j is finished the $k + H_{Job}$-th occurrence of first operation of the same job can start again. In the following we reuse Example 6.1 to get a problem with job chain repetition.

Example 6.3 *The graph given by the precedence constraints for the extended example for the cyclic job shop problem with job repetition is given in Figure 6.5.*

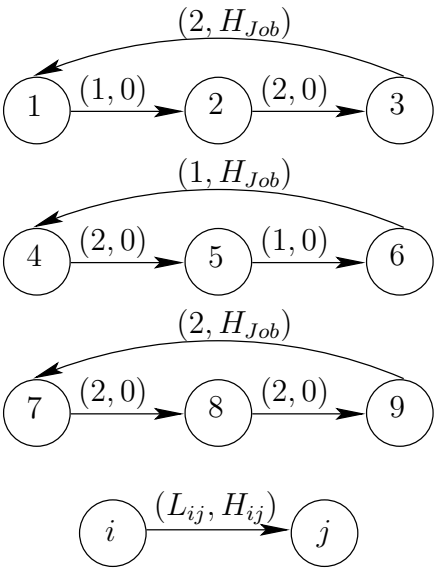

Figure 6.5: Precedence constraints for the cyclic job shop with job repetition instance for Example 6.3

If we set H_{Job} to 1, we get the following optimal schedule with cycle time $\alpha = 7$ which is given in the gantt chart in Figure 6.6.

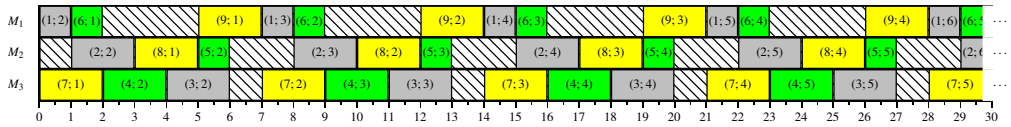

Figure 6.6: Optimal schedule for the Example 6.3 with $H_{Job} = 1$

If we increase the height H_{Job} by one, the optimal cycle time reduces to $\alpha = 6$. The optimal gantt chart can be found in Figure 6.7.

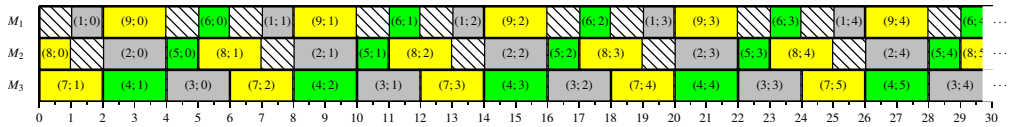

Figure 6.7: Optimal schedule for the Example 6.3 with $H_{Job} = 2$

If $H_{Job} = 1$ holds, the $k+1$-th occurrence of operations of a job can only start if the k-th occurrence of all operations of the same job are finished. However, if we increase H_{Job} to 2, the $k+1$-th occurrence of an operation can already start although the k-th occurrence of the operation is not yet finished. This can be seen in Figure 6.7. In this figure the $k+1$-th occurrence of the first operation of job 3 is already started although the k-th occurrence of the second and third operation of job 3 are still processed by its machines. Thus, the increase of H_{Job} leads to a better utilization of the machine and therefore, to a decrease of the optimal cycle time.

Note that the graph which models the cyclic job shop problem with job chains repetition is not strongly connected. Therefore, we introduce two dummy nodes, the source 0 and the sink $\star$ node. The source node is connected with all first operations of a job with zero length and zero height. All last operations i of a job are connected with the sink node with length p_i and height zero. Finally, the sink node is connected to the source node with zero length and the height is equal to the number of operations. As we showed in Section 3.4, this arc does not lead to a restriction on the problem. It is easy to see that with this extension the graph is strongly connected. This idea is also used to get a strongly connected graph for the following applications.

The third model is widely used in the literature. We call this type of problems **cyclic job shop problems with machine repetition**. It is introduced first by Hitz [33]. He defines a minimal part set (MPS) which must be repeated a certain number of times. As an example, assume we want to produce 100 units of job A, 200 of job B, and 200 of job

C. Then the question is how to set up a MPS. There are various ways to choose a MPS. Normally, selecting the makeup of a MPS is done in advance [43]. If the MPS is fixed to $(1A, 2B, 2C)$, then it has to be produced 100 times to meet the production requirements. Furthermore, the machine processing order in each produced MPS is the same. The aim is, as before, to find a periodic schedule with minimal cycle time.

Lee and Posner [43] extended the disjunctive graph model, developed by Roy and Sussmann [63], to describe solutions for this problem. In the following, we describe how to model these problems in our framework. If we analyse the proposed extension of the disjunctive graph model by Lee and Posner [43], we can easily see that the only difference to the application described before is that the chains on each machine have to be repeated. To model this, we introduce two additional nodes a source node 0_m and a sink node $\star_m$ for each machine $m \in M$. Then each source node 0_m is connected to all operations which are processed on machine m with length and height 0 and all operations $i \in T$ with $M(i) = m$ are connected to the sink node $\star_m$ with length p_i and height 0. Furthermore, each sink node $\star_m$ is connected with the corresponding source node 0_m with delay 0 and height H_{MPS}. Usually in the literature in which this model is discussed the height H_{MPS} is equal to 1. In Seo and Lee [65] the height is set to 2 and they call this problem 'cyclic job shop with overtaking'. In Hall et al. [29] several complexity results for the problem with $H_{MPS} = 1$ are derived. One important result is that if each job has at least three operations, then the problem whether there exists a periodic schedule with cycle time $\alpha \leq \alpha^+$, where α^+ is a given rational number, is $\mathcal{NP}$-complete.

The main difference between these two different heights can be seen easily in the following example.

Example 6.4 *The Example 6.1 is changed so that we get a cyclic job shop problem with machine chains repetition. The graph given by the precedence constraints is shown in Figure 6.8.*

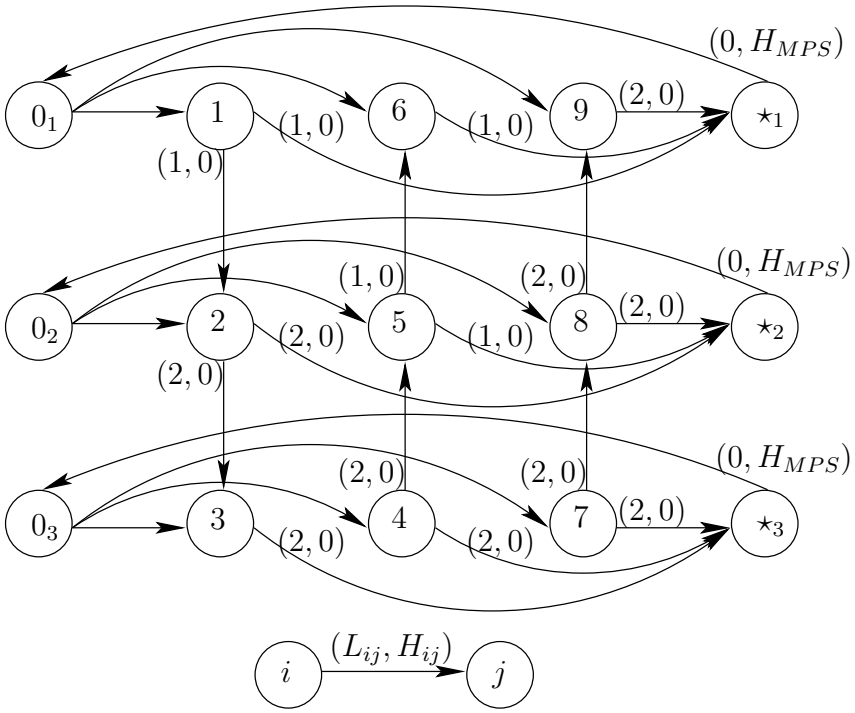

Figure 6.8: Precedence constraints for the cyclic job shop with machine chains repetition instance for Example 6.4

The optimal gantt chart with cycle time $\alpha = 7$ and $H_{MPS} = 1$ is given in Figure 6.9. The optimal gantt chart with $H_{MPS} = 2$ is shown in Figure 6.10. Here the optimal cycle time decreases to $\alpha = 6$.

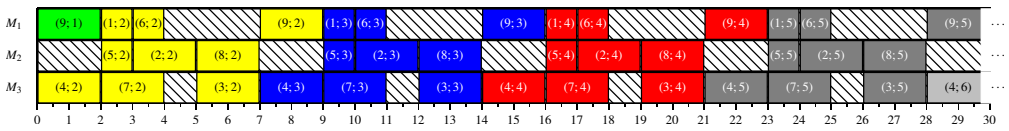

Figure 6.9: Optimal schedule for the Example 6.4 with $H_{MPS} = 1$

If we compare the gantt chart of Figure 6.9 with the gantt chart in Figure 6.10, we can see that in the Figure 6.9 the k-th occurrences of all operations which are processed on the machine must be finished until the next occurrence can start. Due to the increase of

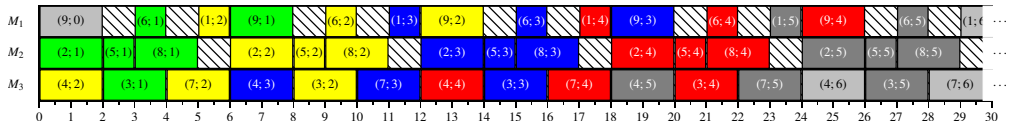

Figure 6.10: Optimal schedule for the Example 6.4 with $H_{MPS} = 2$

the height H_{MPS}, this restriction is relaxed and a new occurrence of an operation can already start although not all operations of the previous occurrences are finished.

Note that all these types of problems can be easily extended to the presence of blocking. The cyclic job shop problem extended by blocking is considered in Brucker and Kampmeyer [9]. McCormick et al. [52] and Song and Lee [68] consider the cyclic job shop problem with machine repetition extended by blocking.

In the following, we present an example for cyclic job shop problem with machine repetition and blocking.

Example 6.5 *The Example 6.4 is changed so that we get a cyclic job shop problem with machine chains repetition and blocking.*

The optimal gantt chart with cycle time $\alpha = 7$ and $H_{MPS} = 1$ is given in Figure 6.11. The optimal gantt chart with $H_{MPS} = 2$ is given in Figure 6.12. Here the optimal cycle time is also $\alpha = 7$.

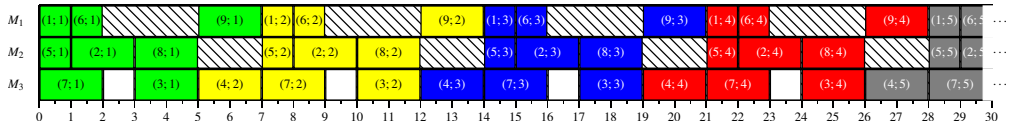

Figure 6.11: Optimal schedule for the Example 6.5 with $H_{MPS} = 1$

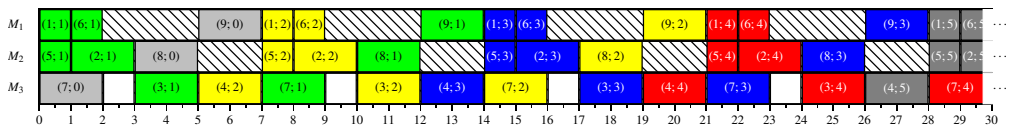

Figure 6.12: Optimal schedule for the Example 6.5 with $H_{MPS} = 2$

In both problems the only time where blocking occurs is after the processing of each occurrence of operation 7 is finished.

6.2 Cyclic Job Shop With Transportation Robots

In this section we describe how to model cyclic scheduling problems in the presence of one transportation robot. Here we distinguish between two cases.

In the first case, we describe a problem in which the robot carries special tools to the machines which are needed for processing operations at this machine. In the second case, the robot carries the different operations from one machine to another. These types of problems are a direct generalization of the classical non-cyclic shop problem with transportation (see e.g. Knust [40]).

Now we start with the first case and show how to model these problems within our framework.

The first problem is discussed in Kuijpers [42] and is called **cyclic machine scheduling with tool transportation**. The problem can be described as follows:

There are given M identical jobs ($j = 1, \ldots, M$) and M identical machines $(1, \ldots, M)$. Each job $j \in \{1, \ldots, M\}$ consists of N operations $(1, \ldots, N)$ with processing time $p_i > 0$ ($i = 1, \ldots, N$). All operations of job j must be processed on the machine j. The succeeding operation of operation i is denoted again by $s(i)$.

There are given N tools ($i = 1, \ldots, N$). For processing operation i of job $j(i)$ tool i must be present at machine $j(i)$. These tools are transported from one machine m to the next machine $m + 1$ cyclically by a robot. The robot can transport only one tool at a time. The tool i for the processing of operation i of job $j(i)$ is taken by the robot from machine $j(i) - 1$ to machine $j(i)$. This transport operation is denoted by $T_{i\,j(i)}$. The corresponding travelling time is denoted by $d_{j(i)-1\,j(i)}$ which also equals to the time to move the robot from machine $j(i)$ back to $j(i) - 1$. Thus, $d_{j(i)-1\,j(i)} = d_{j(i)\,j(i)-1}$ holds. For this application we identify machine M with machine 0. Thus, d_{01} denotes the travelling time from machine M to 1.

The starting time of the k-th occurrence of operation i of job $j(i)$ on machine $j(i)$ is denoted by $t(i; k)$. The starting time of the k-th occurrence of the transport operation $T_{i\,j(i)}$ is denoted by $t(T_{i\,j(i)}; k)$. The set of operations which are performed on the machines is denoted by T, whereas the set of transport operations is denoted by T^{robot}.

Furthermore, the following different types of conjunctions are given:

Type I. Each machine can process only one operation at a time. Therefore, operation i must be finished before the succeeding operation $s(i)$ can start. Thus, the following

inequality must hold

$$t(s(i);k) \geq t(i;k) + p_i$$

for all operation $i \in T$ which has a succeeding operation $s(i)$.

Furthermore, the k-th occurrence of the last operation N of job $j(N)$ (which has no succeeding operation) must be finished before the $k+1$-th occurrence of the first operation of the same job can start. Let $s(i)$ be the first operation of $j(i)$ if i is the last operation of $j(i)$. Thus, we get the following inequality

$$t(s(i);k+1) \geq t(i;k) + p_i$$

which must hold for the last operations of all jobs.

Type II. As the k-th occurrence of operation i needs the tool i for the start of the processing, the k-th occurrence of the transport of tool i to machine $j(i)$ must be finished. Thus,

$$t(i;k) \geq t(T_{i\,j(i)};k) + d_{j(i)-1\,j(i)}$$

for all operations $i \in T$.

Type III. After finishing the k-th occurrence of operation i of job $j(i)$, the tool i can be transported to the next machine $j(i)+1$. So the following inequality must hold for all operations i with $M(i) \in \{1,\ldots,M-1\}$

$$t(T_{i\,j(i)+1};k) \geq t(i;k) + p_i.$$

As the tools are transported cyclically through the machines, the robot carries the tool i to machine 1 after the k-th occurrence of operation i on the last machine M is finished so that the $k+1$-th occurrence of operation i can start on machine 1. Thus,

$$t(T_{i1};k+1) \geq t(i;k) + p_i$$

must hold for all operations with $M(i) = M$.

Type IV. If the robot performs two succeeding transport operations $T_{i\,j(i)}$ and $T_{s(i)\,j(i)}$ which both start at the same machine $j(i)-1$, the time difference between the starting time of these two transport operations must be $2d_{j(i)-1\,j(i)}$ because after finishing the k-th occurrence of transport operation $T_{i\,j(i)}$, which takes $d_{j(i)-1\,j(i)}$, the robot has to move back empty to machine $j(i)-1$, which takes $d_{j(i)\,j(i)-1}$, to perform the k-th occurrence of transport operation $T_{s(i)\,j(i)}$. So, the difference between these two succeeding transport operations must be at least $d_{j(i)-1\,j(i)} + d_{j(i)\,j(i)-1} = 2d_{j(i)-1\,j(i)}$. So we get

$$t(T_{s(i)\,j(i)};k) \geq t(T_{i\,j(i)};k) + 2d_{j(i)-1\,j(i)}$$

for all operations $i \in T$ for which a succeeding operation exists.

If the robot finishes the k-th occurrence of transport operation $T_{N\,j(N)}$, where N is the last operation of the job $j(N)$ (N has no succeeding operation), then the $k+1$-th occurrence of transport operation $T_{1\,j(N)}$ can start after the robot has moved back to machine $j(N)-1$. Thus,

$$t(T_{1\,j(N)}; k+1) \geq t(T_{N\,j(N)}; k) + 2d_{j(N)-1\,j(N)}$$

must hold for the last operations of all jobs.

As the robot can perform only one transport at a time, we have to fix an order between the different transport operations. The robot has to finish one transport operation $T_{i\,j(i)}$ until a new transport operation $T_{i'\,j(i')}$ can start. To start the new transport operation $T_{i'\,j(i')}$, the robot has to arrive empty at the machine $j(i')-1$ to pick up tool i'. Therefore, we get the following disjunctive constraints between the transport operations

$$t(T_{i\,j(i)}; k) + d_{j(i)-1\,j(i)} + d_{j(i)\,j(i')-1} \leq t(T_{i'\,j(i')}; l) \vee$$

$$t(T_{i'\,j(i')}; l) + d_{j(i')-1\,j(i')} + d_{j(i')\,j(i)-1} \leq t(T_{i\,j(i)}; k)$$

for all $i, i' \in T$ with $i \neq i'$ and $k, l \in \mathbb{Z}$.

It is still an open question whether this problem is $\mathcal{NP}$-hard or not.

For this application we also present an example.

Example 6.6 *An instance for the cyclic machine scheduling with tool transportation problem is given in Table 6.2. The instance consists of three jobs each with four operations* 1*,* 2*,* 3 *and* 4*.*

Operation	*1*	*2*	*3*	*4*
Processing time	*2*	*1*	*2*	*1*

Table 6.2: Data for the instance given in Example 6.6

The time it takes the robot to move from one machine m_1 to m_2 equals 1 *if $m_2 \neq m_1$, for all machines $m_1, m_2 \in \{1,2,3\}$. If $m_1 = m_2$, then $d_{m_1\,m_2} = d_{m_1\,m_1}$ equals to 0. The graph given by the precedence constraints is given in Figure 6.13. The gantt chart with*

optimal cycle time $\alpha = 12$ is given in Figure 6.14. Note that all transport operations $T_{ij(i)} \in T^{robot}$ are done by one robot, although all transport operations that arrive at machine m are drawn in the gantt chart as an additional machine.

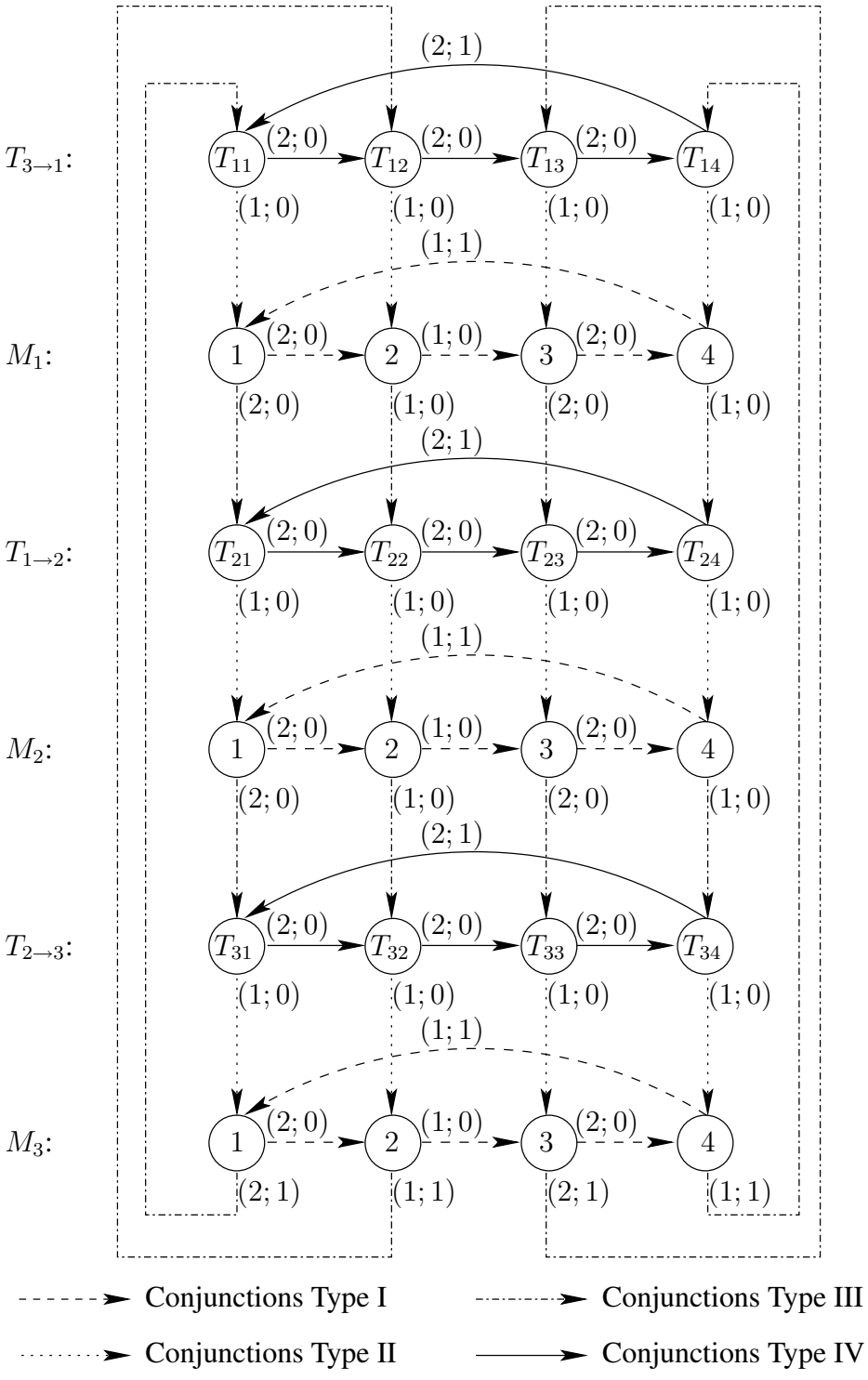

Figure 6.13: Precedence constraints for the Example 6.6

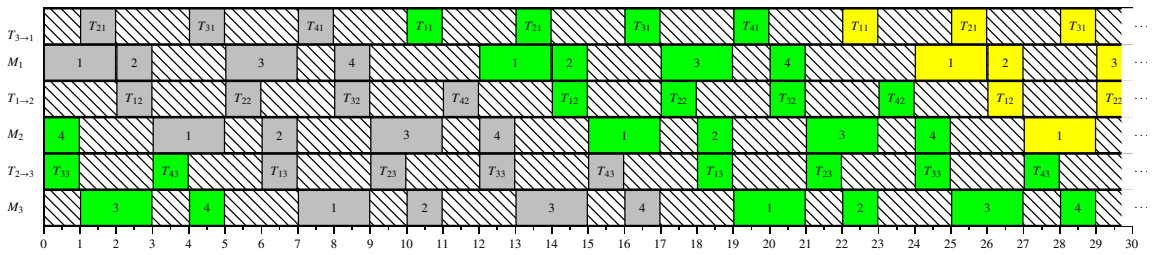

Figure 6.14: Optimal schedule for the Example 6.6

In the gantt chart, we can see that the robot always transports a tool from one machine to another. This is a very special case. In most problems the robot has to perform some empty move, this means that the robot moves empty to a machine to pick up a tool.

This is shown in the gantt chart in Figure 6.15. For example after performing the transport T_{21} the robot moves back empty to machine M_3 to perform the transport T_{31}.

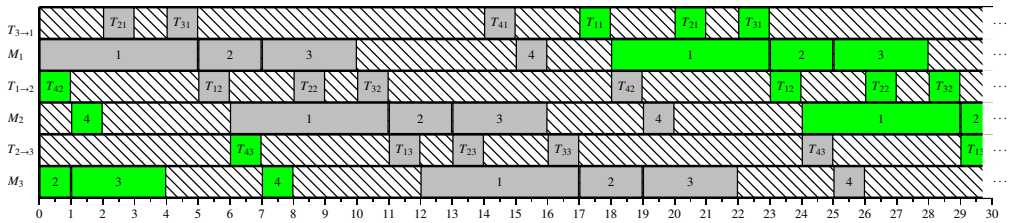

Figure 6.15: Schedule with empty robot moves

Now we consider the second case in which the robot carries instead of tools jobs from one machine to another. This problem is based on the *Cyclic Job Shop Problem with machine chain repetition*. Thus, there exists unlimited storage at each machine. Between two operations $p(i)$ and i of a job $j(i)$ a transport operation T_i occurs, if operation i has a preceding operation, which is denoted by $p(i)$. This transport operation is done by a robot, which takes the job $j(i)$ from machine $M(p(i))$ and brings the job to machine $M(i)$ so that the processing of operation i of $j(i)$ can start. Note that the processing of job $j(i)$ at machine $M(i)$ can only start if the machine $M(i)$ is available.

The transport robot can only perform one transport operation at a time and a job can only picked up by a robot if the robot is empty. The time it takes to bring job $j(i)$ from machine $M(p(i))$ to machine $M(i)$ is denoted by $d_{M(p(i))\,M(i)}$, and the time to load or unload a job from the machine m is denoted by ϵ_m. Thus, the processing time of the transport operation

T_i is defined as $p_{T_i} := \epsilon_{M(p(i))} + d_{M(p(i))\,M(i)} + \epsilon_{M(i)}$. The first operation of each job is always available at its machine.

So we have the following precedence constraints:

Type I. The k-th occurrence of operation i can only start if the k-th occurrence of transport operation T_i is finished. Thus,

$$t(i;k) \geq t(T_i;k) + p_{T_i}$$

must hold for all operations $i \in T$ for which a preceding operation exists. (The first operation of each job is always available at its machine.)

Furthermore, we postulate that the $k+1$-th occurrence of operation i can only start if the k-th occurrence is finished. Therefore, the following inequality must hold for all operation $i \in T$:

$$t(i;k+1) \geq t(i;k) + p_i.$$

Type II. The k-th occurrence of transport operation T_i can only start if the k-th occurrence of $p(i)$ is finished. Thus,

$$t(T_i;k) \geq t(p(i);k) + p_{p(i)}$$

must hold for all operations $i \in T$. Note that for each transport operation T_i there exists a preceding operation $p(i)$.

Additionally, we postulate that the $k+1$-th occurrence of transport operation T_i can only start if the k-th occurrence is finished and the robot arrives empty at machine $M(p(i))$. Therefore, we get

$$t(T_i;k+1) \geq t(T_i;k) + p_{T_i} + d_{M(i)\,M(p(i))}$$

for all transport operations $T_i \in T^{robot}$.

Additionally, we get two types of disjunctive constraints, the disjunctive constraints between the operations which are processed on the same machine and the disjunctive constraints between all transport operations.

As each machine can perform only one operation at a time, we get:

$$t(i;k) + p_i \leq t(i';l) \vee t(i';l) + p_{i'} \leq t(i;k) \tag{6.1}$$

for all $i, i' \in T$ with $i \neq i'$ and $M(i) = M(i')$ and $k, l \in \mathbb{Z}$.

Now we consider two different transport operations T_i and $T_{i'}$. As we have only one robot to perform the transport operations, we must fix an order between the different occurrences of both transport operations. Assume that the k-th occurrence of transport operation T_i is performed before the l-th occurrence of $T_{i'}$. As the robot can transport only one operation at a time, the robot must first finish the k-th occurrence of transport operation T_i, which takes p_{T_i} time units. Then, the robot can directly unloaded job $j(i)$ at machine $M(i)$. Afterwards, the robot must move empty to machine $M(p(i'))$, which takes $d_{M(i)\,M(p(i'))}$ time units, to start the l-th occurrence of transport operation $T_{i'}$. Therefore, we get the following disjunctive constraints:

$$\begin{aligned} t(T_i;k) + p_{T_i} + d_{M(i)\,M(p(i'))} \leq t(T_{i'};l) \vee \\ t(T_{i'};l) + p_{T_{i'}} + d_{M(i')\,M(p(i))} \leq t(T_i;k) \end{aligned} \tag{6.2}$$

for all transport operations T_i and $T_{i'}$ with $T_i \neq T_{i'}$ and for all $k, l \in \mathbb{Z}$.

The situation which is described in the previous paragraphs is a very general situation. Most of the work in the area of cyclic job problems with one transportation robot is focused on the blocking situation, which is described in the following paragraphs. We show that several constraints which are developed before must be adjusted to the new blocking situation. Due to the blocking constraints, each job must be either on a machine or on the robot. Therefore, the following equation holds for all operations $i \in T$:

$$t(i;k) = t(T_i;k) + p_{T_i}. \tag{6.3}$$

Assume that this equation does not hold. This means that operation i cannot be started at machine $M(i)$ right after the job is unloaded. The only reason is that there is another job being processed on machine $M(i)$. Thus, this would lead to a deadlock situation because the robot cannot unload the job at machine $M(i)$ because first there exists no buffer at the machine and second the other operation which is processed on $M(i)$ cannot be unloaded by the robot because the robot is not empty. Therefore, equation (6.3) must hold.

The conjunctive and disjunctive constraints must be adjusted to the new blocking situation. The reformulated conjunctive constraints are:

Type I. As there exists no storage at machine $M(i)$, the processing of the k-th occurrence of operation i starts directly after the k-th occurrence of transport operation T_i is finished. Thus, the following equality must hold

$$t(i;k) = t(T_i;k) + p_{T_i}$$

which is equivalent to the following inequalities:

$$t(i;k) \geq t(T_i;k) + p_{T_i} \wedge \tag{6.4}$$

$$t(T_i; k) \geq t(i; k) - p_{T_i}. \tag{6.5}$$

Furthermore, the $k+1$-th occurrence of operation i can only start if the k-th occurrence of operation i is finished and is transported to machine $M(s(i))$ by the robot, and if the operation i is not the last operation of the job $j(i)$. After the robot has unloaded the job $j(i)$ at machine $M(s(i))$, the robot has to move empty from machine $M(s(i))$ to machine $M(p(i))$ if operation i is not the first operation of job $j(i)$. Then the robot has to load the job $j(i)$, to travel to machine $M(i)$ and finally to unload job $j(i)$ at machine $M(i)$. The time for all these robot moves is

$$p_{T_{s(i)}} + d_{M(s(i))\,M(p(i))} + p_{T_i}$$

if we assume that $d_{M(s(i))\,M(i')} + d_{M(i')\,M(p(i))} \geq d_{M(s(i))\,M(p(i))}$ holds. Thus, we get the following inequality

$$t(i; k+1) \geq t(T_{s(i)}; k) + p_{T_{s(i)}} + d_{M(s(i))\,M(p(i))} + p_{T_i}. \tag{6.6}$$

If operation i has no preceding operation $p(i)$ but a succeeding operation $s(i)$, then the $k+1$-th occurrence of operation i can start directly after the k-th occurrence of operation i is unloaded from machine $M(i)$. Thus, we get

$$t(i; k+1) \geq t(T_{s(i)}; k) + \epsilon_{M(i)}. \tag{6.7}$$

If i is the first and last operation of the job $j(i)$, then the $k+1$-th occurrence of operation i can start directly after the k-th occurrence of operation i is finished. Thus, we get

$$t(i; k+1) \geq t(i; k) + p_i. \tag{6.8}$$

If there exists a preceding operation $p(i)$, then inequality (6.8) can be reformulated due to equation (6.3) to

$$t(i; k+1) \geq t(T_i; k) + p_{T_i} + \max\{p_i, d_{M(i)\,M(p(i))} + p_{T_i}\} \tag{6.9}$$

because the $k+1$-th occurrence can only start if the k-th occurrence is finished and the robot arrives empty at the machine $M(p(i))$.

Summarizing, we get for the $k+1$-th occurrence of operation i the following inequalities:

$$t(i;k{+}1) \geq \begin{cases} t(T_{s(i)};k) + p_{T_{s(i)}} + d_{M(s(i))\,M(p(i))} + p_{T_i} & \text{if } p(i) \text{ and } s(i) \text{ exist,} \\ t(T_{s(i)};k) + \epsilon_{M(i)} & \text{if } s(i) \text{ exists and } p(i) \text{ does not exist,} \\ t(T_i;k) + p_{T_i} + \max\{p_i, d_{M(i)\,M(p(i))} + p_{T_i}\} & \text{if } p(i) \text{ exists and } s(i) \text{ does not exist} \\ t(i;k) + p_i & \text{otherwise} \end{cases}$$

Type II. The transport operation T_i can only start if the processing of operation $p(i)$ is finished. Thus,

$$t(T_i;k) \geq t(p(i);k) + p_{p(i)} \tag{6.10}$$

must hold.

As there exists no storage at machine $M(i)$, the robot can perform the $k+1$-th occurrence of T_i only if the k-th occurrence of transport operation $T_{s(i)}$ is finished and the robot arrives empty at machine $M(p(i))$. So the following inequality must hold

$$t(T_i;k+1) \geq t(T_{s(i)};k) + p_{T_{s(i)}} + d_{M(s(i))\,M(p(i))} \tag{6.11}$$

for all transport operations T_i for which a succeeding transport operation $T_{s(i)}$ exists.

If there exists no succeeding transport operation $T_{s(i)}$, then the $k+1$-th occurrence of T_i can start directly after the k-th occurrence of T_i is finished and the robot has arrived empty at the machine $M(p(i))$. Thus, we get

$$t(T_i;k+1) \geq t(T_i;k) + p_{T_i} + d_{M(i)\,M(p(i))} \tag{6.12}$$

for all transport operations T_i, where i is the last operation of the job $j(i)$.

Summarizing, for the transport operation T_i we get the following inequalities:

$$t(T_i;k+1) \geq \begin{cases} t(T_{s(i)};k) + p_{T_{s(i)}} + d_{M(s(i))\,M(p(i))} & \text{if } T_{s(i)} \text{ exists,} \\ t(T_i;k) + p_{T_i} + d_{M(i)\,M(p(i))} & \text{otherwise.} \end{cases}$$

The disjunctive constraint (6.1) between operations which are processed on the same machine must also be adjusted to the blocking situation. As this situation is the same as for the blocking case described in Section 4, we can reuse the disjunctive constraints (4.6) of this section. So we get:

$$t(b(i);l) + p_i^b \leq t(i';k) \vee t(b(i');k) + p_{i'}^b \leq t(i;l) \tag{6.13}$$

for all operations i, i' with $M(i) = M(i')$, where p_i^b and $b(i)$ are defined as follows:

$$b(i) := \begin{cases} s(i) & \text{if } i \text{ is not the last operation of job } j(i) \\ i & \text{otherwise} \end{cases}$$

and

$$p_i^b := \begin{cases} 0 & \text{if } i \text{ is not the last operation of job } j(i) \\ p_i & \text{otherwise.} \end{cases}$$

Using the fact that the operations are transported by one robot and the constraint (6.3) holds, the disjunctive constraints (6.13) can be reformulated to:

$$t(b(i); l) + p_i^b \leq t(i'; k) \vee t(b(i'); k) + p_{i'}^b \leq t(i; l) \tag{6.14}$$

with

$$b(i) := \begin{cases} T_{s(i)} & \text{if } s(i) \text{ and } p(i') \text{ exist,} & \text{(6.15a)} \\ T_{s(i)} & \text{if } s(i) \text{ exists, } p(i') \text{ does not exist,} & \text{(6.15b)} \\ T_i & \text{if } p(i) \text{ and } p(i') \text{ exist and } s(i) \text{ does not exist,} & \text{(6.15c)} \\ T_i & \text{if } p(i) \text{ exists and } p(i') \text{ and } s(i) \text{ do not exist,} & \text{(6.15d)} \\ i & \text{if } p(i') \text{ exists and } p(i) \text{ and } s(i) \text{ do not exist} & \text{(6.15e)} \\ i & \text{if } p(i'), p(i) \text{ and } s(i) \text{ do not exist} & \text{(6.15f)} \end{cases}$$

and

$$p_i^b := \begin{cases} p_{T_{s(i)}} + d_{M(s(i))\,M(p(i'))} + p_{T_{i'}} & \text{if } s(i) \text{ and } p(i') \text{ exist,} & \text{(6.16a)} \\ \epsilon_{M(i)} & \text{if } s(i) \text{ exists and } p(i') \text{ does not exist} & \text{(6.16b)} \\ p_{T_i} + \max\{p_i, d_{M(i)\,M(p(i'))} + p_{T_{i'}}\} & \text{if } p(i) \text{ and } p(i') \text{ exist, } s(i) \text{ does not exist,} & \text{(6.16c)} \\ p_{T_i} + p_i & \text{if } p(i) \text{ exists and } p(i') \text{ and } s(i) \text{ do not exist,} & \text{(6.16d)} \\ p_i & \text{if } p(i') \text{ exists and } p(i) \text{ and } s(i) \text{ do not exist} & \text{(6.16e)} \\ p_i & \text{if } p(i'), p(i) \text{ and } s(i) \text{ do not exist} & \text{(6.16f)} \end{cases}$$

Note that the disjunctive constraint (6.14) together with (6.15a) and (6.16a) can be reformulated to

$$t(T_{s(i)}; l) + p_{T_{s(i)}} + d_{M(s(i))\,M(p(i'))} \leq t(i'; k) - p_{T_{i'}}$$

or

$$t(T_{s(i)}; l) + p_{T_{s(i)}} + d_{M(s(i))\,M(p(i'))} \leq t(T_{i'}; k) \tag{6.17}$$

as $t(T_{i'}; k) = t(i'; k) - p_{T_{i'}}$ because of equality (6.3). If in case (6.16c) $d_{M(i)\,M(p(i'))} + p_{T_{i'}} > p_i$ holds, then the constraint can also be reformulated to

$$t(T_i; l) + d_{M(i)\,M(p(i'))} \leq t(T_{i'}; k) \tag{6.18}$$

Now we discuss these constraints. For this explanation, we assume that the k-th occurrence of operation i is processed before the l-th occurrence of operation i'. First, we consider the case (6.16a). As the succeeding operation $s(i)$ of operation i exists, the robot has to pick up the job $j(i)$ at machine $M(i)$ and has to transport it to machine $M(s(i))$. This takes $p_{T_{s(i)}}$ time units. As the preceding operation $p(i')$ of i' exists, the robot has to move empty to machine $M(p(i'))$ to pick up job $j(i')$. This empty move takes $d_{M(s(i))\,M(p(i))}$ time units. Finally, the robot can pick up job $j(i')$ at machine $M(p(i))$ and can transport it to machine $M(i')$, which takes $p_{T_{i'}}$ time units.

Now consider the case (6.16b). As there exists no preceding operation of i', the processing of operation i' at machine $M(i')$ can start right after the job $j(i)$ is unloaded from machine $M(i)$ by the robot. The unloading is done in $\epsilon_{M(i)}$ time units.

Now we consider the case (6.16c). As there exists no succeeding operation $s(i)$ of operation i, the processing of i' can start right after the processing of job $j(i)$ at machine $M(i)$ is finished and the robot arrives at machine $M(i')$ with the job $j(i')$. Due to the constraints (6.3), the robot is at machine $M(i)$ at the time the processing of job $j(i)$ starts. Therefore, the robot has to move empty to machine $M(p(i'))$ to pick up job $j(i')$ and has to transport it to machine $M(i')$. This move is done in $d_{M(i)\,M(p(i'))} + p_{T_{i'}}$ time units.

If there exists no succeeding operation $s(i)$ and no preceding operation $p(i')$, as in the case (6.16d), then the processing of operation i' can start directly after the processing of operation i is finished. Due to the constraints (6.3), the finishing time of the processing of operation i is equal to the start of the transport T_i plus its processing time p_{T_i} and the processing time of operation i.

Finally, in the cases (6.16e) and (6.16f) operation i has no preceding and no succeeding operation, this means that the job $j(i)$ consists only of one operation, then the processing of operation i' can start directly after the processing of operation i is finished.

Due to this reformulation, we can conclude that if each job consists at least of two operations, it is sufficient to fix the order between the transport operations because in this case the order on the robot also describes the order of the operations on the corresponding machines.

The disjunctive constraints (6.2) hold also for the blocking case. This can be seen easily. Consider two occurrences of different transport operations $(T_i; k)$ and $(T_{i'}; l)$. Due to constraint (6.14), the k-th occurrence of transport operation T_i can only start if the machine $M(i)$ is empty. Therefore, after the robot has arrived with job $j(i)$ at machine $M(i)$ the job can be directly loaded onto the machine. After this, the robot can move empty to machine $M(p(i'))$ to pick up the job $j(i')$. Thus, the disjunctive constraints (6.2) hold also for the blocking case. Note that this developed model only holds if $d_{ik} + d_{kj} \geq d_{ij}$ for all $i, j, k \in M$.

This proposed approach to model cyclic job shop scheduling problems with one transportation robot and with or without buffers at the machines is very general. Most research is focused on two special problems. The first problem is the **robotic flow shop problem with no buffers at the machines** or **robotic cells**. The second problem is the **reentrant robotic cells problem**.

As the second case is a generalisation of the first case, we give in the following a short overview on robotic cell problems. Afterwards, we present an example for a special robotic cell problem. Finally, we describe the reentrant robotic cell problems.

There are several survey papers on robotic cells (see e.g. Hall et al. [27, 28], Sethi et al. [67]). The following classification is based on the survey article by Crama et al. [20].

A robotic flow shop or robotic cell consists of m machines $M_1, \ldots, M_m$, an input station M_0 and an output station M_{m+1}. Additional to these machines, there exists one robot. All jobs $j \in J$ are initially available at the input station. The job j consists of m operations. Each operation must be processed on a machine, the first operation must be performed on M_1, the second operation on the M_2 and so on. The robot must pick up job j at the input station M_0 and must bring the job to the first machine M_1, so that the processing of the first operation of job j can start. After the last operation is finished on machine M_m the robot has to pick up the job and has to bring it to the output station M_{m+1}. Note that in this case every non-transport operation of job j has a preceding and succeeding transport operation. Thus, the only disjunctive constraints are given by the constraints (6.2) and (6.17).

For each operation i there is a given processing window $[p_i^-, p_i^+]$. This means that the operation i must spend at least p_i^- time units and at most p_i^+ time units at machine $M(i)$. Note that this processing window can be modelled within our framework by using the concepts which are introduced in Section 3.3.

In the literature it is often assumed that the travel speeds are symmetric, which means that $d_{ij} = d_{ji}$ holds for all machines i, j and that the triangle inequality, $d_{ij} + d_{jk} \geq d_{ik}$ for all machines i, j, k, holds.

The aim is to find a periodic schedule with minimal cycle time. There are two important problems in the area of robotic cell scheduling, which are both $\mathcal{NP}$-hard. The first problem is called **robotic flow shop scheduling with identical jobs** in which each operation has a processing window $[p_i^-, p_i^+]$ with $p_i^+ > p_i^-$ and $p_i^+ \neq \infty$. The complexity proof can be found in Lei and Wang [44]. The problem is normally called **single product - single hoist scheduling problem**. Note if $p_i^+ = p_i^-$ holds, then the problems of finding an optimal periodic schedule with minimal cycle time can be solved in polynomial time (see Levner et al. [48]). This restriction is called no-wait restriction. For the case $p_i^+ = \infty$, which means that the processing window is unbounded, the problem can be solved in polynomial time (see Crama and van de Klundert [21]). Brauner et al. [6] showed that the problem becomes strongly $\mathcal{NP}$-hard if the triangle equality is not satisfied.

If we now consider a robotic cell problem with different jobs, in which each operation of a job has an unbounded processing window, $p_i^+ = \infty$, then the problem becomes $\mathcal{NP}$-hard if the number of machine m is greater or equal to 3 (see Hall et al. [28]).

If for robotic cell problems the robot moves are fixed in advance, several scientists use the BCSP to compute the optimal cycle time (see e.g Chen et al. [15], Ioachim et al. [37], Levner and Kats [47]).

In the following, we present two examples. The first example is a Robotic Flow Shop Scheduling Problem with two jobs. The second example is a single product - single hoist scheduling problem with one job.

Example 6.7 ***Robotic Flow Shop Scheduling Problem***

The instance consists of two jobs each with two operations.

The data is given in Table 6.3. The time to move the robot from any machine i to any machine j, $i \neq j$, $i, j \in \{0, 1, 2, 3\}$ equals $d_{ij} = 1$. The time for loading and unloading also equals $\epsilon_m = 1$ for all machines $m \in \{0, 1, 2, 3\}$.

Job	*1*		*2*	
Operation	*1*	*2*	*4*	*5*
Processing time	*2*	*3*	*3*	*1*
Machine	*1*	*2*	*1*	*2*

Table 6.3: Data for the instance given in Example 6.7

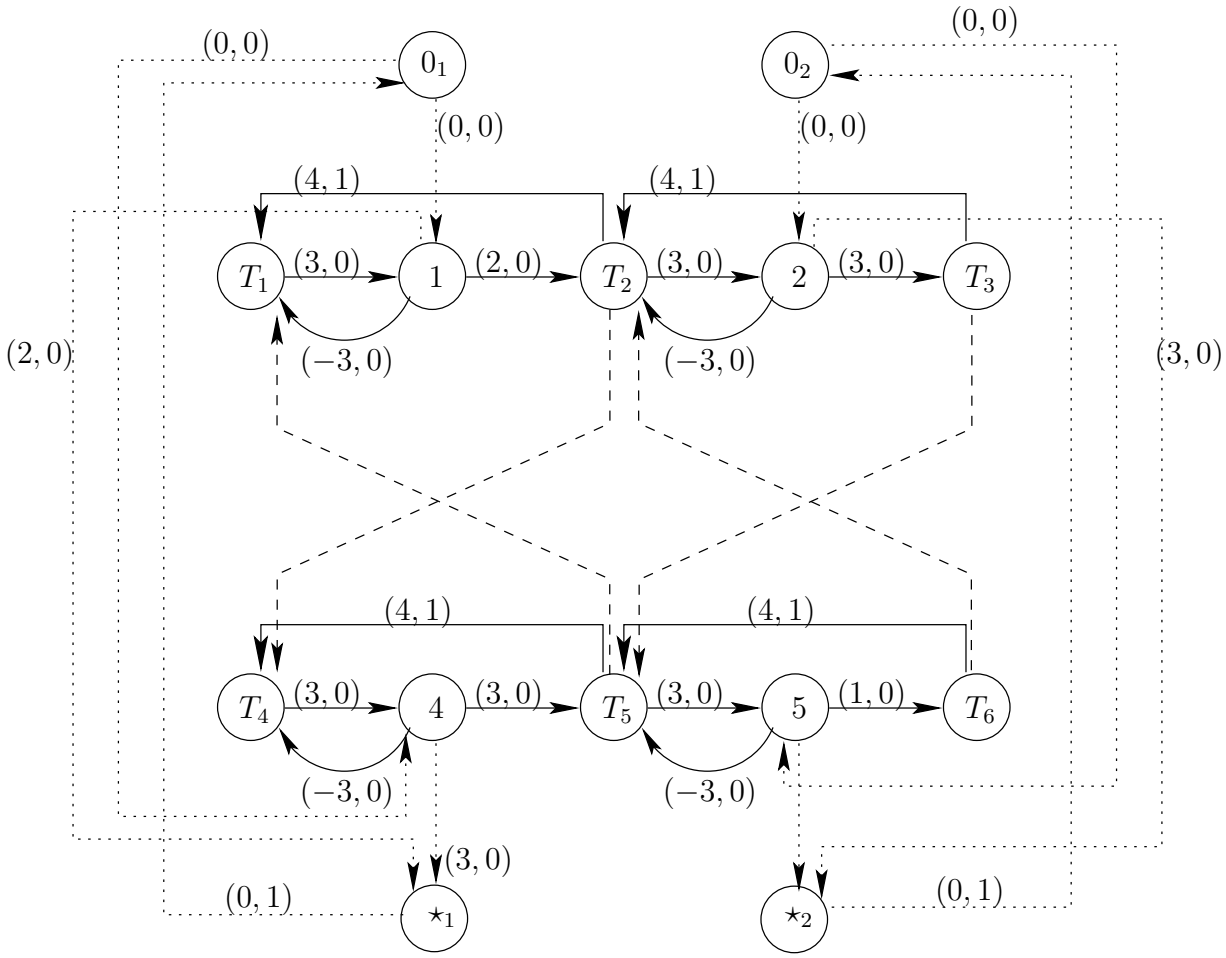

Figure 6.16: Precedence constraints for the Example 6.7

The graph described by the precedence constraints is given in Figure 6.16. Note that in the graph only the necessary constraints are shown. For example, the arcs given by (6.6) *are not drawn in the figure because these constraints are also modeled by the arcs given by the constraints* (6.11) *and* (6.4)*.*

Furthermore, the disjunctive arcs which are introduced by the constraint (6.2) *are not shown in the figure.*

As the robotic cell problems are based on the cyclic job shop problems with machine chains repetition, we must introduce the constraints between the dummy nodes $0_m, \star_m$ *for* $m = 1, 2$*. In the Figure 6.16 these constraints are shown in dotted lines.*

The dashed lines between the transport operations of the job 1 *and the job* 2 *model the disjunctive constraints between the corresponding operations* 1*,* 5 *and* 2*,* 6*. These constraints are derived from the reformulated constraints* (6.17) *and* (6.18)*. The reason for these constraints is the following. Assume the operation* $(1; k)$ *is the first operation which is scheduled on machine* 1*. As there exists no buffer at machine 1, the next operation* $(5; l)$ *can only start on machine 1, if the robot takes the job 1 to the second machine and takes job 2 from the input machine to machine 1.*

The following Table 6.4 shows the constraints leading to the arcs, which are drawn with a solid line in Figure 6.16.

Constraints	*Arcs*
Constraint (6.4)	$(T_1, 1)$, $(T_2, 2)$, $(T_4, 4)$, $(T_5, 5)$
Constraint (6.5)	$(1, T_1)$, $(2, T_2)$, $(4, T_4)$, $(5, T_5)$
Constraint (6.10)	$(1, T_2)$, $(2, T_3)$, $(4, T_5)$, $(5, T_6)$
Constraint (6.11)	(T_2, T_1), (T_3, T_2), (T_5, T_4), (T_6, T_5)

Table 6.4: Constraint-arc assignment

The gantt chart with the optimal cycle time $\alpha = 24$ *is given in Figure 6.17.*

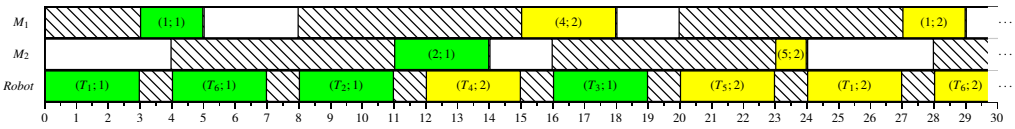

Figure 6.17: Optimal schedule for Example 6.7

Example 6.8 ***Hoist Scheduling Problem***

The instance consists of one job with three operations. Each operation of a job has to be processed on a machine $m = \{1, 2, 3\}$*. The data is given in Table 6.5. The time to move the hoist from any machine* i *to any machine* j*,* $i \neq j$*,* $i, j \in \{0, 1, 2, 3, 4\}$ *equals* $d_{ij} = 1$*. The time for loading and unloading also equals* $\epsilon_m = 1$ *for all machines* $m \in \{0, 1, 2, 3, 4\}$*.*

Job	*1*		
Operation	*1*	*2*	*3*
Minimal processing time	*2*	*10*	*8*
Maximal processing time	*3*	*12*	*10*

Table 6.5: Data for the instance given in Example 6.8

The graph given by the precedence constraints is given in Figure 6.18. The following Table 6.6 shows which constraints leads to which arcs.

Constraints	*Arcs*
Constraint (6.4)	$(T_1, 1)$, $(T_2, 2)$, $(T_3, 3)$
Constraint (6.5)	$(1, T_1)$, $(2, T_2)$, $(3, T_3)$
Constraint (6.10)	$(1, T_2)$, $(2, T_3)$, $(3, T_4)$
Constraint (6.11)	(T_2, T_1), (T_3, T_2), (T_4, T_3)
Time window constraint	$(T_2, 1)$, $(T_3, 2)$, $(T_4, 3)$

Table 6.6: Constraint-arc assignment

The optimal schedule is given in the gantt chart in Figure 6.19. The optimal cycle time is $\alpha = 22$*. The corresponding height of the disjunctive arcs is given in Table 6.7. Note that the disjunctive arcs create together with conjunctive arcs several circuits with negative height and negative delay, e.g. one circuit is* $(T_1, T_4, 3, T_3, 2, T_2, 1, T_1)$ *with delay* -30 *and height* -1*.*

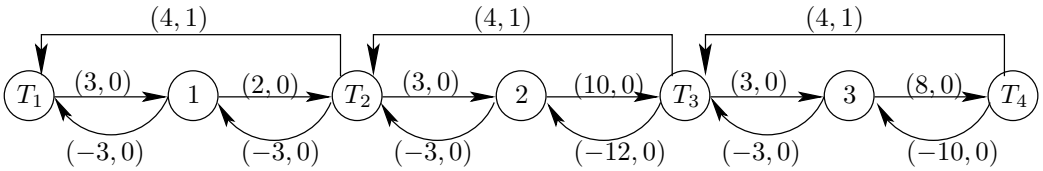

Figure 6.18: Precedence constraints for the Example 6.8

Arc	(T_1, T_3)	(T_1, T_4)	(T_2, T_4)
Height	*0*	*-1*	*-1*
Delay	*4*	*4*	*4*
Alternative Arc	(T_3, T_1)	(T_4, T_1)	(T_4, T_2)
Height	*1*	*2*	*2*
Delay	*4*	*4*	*4*

Table 6.7: Height of the disjunctive arcs

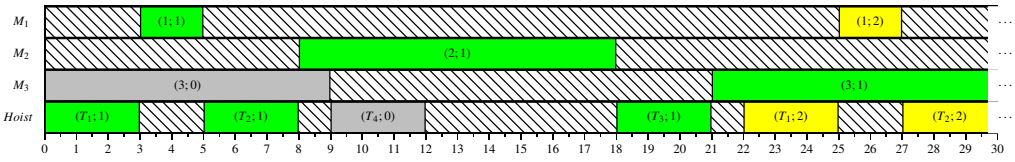

Figure 6.19: Optimal schedule for the Example 6.8

In robotic cell problems, the jobs are processed on machines as in a flow shop. In reentrant robotic cells, this is done in different ways. There exist several different kinds of reentrant shop problems without any transportation, which are mentioned in the literature. Lev and Adiri [46] considered a *V-shop*. This means that operations of a job are processed on the

machines in the following order: $M_1 \rightarrow M_2 \rightarrow \ldots \rightarrow M_{m-1} \rightarrow M_m \rightarrow M_{m-1} \ldots \rightarrow M_2 \rightarrow M_1$. Another possibility is to change the order to $M_1 \rightarrow M_2 \rightarrow \ldots \rightarrow M_m \rightarrow M_1$. This problem is called *chain-reentrant shop* (see Wang et al. [71]) or *loop-reentrant shop* (see Middendorf and Timkovsky [53]). Another variation is considered in Kubiak et al. [41]. This problem is called *hub-reentrant shop* and the operations of a job are processed on the machines in the following order $M_1 \rightarrow M_2 \rightarrow M_1 \rightarrow M_3 \rightarrow \ldots \rightarrow M_1 \rightarrow M_m \rightarrow M_1$. In Steiner and Xue [69] a **loop-reentrant robotic cell** is considered. They show that the problem of finding an optimal periodic schedule for a given instance with three or more machines is $\mathcal{NP}$-hard. Furthermore, they derive a polynomial algorithm for the two-machine case.

6.3 Software Pipelining

Another application of our framework can be found in the area of Software Pipelining (SP). SP is an excellent way to improve parallelism in loops on Very-Long Instruction Word (VLIW) processors [3]. This improvement is done during the compilation of a program. Here it often occurs that loops, especially for-loops, have to be repeated very often. Now the idea is to reschedule the instruction of the loop in such a way that the time of scheduling the loop body is minimized. The loop body is the part of a loop which has to be repeated very often. In the following we restrict this overview to single loops.

To get an idea of the problems in this area we start with an example taken from Fimmel and Müller [24].

Example 6.9 *There are given 8 different tasks $S_1, \ldots, S_8$. Each task has a processing time p_i and must be performed on a processor. In the area of SP we usually talk about processors instead of machines and of tasks instead of operations. The considered loop can be found in Listing 2. The initialization before the loop body or the kernel (line 4 to 11) is called prolog (line 1 and 2). The instructions after the loop are called epilog. In this example the epilog is empty.*

```
1  S5[0] = a;
2  S7[0] = b;
3  for i=1 to N do
4      S1[i] = X[i] + S5[i − 1];
5      S2[i] = S1[i] − S7[i − 1];
6      S4[i] = γ1 ∗ S2[i];
7      S5[i] = S4[i] + S5[i − 1];
8      S3[i] = S2[i] + S5[i];
9      S6[i] = γ2 ∗ S2[i];
10     S7[i] = S6[i] + S7[i − 1];
11     S8[i] = S3[i] − S7[i];
```

Listing 2: Listing for Example 6.9

Tasks S_1, S_2, S_3, S_5, S_7 *and* S_8 *need to perform on the adder processor. Therefore, the processing time of these tasks is* 1. S_4 *and* S_6 *need to be performed on the multiplication processor. The processing time of these operations is* 3.

The schedule for Listing 2 is given in Figure 6.20. The cycle time is $\alpha = 11$.

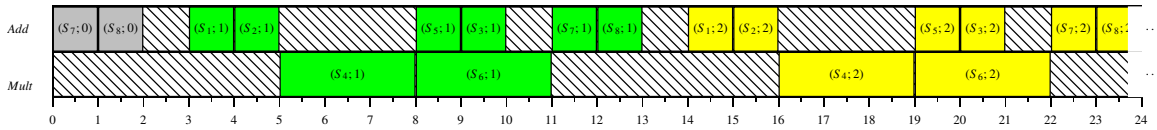

Figure 6.20: Schedule for Listing 2

Now we have to derive the precedence constraints from the described loop. Let us for example consider the tasks S_5. *To perform the* k*-th occurrence of task* S_5, *we need the* k*-th occurrence of* S_4 *and the* $k-1$*-th occurrence of* S_5. *Therefore, we get an arc between* S_4 *and* S_5 *with delay* 3 *and height* 0 *and another arc from* S_5 *to itself with delay* 1 *and height* 1. *The other precedence constraints can be derived in a similar way. Thus, we get the graph given in Figure 6.21 which describes the precedence constraints given in Listing 2. The optimal solution of the problem is given in Figure 6.22. The optimal cycle time is* 8.

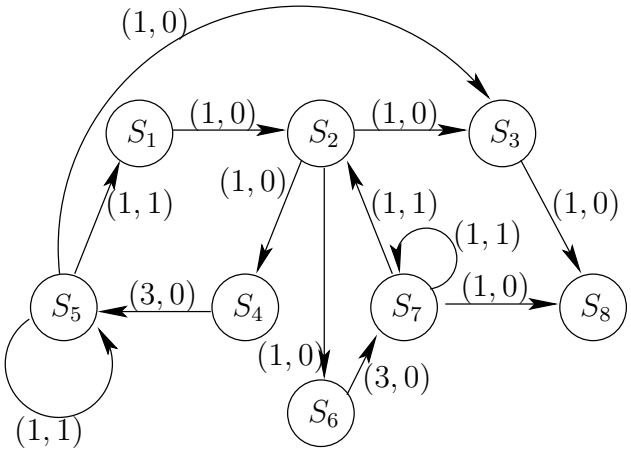

Figure 6.21: Precedence constraints for the Example 6.9

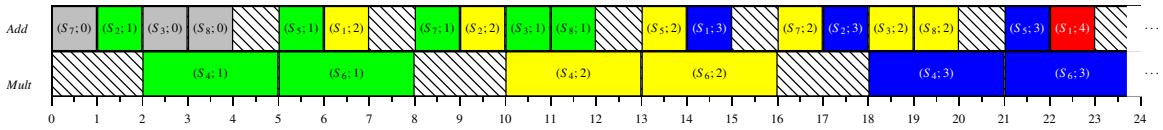

Figure 6.22: Optimal Schedule for Example 6.9

So we can rewrite the loop in Listing 2 to the new loop in Listing 3. As we can see, the prolog and epilog are longer than the prolog and epilog in the original listing. Also the cycle time reduces from $\alpha = 11$ *to* $\alpha = 8$*.*

```
1  S_5[0] = a;
2  S_7[0] = b;
3  S_1[1] = X[1] + S_5[0];
4  S_2[1] = S_1[1] - S_7[0];
5  S_6[1] = γ_2 * S_2[1];
6  S_4[1] = γ_1 * S_2[1];
7  S_5[1] = S_4[1] + S_5[0];
8  S_1[2] = X[2] + S_5[1];
9  for i=1 to N-2 do
10     S_7[i] = S_6[i] + S_7[i-1];
11     S_2[i+1] = S_1[i+1] - S_7[i];
12     S_3[i] = S_2[i] + S_5[i];
13     S_4[i+1] = γ_1 * S_2[i+1];
14     S_8[i] = S_3[i] - S_7[i];
15     S_5[i+1] = S_4[i+1] + S_5[i];
16     S_6[i+1] = γ_2 * S_2[i+1];
17     S_1[i+2] = X[i+2] + S_5[i+1];
18 S_7[N-1] = S_6[N-1] + S_7[N-2];
19 S_2[N] = S_1[N] - S_7[N-1];
20 S_3[N-1] = S_2[N-1] + S_5[N-1];
21 S_4[N] = γ_1 * S_2[N];
22 S_8[N-1] = S_3[N-1] - S_7[N-1];
23 S_5[N] = S_4[N] + S_5[N-1];
24 S_6[N] = γ_2 * S_2[N];
25 S_7[N] = S_6[N] + S_7[N-1];
26 S_3[N] = S_2[N] + S_5[N];
27 S_8[N] = S_3[N] - S_7[N];
```

Listing 3: Reformulated Listing for Example 6.9

The considered example describes a very special case of a software pipelining problem because usually each task needs several processors for its execution or there exist several processors which all can perform the same tasks. The first case can be easily integrated by using the concepts introduced in Section 3.3. So we can consider each task as a multiprocessor task. The second case can also be integrated in our framework by introducing the concept of parallel machines. The only problem is that the idea of parallel machine does not fit directly into our framework. However, if we use a two-stage approach, we can still use our framework. In a first step, we assign to each task a processor. After

this assignment, we have again a problem with dedicated processors. Therefore, in the following we do not consider problems with parallel processors.

So summarizing, problems in the area of software pipelining can be easily integrated into our framework.

Another important problem is register allocation in combination with software pipelining techniques. In the area of SP registers are used to store results of already performed operations. The main advantage of using registers to store the results instead of using the normal memory is the fast writing and reading access of the information in the registers. The concept of writing and reading results from and to the memory instead of using the register is called spilling. To perform spilling, additional instructions need to be created. Therefore, using the register instead of perform spilling is valuable.

Finding an optimal register allocation which uses only k registers for given dependencies between the operations is a $\mathcal{NP}$-hard problem (Sethi [66]). Normally, this assignment is done by using some heuristic based on graph coloring methods (Chaitin et al. [13]). The idea for the heuristic is to consider an undirected graph, where the node set consists of all tasks. If we should not assign a result of an instruction to the same register as the result of another instruction, because both results must be accessible at the same time, then we have to introduce an undirected arc between both instructions. To get a feasible and optimal assignment of the instructions to the registers, we assign to each node in the graph a color so that two nodes which are connected by an arc have a different color. Additionally, the number of colors are limited by k, the number of the registers.

In the following, we propose a way to cover the register requirement within our framework. This is done in a two stage approach because we can consider a register as a parallel machine and therefore, as we only consider problems with dedicated machines, we need in a first step to assign the tasks to a register. During the assignment, we have to consider the number of the registers we can use. This first assignment is done by a heuristic. In a second step, we need to compute the optimal schedule based on the previous register allocation and on the precedence constraints which are derived from the listing of a loop.

Now the only question is how to include the register allocation into the graph of the precedence constraints. Consider the following situation. The two tasks i and i' need to write their result to the same register r. Furthermore, the tasks $i_1, \ldots, i_n$ ($i'_1, \ldots, i'_n$) need to read the result of task i (i') from the register r.

We can model this situation within our framework as follows: We create two additional tasks $s(i)$ and $s(i')$ which need to be processed on the register r. These tasks represent the writing to the register. The processing time of both tasks is zero and the task i (i') is connected with $s(i)$ ($s(i')$) with length p_i ($p_{i'}$) and height 0. Furthermore, the task $s(i)$

($s(i')$) is connected with the task i_a (i'_a) for all $a = 1, \ldots, n$ with length 0 and height $H_{i\,i_a}$ ($H_{i'\,i'_a}$). Additionally, as the task i (i') needs to write the result directly to the register r after the processing on the processor $M(i)$ ($M(i')$) is finished, the task $s(i)$ ($s(i')$) is connected to i (i') with length $-p_i$ ($-p_{i'}$) and height 0.

If the result of the task i is written first to the register r, then the result of task i' can be written to register r as soon as all succeeding operations $i_1, \ldots, i_n$ of $s(i)$ are started, which means that these tasks have read the result of task i from the register. Thus, this situation describes a general blocking situation. So, we get the following constraints

$$\begin{aligned} & t(s(i); l) \geq \max\{t(i'_{a'}; k + H_{s(i')\,i'_{a'}}) | a' = 1, \ldots, n\} \\ & \vee\, t(s(i'); k) \geq \max\{t(i_a; l + H_{s(i)\,i_a}) | a = 1, \ldots, n\} \end{aligned} \tag{6.19}$$

for all tasks $s(i), s(i')$ with $M(s(i)) = M(s(i'))$ and $k, l \in \mathbb{Z}$.

This is equivalent to

$$t(s(i); l) \geq t(i'_{a'}; k + H_{s(i')\,i'_{a'}}) \text{ for all } a' = 1, \ldots, n$$

$$\vee\, t(s(i'); k) \geq t(i_a; l + H_{s(i)\,i_a}) \text{ for all } a = 1, \ldots, n$$

for all $s(i), s(i')$ with $M(s(i)) = M(s(i'))$, $k, l \in \mathbb{Z}$.

Note that in the proposed case the writing operation starts right after the preceding operation is finished. By changing the length, we can also model more general situations. As the writing operations are blocking operations, we also connect all succeeding operations $i_a, a = 1, \ldots, n$ of $s(i)$ with the writing operation $s(i)$ with length p_{i_a} and height 1 because the next occurrence of the writing operation can only start if all succeeding operations are finished.

Therefore, by using the concepts of Section 4 we can include the register allocation problem into our framework. By solving the described problem, we can get two results. First, we can find out that the previous register assignment which is done in the first stage leads to an infeasible schedule or second, we can find a feasible schedule which satisfies all the given constraints.

In the following, we present an example.

Example 6.10 *We extend the Example 6.9 by introducing five different registers. As the $k+1$-th occurrence of the operations S_5 and S_7 need the result of their k-th occurrence, the writing operation $s(S_5)$ and $s(S_7)$ must be assigned to a register so that each writing operation is the only writing operation which writes into this register. The assignment of the writing operations to the register is shown in Table 6.8.*

The optimal solution with cycle time $\alpha = 9$ is given in Figure 6.23.

Register 1	$s(S_1), s(S_4)$
Register 2	$s(S_5)$
Register 3	$s(S_6), s(S_3), s(S_8)$
Register 4	$s(S_7)$
Register 5	$s(S_2)$

Table 6.8: Assignment to the register

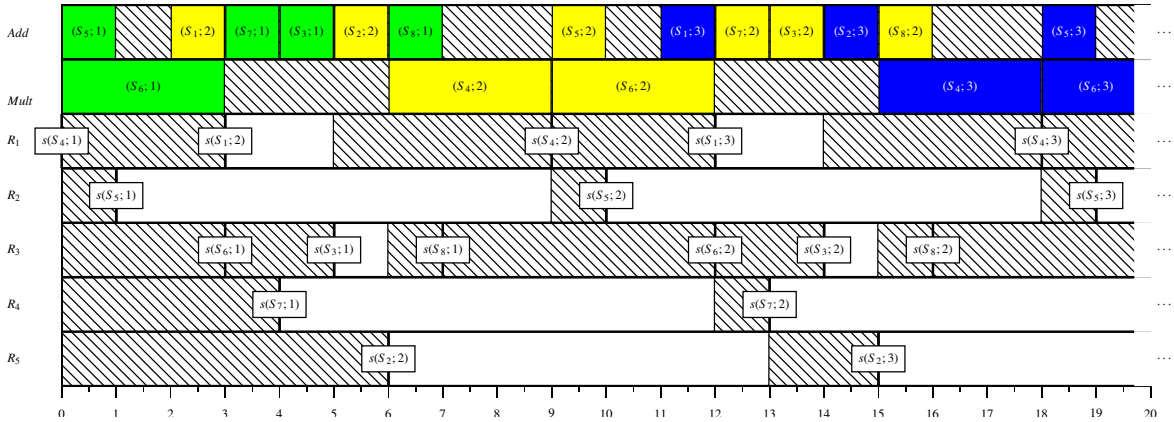

Figure 6.23: Optimal schedule for the problem 6.10

7 Solution Methods

In this section, we present two different approaches to solve the applications which are described in Section 6.

7.1 Solving the Problem With MILP Solver

As we developed for each application a corresponding mixed integer linear program (MILP), it is a straightforward idea to solve the MILP by Cplex (see [1]), which is a MILP solver. Cplex is one of the leading and well-known products to solve MILP in the scheduling community (see e.g. Brucker and Knust [10, 11]). In our experiments, it turns out that Cplex is unable to solve larger instances (more than 10 jobs and 5 machines) in a reasonable time. The only application which can be solved by Cplex is the single hoist-single-product problem. Here it turns out that with our new mixed integer formulation the problem can be solved much faster compared to the formulation which is developed by Phillips and Unger [57] and which is still used to describe the problem in recent papers (see Leung et al. [45]).

Using the two different formulations, we applied Cplex to solve benchmark problems listed in Leung et al. [45]. The tests were performed on a Celeron 1.8GHz computer with operating system Linux and 256MB memory. Table 7.1 summarizes the results of these tests.

	Formulation of [57]		New Formulation	
Instances	CPU time in sec	Number of nodes	CPU time in sec	Number of nodes
P& U	116.911	59816	9.17	789
Cu	1.096	29	0.03	0
Zinc	4.936	1455	0.14	0
Mini	1.266	538	0.41	54
BO1	5.7	2922	3.37	316
BO2	5.968	3286	2.14	398

Table 7.1: Comparison of the different formulation for the single product-single hoist problem

The main result is that the new mixed integer formulation outperforms the previous known formulation.

As the single product-single hoist problem is the only problem which can be solved by Cplex also for larger instances, we need to develop a different approach to solve cyclic machine scheduling problems.

7.2 Solving the Problem With a Meta-Heuristic

To solve the remaining applications like the different cyclic job shop problems, the cyclic job shop problem with blocking and with and without transportation robots, we apply the tabu search method, which was already applied successfully to various non-cyclic scheduling problems (see e.g. Hurink and Knust [35], Nowicki and Smutnicki [56]).

The *tabu search method* is a metaheuristic, which was designed by Glover [25, 26]. In each iteration of this local search method, the current solution is usually replaced by a solution in its neighborhood. The *neighborhood* of a solution s describes the solutions which can be reached in one step from s. A neighborhood is called *opt-connected* if we can get from any feasible solution to the optimal solution by a finite number of steps.

Contrary to the iterative improvement method, also non-improving solutions are accepted during the search process. Thus, it is possible to leave local minima, but it is also possible to visit a solution once again. Therefore, to avoid this kind of cycling, a *tabu list* TL is used which typically stores easy attributes characterizing solutions that should not be considered again for a certain length of time. Here it is important to mention that not a complete description of a solution is stored in the tabu list.

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, an *aspiration criterion* is introduced. This criterion allows the acceptance of a neighbor even if it is forbidden due to the tabu list.

The whole tabu search method stops when a given time limit is exceeded, a maximum number of non-improving solutions are considered or a solution is found which has the same cycle time as the computed lower bound.

In Section 7.4 we discuss the different neighborhoods which we applied to solve the presented problems. Before the description of the neighborhoods, we describe the underlying search space in the following section.

7.3 The Search Space

In the following, we describe the search space for our local search approach. The most general problem we solve is a cyclic job shop problem with blocking and time-window

constraints. If there exists also a transport robot, then the transport operations can be regarded as additional operations, which must be performed on a special machine, namely the transport robot. Therefore, we describe the search space for cyclic job shop problems with blocking and time-window constraints.

Let α together with a vector $HX = (HX_{b(i)\,i'})$ be an optimal solution for a cyclic problem with blocking (4.14) to (4.18). Then α is the minimal solution value of the following GBCSP induced by the $HX_{b(i)\,i'}$-values.

$$min \;\; \alpha \tag{7.1}$$

s.t.

$$t_i + p_i - \alpha HX_{ii'} \;\leq\; t_{i'} \quad (i,i') \in E \tag{7.2}$$

$$t_{b(i)} + p_i^b - \alpha HX_{b(i)\,i'} \;\leq\; t_{i'} \quad (b(i),i') \in D \tag{7.3}$$

with

$$b(i) := \begin{cases} s(i) & \text{if } i \text{ is a blocking operation and } s(i) \neq \star \\ i & \text{otherwise} \end{cases}$$

and

$$p_i^b := \begin{cases} 0 & \text{if } i \text{ is a blocking operation and } s(i) \neq \star \\ p_i & \text{otherwise.} \end{cases}$$

The set D consists of arcs between operations $b(i)$ and i', where i and i' are processed on the same machine $M(i)$.

D is called the set of **disjunctions** while the arcs in E are called **conjunctions**. So, a solution of the problem can be described by a graph $G = (T, E \cup D)$.

Definition 7.1 *A height function $HX : E \cup D \to \mathbb{Z}$ is called **relevant** if the following condition is fulfilled:*
$HX_{ii'} = H_{ii'}$ for all $(i,j) \in E$ and
$HX_{b(i)\,i'} + HX_{b(i')\,i} = 1$ for the pairs $\{(b(i),i'),(b(i'),i)\} \in D$.

We define a consistent height function in the following way:

Definition 7.2 *A height function $HX : E \cup D \to \mathbb{Z}$ is called **consistent** if for each circuit μ one of the following conditions is fulfilled:*

i. The circuit μ has positive height and arbitrary delay.

ii. The circuit μ has zero height and non-positive delay.

iii. The circuit μ has negative height and negative delay.

Furthermore, the height function is called ***feasible (for a given value*** α***)*** *if additionally, the following inequality*

$$\min\{\frac{L(\mu)}{HX(\mu)}|\mu \text{ is a circuit with } HX(\mu) < 0\} \geq$$

$$\alpha \geq \max\{\frac{L(\mu)}{HX(\mu)}|\mu \text{ is a circuit with } HX(\mu) > 0\} \tag{7.4}$$

holds.

Now we introduce the upper bound for the height of an arc $(b(i), i') \in D$.

Assumption 7.3 *There exists a finite value $HX^+_{b(i)\,i'}$ for each arc $(b(i), i') \in D$ with the following property: If we increase the height of an arc $(b(i), i') \in D$ to a value not greater than $HX^+_{b(i)\,i'}$ and therefore, decreases the height of the alternative arc $(b(i'), i)$ by the same amount, then the height function HX is kept consistent. We call $HX^+_{b(i)\,i'}$ the* ***upper bound*** *for the height of the arc $(b(i), i') \in D$.*

In the following sections, we show that the upper bound is a finite value.

7.4 Neighborhoods

This section is divided into several parts. In the first very short part, some basic properties for the local search approach are mentioned. Then we develop and describe the neighborhoods for the different applications. We start with the different cyclic job shop problems without blocking. These concepts are then reformulated for problems with blocking. Afterwards, we adjust the neighborhoods which we develop for the previous two problems to the problem with one transportation robot and blocking. Finally, we consider problems with blocking, one transportation robot, and time-window constraints.

7.4.1 Basic Properties for the Local Search Methods

Let $HX : E \cup D \rightarrow \mathbb{Z}$ be some height function which is feasible for some optimal solution value α for the corresponding GBCSP. Due to Theorem 2.4 there exists at least one circuit μ with value $\alpha = \frac{L(\mu)}{HX(\mu)}$. As mentioned in Section 2 we call this circuit critical.

Assume that the height function keeps to be consistent if we decrease HX_{ij} for some disjunctive arc $(i,j) \in D$ by some positive integer value x. Then the corresponding value of μ increases, which can be seen as follows:

Let μ_1 be the sub-path of μ connecting j with i. Then we get

$$\frac{L(\mu)}{HX(\mu)} = \frac{L(\mu)}{HX(\mu_1) + HX_{ij}} < \frac{L(\mu)}{HX(\mu_1) + HX_{ij} - x}.$$

Thus, we get the following theorem:

Theorem 7.4 *Let HX be a feasible height function and μ be a critical circuit. Then the value of the critical circuit can decrease only if we increase the height of a disjunctive arc $(i,j) \in D$ at least by one.*

In the following we first discuss the neighborhoods for the the different cyclic job shop problems without blocking.

7.4.2 Neighborhoods for Cyclic Job Shop Problems Without Blocking

Notice that for cyclic job shop problems without blocking $L_{ij} = p_i \, \forall \, (i,j) \in E \cup D$ and $p_i > 0$ for all operations $i \in T$ hold. As we mentioned in Definition 7.3, the height function keeps to be consistent when increasing HX_{ij} for a disjunctive arc (i,j) only if $HX_{ij} < HX_{ij}^{+}$ holds.

We now give some conditions under which $HX_{ij} < HX_{ij}^{+}$ holds.

Theorem 7.5 *Let HX be a feasible height function and a be a disjunctive arc from i to j on a corresponding critical circuit in $(T, E \cup D)$. Assume that the conditions*

$$p_i = L_a < \min\{L(\mu) | \mu \textit{ is a path from } i \textit{ to } j \textit{ with at least two arcs}\} \tag{7.5}$$

and

$$L_{ij} \geq p_i \, \forall \, (i,j) \in E \tag{7.6}$$

hold. Then $HX_a < HX_a^{+}$.

Proof: Consider a critical circuit μ. If in μ a disjunctive arc a exists which has the same length and height as a parallel conjunctive arc a', we replace a by a' in the critical circuit. Now assume that in μ a disjunctive arc $a = (i,j)$ exists, otherwise the schedule is

optimal. The sub-path μ_1 of μ from j to i and the arc a form together the critical cycle μ. The corresponding back-going arc from j to i is denoted by b. Thus, $HX_a + HX_b = 1$. As HX is consistent and $L(\mu) > 0$, we also have $HX(\mu) \geq 1$.

Let τ_1 be an arbitrary path from i to j which does not contain the arc a and let τ be the circuit consisting of τ_1 and b. This situation is shown in Figure 7.1.

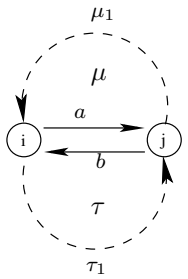

Figure 7.1: Part of the circuits μ and τ

We have to show that $HX(\tau_1) + HX_b > 1$ holds for any arbitrary path from i to j which does not contain the arc a. Then we can reduce the height of the arc b by at least one, and therefore, increase the height of the arc a by at least one. After this change the height function is kept consistent because $HX(\tau_1) + HX_b - 1 \geq 1$ holds. Thus, $HX_a < HX_a^+$.

Because HX is consistent we must have

$$HX(\tau) = HX(\tau_1) + HX_b \geq 1$$

which is equivalent to $HX(\tau_1) \geq HX_a = 1 - HX_b$. Now assume that $HX(\tau_1) = HX_a$. We distinguish between two different cases. In the first case the path τ_1 consists of at least two arcs and in the second case the path τ_1 is a single conjunctive arc c from i to j.

Let us consider the first case. Then

$$\frac{L(\mu)}{HX(\mu)} = \frac{L_a + L(\mu_1)}{HX(\mu_1) + HX_a} \geq \frac{L(\tau_1) + L(\mu_1)}{HX(\mu_1) + HX(\tau_1)} = \frac{L(\tau_1) + L(\mu_1)}{HX(\mu_1) + HX_a} \tag{7.7}$$

holds, because the composition of τ_1 and μ_1 defines a circuit and μ is a critical circuit. The denominator $HX(\mu) = HX(\mu_1) + HX_a$ is greater than one. Thus, (7.7) is equivalent to $L_a + L(\mu_1) \geq L(\tau_1) + L(\mu_1)$ or $L_a \geq L(\tau_1)$ contradicting the assumption.

Now let us consider the second case. The path τ_1 is a single conjunctive arc c. As we assumed that $HX_{\tau_1} = HX_a$, we have $HX_c = HX_a$ and due to condition (7.6) $L_c \geq p_i$. Therefore, we would have chosen the arc c instead of the arc a in the critical cycle μ. □

The conditions (7.5) and (7.6) of Theorem 7.5 are satisfied if $L_{ij} = p_i$ for all $(i,j) \in E \cup D$ and $p_i > 0 \, \forall i \in T$ hold, which is the case for all considered application without blocking. Thus, based on Theorem 7.5 we can now increase every height of a disjunctive arc at least by one and at most by $HX^+_{ij} - HX_{ij}$ and the new height function is still consistent.

Based on this observation we can define our first neighborhood N_1. N_1 assigns to each feasible height function HX the set $N_1(HX)$ of all height functions derived from HX by choosing an arc $(i,j) \in D$ on a critical circuit and replace the corresponding height HX_{ij} by $HX_{ij} + x$ where x is a positive integer value with $1 \leq x \leq HX^+_{ij} - HX_{ij}$.

Now we show that the neighborhood N_1 is opt-connected.

Theorem 7.6 *The neighborhood N_1 is opt-connected.*

Proof: We have to show that we can get from any feasible height function HX to an optimal height function HX^* by a finite sequence of moves. This is accomplished by the following procedure. Assume that HX is the currently reached height function. Then we choose an arc $(i,j) \in D$ on a critical circuit with respect to HX such that $HX_{ij} < HX^*_{ij}$ and replace HX_{ij} by $\overline{HX}_{ij} = HX_{ij} + x$ and HX_{ji} by $\overline{HX}_{ji} = HX_{ji} - x$ with $1 \leq x \leq \min\{HX^*_{ij}, HX^+_{ij}\}$. $\overline{HX}$ is feasible and consistent because of Theorem 7.5. If no such arc exists, then for all arcs (i,j) on a critical circuit μ we have $HX_{ij} \geq HX^*_{ij}$. Thus,

$$\frac{\sum_{(i,j)\in\mu} L_{ij}}{\sum_{(i,j)\in\mu} HX_{ij}} \leq \frac{\sum_{(i,j)\in\mu} L_{ij}}{\sum_{(i,j)\in\mu} HX^*_{ij}}$$

which implies that HX must be optimal as well because μ is critical.

Otherwise, due to $\overline{HX}_{ij} = HX_{ij} + x \leq HX^*_{ij}$ and

$$\overline{HX}_{ji} = HX_{ji} - x = 1 - HX_{ij} - x = 1 - \overline{HX}_{ij} \geq 1 - HX^*_{ij} = HX^*_{ji}$$

the distance

$$\sum_{(i,j)\in D} |HX_{ij} - HX^*_{ij}|$$

decreases by $2x \geq 2$. Therefore, the procedure must reach an optimal solution after a finite number of steps. □

Furthermore, we derive further neighborhoods based on a block theorem which generalizes a corresponding block theorem for the classical job-shop problem (see Brucker [7], Theorem 6.18).

Let HX be a feasible height function and let μ be a critical circuit with respect to HX. A sub-path of μ containing at least one arc is called *block*, if it is a maximal sub-path containing only disjunctive arcs.

Theorem 7.7 *Let μ be a critical circuit for a feasible height function HX and let (j,k) be an arc of a block B in μ such that (j,k) is different from the first and last arc in B. Then the cycle time is not improved by increasing HX_{jk} by any integer x with $HX^{+}_{jk} - HX_{jk} \geq x \geq 1$.*

As we generalize this theorem in Section 7.4.3, we do not give the proof for this theorem here. However, we present now in a very short form the idea of the proof: Let the block B in μ consists of the nodes (i,j,k,l). Then, we can show that after increasing the height of HX_{jk} the value of the circuit μ' is greater or equal to the value of μ, where μ' is equal to μ except the block B is replaced by (i,k,j,l).

Based on Theorem 7.7 we define a neighborhood N_2 which only increases the HX_{ij}-values of the first or last arcs of a block. It is still an open question even for the non-cyclic case whether the neighborhood is opt-connected.

A direct conclusion of Theorem 7.7 is the following lemma.

Lemma 7.8 *Let μ be a critical circuit for a feasible height function HX and let i_k be not the first or the last operation in a block B in μ. If we do not change the height of the arcs (l,i_k) for all predecessors l of i_k in B by at least one or the height of the arcs (i_k,l) for all successors l of i_k in B by at least one, then the cycle time is not improved.*

Proof: We denote the cycle time before changing any height with α. In the following we show that if the height of all arcs starting at a predecessor of i_k in B are not increased, then the new cycle time is greater or equal to the value of the circuit μ. The block B is given in Figure 7.2. Note that the heights of the arcs (i_{k-l}, i_{k-l-1}), $(i_{k-l-1}, i_{k-l-2}), \ldots, (i_{k-2}, i_{k-1})$ are never changed during this process.

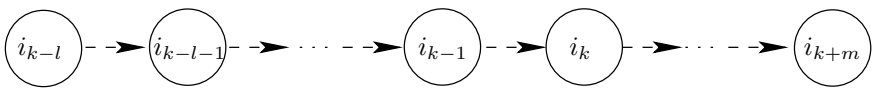

Figure 7.2: Block B in proof of Lemma 7.8

Assume the height of at least one of the arcs (i_{k-b}, i_k) with $b \in \{1, \ldots, l\}$ is not increased. Let the arc (i_{k-a}, i_k) be the arc with the lowest index a with this property.

We show this result by induction. We start with $a = 1$. As the arc (i_{k-1}, i_k) is not changed, there exists no arc in the critical circuit which is changed. So, the new cycle time α_1 is greater or equal to the cycle time α.

The statement holds now for $a = r$. This means that all arcs (i_{k-c}, i_k) with $c < r$ are increased at least by one and the cycle time α_r is greater or equal to the value of the circuit μ, where in μ the block B is replace by the block B^r. Due to Theorem 7.7 and the idea of the proof this value is greater or equal to α. The block B^r is given in Figure 7.3.

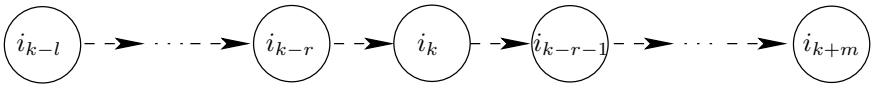

Figure 7.3: Block B^r in proof of Lemma 7.8

Now we show that the statement holds also for $a = r + 1$. As the arc (i_{k-r}, i_k), which height is increased, is not the first or the last arc in the block B^r, due to Theorem 7.7 the cycle time α^{r+1} is greater or equal to α^r and therefore, greater or equal to α. □

A direct consequence of the previous lemma is the following theorem.

Theorem 7.9 *Let S together with the height function HX^S and cycle time α_S be a feasible solution for a cyclic job shop problem. Let S' together with the height function $HX^{S'}$ be another feasible solution with cycle time $\alpha_{S'} < \alpha_S$.*

Then, there exists at least one operation k in a block B in a critical circuit μ of S for which the height of the arcs (l, k) is increased by at least one for all predecessors l of k in the block B or the height of the arcs (k, l) is increased by at least one for all successors l of k in the block B.

Due to this observations, we introduce a neighborhood N_3 which is defined by the following moves defined for the interior vertices j of blocks of a critical circuit. There are two neighbors associated with j. The first neighbor is constructed by increasing $HX_{i_1 j}$ by one, where i_1 is the first predecessor of j in the block. The height can be always increased because of Theorem 7.5. Then we increase $HX_{i_2 j}$ by one, if $HX_{i_2 j} < HX^+_{i_2 j}$, where i_2 is the predecessor of i_1 in the block. If $HX_{i_2 j}$ is increased, then we increase $HX_{i_3 j}$

by one if $HX_{i_3\,j} < HX^+_{i_3\,j}$, etc. Finally if $HX_{i_{l-1}\,j}$ is increased, we increase $HX_{i_l\,j}$, if $HX_{i_l\,j} < HX^+_{i_l\,j}$ holds, where i_l is the first vertex in the block. Symmetrically, the second neighbor is constructed by increasing successively $HX_{j\,k_1}, HX_{j\,k_2}, \ldots, HX_{j\,k_h}$ where k_h is the last vertex in the block.

Theorem 7.10 *The neighborhood N_3 is opt-connected.*

Proof: The proof is similar to the proof of Theorem 7.6. Let HX^* be an optimal height function and HX be the actual height function. Let μ be a critical circuit with respect to HX. If HX is not optimal, then due to Theorem 7.9 there must exist at least one node k in a block B in μ for which either the height of all arcs (l,k) is increased by one for all predecessors l of k in B or the height of all arcs (k,l) is increased by one for all successors l of k in B. Assume that the height for all predecessors of k must be increased. Thus, $HX_{lk} < HX^*_{lk}$ for all predecessors l of k in B. Therefore, the distance

$$\sum_{(i,j)\in D} |HX_{ij} - HX^*_{ij}|$$

decreases by at least 2 because due to Theorem 7.5 the height of the arc (a,k) can always be increased, where a is the direct predecessor of k. □

These three different neighborhoods are applied to the following applications: cyclic job shop, cyclic job shop with job repetition and cyclic job shop with machine repetition. The computation results are given in Section 8.

Now we describe several neighborhoods for the problems with blocking.

7.4.3 Neighborhoods for Cyclic Job Shop Problems With Blocking

In this section, we first derive neighborhoods for the cyclic job shop problem and for the cyclic job shop problem with job repetition. We assume that each operation of a job is processed on a different machine. After this description, these neighborhoods are adapted to the cyclic job shop problems with machine repetition.

Neighborhoods for the Cyclic Job Shop Problem With Blocking

Unfortunately, due to the blocking restrictions, the condition Theorem 7.5 does not hold anymore. This can also be seen in the following example. There exists a disjunctive arc on a critical circuit which cannot be increased because the height of the arc is equal to the upper bound. Thus, increasing the height would lead to an infeasible solution.

Example 7.11 *The example consists of 2 jobs. The first job consists of five operations, where the second job consists of 2 operations. The data for the problem is given in Table 7.11.*

Jobs	*1*					*2*	
Operations	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Processing time	*3*	*3*	*4*	*2*	*3*	*1*	*1*
Machine	*1*	*2*	*3*	*1*	*2*	*1*	*2*

Table 7.2: Data for Example 7.11

The precedence constraints are given by the graph G in Figure 7.4. Note that not all conjunctive arcs imposed by the loop constraints (see (4.12)*) are shown in the graph G. As an example the arc $(7, 6)$ is shown. The height of this arc is 1. The height of the arc $(\star, 0)$ is also 1.*

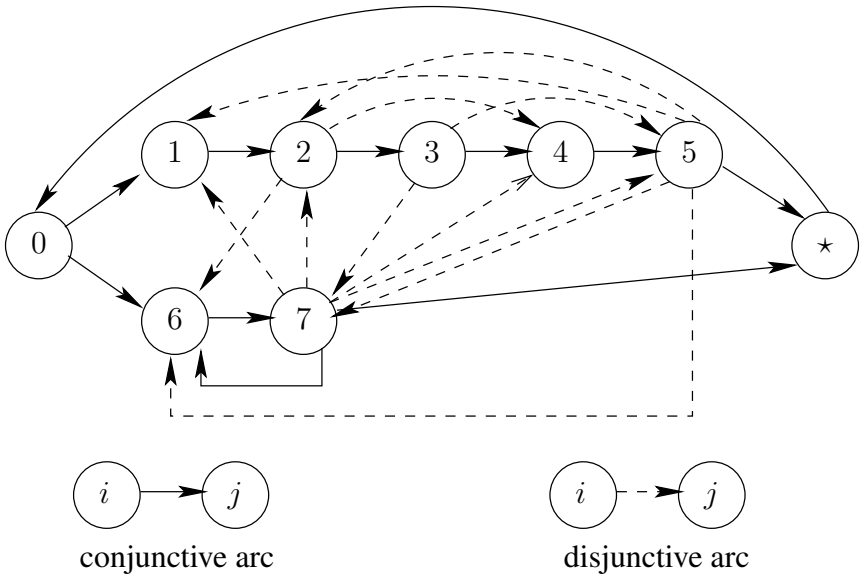

Figure 7.4: Precedence constraints for Example 7.11

The heights of all other conjunctive arcs are zero.

The heights and the delays of the disjunctive arcs are given in Table 7.3.

Arc	*(2,6)*	*(2,4)*	*(3,7)*	*(3,5)*	*(5,6)*	*(5,7)*
Height	*0*	*0*	*0*	*0*	*0*	*0*
Delay	*0*	*0*	*0*	*0*	*0*	*3*
Alternative arc	*(7,1)*	*(5,1)*	*(7,2)*	*(5,2)*	*(7,4)*	*(7,5)*
Height	*1*	*1*	*1*	*1*	*1*	*1*
Delay	*0*	*0*	*1*	*3*	*0*	*1*

Table 7.3: Height of the disjunctive arcs

The corresponding gantt chart with cycle time $\alpha = 16$ *is given in Figure 7.5. The cycle time of the optimal solution, which is computed by solving the corresponding mixed integer program, is* $\alpha = 15$*. The gantt chart of the optimal solution is given in Figure 7.6. Note that after each occurrence of operation* 6 *the machine* M_1 *is blocked. This is also shown in the gantt chart.*

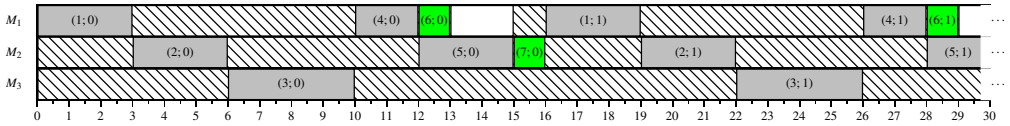

Figure 7.5: Gantt chart for Example 7.11

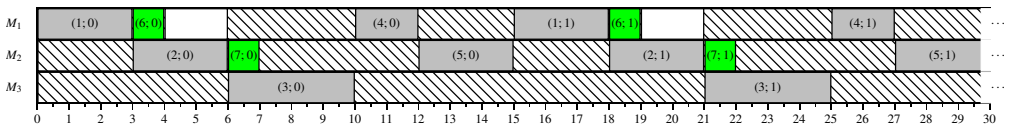

Figure 7.6: Optimal solution

A critical circuit is $(1, 2, 3, 4, 5, 7, \star, 0, 1)$ *and the value is* 16*. The only disjunctive arc on this circuit is* $(5, 7)$*. To get a possibly better solution we need to increase the height of this arc to* 1 *and decrease the height of the arc* $(7, 5)$ *to zero. However, this would lead to an infeasible solution because the height of the circuit* $(5, 6, 7, 5)$ *is zero, where the delay is positive. Due to Theorem 2.1 we get an infeasible solution.*

The delay of the path $\mu = (5, 6, 7)$ *is equal to* $L_{56} + L_{67} = 1$ *and the delay of the arc* $(5, 7)$ *is equal to* $L_{57} = 3$*. Thus, the constraint* (7.5) *of Theorem 7.5 is violated.*

We usually cannot find disjunctive arcs on a critical circuit which can be increased. Therefore, we need to find a way to increase the height of an arc even if the new height is larger than the upper bound.

First, we generalize Theorem 7.7 for blocking situations.

Let HX be a feasible height function and let μ be a critical circuit with respect to HX.

A sub-path in μ containing at least one arc is called **block** if the following conditions are fulfilled:

- Two operations i and i' which are processed successively on the same machine are connected either by a disjunctive arc from i to i', where operation i is a nonblocking operation or has no successor, or by a conjunctive and a disjunctive arc, where operation i is a blocking operation.
- The sub-path is a maximal sub-path with this property.

Theorem 7.12 *Let* μ *be a critical circuit for a feasible height function* HX *and let* $(b(k), l)$ *be an arc of a block* B *in* μ *such that* $(b(k), l)$ *is different from the first and last arc in* B*. Then the cycle time is not improved by increasing* $HX_{b(k)\,l}$ *by any integer* $x \geq 1$.

Proof:

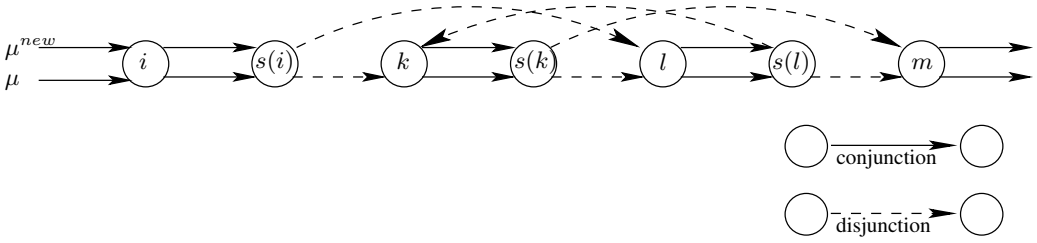

Figure 7.7: The block B in the proof of Theorem 7.12

First, for two operations c and d which are processed on the same machine, we need to define a path τ_{cd} from operation c to d. τ_{cd} consists of three operations c, $s(c)$ and d, if c is a blocking operation, otherwise path τ_{cd} consists of two operations c and d. The delay and height of this path τ_{cd} is $L(\tau_{cd}) = p_c$ and $HX(\tau_{cd}) = HX_{b(c)\,d}$, respectively.

Let us consider a situation, where operations i, k, l and m are processed on the same machine. In Figure 7.7 i, k and l are assumed to be blocking operations. In the following, we consider a more general situation.

Let τ_{im}^{old} be the path consisting of the sub-paths τ_{ik}, τ_{kl} and τ_{km}. We denoted the height of the path τ_{im}^{old} with respect to HX by

$$HX^{old} = HX(\tau_{ik}) + HX(\tau_{kl}) + HX(\tau_{lm})$$

and let HX^{new} be the height of the path τ_{im}^{new} which consists of the sub-paths τ_{il}, τ_{lk} and τ_{km} after increasing the height of the arc $(b(k), l)$ (i.e. decreasing the height of $(b(l), k)$.

Furthermore, let μ' be the circuit if in μ the path τ_{im}^{old} is replaced by τ_{im}^{new} and let $HX^{new}(\mu')$ be the height of μ' after increasing $HX_{b(k)\,l}$.

Then it is sufficient to show that

$$HX^{new} \leq HX^{old} \tag{7.8}$$

because this implies that

$$\frac{L(\mu)}{HX(\mu)} \leq \frac{L(\mu')}{HX^{new}(\mu')} \tag{7.9}$$

because $L(\mu) = L(\mu')$.

To prove (7.8), we firstly derive two inequalities which are consequences of the fact that HX is consistent. For the circuit which consists of the paths τ_{ik}, τ_{kl} and τ_{li} and which has a positive delay we have:

$$\begin{aligned} 1 &\leq HX(\tau_{ik}) + HX(\tau_{kl}) + HX(\tau_{li}) && (7.10)\\ &= HX_{b(i)\,k} + HX_{b(k)\,l} + HX_{b(l)\,i} && (7.11)\\ &\quad \text{As } HX_{b(i)\,l} + HX_{b(l)\,i} = 1 \text{ we get} \\ &= HX_{b(i)\,k} + HX_{b(k)\,l} + 1 - HX_{b(i)\,l} && (7.12) \end{aligned}$$

Therefore, we have

$$HX_{b(i)\,k} + HX_{b(k)\,l} \geq HX_{b(i)\,l} \tag{7.13}$$

Similarly, because $HX_{b(k)\,m} + HX_{b(m)\,k} = 1$ holds, we get for the circuit which consists of the paths τ_{kl}, τ_{lm} and τ_{mk}

$$HX_{b(k)\,l} + HX_{b(l)\,m} \geq HX_{b(k)\,m}. \tag{7.14}$$

Thus,

$$\begin{aligned} HX^{new} &= HX_{b(i)\,l} + HX_{b(l)\,k} - x + HX_{b(k)\,m} && (7.15)\\ &\leq HX_{b(i)\,k} + HX_{b(k)\,l} + HX_{b(l)\,k} - x + HX_{b(k)\,m} && (7.16)\\ &= HX_{b(i)\,k} + 1 - x + HX_{b(k)\,m} && (7.17)\\ &\leq HX_{b(i)\,k} + HX_{b(k)\,m} && (7.18)\\ &\leq HX_{b(i)\,k} + HX_{b(k)\,l} + HX_{b(l)\,m} && (7.19)\\ &= HX^{old} && (7.20) \end{aligned}$$

(7.15) is smaller or equal to (7.16) due to (7.13). (7.16) and (7.17) are equal because $HX_{b(k)\,l} + HX_{b(l)\,k} = 1$. (7.18) is smaller or equal to (7.19) due to (7.14). □

The remaining part of this section deals with the problem that the height of a disjunctive arc is equal to the upper bound.

For non-cyclic job shop and flow shop problems with blocking we can show the following theorem. A proof for this theorem can be found in Nieberg [55].

Theorem 7.13 *For the non-cyclic job shop problem we can always find a new feasible solution by eliminating in a feasible solution one job and putting the job at the end of the schedule by reversing all disjunctive arcs which start at operations of this job.*

Now we want to use the idea of this theorem to compute a new consistent and relevant height function for cyclic job shop scheduling problems with blocking.

First, we have to develop another bound for the height of a disjunctive arc. Due to the given conjunctive arcs, we can show the following lemma.

Lemma 7.14 *If $(b(i), i') \in D$ is a disjunctive arc, then there exists no consistent and relevant height function HX satisfying $HX_{b(i)\,i'} > HX_{\star 0}$ or $HX_{b(i')\,i} < 1 - HX_{\star 0}$.*

Proof: Assume there exists a disjunctive arc $(b(i), i') \in D$ with $HX_{b(i)\,i'} > HX_{\star 0}$ i.e. $HX_{b(i)\,i'} \geq HX_{\star 0} + 1$, where HX is relevant. Thus, the alternative arc $(b(i'), i)$ has the height $HX_{b(i')\,i} = 1 - HX_{b(i)\,i'} \leq -HX_{\star 0} < 1 - HX_{\star 0}$.

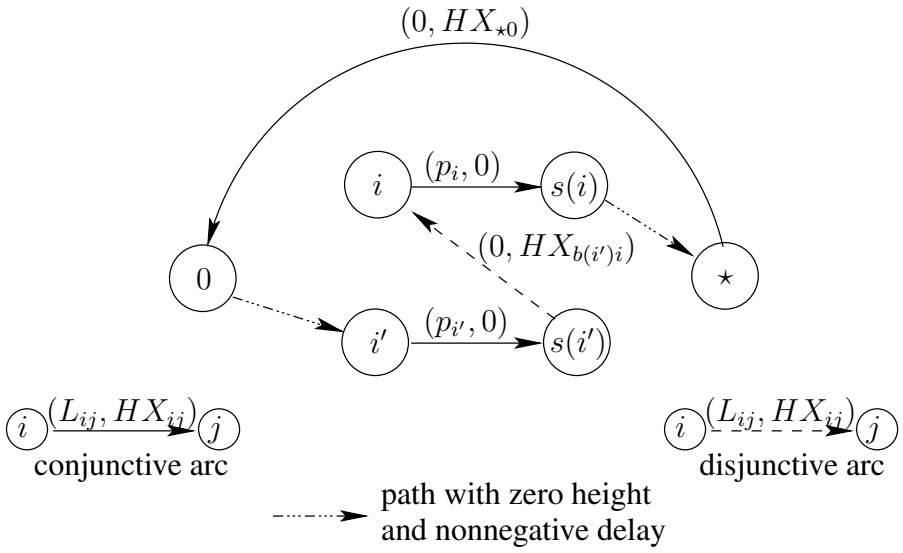

Figure 7.8: Situation in Proof of Lemma 7.14

Furthermore, there exists a path μ_1 from i to $\star$ which has a positive delay and zero height. Additionally, there exists a path μ_2 from 0 to $b(i')$ with non-negative delay and zero height. Both paths μ_1 and μ_2 only consist of conjunctive arcs. Together with the arc $(\star, 0)$, which has zero delay and height $HX_{\star 0}$, these paths form a path μ and have a positive delay and height $HX_{\star 0}$. If operation i and i' are blocking operations, then the situation is shown in Figure 7.8.

The path μ and the alternative arc $(b(i'), i)$ form a circuit with $HX_{b(i')\,i'} + HX_{\star 0} < 1 - HX_{\star 0} + HX_{\star 0} = 1$, i.e. with non-positive height and positive delay. Thus, HX is not consistent. □

We now develop an algorithm which allows us to increase the height of a disjunctive arc $(b(i), i')$ on a critical circuit even if the current height is equal to the upper bound.

To show that our approach always finds a feasible solution, we introduce the **expanded graph** G_{exp}.

G_{exp} consists of an infinite number of nodes $(i; k)$ $(i \in T, k \in \mathbb{Z})$. The node $(i; k)$ represents the k-th occurrence of the operation i. The node set of the expanded graph G_{exp}

is denoted by T_{exp}. Furthermore, if there exists an arc (i, i') in G with delay $L_{ii'}$ and height $HX_{ii'}$, then in the graph G_{exp} there exist arcs between the nodes $(i; k)$ and $(i'; k + HX_{ii'})$ with length $L_{ii'}$ for all $k \in \mathbb{Z}$. The following lemma describes a relationship between circuits with positive length in G_{exp} and circuits with zero height and positive delay in G.

Lemma 7.15 *There exists a circuit with positive length in G_{exp} if and only if there exists a circuit with zero height and positive delay or a circuit with negative height and non-negative delay in G.*

Proof: Assume there exists a circuit μ with positive length in G_{exp} which consists of the following operations $((i_1; k_1), (i_2; k_2), \ldots, (i_l; k_l), (i_1; k_1))$.

We now show that there exists either a circuit with zero height and positive delay or a circuit with negative height and non-negative delay in G. As each arc in G_{exp} is derived from an arc in G, all arcs $((i_p; k_p), (i_{p+1}; k_{p+1}))$ for $1 \leq p \leq l$ in the circuit μ with length $L_{i_p\, i_{p+1}}$ and height $HX_{i_p\, i_{p+1}} = k_{p+1} - k_p$ exist in the graph G. Note that we identify the node $(i_{l+1}; k_{l+1})$ with $(i_1; k_1)$. The height of the circuit μ in G is $\sum_{(i,j)\in\mu} HX_{ij} = 0$ and the length is $\sum_{(i,j)\in\mu} L_{ij}$ positive.

The circuit μ in G can consist of several simple circuits. If one of the simple circuits has a positive height, then there must exist at least one circuit μ' with negative height because the sum of the height of all simple circuits is equal to zero. As all arcs have non-negative length, the length of the circuit μ' is greater or equal to zero. Thus, the circuit μ' violates the conditions of Theorem 2.1 because a circuit with negative height must also have a negative delay. If all simple circuits have height zero, then there must exist at least one circuit μ' with positive delay because the delay of the circuit μ is positive. Thus, the height of the circuit μ' is zero and the length is positive. This circuit also violates the conditions of Theorem 2.1 because any circuit with zero height must have a non-positive delay. Summarizing, a circuit with positive length in G_{exp} leads to a circuit with negative height and non-negative delay or to circuit with zero height and positive delay.

Now assume that there exists a circuit $\mu = (i_1, i_2, \ldots, i_l, i_1)$ in G with negative height and non-negative delay. Thus, there exists a path $\mu_1 = ((i_1; k_1), (i_2; k_2), \ldots, (i_l; k_l), (i_1, k'_1))$ in G_{exp} with $k_2 = k_1 + HX_{i_1 i_2}$ and $k_\nu = k_{\nu-1} + HX_{i_{\nu-1}\, i_\nu}$ for all $3 \leq \nu \leq l + 1$. We again identify node (i_1, k'_1) with (i_{l+1}, k_{l+1}). As the height of the circuit μ in G is negative, $k'_1 < k_1$ holds. Now we add the arcs $((i_1; k), (i_1; k + 1))$, if i_1 is a non-blocking operation (see Figure 7.9.a), or $((i_1; k), (s(i_1); k))$ and $((s(i_1); k), (i_1; k + 1))$, if i_1 is a blocking operation (see Figure 7.9.b), to the path μ_1 until $k'_1 = k_1$. These arcs exist due to the loop constraints (see (4.12)). Thus, we get a circuit in G_{exp} with positive length because the length of the path from the node (i_1, k'_1) to the node (i_1, k_1) is positive.

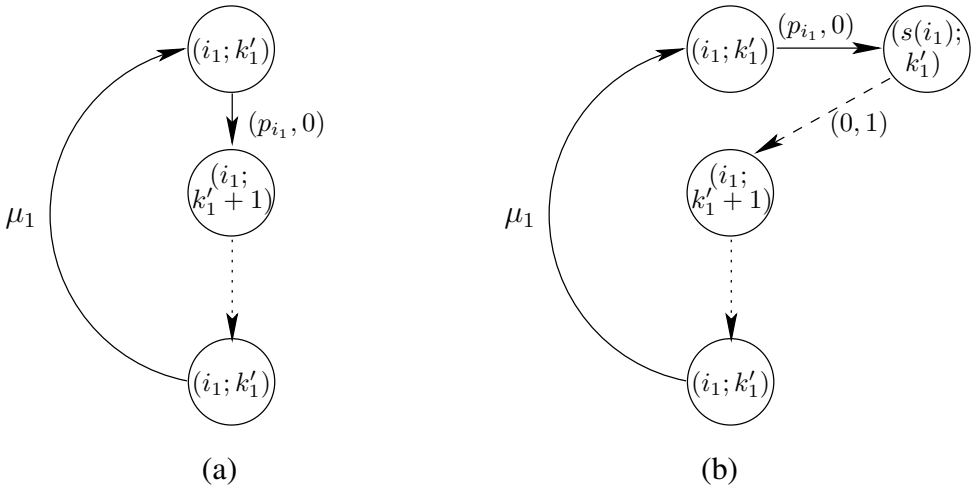

Figure 7.9: The path μ_1 in the proof of Lemma 7.15

Now assume that there exists a circuit $\mu = (i_1, i_2, \ldots, i_l, i_1)$ in G with zero height and positive delay in G. Thus, there exists a circuit $\mu_1 = ((i_1; k_1), (i_2; k_2), \ldots, (i_l; k_l), (i_1, k'_1))$ in G_{exp} with $k_2 = k_1 + HX_{i_1 i_2}$ and $k_\nu = k_{\nu-1} + HX_{i_{\nu-1} i_\nu}$ for all $3 \leq \nu \leq l + 1$. We again identify node (i_1, k'_1) with (i_{l+1}, k_{l+1}). As the height of the μ is zero, $k'_1 = k_1$ holds. Thus, the circuit μ_1 is a circuit with positive length in G_{exp}. □

Now we want to analyse the impact on the graph G_{exp} if we change the height of a disjunctive arc. Assume the operations i and i' are processed on the same machine. Therefore, there exists a disjunctive arc between $b(i)$ and i' with height $HX_{b(i)\,i'}$ and the corresponding alternative arc between $b(i')$ and i with height $HX_{b(i')\,i} = 1 - HX_{b(i)\,i'}$. In G_{exp} there exist disjunctive arcs between $(b(i); k)$ and $(i'; k + HX_{b(i)\,i'})$ and the alternative arcs between $(b(i'); k + HX_{b(i)\,i'})$ and $(i; k + HX_{b(i)\,i'} + HX_{b(i')\,i})$ for all $k \in \mathbb{Z}$. Note that $HX_{b(i)\,i'} + HX_{b(i')\,i} = 1$.

If we now increase the height of the disjunctive arc $(b(i), i')$ by one and decreases the height of the alternative arc by the same amount, we have to replace the arcs

$$((b(i); k), (i'; k + HX_{b(i)\,i'})) \tag{7.21}$$

by

$$((b(i); k), (i'; k + HX_{b(i)\,i'} + 1))$$

and the alternative arcs

$$((b(i'); k + HX_{b(i)\,i'}), (i; k + HX_{b(i)\,i'} + HX_{b(i')\,i})) \tag{7.22}$$

by

$$((b(i'); k + HX_{b(i)\,i'} + 1), (i; k + HX_{b(i)\,i'} + 1 + HX_{b(i')\,i} - 1))$$

for all $k \in \mathbb{Z}$ in the graph G_{exp}.

Before the increase of the height the following inequalities hold due to the disjunctive arcs (7.21) and (7.22) and (4.7):

$$t(i; k) + p_i \leq t(b(i); k) + p_i^b \leq t(i'; k + HX_{b(i)\,i'}) \tag{7.23}$$

and

$$t(i'; k) + p_{i'} \leq t(b(i'); k) + p_i^b \leq t(i; k + HX_{b(i')\,i}).$$

If we now increase the height $HX_{b(i)\,i'}$ by one and decreases $HX_{b(i')\,i}$ by one, we get

$$t(i; k) + p_i \leq t(b(i); k) + p_i^b \leq t(i'; k + HX_{b(i)\,i'} + 1)$$

and

$$t(i'; k) + p_{i'} \leq t(b(i'); k) + p_{i'}^b \leq t(i; k - 1 + HX_{b(i')\,i}) = t(i; k - HX_{b(i)\,i'})$$

must hold because $HX_{b(i')\,i} + HX_{b(i)\,i'} = 1$. This implies

$$t(i'; k + HX_{b(i)\,i'}) + p_{i'} \leq t(i; k). \tag{7.24}$$

By comparison inequalities (7.23) and (7.24) we get the following lemma.

Lemma 7.16 *An increase of the height by one of a disjunctive arc $(b(i), i')$ leads to a change in the processing order of the corresponding operations in G_{exp}.*

Let us now consider the disjunctive arc $(b(i'), i'')$ with height $HX_{b(i')\,i''} < HX_{\star 0}$ which lies on a critical circuit in G. Assume that the height function HX is consistent and that the height of this arc is equal to the height of the upper bound. Thus, an increase would lead to an infeasible solution. Now we want to describe an algorithm which changes the height of some arcs to get a new consistent height function in which the height of the disjunctive arc $(b(i'), i'')$ is increased. Note that, due to Lemma 7.14, we can only increase the height of a disjunctive arc $(b(i'), i'')$ by one, if $HX_{b(i')\,i''} < HX_{\star 0}$ holds because otherwise, we cannot find a feasible solution.

The main idea of our algorithm can be described by the following three steps.

i. Fix a set L_0 in which each operation occurs exactly once (with possibly different occurrence numbers) in G_{exp}. However, the occurrence number k_j for all operations $i \in O(j)$ of the same job $j \in J$ in the set L_0 must be the same, i.e. $L_0 = \{(i; k_{j(i)}) | i \in T\}$.

ii. Replace all disjunctive arcs $((b(c); k_{j(c)}), (d; k_{j(c)} + HX_{b(c)\,d}))$ by $((b(c); k_{j(c)}), (d; k_{j(d)}))$ and the corresponding alternative arcs by $((b(d); k_{j(d)}), (c; k_{j(c)} + 1))$, if $k_{j(c)} + HX_{b(c)\,d} \notin \{k_{j(d)}, k_{j(d)} + 1\}$.

For the last step we need to define sets L_l for all $l \in \mathbb{Z}$. Due to the periodic constraints, the operations of the set L_0 are repeated in the same pattern infinitely often. Therefore, we introduce the sets $L_l = \{(i; k_{j(i)} + l) | i \in T\}$. We call the sets L_{-l} with $l > 0$ preceding sets and the sets L_l with $l > 0$ succeeding sets.

iii. Change the processing orders of the operations in the sets L_l so that all operations of job $j' := j(i')$ are processed as the last operations in the sets L_l for all $l \in \mathbb{Z}$.

We show that these three steps lead to a new consistent height function in which the height of the arc $(b(i'), i'')$ is increased.

In the following, we describe the three steps of our proposed algorithm in more detail. Before this, we want to illustrate our approach with the following example.

Example 7.17 *Consider the example in Table 7.4, which consists of three jobs, each with three operations and a source* 0 *and sink node* $\star$*. The height of all conjunctive arcs is* 0*, except the height of the arc* $(\star, 0)$ *which equals 2. The heights of the disjunctive arcs are given in Table 7.5.*

Jobs	*1*			*2*			*3*		
Operations	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Processing time	*2*	*1*	*2*	*1*	*5*	*2*	*3*	*2*	*1*
Machine	*1*	*2*	*3*	*2*	*3*	*1*	*1*	*3*	*2*

Table 7.4: Job-shop problem with three jobs, each with three operations

Arc	*(2,6)*	*(2,7)*	*(3,4)*	*(3,9)*	*(3,5)*	*(3,8)*	*(5,9)*	*(6,8)*	*(6,7)*
Height	*0*	*1*	*1*	*1*	*1*	*2*	*1*	*1*	*2*
Alternative arc	*(6,1)*	*(8,1)*	*(5,2)*	*(9,2)*	*(6,3)*	*(9,3)*	*(9,4)*	*(9,5)*	*(8,6)*
Height	*1*	*0*	*0*	*0*	*0*	*-1*	*0*	*0*	*-1*

Table 7.5: Height of the disjunctive arcs

The cycle time is $\alpha = 9$*. A critical circuit consists of the nodes* $(3, 5, 6, 8, 9, 3)$*. The upper bound for the disjunctive arc* $(3, 5)$ *is* 1 *because an increase of the height of the arc* $(3, 5)$ *leads to an infeasible solution. Now we describe how our algorithm changes the height function to get a new feasible solution.*

In a first step we need to fix the set L_0*. The first occurrences of all operations of job 1 and the second occurrences of all operations of job 2 must be in the set* L_0*. Additionally, we choose the occurrence number* 2 *for all operations of the third job.*

So, L_0 *includes the following operations*

$$L_0 := \{(1; 1), (2; 1), (3; 1), (4; 2), (5; 2), (6, 2), (7, 2), (8, 2), (9, 2)\}.$$

This situation is shown in Figure 7.10. Note that in this and in the following figures not all disjunctive arcs are drawn.

The color of the nodes describes to which set the nodes belong. As we see in this figure, there exists the arc $((8; 2), (6; 1))$*, which starts in* L_0 *and ends in* L_{-1}*. Additional to this arc, the arc* $((2; 1), (6; 1))$ *also starts in* L_0 *and ends in* L_{-1}*.*

Due to the second step of our algorithm, we need to change these two arcs. The arc $((8; 2), (6; 1))$ *is replaced by* $((8; 2), (6; 2))$*, the alternative arc* $((6; 1), (8; 3))$ *is replaced by* $((6; 2), (8; 3))$*. The arc* $((2; 1), (6; 1))$ *is replaced by* $((2; 1), (6, 2))$ *and the alternative arc by* $((6; 2), (2; 2))$*. Note that we have also to change the other occurrences of these arcs in the same way. Therefore, we get the situation which is shown in Figure 7.11. To get a new consistent and relevant height function we have now to process all operations of job* 1*, i.e. operations 1,2, and 3 as last in the set* L_0*. This situation is shown in Figure 7.12. The cycle time of the new solution is* $\alpha = 19$*.*

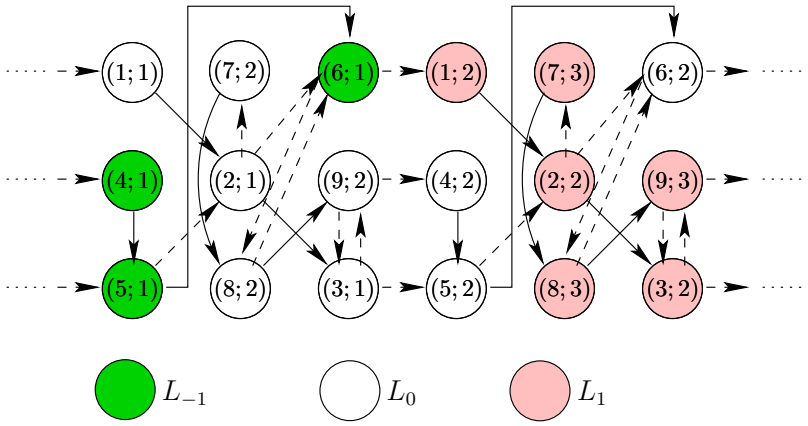

Figure 7.10: Situation after fixing the set L_0

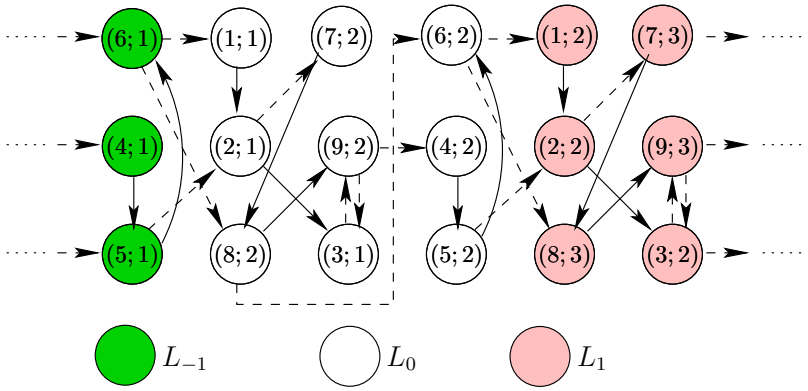

Figure 7.11: Situation after replacing the arcs

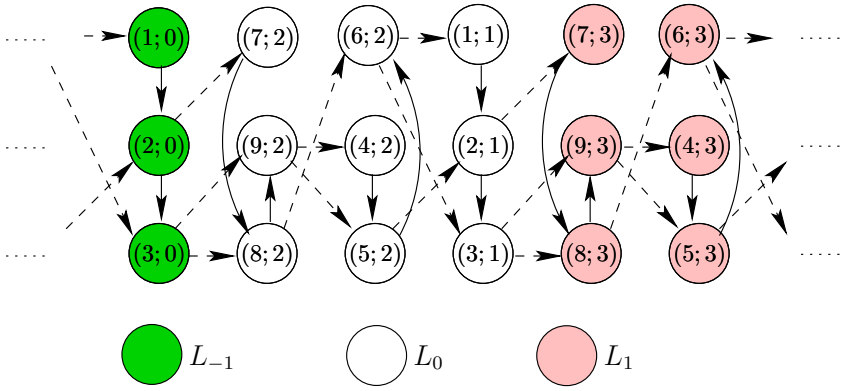

Figure 7.12: New solution

To define the set L_0, we have to fix an occurrence number k_j for each job $j \in J$. The occurrence number of job j' is set to $k_{j'} := 0$. The occurrence number of the job $j'' := j(i'')$ depends on the occurrence number of job j' as follows: $k_{j''} := k_{j'} + HX_{b(i')\,i''} = HX_{b(i')\,i''}$. For all other jobs $j \in J \setminus \{j'\}$ we fix the occurrence number such that

$$k_j < k_{j'} + HX_{\star 0} = HX_{\star 0} \tag{7.25}$$

and

$$1 - HX_{\star 0} \leq \max_{j \in J'} k_j - \min_{j \in J'} k_j \leq HX_{\star 0} \tag{7.26}$$

for all $J' \in \mathcal{P}(J)$ hold.

Thus, we need to fix the k_j-variables for all $j \in J \setminus \{j', j''\}$ to satisfy the conditions (7.25) and (7.26). This is done by a heuristic, which is presented later. This concludes the first step of our algorithm.

For the second step we have to change all disjunctive arcs which start at operations in the set L_0 and end at operations of a preceding set $L_{-\nu}, \nu \geq 1$ or end at operations of a succeeding sets L_ν, $\nu \geq 2$. Afterwards, we show that after the second step the new graph G^{new}_{exp} has no circuit with positive length, implying that G^{new} is consistent (see Lemma 7.15).

To change the disjunctive arcs, we substitute

$$((b(c); k_{j(c)}), (d; k_{j(d)} - \nu))$$

with $\nu \geq 1$ and $(c; k_{j(c)}), (b(c); k_{j(d)}) \in L_0$ by

$$((b(c); k_{j(c)}), (d; k_{j(d)})).$$

Due to the above substitution, we need to change the height of the arc $(b(c), d)$ in G from $HX_{b(c)\,d} := k_{j(d)} - \nu - k_{j(c)}$ to $HX^{new}_{b(c)\,d} := k_{j(d)} - k_{j(c)}$. Thus, the height of arc $(b(c), d)$ is increased by ν. Therefore, we have to decrease the height of the alternative arc $(b(d), c)$ which is equal to $HX_{b(d)\,c} = 1 - HX_{b(c)\,d} = 1 - k_{j(d)} + \nu + k_{j(c)}$ by ν. Thus, $HX^{new}_{b(d)\,c} = 1 - k_{j(d)} + k_{j(c)}$. Therefore, we have to replace the alternative arc of $((b(c); k_{j(c)}), (d; k_{j(d)} - \nu))$ which is

$$((b(d); k_{j(d)} - \nu), (c; k_{j(d)} - \nu + 1 - k_{j(d)} + \nu + k_{j(c)}))$$

or equivalently

$$((b(d); k_{j(d)} - \nu), (c; 1 + k_{j(c)}))$$

by

$$((b(d); k_{j(d)}), (c; k_{j(d)} + 1 - k_{j(d)} + k_{j(c)}))$$

or

$$((b(d); k_{j(d)}), (c; 1 + k_{j(c)})).$$

Note that $(c; 1 + k_{j(c)}) \in L_1$ holds.

Due to the above changes in G^{new}_{exp}, there exists no disjunctive arc which starts at operation $(b(c); k_{j(c)}) \in L_0$ and ends at operation $(d; k_{j(d)} + \nu) \in L_\nu$, $\nu \geq 2$ because for the corresponding alternative arc $((b(d); k_{j(d)} + \nu), (c; k_{j(c)} + 1))$ the conditions $(b(d); k_{j(d)} + \nu) \in L_\nu$ and $(c; k_{j(c)} + 1) \in L_1$ would hold or equivalently $(b(d); k_{j(d)}) \in L_0$ and $(c; k_{j(c)} + 1 - \nu) \in L_{1-\nu}$, $\nu \geq 2$. However, as the disjunctive arc $((b(d); k_{j(d)}), (c; k_{j(c)} + 1 - \nu))$ and the corresponding alternative arc are already substituted, there exists no disjunctive arc which starts at operations in L_0 and ends at operations in L_ν, $\nu \geq 2$.

Therefore, after this change all disjunctive arcs which start at operations in L_0 end at operations in L_0 or in L_1.

To show that G^{new}_{exp} has no circuits with positive length, we need the following lemma.

Lemma 7.18 *If there exists a disjunctive arc $((b(c); k_{j(c)}), (d; k_{j(d)} - \nu))$ with $\nu > 0$ and $(b(c); k_{j(c)}), (d; k_{j(d)}) \in L_0$ in G_{exp}, then there exists also a path with positive length from $(b(c); k_{j(c)})$ to $(d; k_{j(d)})$ in G_{exp}.*

Proof: The alternative arc of $((b(c); k_{j(c)}), (d; k_{j(d)} - \nu))$ is $((b(d); k_{j(d)} - \nu), (c; 1 + k_{j(c)}))$, so there exists a path from $(b(c); k_{j(c)})$ to $(d; k_{j(d)} - \nu)$. Additionally, there exists

a disjunctive arc $((b(c); k_{j(c)} + 1), (d; k_{j(d)} - \nu + 1))$ and the alternative arc $((b(d); k_{j(d)} - \nu + 1), (c; 2 + k_{j(c)}))$. So, there exists a path from $(c; k_{j(c)})$ to $(d; k_{j(d)} - \nu + 1)$. If d is a blocking operation then the path also includes the arc $((d; k_{j(d)} - \nu), (s(d); k_{j(d)} - \nu))$. This process can be repeated until we get a path from $(c; k_{j(c)})$ to $(d; k_{j(d)})$. We can easily verify that the length of the path is positive (A similar idea is used in the proof of Lemma 7.15). □

Now we show that after the second step there exists no circuit with positive length in G_{exp}^{new}. First we show that there exists no path μ in G_{exp}^{new} that connects an operation of the set L_0 with an operation of a previous set $L_{-\nu}$ with $\nu \geq 1$.

Lemma 7.19 *There exists no path μ in G_{exp}^{new} from $(c; k_{j(c)}) \in L_0$ to $(d; k_{j(d)} - \nu) \in L_{-\nu}$ with $\nu > 0$.*

Proof: Assume there exists a path μ from $(c; k_{j(c)}) \in L_0$ to $(d; k_{j(d)} - \nu) \in L_{-\nu}$. Due to the change of the disjunctive arcs, all disjunctive arcs start at operations in L_0 and end at operations in L_0 or L_1. The conjunctive arcs between the operations of a job start in L_0 and end in L_0. Therefore, an arc $((\star, k), (0, k + HX_{\star 0}))$ must be in the path.

We denote the last operation on the path μ before the node $(\star, k)$ with $(e; k)$ and the first operation after the node $(0; k + HX_{\star 0})$ with $(f; k + HX_{\star 0})$. As all arcs except the arc $((\star, k), (0, k + HX_{\star 0}))$ connect nodes in L_k with nodes in L_k or L_{k+1}, the node $(e; k)$ can be in any set $L_{k'}$ with $k' \geq 0$. With the same argument we can conclude that the node $(f; k + HX_{\star 0})$ can be in any set $L_{k''}$ with $k'' \leq -\nu$.

Thus, $(e; k - k') \in L_0$ and $(f; k + HX_{\star 0} - k'') \in L_0$ hold. Now we check whether inequality (7.26) holds.

If $k - k' \geq k + HX_{\star 0} - k''$ we get

$$\begin{aligned}
& \max\{k_{j(e)}, k_{j(f)}\} - \min\{k_{j(e)}, k_{j(f)}\} \\
= \; & \max\{k - k', k + HX_{\star 0} - k''\} - min\{k - k', k + HX_{\star 0} - k''\} \\
= \; & k - k' - (k + HX_{\star 0} - k'') \\
= \; & -k' - HX_{\star 0} + k'' \\
\leq \; & 0 - HX_{\star 0} - \nu \\
< \; & 1 - HX_{\star 0}
\end{aligned}$$

For the other case we get

$$\begin{aligned} & \max\{k_{j(e)}, k_{j(f)}\} - \min\{k_{j(e)}, k_{j(f)}\} \\ = \; & k + HX_{\star 0} - k'' - (k - k') \\ = \; & HX_{\star 0} - k'' + k' \\ \geq \; & HX_{\star 0} + \nu \\ > \; & HX_{\star 0} \end{aligned}$$

Thus, both cases lead to a contradiction to inequality (7.26). Therefore, there cannot exist a path μ from $(c, k_{j(c)}) \in L_0$ to $(d; k_{j(d)} - \nu) \in L_{-\nu}, \nu > 0$. □

As all operations and arcs in L_0 are repeated in the same pattern, this results can be generalized to

Lemma 7.20 *There exists no path μ in G_{exp}^{new} from $(c; k')$ to $(d; k'')$ with $(c; k') \in L_k$ and $(d; k'') \in L_{k-\nu}$, $\nu > 0$.*

With Lemma 7.18 and Lemma 7.20 we can now show that the second step of our algorithm leads to no circuit with positive length in the expanded graph G_{exp}^{new}.

Theorem 7.21 *After performing the second step of our proposed algorithm there exists no circuit with positive length in the graph G_{exp}^{new}.*

Proof: Assume in G_{exp}^{new} there exists a circuit μ with positive length. The nodes in this circuit must belong to the same set L_k because if a node in this circuit is in L_k and another node is in $L_{k+\nu}$, $\nu > 0$, then, as there exists no path from any node in $L_{k+\nu}$ back to a node in L_k (see Lemma 7.20), these nodes cannot be in a circuit with positive length in G_{exp}^{new}. Additionally, due to the periodic constraints, this circuit exists infinitely often in the graph G_{exp}^{new}. Therefore, we can assume that the only nodes in the circuit μ except the $\star$ and 0 nodes belong to the set L_0. Furthermore, there must be at least one arc in this circuit which was not in G_{exp} because otherwise the circuit also exists in G_{exp}. Each arc from $(i; k) \in L_0$ to $(i'; k') \in L_0$ exists also in G_{exp} or is a new arc in G_{exp}^{new}. If the arc is a new arc, then there exists an arc from $((i; k)$ to $(i'; k' - \nu)$ with $\nu > 0$ in G_{exp}. Due to Lemma 7.18, there exists also a path with positive length from $((i; k)$ to $(i'; k')$ in G_{exp}. Therefore, the circuit μ must be also in G_{exp}. This leads to a contradiction because in G_{exp} there exists no circuit with positive length. □

The last step to get a new relevant and consistent height function is to change the processing orders in the set L_0 (and thus also in the sets L_l) so that all operations of the job $j' = j(i')$ are processed as last operations in the sets.

The following theorem shows that this leads to a new relevant and consistent height function.

Theorem 7.22 *If we change the order in the set L_0 so that all operations of job j' are scheduled as last, then we get a new consistent height function.*

Proof: Before changing the order in the set L_0 all disjunctive arcs which start at operations of job j' in L_0 end either at operations in L_0 or L_1. The set of disjunctive arcs which start at operations of job j' in L_0 and end at operation in L_0 is denoted with R.

Consider a disjunctive arc $((b(i');k_{j'}),(i'';k_{j(i'')}))$ in R and its alternative arc $((b(i'');k_{j(i'')}),\ (i';k_{j'}+1))$ with $(b(i');k_{j'}),(i'';k_{j(i'')}) \in L_0$ and $(i';k_{j'}+1) \in L_1$. These arcs are replaced by $((b(i'');k_{j(i'')}),(i';k_{j'}))$ and $((b(i');k_{j'}),(i'';k_{j(i'')}+1))$ with $(b(i'');k_{j(i'')}),\ (i';k_{j'}) \in L_0$ and $(i'';k_{j(i'')}+1) \in L_1$. This is also done for all the other arcs in R and its alternative arcs.

Therefore, the height $HX_{b(i')\,i''} = k_{j(i'')}$ of the arc $(b(i'),i'')$ is increased by one and the height $HX_{b(i'')\,i'} = 1 - k_{j(i'')}$ of the alternative arc $(b(i''),i')$ is decreased by one (see Lemma 7.16). So, the new heights are smaller or equal to $HX_{\star 0}$ and greater or equal to $1 - HX_{\star 0}$ because $k_{j(i'')} < HX_{\star 0}$ and $1 - k_{j(i'')} > 1 - HX_{\star 0}$.

Thus, after replacing these arcs all new disjunctive arcs which end at operations of j' in L_0 start at operations in L_0 and all new and old disjunctive arcs which start at operations of j' in L_0 end at operations in L_1.

Furthermore, there exists no path from the last operation $(l;k_{j'})$ of job j' in L_0 to a first operation $(a;k_{j(a)})$ of a job $j(a) \neq j'$ in L_0 passing the nodes $(\star;k_{j'})$ and $(0;k_{j'}+HX_{\star 0})$ because in this case the occurrence number $k_{j(a)}$ of operation a must be $k_{j'}+HX_{\star 0}$, which violates the constraint (7.25).

Now assume that after replacing all the arcs in R and their alternative arcs, there exists a circuit μ in G_{exp}^{new} with positive length. Due to Theorem 7.21, there exists no circuit with positive length in the graph G_{exp}^{new} before replacing the arcs. Thus, there must be at least one new disjunctive arc in the circuit μ. Since we only insert arcs which start at operations in L_0 and end at operations in L_0 or L_1 all nodes in μ must be in L_0 because there exists no path from nodes in L_1 to nodes in L_0. As each new disjunctive arc, which connects nodes in L_0, ends at operation of job j' and each arc which starts at operation of job j' in L_0 ends at operation in L_1, there cannot exist a new disjunctive arc in the circuit μ. Thus, there cannot exist a circuit with positive length in G_{exp}^{new} after replacing the arcs. □

Due to the periodic constraints changing the processing orders in the set L_0 also lead to a change in the processing orders of the other sets.

To sum up, we have shown that our algorithm computes a new consistent and relevant height function in which the height of the disjunctive arc $(b(i'), i'')$ is increased. Now the only open question is how to compute the k_j-variables for all $j \in J \setminus \{j', j''\}$.

We propose two different heuristics.

Heuristic H_1

An easy way to get the set L_0 is to add, in addition to the $k_{j'}$-th occurrence of job j' and the $(k_{j'} + HX_{b(i')\,i''})$-th occurrence of job j'', the $k_{j'}$-th occurrences of all other jobs $j \in J \setminus \{j', j''\}$ to the set L_0. We call this heuristic H_1. Now we show that the constraints (7.25) and (7.26) are fulfilled. As k_j is equal to $k_{j'} = 0$ for all $j \in J \setminus \{j', j''\}$ and $HX_{\star 0} \geq 1$, constraint (7.25) is fulfilled.

For $J' \in \mathcal{P}(J)$ the value $\max_{j \in J'} k_j - \min_{j \in J'} k_j$ is either 0 or $HX_{b(i')\,i''}$ which implies that (7.26) holds because $HX_{\star 0} \geq 1$ implies $1 - HX_{\star 0} \leq 0 \leq HX_{\star 0}$ and by Lemma 7.14 we also have $1 - HX_{\star 0} \leq HX_{b(i')\,i''} \leq HX_{\star 0}$. Therefore, the constraints (7.25) and (7.26) are fulfilled.

The disadvantage of this set is that it might be possible that we insert occurrence of jobs to this set which are not processed near the occurrences of the operations of the jobs j' and j'' in L_0.

Heuristic H_2

The second heuristic H_2 inserts the operations to the set L_0 which are scheduled near the operations of job j'.

First, we compute the starting times of the $k_{j'}$-th occurrence of all operations of job j'. Then we compute a time interval with length α for each machine $m \in M$, where the starting time is defined by $\min\{t(i; k_{j'}) | i \in O(j') \text{ and } M(i) = m\}$ and α is the cycle time of the actual solution. Finally, we add all jobs $j := j(i)$ for which the first operation $(i; l)$ starts in these time intervals with some occurrence number l to the set L_0. If there exists a job j which first operation is not scheduled in the computed time intervals, then we fix the occurrence number k_j to 0.

If for the computed occurrence number l the inequality $l \geq k_{j'} + HX_{\star 0}$ holds, then we set $l := (k_{j'} + HX_{\star 0} - 1)$ as occurrence number of all operations of the job $j(i)$. So, the occurrence number for job $j(i)$ is

$$k_j := \min\{l, k_{j'} + HX_{\star 0} - 1\} \leq HX_{\star 0} - 1. \tag{7.27}$$

If $k_j - \min_{j' \in J'} k_{j'} > HX_{\star 0}$ holds, then we set $k_j := \min_{j' \in J'} k_{j'} + HX_{\star 0}$. If $\max_{j' \in J'} k_{j'} - k_j > HX_{\star 0}$ holds, then we set $k_j := \max_{j' \in J'} k_{j'} - HX_{\star 0}$.

Thus, we get the following four different cases for the occurrence number k_j of job j:

i. $k_j = 0$ or

ii. $k_j = \min\{l, k_{j'} + HX_{\star 0} - 1\}$, if $\max\{\max_{j' \in J'} k_{j'}, k_j\} - \min\{\min_{j' \in J'} k_{j'}, k_j\} \leq HX_{\star 0}$ or

iii. $k_j = \min_{j' \in J'} k_{j'} + HX_{\star 0}$, if $\min\{l, k_{j'} + HX_{\star 0} - 1\} - \min_{j' \in J'} k_{j'} > HX_{\star 0}$ or

iv. $k_j = \max_{j' \in J'} k_{j'} - HX_{\star 0}$, if $\max_{j' \in J'} k_{j'} - \min\{l, k_{j'} + HX_{\star 0} - 1\} > HX_{\star 0}$.

In the following we show that the constraints (7.25) and (7.26) are fulfilled. We show now that after we insert a new job j to the set L_0 that

$$k_{j'} < HX_{\star 0} \tag{7.28}$$

for all $j' \in J' \cup \{j\}$ and

$$1 - HX_{\star 0} \leq \max_{j' \in J''} k_{j'} - \min_{j' \in J''} k_{j'} \leq HX_{\star 0} \tag{7.29}$$

for all $J'' \in \mathcal{P}(J' \cup \{j\})$hold , where J' is the set of already inserted jobs.

After inserting the first two jobs j' and j'' into L_0 with $k_{j'} = 0$ and $k_{j''} = HX_{b(i')\,i''}$ inequalities (7.29) and (7.28) hold.

Now we have to show that after adding the job j with occurrence number k_j to the set L_0 inequalities (7.29) and (7.28) hold.

We start with the first case, i.e. $k_j = 0$ holds. We can easily see with the same arguments as for heuristic H_1 that (7.28) and (7.29) are fulfilled.

Let us now consider the second case, i.e.

$$k_j = \min\{l, k_{j'} + HX_{\star 0} - 1\}$$

and

$$\max\{\max_{j' \in J'} k_{j'}, k_j\} - \min\{\min_{j' \in J'} k_{j'}, k_j\} \leq HX_{\star 0}$$

hold. Inequality (7.28) is fulfilled due to inequality (7.27).

First we derive some bounds for the maximum and minimum of $\max_{j' \in J''} k_{j'} - \min_{j' \in J''} k_{j'}$ for all $J'' \in \mathcal{P}(J' \cup \{j\})$. The maximal value is

$$\max\{\max_{j' \in J''} k_{j'} - \min_{j' \in J''} k_{j'} | J'' \in \mathcal{P}(J' \cup \{j\})\} = \max_{j \in J' \cup \{j\}} k_j - \min_{j \in J' \cup \{j\}} k_j \tag{7.30}$$

and the minimal value is

$$\min\{\max_{j'\in J''} k_{j'} - \min_{j'\in J''} k_{j'} | J'' \in \mathcal{P}(J' \cup \{j\})\} \geq 0. \tag{7.31}$$

As for the second case $\max\{\max_{j'\in J'} k_{j'}, k_j\} - \min\{\min_{j'\in J'} k_{j'}, k_j\} \leq HX_{\star 0}$ holds, inequality (7.29) is fulfilled due to (7.30).

Now we analyse the third case, i.e. $k_j = \min_{j'\in J'} k_{j'} + HX_{\star 0}$ and $\min\{l, k_{j'} + HX_{\star 0} - 1\} - \min_{j'\in J'} k_{j'} > HX_{\star 0}$ First, we show that $k_j < HX_{\star 0}$ holds. As $HX_{\star 0} - 1 \geq \min\{l, k_{j'} + HX_{\star 0} - 1\} > HX_{\star 0} + \min_{j'\in J'} k_{j'} = k_j$ holds, (7.28) holds. Now we check if (7.29) holds. We get

$$\max_{j\in J'\cup\{j\}} k_j = \max\{\max_{j\in J'} k_j, \min_{j'\in J'} k_{j'} + HX_{\star 0}\}.$$

If $\max_{j\in J'\cup\{j\}} k_j = \max_{j\in J'} k_j$, then we get

$$\begin{aligned} & \max\{\max_{j'\in J''} k_{j'} - \min_{j'\in J''} k_{j'} | J'' \in \mathcal{P}(J' \cup \{j\})\} \\ = \; & \max_{j\in J'} k_j - \min_{j\in J'\cup\{j\}} k_j \\ = \; & \max_{j\in J'} k_j - \min_{j\in J'} k_j \\ \leq \; & HX_{\star 0}. \end{aligned}$$

If $\max_{j\in J'\cup\{j\}} k_j = \min_{j'\in J'} k_{j'} + HX_{\star 0}$, then we get

$$\begin{aligned} & \max\{\max_{j'\in J''} k_{j'} - \min_{j'\in J''} k_{j'} | J'' \in \mathcal{P}(J' \cup \{j\})\} \\ = \; & \min_{j'\in J'} k_{j'} + HX_{\star 0} - \min_{j\in J'\cup\{j\}} k_j \\ = \; & HX_{\star 0} \\ \leq \; & HX_{\star 0}. \end{aligned}$$

Thus, for both cases the inequality (7.29) is fulfilled.

Finally, we consider the fourth case, i.e. $k_j = \max_{j'\in J'} k_{j'} - HX_{\star 0}$ and $\max_{j'\in J'} k_{j'} - \min\{l, k_{j'} + HX_{\star 0} - 1\} > HX_{\star 0}$ hold. First, we need to show that $k_j < HX_{\star 0}$ holds. This is equivalent to $\max_{j'\in J'} k_{j'} - HX_{\star 0} < HX_{\star 0}$. This holds because $\max_{j'\in J'} k_{j'} < HX_{\star 0}$. Thus, the inequality (7.28) holds. Now we show that inequality (7.29) holds. We get

$$\min_{j\in J'\cup\{j\}} k_j = \min\{\min_{j\in J'} k_j, \max_{j'\in J'} k_{j'} - HX_{\star 0}\}.$$

If $\min_{j \in J' \cup \{j\}} k_j = \min_{j \in J'} k_j$, then we get

$$\begin{aligned} & \max\{\max_{j' \in J''} k_{j'} - \min_{j' \in J''} k_{j'} | J'' \in \mathcal{P}(J' \cup \{j\})\} \\ = & \max_{j \in J' \cup \{j\}} k_j - \min_{j \in J'} k_j \\ = & \max_{j \in J'} k_j - \min_{j \in J'} k_j \\ \leq & HX_{\star 0}. \end{aligned}$$

If $\min_{j \in J' \cup \{j\}} k_j = \max_{j' \in J'} k_{j'} - HX_{\star 0}$ holds, we get

$$\begin{aligned} & \max\{\max_{j' \in J''} k_{j'} - \min_{j' \in J''} k_{j'} | J'' \in \mathcal{P}(J' \cup \{j\})\} \\ = & \max_{j \in J' \cup \{j\}} k_j - \max_{j' \in J'} k_{j'} + HX_{\star 0} \\ = & HX_{\star 0} \\ \leq & HX_{\star 0}. \end{aligned}$$

Thus, for both cases the inequality (7.29) is fulfilled. Summarizing, we showed that after inserting a new job j into the set L_0 the inequalities (7.28) and (7.29) hold. So, after adding all jobs to the set L_0 the constraints (7.25) and (7.26) are fulfilled.

For the description of the neighbourhood N_4 we need to define the arc set A and the corresponding height function HX^A. The arc set A is defined by all arcs which are changed during the repair algorithm. The corresponding height function HX^A is defined by replacing the heights of all arcs in A in the old height function HX by the computed height values of our algorithm. Thus, we can define N_4:

To each feasible height function HX the set $N_4(HX)$ of all height functions derived from HX is assigned as follows: We start with choosing an arc $(b(i'), i'') \in D$ in G on a critical circuit and replace the height $HX_{b(i')\,i''}$ by $HX_{b(i')\,i''} + x$ with $x = HX^+_{b(i')\,i''} - HX_{b(i')\,i''}$ if $x > 0$ or we replace the height function HX by HX^A.

To show that the neighbourhood N_4 is not opt-connected, we reuse Example 7.11.

Example 7.23 *To increase the height of the arc $(5, 7)$, we use the proposed repair algorithm, which is used in neighborhood N_4 to get a new feasible height function. The repair algorithm leads to the following new solution, which is presented in the gantt chart in Figure 7.13. The height of the disjunctive arcs is given in Table 7.6.*

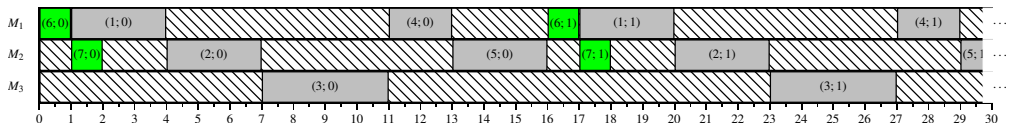

Figure 7.13: Gantt chart for Example 7.23

Arc	*(2,6)*	*(2,4)*	*(3,7)*	*(3,5)*	*(5,6)*	*(5,7)*
Height	*1*	*0*	*1*	*0*	*1*	*1*
Alternative arc	*(7,1)*	*(5,1)*	*(7,2)*	*(5,2)*	*(7,4)*	*(7,5)*
Height	*0*	*1*	*0*	*1*	*0*	*0*

Table 7.6: Height of the disjunctive arcs

The critical circuit is $(1, 2, 3, 4, 5, \star, 0, 6, 7, 1)$. *The only disjunctive arc in this circuit is* $(7, 1)$. *Due to the circuit* $(2, 6, 7, 2)$, *we must again apply the repair algorithm to get a new feasible solution in which the height of the arc* $(7, 1)$ *is increased. After applying the repair algorithm, we get back to the solution in Figure 7.5. So, the algorithm cannot reach the optimal solution which is given in Figure 7.6.*

Now we want to describe another way of getting a new and feasible height function which is based on the previous computed arc set A. We now compute a subset A' of this arc set A, which also leads to a new and feasible height function. Note that A' and A can be equal.

This algorithm works as follows: As the disjunctive arc $(b(i'), i'')$ lies on a critical circuit the height of this arc must be changed to $HX^A_{b(i')\,i''}$. Thus, we increase the height of this arc and decrease the height of the alternative arc. Now it can happen, that in the graph G there exists a circuit μ which does not fulfill the conditions of Theorem 2.1. If this is the case, there must exist an arc a_1 in the circuit μ which is also in the arc set A. Therefore, we add this arc a_1 to the set A' and change the height of arc a_1 so that it equals to $HX^A_{a_1}$ and check again whether the height function is relevant and consistent. If the height function is still not relevant and consistent, we can again find an arc a_2 on a circuit which does not fulfill the conditions of Theorem 2.1 and which is also in the arc set A. The arc is also added to the arc set A' and the height is also changed to $HX^A_{a_2}$, etc. The algorithm stops, if we get a feasible height function. As the algorithm only changes the height of arcs which are in the arc set A the algorithm always finds a feasible solution.

In each circuit μ which does not fulfill the conditions of Theorem 2.1 there must exist an arc which is also in the arc set A because otherwise the circuit would still exist after replacing HX by HX^A. As replacing HX by HX^A always leads to a feasible solution, this cannot happen.

Based on this algorithm a new neighborhood N_5 can be defined. If the height of a disjunctive arc on a critical circuit cannot be increased, a new height function is computed based on the arc set A'.

Note by applying the neighborhood N_5 on the instance given in Example 7.11 we can reach the optimal solution. However, it is still an open question whether the neighborhood is opt-connected.

Now we want to illustrate the neighborhood N_5 with the continuation of Example 7.17.

Example 7.24 *If we apply neighbourhood N_5 to the problem presented in Example 7.17, we gets a solution with cycle time $\alpha = 14$. This solution is shown in Figure 7.14. The only difference between the solutions computed by N_4 and N_5 is the processing order between the operations $(6; k+1)$ and $(1; k)$ for all $k \in \mathbb{Z}$. In the solution computed by neighbourhood N_5 the operations $(1; k)$ are processed before the operations $(6; k+1)$. In the solution computed by neighbourhood N_4 the operations $(6; k+1)$ are processed before the operations $(1; k)$.*

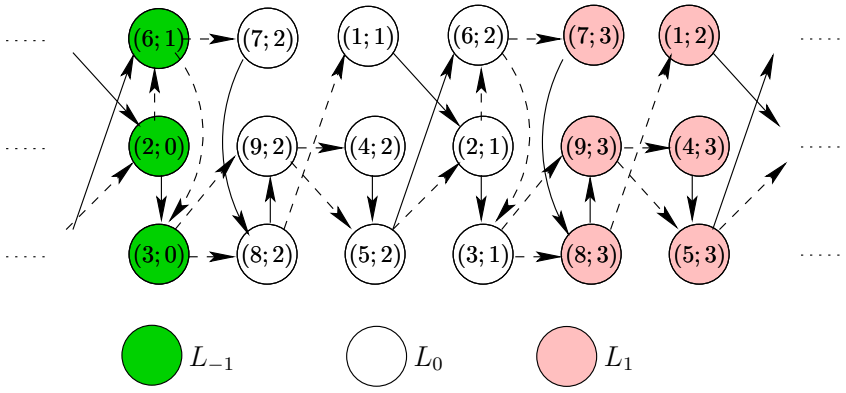

Figure 7.14: Solution computed by neighbourhood N_5

Now we present a slightly different repair algorithm based on a result for the non-cyclic job shop, which again can be found in Nieberg [55].

Theorem 7.25 *For the non-cyclic job shop problems we can always find a new feasible solution by eliminating in a feasible solution one job and putting the job at the beginning of the schedule by reversing all disjunctive arcs which end at operations of this job.*

To use this idea, we have to change the conditions (7.25) and (7.26) of the set L_0 for the new repair algorithm to:

For all jobs $j \in J \setminus \{j', j''\}$ the following inequalities for the occurrence number must hold

$$k_j > k_{j''} - HX_{\star 0} \tag{7.32}$$

and

$$1 - HX_{\star 0} \leq max_{j \in J'} k_j - min_{j \in J'} k_j \leq HX_{\star 0} \tag{7.33}$$

for all $J' \in \mathcal{P}(J)$.

So, we can now describe a new algorithm to compute a new feasible solution in which the height of a disjunctive arc $(b(i'), i'')$ on a critical circuit is increased. This algorithm is quite similar to the previous proposed algorithm.

i. Fix a set L_0 in which each operation occurs exactly once in G_{exp}. The occurrence number k_j for all operations $i \in O(j)$ of the same job $j \in J$ in the set L_0 must be the same. Furthermore, $k_{j'} := 0$, $k_{j''} := HX_{b(i')\, i''}$ and the inequalities (7.32) and (7.33) must hold.

ii. Replace all disjunctive arcs $((b(c); k_{j(c)}), (d; k_{j(c)} + HX_{b(c)\, d}))$ by $((c; k_{j(c)}), (d; k_{j(d)}))$ and the corresponding alternative arc by $((d; k_{j(d)}), (c; k_{j(c)} + 1))$, if $k_{j(c)} + HX_{b(c)\, d} \notin \{k_{j(d)}, k_{j(d)} + 1\}$.

iii. Change the processing orders of the operations in the set L_l so that all operations of job j'' are processed as the first operations in the set L_l for all $l \in \mathbb{Z}$.

Note that the second step is the same for both algorithms. Then we can show with the same ideas as for the first algorithm that after the second step of the algorithm there exists no circuit with positive length in G^{new}_{exp}. Now we explain that the third step of the second algorithm leads to a new relevant and consistent height function.

Theorem 7.26 *If we change the order in the set L_0 so that all operations of job j'' are scheduled as first, then we get a new consistent height function.*

Proof: Before changing the order in the set L_0 all disjunctive arcs which end at operations of job j'' in L_0 start either at operations in L_{-1} or L_0. The set of disjunctive arcs which end at operations of job j'' in L_0 and start at operations in L_0 is denoted with R.

Consider a disjunctive arc $((b(i');k_{j(i')}),(i'';k_{j''}))$ in R and its alternative arc $((b(i'');k_{j''}),(i';k_{j(i')}+1))$ with $(b(i');k_{j'}),(i'';k_{j(i'')}) \in L_0$ and $(i';k_{j'}+1) \in L_1$. These arcs are replaced by $((b(i'');k_{j''}),(i';k_{j(i')}))$ and $((b(i');k_{j(i')}),(i'';k_{j''}+1))$ with $(b(i'');k_{j''})$, $(i';k_{j(i')}) \in L_0$ and $(i'';k_{j''}+1) \in L_1$. This is also done for all the other arcs in R and its alternative arcs.

Therefore, the height $HX_{b(i')\,i''} = k_{j''} - k_{j(i')}$ of the arc $(b(i'),i'')$ is increased by one and the height $HX_{b(i'')\,i'} = 1 + k_{j(i')} - k_{j''}$ of the alternative arc $(b(i''),i')$ is decreased by one. So, the new heights are smaller or equal to $HX_{\star 0}$ and greater or equal to $1 - HX_{\star 0}$ because $k_{j''} - k_{j(i')} < HX_{\star 0}$ and $1 - k_{j''} + k_{j(i')} > 1 - HX_{\star 0}$ hold due to constraint (7.32).

Thus, after replacing these arcs all new arcs which start at operations of j'' in L_0 end at operations in L_0 and all old and new arcs which end at operations of j'' in L_0 start at operations in L_{-1}.

Furthermore, there exists no path from a last operation l of a job j, $j \neq j''$, in L_0 to the first operation $(a;k_{j''})$ of job j'' in L_0 passing the nodes $(0;k_{j''})$ and $(\star;k_{j''} - HX_{\star 0})$ because in this case the occurrence number of l must be $k_{j''} - HX_{\star 0}$, which violates the constraint (7.32).

Now assume that after replacing all the arcs in R and their alternative arcs, there exists a circuit μ in G_{exp}^{new} with positive length. Due to Theorem 7.21, there exists no circuit with positive length in the graph G_{exp}^{new} before replacing the arcs. Thus, there must be at least one new disjunctive arc in the circuit μ. Since we only insert arcs which end at operations in L_0 and start at operations in L_{-1} or L_0 all nodes in μ must be in L_0 because there exists no path from nodes in L_0 to nodes in L_{-1}. As each new disjunctive arc which connects nodes in L_0 starts at operation of job j'' and each arc which ends at operation of job j'' starts at operation in L_{-1}, there cannot exist a new arc in the circuit μ. Thus, there cannot exist a circuit with positive length in G_{exp}^{new} after replacing the arcs. □

The heuristics H_1 and H_2 can be easily adapted to the second repair algorithm.

Therefore, we can now define two additional neighborhoods N_6 and N_7 which are both based on the second algorithm. The arc set which is changed while applying the second algorithm is denoted by B and the corresponding height function is denoted by HX^B. Note that we can also compute a smaller set $B' \subset B$, leading to a new feasible solution by using the same technique which is used to compute the set $A' \subset A$ for the neighborhood N_5. So, the neighborhood N_6 (N_7) is defined as follows:

To each feasible height function HX the set $N_6(HX)$ $(N_7(HX))$ of all height functions derived from HX is assigned as follows: We start with choosing an arc $(b(i'), i'') \in D$ in G on a critical circuit and replace the height $HX_{b(i')\,i''}$ by $HX_{b(i')\,i''} + x$ with $x = HX^{+}_{b(i')\,i''} - HX_{b(i')\,i''}$ if $x > 0$ or we replace the height function HX by HX^B $(HX^{B'})$.

Finally, we propose two combined neighborhoods N_8 and N_9. N_8 (N_9) assigns to each feasible height function HX the set $N_8(HX)$ $(N_9(HX))$ of all height functions derived from HX by choosing an arc $(b(i'), i'') \in D$ on a critical circuit and

- replacing the corresponding $HX_{b(i')\,i''}$-value by $HX_{b(i')\,i''}+x$, with $x = HX^{+}_{b(i')\,i''} - HX_{b(i')\,i''}$ if $x > 0$ or
- replacing the height function by HX^A $(HX^{A'})$ or
- replacing the height function by HX^B $(HX^{B'})$.

Furthermore, due to Theorem 7.12, we introduce the neighborhoods N_i^{Block} for $i = 4, \ldots, 9$. In this neighborhood, we only change the height of the first or the last arc of a block.

Now we want to analyse the case that several operations of a job are processed on the same machine. We adapt our proposed repair algorithm for this new situation. First we have to derive new upper and lower bounds for this situation.

Lemma 7.27 *Consider the disjunctive arc $(b(i'), i'')$ with $j(b(i')) = j(i'')$. The operation i' is a predecessor of i''. Then*

$$1 - HX_{\star 0} \leq HX_{b(i')\,i''} \leq 0 \tag{7.34}$$

and

$$1 \leq HX_{b(i'')\,i'} \leq HX_{\star 0} \tag{7.35}$$

must hold.

Proof: The operations i' and i'' need to be processed on the same machine. Thus, there exists a disjunctive arc $(b(i'), i'')$ and its alternative arc $(b(i''), i')$. Due to the conjunctive arcs there exists a path from i' to $b(i'')$ with height 0. Therefore, the height of the disjunctive arc $(b(i''), i')$ must be greater than 1 and due to Lemma 7.14 smaller or equal to $HX_{\star 0}$. Thus,

$$1 \leq HX_{b(i'')\,i'} \leq HX_{\star 0}$$

holds. This is equivalent to

$$1 \leq 1 - HX_{b(i')\,i''} \leq HX_{\star 0}$$

or

$$0 \geq HX_{b(i')\,i''} \geq 1 - HX_{\star 0}.$$

□

So, we can only find a new feasible height function in which the height of the arc $(b(i'), i'')$ is increased if $b(i')$ is a predecessor of i'' and $HX_{b(i')\,i''} < 0$ hold or $b(i')$ is a successor of i'' and $HX_{b(i')\,i''} < HX_{\star 0}$ hold. Thus, the case is only relevant if $HX_{\star 0} > 1$ because otherwise the heights of the disjunctive arcs cannot be changed (see constraints (7.34) and (7.35)).

We first derive an algorithm for the case that $b(i')$ is a predecessor of i''. The main idea is to divide the job $j(i')$ into at least two jobs and at most $HX_{\star 0}$ jobs. In the following we assume that the job is divided into two new jobs j'_1 and j'_2. The job j'_1 consists of the first operations of the job $j(i')$ up to operation $b(i')$. The job j'_2 consists of the rest of the operations.

If $b(i')$ is a successor of i'', then we divided the job also into two new jobs j'_1 and j'_2. The job $j_2(i')$ consists of the first operations of the job $j(i')$ up to the predecessor of i' and the second job j'_1 consists of the rest of the operations.

Now we need again to fix the set L_0 in which each operation occurs exactly once in G_{exp}. However, the occurrence number k_j for all operations $i \in O(j)$ of the same job $j \in J \cup \{j'_1\} \cup \{j'_2\} \setminus \{j\}$ in the set L_0 must be the same. The occurrence number of job j'_1 is set to $k_{j'_1} := 0$ and the occurrence number of job j'_2 is set to $k_{j'_2} := HX_{b(i')\,i''}$. For all other jobs we fix the occurrence number such that constraints (7.25) and (7.26) are fulfilled. Then we can perform the last two steps of our proposed algorithm which leads to a new consistent and feasible height function in which the height of the arc $(b(i'), i'')$ is increased by one.

Now we want to illustrate our new algorithm by the following example.

Example 7.28 *The example is the same as in Example 7.11. The only difference is that the height of the arc* $(\star, 0)$ *is set to* 2*. The height of the disjunctive arcs is given in Table 7.7.*

Arc	*(2,6)*	*(2,4)*	*(3,7)*	*(3,5)*	*(5,6)*	*(5,7)*
Height	*0*	*0*	*0*	*0*	*1*	*1*
Length	*0*	*0*	*0*	*0*	*0*	*3*
Alternative arc	*(7,1)*	*(5,1)*	*(7,2)*	*(5,2)*	*(7,4)*	*(7,5)*
Height	*1*	*1*	*1*	*1*	*0*	*0*
Length	*0*	*0*	*1*	*3*	*0*	*1*

Table 7.7: Height of the disjunctive arcs

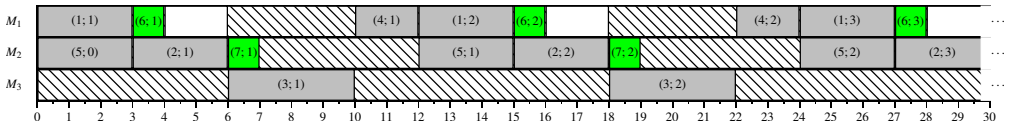

Figure 7.15: Gantt chart with cycle time $\alpha = 12$ for Example 7.28

The corresponding gantt chart with $\alpha = 12$ *is given in Figure 7.15. The critical circuit is* $(1, 2, 3, 4, 5, 1)$. *Thus, the only disjunctive arc which must be changed is* $(5, 1)$. *Both operations of this arc belong to the same job. Therefore, we divide the first job into two jobs* 1 *with operations* $1, 2$ *and* 3 *and job* $1'$ *with operations* 4 *and* 5. *We choose* $k_{1'} = 1, k_1 = 0$ *and* $k_2 = 0$. *After applying the algorithm we get the optimal solution with* $\alpha = 9$. *The gantt chart is given in Figure 7.16.*

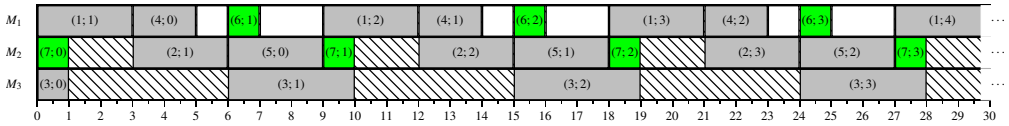

Figure 7.16: Gantt chart with cycle time $\alpha = 9$ for Example 7.28

In the previous paragraphs, we developed neighborhoods, which can be applied to cyclic job shop problems with blocking. In the following, we check whether these neighborhoods can be adapted to cyclic job shop problems with job chain and machine chain repetition. Furthermore, we have also to adjust the conditions (7.25) and (7.26) to the new situation.

Cyclic Job Shop Problems With Job Chain Repetition

For cyclic job shop problems with job chain repetition there exists no conjunctions between the different jobs. Therefore, we cannot develop an upper bound on the height of a disjunctive arc.

This leads to the following new result for the occurrence numbers of the jobs in the set L_0, which need to be computed during the repair algorithm.

Assume that the disjunctive arc $(b(i'), i'')$ lies in a critical circuit and $HX^+_{b(i')\,i''} = HX_{b(i')\,i''}$ holds. Then we fix $k_{j(i')} = 0$ and $k_{j(i'')} = HX_{b(i')\,i''}$. For all other jobs we can choose any integer value $k_j \in \mathbb{Z}\ \forall\, j \in J \setminus \{j(i'), j(i'')\}$.

Now we have to check after we apply the proposed algorithm for cyclic job shop problems with blocking to problems with job chain repetition whether the algorithm still computes a new feasible solution. Therefore, we must check whether Lemma 7.19 and Theorem 7.21 and 7.22 still hold because the proofs of the lemma and the theorems are based on conditions 7.25 and 7.26. All other developed lemmas are based on special properties of the expanded graph G_{exp}.

As the arcs from the last operation of a job to the first operation of a job are the only conjunctive arcs which connect operations of a set L_k with operations of a set $L_{k'}$ with $k' > k$, Lemma 7.19 is immediately fulfilled. Theorem 7.21 and 7.22 are fulfilled, too. Therefore, we can reuse the developed neighborhoods also for cyclic job shop problems with blocking and with job chain repetition.

Also the heuristics H_1 and H_2 can be reused. The only difference is that we do not have to check whether feasible occurrence numbers are computed.

Cyclic Job Shop Problems With Machine Chain Repetition

Now we have to adapt the neighborhoods for cyclic job shop problems with machine chain repetition. For these problems, we can again compute an upper bound for a disjunctive arc based on the given conjunctions.

Lemma 7.29 *If $(b(i), i') \in D$ is a disjunctive arc with $M(i) = M(i')$, then there exists no consistent and relevant height function HX satisfying $HX_{b(i)\,i'} > HX_{MPS}$ or $HX_{b(i')\,i} < 1 - HX_{MPS}$.*

Proof: Consider the situation in Figure 7.17. Then we can easily adapt the proof of Lemma 7.14 to prove this result.

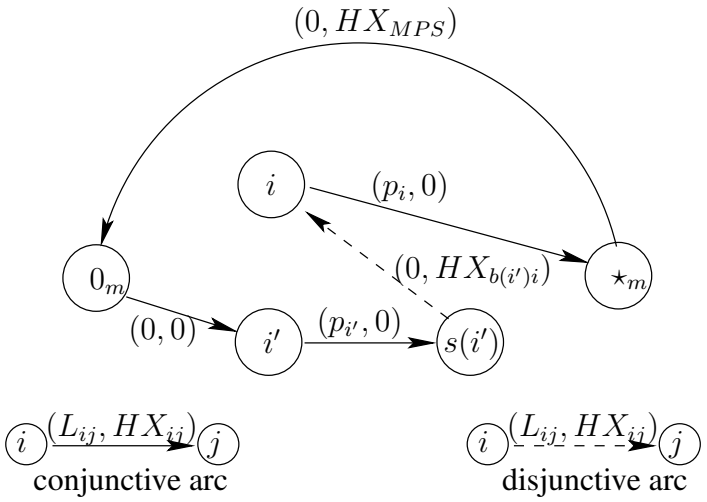

Figure 7.17: Situation in Proof of Lemma 7.29

□

Thus, the new constraints on the occurrence number are:

Consider again the disjunctive arc $(b(i'), i'')$ with $HX_{b(i')\,i''} < HX_{MPS}$ which height must be increased. We again denoted the job $j(i')$ by j' and $j(i'')$ by j''. Then we fix $k_{j(i')} := 0$ and $k_{j(i'')} := HX_{b(i')\,i''}$. For all other jobs $j \in J \setminus \{j', j''\}$ we fix the occurrence number such that

$$k_j < HX_{MPS} \tag{7.36}$$

and

$$1 - HX_{MPS} \leq \max_{j \in J'} k_j - \min_{j \in J'} k_j \leq HX_{MPS} \tag{7.37}$$

for all $J' \in \mathcal{P}(J)$ hold.

Note, heuristic H_1 computes feasible occurrence numbers. Furthermore, we can easily change the heuristic H_2 in such a way that this heuristic also computes feasible occurrence numbers.

Now we have check whether Lemma 7.19 is still valid for the cyclic job shop problems with blocking and machine chain repetition.

Lemma 7.30 *If* (7.37) *holds, then there exists no path* μ *in* G^{new}_{exp} *from* $(c; k_{j(c)}) \in L_0$ *to* $(d; k_{j(d)}) \in L_{-\nu}$ *with* $\nu > 0$.

Proof: Assume there exists a path μ from $(c; k_{j(c)}) \in L_0$ to $(d; k_{j(d)}) \in L_{-\nu}$. Due to the change of the disjunctive arcs, all disjunctive arcs start at operations in L_0 and end at operations in L_0 or L_1. The conjunctive arcs between the operations of a job start in L_0 and end in L_0. Therefore, at least one of the arcs $((\star_m, k), (0_m, k + HX_{MPS}))$ for some $m = 1, \ldots, M$ connects an operation of a set L_k with $k \geq 0$ with an operation of a set $L_{k'}$ with $k' \leq -\nu$. Assume that the arc $((\star_{m'}; k), (0_{m'}; k + HX_{MPS}))$ is one of these arcs.

We denote the last operation on the path μ before the node $(\star_{m'}; k)$ with $(e; k)$ and the first operation after the node $(0_{m'}; k + HX_{MPS})$ with $(f; k + HX_{MPS})$. The node $(e; k)$ can be in any set $L_{k'}$ with $k' \geq 0$ and the node $(f; k + HX_{MPS})$ can be in any set $L_{k''}$ with $k'' \leq -\nu$. Thus, $(e; k - k') \in L_0$ and $(f; k + HX_{MPS} - k'') \in L_0$ hold.

Now we check whether inequality (7.37) holds. If $k - k' \geq k + HX_{MPS} - k''$ we get

$$\begin{aligned}
& \max\{k_{j(e)}, k_{j(f)}\} - \min\{k_{j(e)}, k_{j(f)}\} \\
= \; & \max\{k - k', k + HX_{MPS} - k''\} - min\{k - k', k + HX_{MPS} - k''\} \\
= \; & k - k' - (k + HX_{MPS} - k'') \\
= \; & -k' - HX_{MPS} + k'' \\
\leq \; & 0 - HX_{MPS} - \nu \\
< \; & 1 - HX_{MPS}.
\end{aligned}$$

For the other case we get

$$\begin{aligned}
& \max\{k_{j(e)}, k_{j(f)}\} - \min\{k_{j(e)}, k_{j(f)}\} \\
= \; & k + HX_{MPS} - k'' - (k - k') \\
= \; & HX_{MPS} - k'' + k' \\
\geq \; & HX_{MPS} + \nu \\
> \; & HX_{MPS}.
\end{aligned}$$

Thus, both cases lead to a contradiction to inequality (7.37). □

Finally, we can easily verify that Theorems 7.21 and 7.22 are still valid for job shop problem with blocking and machine chain repetition.

Therefore, the proposed algorithm can compute to each solution a new feasible solution in which the height of the arc $(b(i'), i'')$ is increased, if conditions (7.36) and (7.37) are fulfilled. To solve instances of the job shop problems with blocking and machine chain repetition, we apply the adjusted neighborhoods N_i with $i = 4, \ldots, 9$.

7.4.4 Neighborhoods for Robotic Cell Problems

In this section, we discuss how to adjust the developed neighborhoods to the robotic cell problems without time window constraints. For these problems we assume that the triangle inequality $d_{ij} + d_{jk} \geq d_{ik}$ holds for all machines i, j, k. As the robotic cell problems are based on the cyclic job shop problems with machine chain repetition, we can apply the neighborhoods which are developed for the problem without transportation. However, as there exist several operations, namely the transport operations of a job, which are processed on the same machine, the robot, we show in the following that the height of the disjunctive arcs between these operations can be always increased. As mentioned in Section 6.2 each non-transport operation has a preceding and succeeding transport operation. So, we have the following disjunctive constraints for a robotic cell problem:

- For all transport operations T_i and $T_{i'}$ with $T_i \neq T_{i'}$ and $k, l \in \mathbb{Z}$ we get

$$\begin{aligned} &t(T_i; k) + p_{T_i} + d_{M(i)\,M(p(i'))} \leq t(T_{i'}; l) \\ \vee\, &t(T_{i'}; l) + p_{T_{i'}} + d_{M(i')\,M(p(i))} \leq t(T_i; k) \end{aligned} \tag{7.38}$$

- For all operations i and i' with $M(i) = M(i')$, $i \neq i'$ and $k, l \in \mathbb{Z}$ we get

$$\begin{aligned} &t(T_{s(i)}; l) + p_{T_{s(i)}} + d_{M(s(i))\,M(p(i'))} \leq t(T_{i'}; k) \\ \vee\, &t(T_{s(i')}; l) + p_{T_{s(i')}} + d_{M(s(i'))\,M(p(i))} \leq t(T_i; k). \end{aligned} \tag{7.39}$$

  Note that each operation in a job is processed on a different machine and therefore, $j(i) \neq j(i')$ holds.

First, we consider two transport operations T_i and T_{i+1} with $j(T_i) = j(T_{i+1})$.

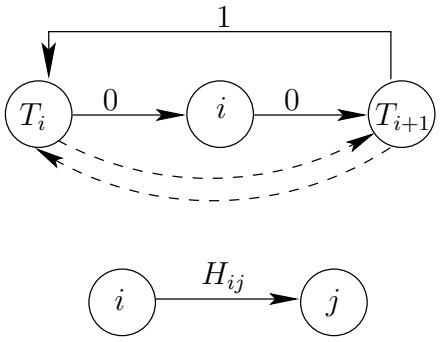

Figure 7.18: The conjunctive and disjunctive arcs between T_i and $T_{i'}$

Now we show that we cannot change the height of the disjunctive arcs (T_i, T_{i+1}) and (T_{i+1}, T_i). In Figure 7.18 the transport operation T_i and T_{i+1} and the operation i are shown. Additionally, to this, the conjunctive arcs between T_i and T_{i+1}, due to constraint (6.11), between T_i and i, due to constraint (6.4), and between i and T_{i+1}, due to (6.10), are shown. For this figure, we can conclude that the lower bound for the height of the arc (T_i, T_{i+1}) is 0. The lower bound of (T_{i+1}, T_i) is 1. Thus, $HX_{T_i\,T_{i+1}} \geq 0$ must hold. This is equivalent to $1 - HX_{T_{i+1}\,T_i} \geq 0$ or $1 \geq HX_{T_{i+1}\,T_i}$. As the lower bound of the arc (T_{i+1}, T_i) is 1, we get $1 \geq HX_{T_{i+1}\,T_i} \geq 1$ So, $HX_{T_{i+1}\,T_i}$ must be fixed to 1 and $HX_{T_i\,T_{i+1}}$ must be fixed to 0. Therefore, we can consider these arcs as conjunctive constraints.

Now consider a disjunctive arc between two transport operations T_i and $T_{i'}$ with $i \pm 1 \neq i'$ which is derived from (7.38). Furthermore, we assume that there exists no parallel disjunctive arc between T_i and $T_{i'}$ which is derived from (7.39). Thus, the corresponding arc of $(T_i, T_{i'})$ is $(T_{i'}, T_i)$. Now we want to show that the height of the disjunctive arc $(T_i, T_{i'})$ can be increased at least by one if the arc is in a critical circuit. Therefore, we have to check whether the conditions of Theorem 7.5 are fulfilled. We can easily check that constraint (7.6) of Theorem 7.5 is fulfilled.

Now we show that constraint (7.5) of Theorem 7.5 is also fulfilled. Consider a path μ from the transport operation T_i to any other transport operation T_j which does not pass the nodes i and T_{i+1}. So,

$$L(\mu) \geq p_{T_i} + d_{M(i)\,M(p(j))}$$

holds, because the length of an arc which start at transport operation T_i and does not end at operation i is greater or equal to $p_{T_i} + d_{M(i)\,M(p(j))}$ (see also (7.38) and (7.39)). If the path μ passes the nodes i and T_{i+1}, we get

$$\begin{aligned} L_{T_i\,i} + L_{i\,T_{i+1}} + L(\mu_1) &\geq p_{T_i} + p_i + p_{T_{i+1}} + d_{M(i+1)\,M(p(j))} \\ &= p_{T_i} + p_i + \epsilon_{M(i)} + d_{M(i)\,M(i+1)} + \epsilon_{M(i+1)} + d_{M(i+1)\,M(p(j))} \\ &\geq p_{T_i} + d_{M(i)\,M(i+1)} + d_{M(i+1)\,M(p(j))} \\ &\geq p_{T_i} + d_{M(i)\,M(p(j))}, \end{aligned}$$

where μ_1 is the sub-path of μ which connects T_{i+1} and T_j.

Thus, for the delay of a path μ from T_i to any other transport operation T_j

$$L(\mu) \geq p_{T_i} + d_{M(i)\,M(p(j))}$$

holds.

If we now consider a path μ from T_i to $T_{i'}$ with at least two arcs, the path μ must pass at least one other transport operation $T_j \neq T_{i'}$. So, we get

$$L(\mu) > p_{T_i} + d_{M(i)\,M(p(i'))}$$

because the processing time of any transport operation is greater than zero. Thus, we can apply Theorem 7.5 and therefore, the height of the disjunctive arcs between the transport operation T_i and $T_{i'}$ can be increased at least by one. If there exists a parallel disjunctive arc between T_i and $T_{i'}$ which is derived from (7.39), we can apply the proposed repair algorithm for the cyclic job shop problem with machine chains repetition and blocking. Summarizing, to solve robotic cell problems we apply the adjusted neighborhoods N_i with $i = 4, \ldots, 9$.

Now we consider the case that there exist time window constraints for the robotic cell problems. Assume that we want to increase the height of the disjunctive arc $(b(i'), i'')$. Without the given time window constraints we showed in the previous sections that there exists a feasible height function in which the height of the arc $(b(i'), i'')$ is increased by one. Unfortunately, this cannot be proved if there exist time window constraints.

Therefore, we propose the following neighborhood for the robotic cell problems with time window constraints which does not find in all cases a new feasible height function. The idea is quite similar to the idea of the neighborhood N_5.

We increase the height of the disjunctive arc $(b(i'), i'')$ by one. If $HX_{ij}^{+} = HX_{ij}$ holds, then we get at least one circuit μ which does not fulfill the conditions of Theorem 2.1. Thus, we must change the height of a disjunctive arc in this circuit. Therefore, the algorithm checks if there exists a disjunctive arc in the circuit μ which can be increased and which leads to a new feasible solution. If there does not exist such an arc, the algorithm chooses an arbitrary arc in this circuit, which height is increased by one. Afterwards, we check again whether there exists a circuit which does not fulfill the conditions of Theorem 2.1 and starts again with searching for a disjunctive arc which height can be increased.

The process stops after a given number C of iterations or after a new feasible height function is found.

We denote this neighborhood by N_{10}^{C}.

Up to now, we presented several different neighborhoods, which can be used to solve different cyclic scheduling problems. In the following subsection we describe two different start heuristics.

7.5 Start heuristics

In this section, we describe how we compute a starting solution for the different application, for which we implemented the tabu search. We present two different heuristics. The first heuristic can be used to compute a starting solution for all different applications. The second heuristic can only be applied on the different cyclic job shop applications without blocking.

We start with the first heuristic SH_1. All applications with and without blocking have in common that we get a feasible solution if we schedule one job after another. Thus, the first heuristic computes a feasible and consistent height function by scheduling job 1 as the first job, job 2 as the second job and so on, until all jobs are scheduled.

The second heuristic is a more complex heuristic, which can be applied on problems without blocking. Note that in this case the delay of the conjunctive and disjunctive arcs are positive integer values. Thus, a height function HX is called consistent if each circuit μ has a positive height.

Due to this definition of a consistent height function, we can derive the following upper and lower bounds on the height of a disjunctive arc. Both bounds can be computed in polynomial time.

Lemma 7.31 *Assume that the heights of the arcs (i,j) and (j,i) are not fixed. Then*

$$HX_{ij}^{+} := \min\{HX(\tau) | \tau \textit{ is a path from } i \textit{ to } j \textit{ in } (T, E \cup D \setminus \{(i,j)\}\} \tag{7.40}$$

*is an **upper bound** on the height of the arc (i,j) and*

$$HX_{ij}^{-} := 1 - \min\{HX(\tau) | \tau \textit{ is a path from } j \textit{ to } i \textit{ in } (T, E \cup D \setminus \{(j,i)\}\} \tag{7.41}$$

*is a **lower bound** on the height of the arc (i,j).*

Proof: First, we show that HX_{ij}^{+} is an upper bound, then we show that HX_{ij}^{-} is a lower bound.

In order to get a relevant and consistent height function the following inequality must hold for every path τ from i to j:

$$HX(\tau) + HX_{ji} \geq 1.$$

So, it must also hold for the path τ^{-} with the lowest height value. Thus, we get

$$\begin{aligned} HX(\tau^{-}) + HX_{ji} &\geq 1 \\ \text{or } HX_{\tau^{-}} &\geq HX_{ij} \end{aligned}$$

Therefore, as $HX_{\tau^-} = HX_{ij}^+$ holds, HX_{ij}^+ is an upper bound on the height of the arc (i, j).

The lower bound can be proved in a similar way. In order to get a relevant and consistent height function the following inequality must hold for every path τ from j to i:

$$HX(\tau) + HX_{ij} \geq 1.$$

So, it must also hold for the path τ^- with the lowest height value. Thus, we get

$$\begin{aligned} HX(\tau^-) + HX_{ij} &\geq 1 \\ \Leftrightarrow HX_{ij} &\geq 1 - HX(\tau^-) \end{aligned}$$

Therefore, as $1 - HX(\tau^-) = HX_{ij}^-$ holds, HX_{ij}^- is a lower bound on the height of the arc (i, j). □

Based on these two bounds we develop the second heuristic. The heuristic starts with $G := (T, E)$ and a relevant and consistent height function HX defined on the set E. The function HX is extended by $(i, j), (j, i) \in D \setminus \overline{E}$ by setting HX_{ij} to HX_{ij}^+ and HX_{ji} to $1 - HX_{ij}^+$, where $\overline{E}$ desribes the set of disjunctive and conjunctive arcs for which the height is already fixed.

Now the question is how to choose a disjunctive arc (i, j), which is added to the current arc set $\overline{E}$. To choose an arc, we first compute the current cycle time $\alpha_{\overline{E}}$ for the graph $G := (T, \overline{E})$ and choose an operation k from a critical circuit.

Then we compute the following two values a_{ij}^+ and a_{ji}^+. The maximum value of these two values is denoted by $a_{max}^+(i, j)$.

$$\begin{aligned} a_{ij}^+ = \; & \max\{L(\mu) - \alpha_{\overline{E}} HX(\mu) | \mu \text{ is a path from } k \text{ to } i\} + \\ & L_{ij} - \alpha_{\overline{E}} HX_{ij}^+ + \\ & \max\{L(\tau) - \alpha_{\overline{E}} HX(\tau) | \tau \text{ is a path from } j \text{ to } k\} \end{aligned} \tag{7.42}$$

$$\begin{aligned} a_{ji}^+ = \; & \max\{L(\mu) - \alpha_{\overline{E}} HX(\mu) | \mu \text{ is a path from } k \text{ to } j\} + \\ & L_{ji} - (1 - \alpha_{\overline{E}} HX_{ij}^+) + \\ & \max\{L(\tau) - \alpha_{\overline{E}} HX(\tau) | \tau \text{ is a path from } i \text{ to } k\} \end{aligned} \tag{7.43}$$

The next arc (i,j) which is added to the arc set $\overline{E}$ is the arc with the minimal $a^+_{max}(i,j)$-value.

Summarizing, we have the following procedure SH_2 which provides a relevant and consistent height function.

```
1  Ē := E;
2  D := List of all disjunctions (i,j) with i < j;
3  while D ≠ ∅ do
4      a⁺_min := ∞;
5      α_Ē := Actual cycle time of G := (T, Ē);
6      for each arcs (i,j) ∈ D do
7          Calculate HX⁺_ij according to (7.40);
8          Calculate HX⁻_ij according to (7.41);
9          if (HX⁻_ij > HX⁺_ij) then
10             return No_consistent_solution;
11         Compute a⁺_ij according to (7.42);
12         Compute a⁺_ji according to (7.43);
13         a⁺_max(i,j) := max{a⁺_ij, a⁺_ji};
14         if (a⁺_min > a⁺_max) then
15             a⁺_min := a⁺_max;
16             (i_c, j_c) := (i,j);
17     D := D \ (i_c, j_c);
18     Set HX_{i_c j_c} := HX⁺_{i_c j_c};
19     Ē := Ē ∪ {(i_c, j_c), (j_c, i_c)};
```

Listing 4: Procedure SH_2

Now we show that the second heuristic always computes a relevant and consistent height function.

Theorem 7.32 *The procedure SH_2 computes always a relevant and consistent height function.*

Proof: The procedure terminates with an inconsistent height function, if $HX^-_{ij} > HX^+_{ij}$ holds. This is equivalent to

$$1 - HX^+_{ji} > HX^+_{ij}$$

or

$$1 > HX^+_{ij} + HX^+_{ji}.$$

Thus, the procedure cannot find a consistent height function if there exists a circuit with non-positive height. However, as the height of a disjunctive arc is fixed to the upper bound, there cannot exist a circuit with non-positive height in the graph $G = (T, \overline{E})$ after the end of the while-loop. Thus, in each step of the while-loop $HX_{ij}^{-} \leq HX_{ij}^{+}$ holds. $\square$

Up to now, we have presented the neighborhoods and the start heuristics for our tabu search. In the following section, we present some implementation details and the computational results.

8 Implementation and Computational Results

In this section, we describe some implementation details and report some computational results. We implemented all algorithms in C and tested these algorithms on a Celeron 1.8GHz computer with operation system Linux and 256MB general storage.

First, we present the test data. Then in Section 8.2 we describe the detailed settings for our tabu search procedure. In Section 8.3 we finally present the computational results for several different applications.

8.1 Test Data

We tested our algorithms on the following applications:

- cyclic job shop with and without blocking and $HX_{\star 0} \in \{1, 2\}$,
- cyclic job shop with machine chains repetition with and without blocking and $HX_{MPS} \in \{1, 2\}$,
- cyclic job shop with job chains with and without blocking and $HX_{Job} \in \{1, 2\}$ and
- cyclic job shop with machine chains and blocking and one transportation robot and $HX_{MPS} = \{1, 2\}$.

As there exists no benchmark instances for all these different applications we transform the job-shop benchmark instances given in Table 8.1 into instances for the consider applications.

Instance	#jobs	#machines	#operations
la01	10	5	50
la02	10	5	50
la03	10	5	50
la04	10	5	50
la05	10	5	50
la06	15	5	75
la07	15	5	75
la08	15	5	75
la09	15	5	75
la10	15	5	75

Instance	#jobs	#machines	#operations
la11	20	5	100
la12	20	5	100
la13	20	5	100
la14	20	5	100
la15	20	5	100
la16	10	10	100
la17	10	10	100
la18	10	10	100
la19	10	10	100
la20	10	10	100
la21	15	10	150
la22	15	10	150
la23	15	10	150
la24	15	10	150
la25	15	10	150
la26	20	10	200
la27	20	10	200
la28	20	10	200
la29	20	10	200
la30	20	10	200
la31	30	10	300
la32	30	10	300
la33	30	10	300
la34	30	10	300
la35	30	10	300
la36	15	15	225
la37	15	15	225
la38	15	15	225
la39	15	15	225

Table 8.1: Job-Shop benchmark problems

These job shop problems are taken from [5].

For the problem with one transport robot we use as underlying problem the instances $la01$ and $la06$. For the transportation times d_{ij} and the un- and loading times ϵ_i for the robot we consider the following cases:

- $d_{ij} = d$ for all $i, j \in M$ and $\epsilon_i = \epsilon$ for all $i \in M$. The names of the test instances are $la0i\_1\_d\_\epsilon$ with $i \in \{1, 6\}$.
- d_{ij}, ϵ_i are randomly generated from an interval $[1, D_{MAX}]$, $[1, \epsilon_{MAX}]$. The chosen d_{ij} values are adjusted so that the triangle inequality holds. The names of the test instances are $la0i\_2\_D_{MAX}\_\epsilon_{MAX}$ with $i \in \{1, 6\}$.
- $d_{ij} = D * |i - j|$ for all $i, j \in M$ and $\epsilon_i = \epsilon$ for all $i \in M$. The names of the test instances are $la0i\_3\_D\_\epsilon$ with $i \in \{1, 6\}$.

The name of the test set for the problems with one transportation robot is JR.HX_{MPS}.

The names of the test instances for the applications without one transportation robot are J.$HX_{\star 0}$ for the cyclic job shop problems, JM.HX_{MPS} for the cyclic job shop problems with machine chains repetition and JC.HX_{Job} for the cyclic job shop problems with job chains repetition.

8.2 Experiments Setup

Several tests have been performed to investigate the influence of

- the neighborhood,
- the choice of a strategy for selecting a neighbor,
- definition of the tabu status and
- organization of the tabu list.

From these preliminary tests we chooses the neighborhoods N_2 and N_3 for the different cyclic job problem without blocking and N_9 and N_9^{Block} together with heuristic H_1 and H_2 for the cyclic job shop problems with blocking. For the problem with one transportation robot we use the adjusted neighborhoods N_9 and N_9^{Block} together with heuristics H_1 and H_2.

In each iteration of the tabu search procedure the current solution is replace by the neighbor with the best solution value.

We use the following type of attributes TB to describe moves. As in the neighborhoods N_3, N_9 and N_9^{Block} several arcs are changed we consider only the first arc which is changed to describe moves. Thus, TB is given by

- the first arc $(b(i), i')$ for which the height $HX_{b(i),i'}$ is changed and
- the value $HX_{b(i)i'}$ before its change.

In connection with TB we choose the following aspiration criterion: accept tabu moves if it improves the best solution found so far.

To decide whether a neighbor is tabu or not we check for all changed arcs whether one of these arcs is in the tabu list TL. If one of the arcs is in the tabu list and the neighbor does not improve the best solution found so far, we cannot accept the neighbor as a new starting solution for the next iteration of the tabu search.

We organize the tabu list TL in the following way: We start with an empty list. Then the attributes of all visited neighbors are inserted into the list until the list has reached a maximal length L_{MAX}. In this case, the oldest entry is replaced by the attribute of the actual solution. If all neighbors of a solution are tabu, then the oldest entries are deleted one by one until one neighbor is not tabu. The maximal tabu list length is computed by $L_{MAX} = n \cdot m \cdot k$ with $k \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$, where n is the number of jobs and m is the number of machines. For the problems without blocking we use as a start heuristic SH_1 and SH_2. For the problems with blocking we use only SH_1.

We decided to stop each run of our tabu search procedure after 1000 non-improving steps or after 3600 seconds of computation time.

8.3 Computational Results

In this section, we present the computational results for the different applications. The section is divided into three parts. In the first part, we consider the applications without blocking, then the applications with blocking and finally, we discuss the results for the problem with blocking and one transportation robot.

In each part, we first present the best results for each test instance. Furthermore we analyse which neighborhood contributes most to the best results. For the analysis we compute the following values:

- the average value Δavg for the relative deviation

$$\Delta := \frac{\alpha_H - \alpha_{Best}}{\alpha_{Best}} \cdot 100\%$$

  of the cycle time α_H provided by the heuristic from the best solution α_{Best}.

- the average computation time ΔCPU in seconds.

Finally, we compare our best results with the optimal results which can be found in the literature.

Cyclic Scheduling Without Blocking

The best results for the different type of applications without blocking are given in Table 8.2. If the best result is equal to the computed lower bound, which means that we have found the optimal solution, then we add the symbol $*$ to the best result.

	Cyclic Job Shop		Cyclic Job Shop with job chains		Cyclic Job Shop with machine chains	
Instance	$H_{\star 0}=1$	$H_{\star 0}=2$	$H_{Job}=1$	$H_{Job}=2$	$H_{MPS}=1$	$H_{MPS}=2$
la01	666.0*	666.0*	666.0*	666.0*	666.0*	666.0*
la02	655.0	635.0*	635.0*	635.0*	635.0*	635.0*
la03	603.0	588.0*	588.0*	588.0*	588.0*	588.0*
la04	590.0	537.0*	537.0*	537.0*	556.0	537.0*
la05	593.0*	593.0*	593.0*	593.0*	593.0*	593.0*
la06	926.0*	926.0*	926.0*	926.0*	926.0*	926.0*
la07	890.0	869.0*	869.0*	869.0*	869.0*	869.0*
la08	863.0*	863.0*	863.0*	863.0*	863.0*	863.0*
la09	951.0*	951.0*	951.0*	951.0*	951.0*	951.0*
la10	958.0*	958.0*	958.0*	958.0*	958.0*	958.0*
la11	1222.0*	1222.0*	1222.0*	1222.0*	1222.0*	1222.0*
la12	1039.0*	1039.0*	1039.0*	1039.0*	1039.0*	1039.0*
la13	1150.0*	1150.0*	1150.0*	1150.0*	1150.0*	1150.0*
la14	1292.0*	1292.0*	1292.0*	1292.0*	1292.0*	1292.0*
la15	1207.0*	1207.0*	1207.0*	1207.0*	1207.0*	1207.0*
la16	962.0	660.0*	756.0	660.0*	781.0	660.0*
la17	785.0	683.0*	683.0*	683.0*	713.0	683.0*
la18	861.0	623.0*	708.5	623.0*	768.0	623.0*
la19	852.0	685.0*	721.0	685.0*	794.0	685.0*
la20	902.0	744.0*	758.0	744.0*	769.0	744.0*
la21	1070.0	935.0*	935.0*	935.0*	963.0	935.0*
la22	960.0	830.0*	830.0*	830.0*	875.0	830.0*
la23	1032.0*	1032.0*	1032.0*	1032.0*	1032.0*	1032.0*
la24	955.0	857.0*	883.0	857.0*	924.5	857.0*
la25	996.0	864.0*	872.0	864.0*	902.0	864.0*
la26	1218.0*	1218.0*	1218.0*	1218.0*	1218.0*	1218.0*

	Cyclic Job Shop		Cyclic Job Shop with job chains		Cyclic Job Shop with machine chains	
Instance	$H_{\star 0} = 1$	$H_{\star 0} = 2$	$H_{Job} = 1$	$H_{Job} = 2$	$H_{MPS} = 1$	$H_{MPS} = 2$
la27	1293.0	1188.0*	1188.0*	1188.0*	1197.0	1188.0*
la28	1242.0	1216.0*	1216.0*	1216.0*	1216.0*	1216.0*
la29	1212.0	1105.0*	1105.0*	1105.0*	1105.0*	1105.0*
la30	1355.0*	1355.0*	1355.0*	1355.0*	1355.0*	1355.0*
la31	1784.0*	1784.0*	1784.0*	1784.0*	1784.0*	1784.0*
la32	1850.0*	1850.0*	1850.0*	1850.0*	1850.0*	1850.0*
la33	1719.0*	1719.0*	1719.0*	1719.0*	1719.0*	1719.0*
la34	1721.0*	1721.0*	1721.0*	1721.0*	1721.0*	1721.0*
la35	1888.0*	1888.0*	1888.0*	1888.0*	1888.0*	1888.0*
la36	1305.0	1028.0*	1153.0	1028.0*	1204.5	1028.0*
la37	1483.0	980.0*	1223.0	980.0*	1326.0	980.0*
la38	1267.0	876.0*	1123.0	876.0*	1157.5	876.0*
la39	1274.0	1012.0*	1137.0	1012.0*	1172.0	1012.0*

Table 8.2: Best results for the different cyclic job shop problems without blocking

As we can see, our tabu search approach computes very good results, especially for the problems with height equal to two. Here we found for all instances and all different applications an optimal solution. It turns out that the computed lower bound is very tight.

In Table 8.3 we compare the different neighborhoods. The main result is that on the average both neighborhoods N_2 and N_3 lead almost to the same results. With the neighborhood N_3 we get slightly better results. Quite interesting is also the impact of the start heuristics. In almost all cases, the second start heuristic leads to better results for both neighborhoods. With the first start heuristic, we get only better results for the test set JC.1.

Test set	Neighborhood N_2 with				Neighborhood N_3 with			
	SH_1		SH_2		SH_1		SH_2	
	Δavg	ΔCPU	Δavg	ΔCPU	Δavg	ΔCPU	Δavg	ΔCPU
J.1	1.74	656	0.46	129	0.48	762	0.21	241
J.2	0.08	463	0.03	83	0.00	372	0.00	250
JC.1	0.49	479	1.00	120	0.25	437	0.47	148
JC.2	0.00	376	0.00	66	0.00	291	0.00	66
JM.1	1.21	737	0.55	139	0.93	790	0.17	256
JM.2	0.00	456	0.00	106	0.00	388	0.00	266

Table 8.3: Comparison between the different neighborhoods and start heuristics

But the main disadvantage of the second start heuristic is the computational time which is needed to compute a feasible solution. The average computational time ΔCPU and the average value Δavg for the relative deviation from cycle time α_H from the best solution α_{Best} is given in Table 8.4. Here α_H is provided by the start heuristic SH_2.

Test set	Δavg	ΔCPU
J.1	16.71	445.05
J.2	5.43	418.61
JC.1	30.20	467.41
JC.2	45.55	466.00
JM.1	12.81	687.39
JM.2	2.99	690.19

Table 8.4: Average computational time for start heuristic SH_2

For almost every test set SH_2 needs the same computational time as the tabu search with the first start heuristic. The average computational time for the first start heuristic is less than 2 seconds.

Finally, we compare the best results for the cyclic job shop with height one known from the literature with our best results. Minimizing the cycle time for the cyclic job shop with height one is equivalent to minimizing the maximal completion time for the classical non-cyclic job shop problem. Here the deviation between our best results and the optimal

results is 1.18%. Thus, the results of our solution method are also very good for this kind of problem.

Cyclic Scheduling With Blocking

In Table 8.5 the best results for the different type of problems with blocking are given.

	Cyclic Job Shop		Cyclic Job Shop with job chains		Cyclic Job Shop with machine chains	
Instance	$H_{\star 0}=1$	$H_{\star 0}=2$	$H_{Job}=1$	$H_{Job}=2$	$H_{MPS}=1$	$H_{MPS}=2$
la01	793.0	775.0	776.0	775.0	775.0	775.0
la02	793.0	748.0	740.0	740.0	757.0	744.0
la03	715.0	656.0	656.0	656.0	656.0	656.0
la04	743.0	677.0	677.0	666.0	714.0	677.0
la05	671.0	645.0	642.0	642.0	662.0	645.0
la06	1145.0	1072.0	1094.0	1094.0	1099.0	1094.0
la07	1048.0	1041.0	1024.0	1016.0	997.0	975.0
la08	1099.0	1071.0	1056.0	1053.0	1087.0	1066.0
la09	1202.0	1187.0	1179.0	1139.0	1173.0	1173.0
la10	1159.0	1113.0	1118.0	1116.0	1121.0	1112.0
la11	1553.0	1460.0	1511.0	1486.0	1509.0	1505.0
la12	1304.0	1304.0	1327.0	1325.0	1319.0	1299.0
la13	1499.0	1468.0	1469.0	1461.0	1465.0	1486.0
la14	1525.0	1498.0	1477.0	1477.0	1539.0	1493.0
la15	1598.0	1513.0	1507.0	1507.0	1542.0	1514.0
la16	1097.0	956.0	942.0	942.0	1031.0	953.0
la17	969.0	868.0	854.0	850.0	931.0	843.0
la18	1046.0	893.0	910.0	910.0	1004.0	879.0
la19	1094.0	920.0	911.0	911.0	1035.0	908.0
la20	1111.0	960.0	934.0	934.0	1018.0	1004.0
la21	1572.0	1432.0	1413.0	1413.0	1500.0	1441.0
la22	1442.0	1327.0	1348.0	1348.0	1390.0	1331.0
la23	1532.0	1422.0	1489.0	1475.0	1486.0	1443.0
la24	1423.0	1372.0	1446.0	1410.0	1478.0	1408.0
la25	1462.0	1375.0	1387.0	1317.0	1439.0	1380.0
la26	2121.0	1941.0	2016.0	1961.0	2030.0	1925.0
la27	2151.0	1997.0	2053.0	2053.0	2117.0	2038.0
la28	2063.0	2004.0	2013.0	1947.0	2121.0	2010.0
la29	1956.0	1834.0	1965.0	1938.0	1894.0	1813.0
la30	2140.0	2019.0	1924.0	1924.0	2136.0	2108.0
la31	3208.0	3073.0	3388.0	3248.0	3062.0	3097.0

	Cyclic Job Shop		Cyclic Job Shop with job chains		Cyclic Job Shop with machine chains	
Instance	$H_{\star 0}=1$	$H_{\star 0}=2$	$H_{Job}=1$	$H_{Job}=2$	$H_{MPS}=1$	$H_{MPS}=2$
la32	3569.0	3345.0	3661.0	3553.0	3418.0	3299.0
la33	3161.0	3103.0	3059.0	2905.0	3246.0	3042.0
la34	3432.0	3206.0	3015.0	2963.0	3352.0	3183.0
la35	3361.0	3134.0	3193.0	3139.0	3232.0	3155.0
la36	1916.0	1716.0	1751.0	1649.0	1769.0	1714.0
la37	2014.0	1840.0	1867.0	1867.0	1918.0	1868.0
la38	1897.0	1597.0	1637.0	1559.0	1822.0	1621.0
la39	1852.0	1697.0	1696.0	1696.0	1788.0	1716.0

Table 8.5: Best results for the different cyclic job shop problems without blocking

In Table 8.6 we compare the different tested neighborhoods and the heuristics H_1 and H_2. It turns out that on average the use of heuristic H_1 leads to better results even for the problems with height 2. For the most problems, the neighborhoods based on the block approach contributes the most to the best results.

One disadvantage of our local search approach is the high computation time which is due to our proposed repair algorithm. We use, as already mentioned, two stop criteria for the tabu search, the time limit and the criterion based on visiting non-improving solutions. For these problems with blocking the stop criterion is in most cases the given time limit of 3600 sec. We can assume that the results get better if we skip the time limit.

Test set	Neighborhood N_9 with				Neighborhood N_9^{Block} with			
	H_1		H_2		H_1		H_2	
	Δavg	ΔCPU	Δavg	ΔCPU	Δavg	ΔCPU	Δavg	ΔCPU
J.1	1.00	2482	2.89	2304	2.23	2415	2.48	2359
J.2	2.10	2494	1.88	2384	1.71	2533	2.07	2411
JC.1	1.69	2444	3.11	2891	1.20	2494	4.26	2929
JC.2	1.91	2603	5.74	2911	2.28	2558	5.90	2932
JM.1	1.49	2650	2.44	2551	1.29	2641	2.53	2548
JM.2	3.38	2786	1.83	2568	2.54	2660	1.87	2743

Table 8.6: Comparison between the different neighborhoods and heuristics

To sum up, the more complex heuristic H_2 is only useful for the cyclic job shop with machine chain repetition and not for all problems with height 2.

In Table 8.7 we compare our best results with the best results for the classical job shop problem with blocking known from the literature. Here again our tabu search approach computes good results. In the same table we also give the average deviation $\Delta avg\ LB$ of the best results to the lower bound. Based on this value we can conclude that the lower bound is not very useful to measure the results of the tabu search approach.

Test set	Δavg	$\Delta avg\ LB$
J.1	2.06	37.64

Table 8.7: Comparison between the best solutions, the optimal solutions, and the lower bound

Cyclic Scheduling With Blocking and One Transportation Robot

In Table 8.8 the best results for the cyclic job shop with blocking and one transportation robot are given.

	Cyclic Job Shop with blocking and one transportation robot	
Instance	$H_{MPS} = 1$	$H_{MPS} = 2$
la01_1_2_2	1146.0	1130.0
la01_1_2_3	1213.0	1213.0
la01_1_3_2	1217.0	1113.0
la01_1_3_3	1311.0	1211.0
la01_1_4_2	1182.0	1182.0
la01_1_4_3	1290.0	1270.0
la01_2_2_2	1146.0	1130.0
la01_2_2_3	1223.0	1196.0
la01_2_3_2	1073.0	1073.0
la01_2_3_3	1199.0	1199.0
la01_2_4_2	1173.0	1125.0
la01_2_4_3	1265.0	1185.0
la01_3_2_2	1237.0	1225.0
la01_3_2_3	1362.0	1347.0
la01_3_3_2	1387.0	1311.0

Instance	Cyclic Job Shop with blocking and one transportation robot	
	$H_{MPS} = 1$	$H_{MPS} = 2$
la01_3_3_3	1432.0	1375.0
la01_3_4_2	1558.0	1500.0
la01_3_4_3	1636.0	1636.0
la06_1_2_2	1847.0	1830.0
la06_1_2_3	1801.0	1801.0
la06_1_3_2	1741.0	1741.0
la06_1_3_3	2045.0	2029.0
la06_1_4_2	1962.0	1962.0
la06_1_4_3	2049.0	2049.0
la06_2_2_2	1811.0	1811.0
la06_2_2_3	1783.0	1783.0
la06_2_3_2	1773.0	1773.0
la06_2_3_3	1854.0	1854.0
la06_2_4_2	1805.0	1805.0
la06_2_4_3	1950.0	1881.0
la06_3_2_2	1984.0	1984.0
la06_3_2_3	2117.0	2115.0
la06_3_3_2	2102.0	2102.0
la06_3_3_3	2334.0	2245.0
la06_3_4_2	2437.0	2408.0
la06_3_4_3	2710.0	2677.0

Table 8.8: Best results for the cyclic job shop with blocking and one transportation robot

In Table 8.9 the different tested neighborhoods together with the heuristics H_1 and H_2 are compared. Here it turns out that the use of heuristic H_2 leads to better results for both test sets.

As for the problems with blocking, the computation time is very high. Again, the given time limit of 3600 seconds stops in the most cases our tabu search. We can assume that the results get better if we skip the time limit.

Test set	Neighborhood N_9 with				Neighborhood N_9^{Block} with			
	H_1		H_2		H_1		H_2	
	Δavg	ΔCPU	Δavg	ΔCPU	Δavg	ΔCPU	Δavg	ΔCPU
JR.1	1.36	3624	1.24	3616	1.65	3625	1.37	3613
JR.2	4.58	3623	3.28	3629	4.70	3643	3.15	3627

Table 8.9: Comparison between the different neighborhoods and heuristics

To sum up, for problems with blocking and one transportation robot the neighborhood N_9 and N_9^{Block} together with heuristic H_2 lead to the best results. As there exists no computational results for this type of problem in the literature, we cannot give any comparison.

9 Concluding remarks

In this thesis, we develop a general framework to describe and to model various cyclic scheduling problems. The foundation for this model is on the one hand the GBCSP and on the other hand, the alternative graph model that describes blocking situations. The latter is known from the classical non-cyclic scheduling literature.

For the GBCSP, we develop the necessary theoretical background to solve general problems without any resource conflicts. Furthermore, we generalize Howard's Algorithm, a practicably very fast algorithm to compute the optimal cycle time. We also adapt the alternative graph model to model cyclic problems with blocking.

We analyse several cyclic scheduling problems both with and without blocking, and transportation robots proposed in the literature, and we show how to model these problems within our general framework. To model special applications in the area of software pipelining we generalize the alternative graph model.

In the last part of this thesis, we develop a local search method to solve general cyclic scheduling problems. To evaluate our new solution method, we test this method with several cyclic scheduling problems from the literature. Here, it turns out that the solution method performs very well on problems without blocking and without any transportation robots and on problems with blocking.

Another result of the evaluation of our local search method is that the computed lower bounds for problems with blocking are not very tight. Therefore, it would be necessary to develop better methods and algorithms to compute such lower bounds for problems with blocking.

Also, new start heuristics for these kinds of problems should be developed. Another interesting aspect would be to analyse the relationship between the flow time and the cycle time. It would be interesting to see how our local search approach works on the problem of minimizing the flow time for a given upper bound on the cycle time. Furthermore, new solution methods for problems with time window constraints, as exist in hoist scheduling problems, should be developed. Although we do not think that a repair algorithm similar to the algorithm proposed by us for the problems with blocking and time windows can be developed. This is due to the fact that it is already a hard problem to find a feasible solution when some of the disjunctive constraints are fixed in advance. Therefore, new ideas for a local search approach must be developed. One idea could be to allow moves to infeasible solutions during the search process.

Another area of further research would be to analyse other software pipelining problems in order to get a better understanding of the types of problems which occur in this area. This could lead to more discussions and exchange between the different research groups.

10 Bibliography

[1] ILOG CPLEX: Mathematical Programming Optimizer.

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

[3] V. Allan, R. Jones, R. Lee, and S. Allan (1995). Software Pipelining. *ACM Computing Surveys* 27(3), 367–432.

[4] E. Balas (1969). Machine sequencing via disjunctive graphs: An implicit enumeration algorithm. *Operations Research* 17(6), 941–957.

[5] J.E. Beasley (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society* 41(11), 1069–1072.

[6] N. Brauner, G. Finke, and W. Kubiak (1997). A proof of the Lei and Wang claim. Technical report, Laboratoire Leibnitz, Institut IMAG, Grenoble.

[7] P. Brucker (2004). *Scheduling algorithms*. Fourth edition. Berlin: Springer-Verlag.

[8] P. Brucker and T. Kampmeyer (2005). Tabu search algorithms for cyclic Machine scheduling problems. *Journal of Scheduling* 8, 303–322.

[9] P. Brucker and T. Kampmeyer (2005). Tabu search algorithms for cyclic machine scheduling problems with blocking. In G. Kendall, L. Lei, and M. Pinedo, eds., *Proceedings of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications*, volume 1, 107–108.

[10] P. Brucker and S. Knust (2000). A linear programming and constraint propagation-based lower bound for the RCPSP. *European Journal of Operational Research* 127, 355–362.

[11] P. Brucker and S. Knust (2003). Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research* 149(2), 302–313.

[12] J. Carlier and P. Chrétienne (1988). *Les Problèmes d'ordonnancement*. Masson, Paris.

[13] G. Chaitin, M. Auslander, A. Chandra, J. Cocke, M. Hopkins, and P. Markstein (1981). Register allocation via coloring. *Comput. Lang.* 6(1), 47–57.

[14] A. Che, C. Chengbin, and E. Levner (2003). A polynomial algorithm for 2-degree cyclic robot scheduling. *European Journal of Operational Research* 145, 31–44.

[15] H. Chen, C. Chu, and J.-M. Proth (1995). Cyclic hoist scheduling based on graph theory. In *Emerging Technologies and Factory Automation, 1995. ETFA '95, Proceedings., 1995 INRIA/IEEE Symposium on*, 451–459.

[16] P. Chrétienne (1991). The basic cyclic scheduling problem with deadlines. *Discrete Applied Mathematics* 30, 109–123.

[17] V. Chvatal (1983). *Linear programming.* New York - San Francisco: W. H. Freeman and Company.

[18] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. Mc Gettrick, and J.P. Quadrat (1998). Numerical Computation of Spectral Elements in Max-Plus Algebra. In *Proc. IFAC Conf. on Sys. Structure and Control*.

[19] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot (1985). A linear system theoretic view of discrete event process and its user for performance evaluation in manufacturing. *IEEE Transactions on Automatic control* 30(3), 210–220.

[20] Y. Crama, V. Kats, J. van de Klundert, and E. Levner (2000). Cyclic scheduling in robotic flow-shops. *Annals of Operations Research* 96, 97–124.

[21] Y. Crama and J. van de Klundert (1997). Cyclic scheduling of identical parts in a robotic cell. *Operations Research* 45(6), 952–965.

[22] G.B. Dantzig, W.O. Blattner, and M.R. Rao (1966). *Finding a cycle in a graph with minimum cost to time ratio with application to a ship routing problem*, 77–84. New York: Gordon and Breach.

[23] A. Dasdan, R.K. Gupta, and S.S. Irani (1998). An Experimental Study of Minimum Mean Cycle Algorithms. Technical Report 98-32, University of California.

[24] D. Fimmel and J. Müller (2001). Optimal software pipelining under resoucre constraints. *International Journal of Foundations of Computer Science* 12(6), 697–718.

[25] F. Glover (1989). Tabu search. I. *ORSA Journal on Computing* 1, 190–206.

[26] F. Glover (1990). Tabu search. II. *ORSA Journal on Computing* 2, 4–32.

[27] N. G. Hall, H. Kamoun, and C. Sriskandarajah (1998). Scheduling in robotic cells: Complexity and steady state analysis. *European Journal of Operational Research* 109(1), 43–65.

[28] N.G. Hall, H. Kamoun, and C. Sriskandarajah (1997). Scheduling in robotic cells: Classification, two and three machine cells. *Operations Research* 45(3), 421–439.

[29] N.G. Hall, T.E. Lee, and M.E. Posner (2002). The Complexity of Cyclic Shop Scheduling Problems. *Journal of Scheduling* 5(4), 307–327.

[30] C. Hanen (1994). Study of a NP-hard cyclic scheduling problem: The recurrent job-shop. *European Journal of Operational Research* 72, 82–101.

[31] C. Hanen and A. Munier (1995). Cyclic scheduling on parallel processors: on overview. In P. Chretienne, E.G. Coffman, J.K. Lenstra, and Z. Liu, eds., *Scheduling Theory and Its Applications*, chapter 4, 194–226. New York: John Wiley and Sons.

[32] C. Hanen and A. Munier-Kordon (2004). Periodic Schedules For Linear Precedence Constraints. Technical report, Laboratoire LIP6, Paris.

[33] K.L. Hitz (1980). Scheduling of Flexible Flow Shops II. Technical Report LIDS-R-1049, Laboratory for Information and Decision Systems, MIT, Cambridge, USA.

[34] R.A. Howard (1960). *Dynamic Programming and Markov Processes*. New York, NY: Technology Press & Wiley.

[35] J. Hurink and S. Knust (2002). A tabu search algorithm for scheduling a single robot in a job-shop environment. *Discrete Applied Mathematics* 119, 181–203.

[36] I. Ichimori and F. Soumis (1995). Schedule efficiency in a robotic production cell. *International Journal of Flexible Manufacturing Systems* 7, 5–26.

[37] I. Ioachim, E. Sanlaville, and M. Lefebvre (2001). The basic cylcic scheduling model for robotic flow shops. *INFOR - Information systems and operational research* 39(3), 257–277.

[38] R.M. Karp and J.B. Orlin (1981). Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics* 3, 37–45.

[39] V. Kats and E. Levner (1998). Cyclic scheduling of operations for a part type in an FMS handled by a single robot: a parametric critical-path approach. *The International Journal of Flexible Manufacturing Systems* 10, 129–138.

[40] S. Knust (1999). *Shop-scheduling problems with transportation*. Ph.D. thesis, Universität Osnabrück, Fachbereich Mathematik/Informatik.

[41] W. Kubiak, S.X.C. Lou, and Y. Wang (1996). Mean flow time minimization in reentrant job shops with hub. *Operations Research* 44, 764–776.

[42] C. Kuijpers (2001). *Cyclic Machine Scheduling with Tool Transportation*. Enschede: Disseration, University of Twente.

[43] T.E. Lee and M. Posner (1997). Performance Measures and Schedules in Periodic Job Shop. *Operations Research* 45, 72–91.

[44] L. Lei and T.J. Wang (1989). A proof: the cyclic hoist scheduling problem is NP-complete. Working paper 890016, Rudgers University.

[45] J.M.Y. Leung, G. Zhang, X. Yang, R. Mak, and K. Lam (2004). Optimal Cyclic Multi-Hoist Scheduling: A Mixed Integer Programming Approach. *Operations Research* 52(6), 965–976.

[46] V. Lev and I. Adiri (1984). V-shop scheduling. *European Journal of Operational Research* 18, 51–56.

[47] E. Levner and V. Kats (1998). A parametrical critical path problem and an application for cyclic scheduling. *Discrete Applied Mathematics* 87, 149–158.

[48] E. Levner, V. Kats, and V.E. Levit (1997). An improved algorithm for a cyclic robotic scheduling problem. *European Journal of Operational Research* 97, 500–508.

[49] E.V. Levner (1969). Optimal planning of parts machining on a number of machines. *Automation and Remote Control* 12, 1972–1981.

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

[51] H. Matsuo, J.S. Shang, and R.S. Sullivan (1991). A crane scheduling problem in a computer-integrated manufacturing environment. *Management Science* 17, 587–606.

[52] S.T. McCormick, M. Pinedo, S. Shenker, and B. Wolf (1989). Sequencing in an assembly line with blocking to minimize cycle time. *Operational Research* 37(6), 925–935.

[53] M. Middendorf and V.G. Timkovsky (2002). On Scheduling cycle shops: Classification, complexity and approximation. *Journal of Scheduling* 5(2), 135–169.

[54] A. Munier (1996). The basic cyclic scheduling problem with linear precedence constraints. *Discrete Applied Mathematics* 64, 219–238.

[55] T. Nieberg (2002). *Tabusuche für Flow-Shop und Job-Shop Probleme mit begrenztem Zwischenspeicher*. Master's thesis, University of Osnabrück.

[56] E. Nowicki and C. Smutnicki (1996). A fast tabu search algorithm for the job shop problem. *Management Science* 42(6), 797–813.

[57] L.W. Phillips and P.S. Unger (1976). Mathematical programming solution of a hoist scheduling program. *AIIE Transactions* 8(2), 219–225.

[58] M. Pinedo (2002). *Scheduling: Theory, Algorithms and Systems*. 2nd edition. Prentice Hall.

[59] B.R. Rau and J.A. Fisher (1993). Instruction-level parallel processing: History, overview and perspective. *Journal of Supercomputing* 7, 9–50.

[60] R. Reiter (1968). Scheduling parallel computations. *Journal of the ACM* 15, 590–599.

[61] I. V. Romanovskii (1967). Optimization of stationary control of a discrete deterministic process. *Cybernetics* 3, 52–62.

[62] R. Roundy (1992). Cyclic Schedules for job-shops with identical jobs. *Mathematics of Operations Research* 17, 842–865.

[63] B. Roy and B. Sussmann (1964). Les problèmes d'Ordonnancement avec Constraints Disjonctives. Note DS no. 9 bis, SEMA, Paris.

[64] Harald Scheid (1994). *Zahlentheorie, 2nd edition*. BI Wissenschaftsverlag.

[65] J.W. Seo and T.E. Lee (2002). Steady-State Analysis and Scheduling of Cyclic Job Shops with Overtaking. *The International Journal of Flexible Manufacturing Systems* 14, 291–318.

[66] R. Sethi (1975). Complete register allocation problems. *SIAM Journal of Computing* 4(3), 226–248.

[67] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak (1992). Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems* 4, 331–358.

[68] J.S. Song and T.E. Lee (1998). Petri net modeling and scheduling for cyclic job shop with blocking. *Computers and Industrial Engineering* 34(2), 281–295.

[69] G. Steiner and Z. Xue (205). Scheduling in reentrant robotic cells: Algorithms and complexity. *Journal of Scheduling* 8, 25–48.

[70] R.E. Tarjan (1972). Depth first search and linear graph algorithms. *SIAM Journal on Computing* 1(2), 146–160.

[71] M.Y. Wang, S.P. Sethi, and S.L. van de Velde (1997). Minimizing makespan in a class of reentrant shops. *Operations Research* 45, 702–712.

[72] N.E. Young, R.E. Tarjan, and J.B. Orlin (1991). Faster parametric shortest path and minimum-balance algorithms. *Networks* 21, 205–221.