A New Procedure for the Cyclic Job Shop Problem

Martin Fink [*,**] **Touria Ben Rahhou** [**] **Laurent Houssin** [**]

[*] *TUM-School of Management, Technische Universität München, 80333 Munich, Germany*
[**] *CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France*

(e-mail: {mfink , tbenrahh , houssin}@laas.fr)

Abstract: The topic of this paper is the cyclic job shop problem which aims at minimizing the cycle time under precedence and resource constraints. Based on graph theory, we propose a new branch and bound enumeration procedure to solve this problem. We compare the new procedure with other exact methods and present numerical test results.

Keywords: Cyclic scheduling, Discrete optimization, Branch and bound algorithm, Job shop problem.

1. INTRODUCTION

In classical scheduling, a set of tasks is executed once while the determined schedule optimizes objective functions such as the makespan or earliness-tardiness. In contrast, cyclic scheduling means performing a set of generic tasks infinitely often while minimizing the time between two occurrences of the same task. Cyclic scheduling has multiple applications, such as robotics (see Kats and Levner (1997)), manufacturing systems (see Pinedo (2005) and Hillion and Proth (1989)) or multiprocessor computing (see Hanen and Munier (1995b)). It has been studied from multiple perspectives, since there exist several possible representations of the problem such as graph theory, integer linear programming, Petri nets or $(max, +)$ algebra. An overview over cyclic scheduling problems and the different approaches can be found in Hanen and Munier (1995a).

We consider the cyclic job shop problem (CJSP), which is proven to be NP-hard by Hanen (1994) and employ graph theory to represent the problem (see Brucker and Kampmeyer (2008) and Hanen and Munier (1995a)). The goal of the CJSP is to find a periodic schedule with minimal asymptotic cycle time that satisfies all uniform and disjunctive constraints. However, only two exact solution procedures exist to solve this problem, to the best of our knowledge. First, a mixed integer linear program (MILP) formulation of the CJSP described in Brucker and Kampmeyer (2008) and Hanen (1994) which can be solved with linear programming software. Second, a branch and bound procedure based on conservative height functions and amplitudes which is described in Hanen (1994).

In this paper, we propose a new branch and bound enumeration procedure that derives bounds on shift events using the consistency of a graph and evaluates different solutions using the $(max, +)$ algebra version of Howard's algorithm (see Cochet-Terrasson et al. (1998)). Subsequently, we compare our approach to two existing solution procedures to evaluate its performance.

The paper is organized as follows. In section 2, we first present the underlying general basic cyclic scheduling problem (GBCSP) and briefly highlight algorithms to solve it. Subsequently, we outline the CJSP and present existing solution procedures which we will compare to our procedure. In section 3, we introduce a new branch and bound procedure for the CJSP, which is compared with the two existing solution procedures for the CJSP in section 4. Computational results are given and discussed. In section 5, we draw conclusions and provide an outlook on future research in the field of CJSP and solution procedures.

2. PROBLEM DEFINITION AND NOTATION

In this section, we first present the general basic cyclic scheduling problem (GBCSP) and several algorithms to solve it. Subsequently, we focus on the cyclic job shop problem (CJSP), recall the problem notation and definition, and summarize characteristics of the two existing solution procedures that are to be compared with our procedure.

2.1 The General Basic Cyclic Scheduling Problem

The GBCSP is characterized by a set of generic tasks $\mathcal{T} = \{1, ..., n\}$ that is repeated infinitely often. Each task i has a corresponding processing time p_i and $< i, k >$ denotes its k^{th} occurrence. In a GBCSP, tasks are linked by uniform precedence constraints that define which of the two tasks precedes the other.

Solving a GBCSP means assigning a starting time $t(i, k)$ to each occurrence $< i, k >$. Given a periodic schedule with cycle time α we obtain

$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N}: \quad t(i, k) = t(i, 0) + \alpha k. \tag{1}$$

Knowing that the occurrence $< i, k+1 >$ cannot begin before the end of the occurrence $< i, k >$, a non-reentrance constraint can be written as

$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N}: \quad t(i,k+1) \geq t(i,k) + p_i. \tag{2}$$

A directed graph $G = (T, E)$ with a set of nodes T and a set of arcs E can be associated with a GBCSP such that a node (resp. an arc) of G corresponds to a task (resp. precedence constraints) in the GBCSP. Each arc (i, j) of G is equipped with two values: the delay or length $L_{ij} \in \mathbb{Q}$ and the height (also called event shift) $H_{ij} \in \mathbb{Z}$. Uniform precedence constraints can be expressed as follows:

$$\forall (i,j) \in E, \forall k \in \mathbb{N}: \quad t(i,k) + L_{ij} \leq t(j, k + H_{ij}). \tag{3}$$

2.2 Existing solution procedures for the GBCSP

Once the graph G is obtained, one can check if the GBCSP is consistent with the following theorem (Hanen (1994), Carlier and Chrétienne (1988), Cohen et al. (1985)).

Theorem 1. The GBCSP is feasible if and only if any circuit in the associated graph G has a positive height.

Moreover, it can be shown that the minimum cycle time of the system is given by the critical circuit of the graph. Several algorithms exist to compute the critical circuit. In the following, we briefly present three different ways to solve this problem. We can first mention the algorithm from Gondran and Minoux (1995) which runs in $O(n^3 log(n))$. Second, Karp's algorithm Karp (1978); Dasdan and Gupta (1998) can be used to find the critical circuit in $O(n^3)$. However, Karp's algorithm only works when for all existing H_{ij}, we have $H_{ij} = 1$. When $H_{ij} \geq 0$, a remodeling is necessary (see Baccelli et al. (1992)). In cyclic scheduling, also negative heights can be found frequently. (*cf.* the basic example of Hanen (1994)). A first attempt for considering negative heights in Karp's algorithm was made in Houssin (2011).

Howard's algorithm (Howard (1960)) was originally designed for Markov decision processes. In Cochet-Terrasson et al. (1998), it was adapted to the framework of $(max, +)$ algebra and can be used to compute the critical circuit of a double weighted directed graph. Although the complexity remains unproved it shows excellent performance and almost linear running time.

2.3 The Cyclic Job Shop Problem

The major difference between the CJSP and the GBCSP is that the GBCSP processes each task on its own dedicated machine whereas in the CJSP, the number of machines is smaller than the number of tasks. As a consequence, machines play an important role and need to be added to the CJSP. The CJSP can therefore be interpreted as a GBCSP equipped with resource constraints. We consider a set of machines $\mathcal{M} = \{1, ..., m\}$ with $m < n$. Task i is processed on machine $M(i) \in \mathcal{M}$ for its whole processing time p_i. We refer to an event when two occurrences of different tasks using the same machine overlap as a resource conflict, which can be resolved by adding resource or disjunctive constraints.

We denote by T_s the set of tasks to be processed on machine $s \in \mathcal{M}$. We can express disjunctive resource constraints as following:

$$\forall s \in \mathcal{M}, \forall i, j \in T_s \forall k, l \in \mathbb{N}:$$
$$t(i,k) \leq t(j,l) \Rightarrow (t(i,k) + p_i \leq t(j,l)) \; or \; (i = j \; and \; k = l). \tag{4}$$

As in the GBCSP, a directed graph $G = (T, E)$ can be associated with a CJSP such that a node (resp. an arc) of G corresponds to a task (resp. constraints) in the CJSP. Each uniform arc (i, j) of G again has two values $L_{ij} = p_i$ and H_{ij}. Each disjunctive arc also features two values: the delay $L_{ij} = p_i$ and the event shift $K_{ij} \in \mathbb{Z}$. For each machine s, if $i \in T_s$ and $j \in T_s$ then: $K_{ij} + K_{ji} = 1$ (see Hanen (1994) for further details). Two dummy nodes are introduced in the model. They represent the start of an occurrence k and the end of this occurrence. The arc between these two nodes is valuated with no processing time and with a non negative height. This height represents the maximum work in progress of the system. Further considerations of the value of the height are available in Brucker and Kampmeyer (2008).

Example The example consists of four generic tasks grouped into two jobs and two machines. Table 1 indicates details of this CJSP and Figure 1 shows the associated graph. The nodes 1 (start) and 6 (end) are dummy nodes.

Job	1		2	
Task	2	3	4	5
Processing time	5	4	2	3
Machine	1	2	1	2

Table 1. Data for the CJSP example

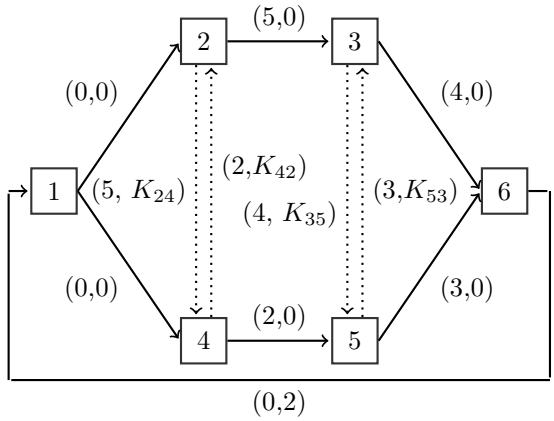

Fig. 1. Graph example

2.4 Existing Solution Procedures for the CJSP

The CJSP as described in chapter 2.3 can be solved by two existing procedures:

- The mixed integer linear program (MILP) as described in Hanen (1994) or Brucker and Kampmeyer (2008) .
- The branch and bound enumeration procedure from Hanen (1994).

In the following, we briefly sum up the major characteristics of both approaches.

Mixed Integer Linear Program The CJSP can be transformed to a MILP. The remodeling is necessary because the resource constraint features a non-linear component with $\alpha \times K_{ij}$. Thus, the throughput $\tau = \alpha^{-1}$ is introduced together with a new variable $u_i = t_i \times \alpha^{-1}$, which leads to a linear formulation that can be solved with a linear programming solver.

$$\max \tau \tag{2.5}$$
$$\text{s.t.}$$
$$u_j - u_i \geq \tau \times p_i - H_{ij} \tag{2.6}$$
$$u_j - u_i \geq \tau \times p_i - K_{ij} \tag{2.7}$$
$$K_{ij} + K_{ji} = 1 \tag{2.8}$$

Branch and Bound Procedure from Hanen In the considered paper from Hanen (1994), the author presents a branch and bound procedure that - unlike the general MILP formulation - is tailored to the graph representation of the CJSP. It is based on two concepts: conservative height functions and amplitudes.

The author proves that a feasible conservative height function is associated with a set of feasible schedules, out of which the best solution can be determined by solving an instance of the GBCSP with the algorithm from Gondran and Minoux (1995) in $O(n^3 log(n))$. Amplitudes are employed to derive upper bounds on the height function.

An adjustment procedure is used to evaluate nodes. It iteratively increases amplitude lower bounds and decreases height function upper bounds until either no changes are made or a circuit with positive amplitude is detected and the solution hence is unfeasible. Since Hanen's procedure requires an upper bound on the cycle time to start with, a heuristic is developed for this purpose in Hanen (1994).

3. A NEW BRANCH AND BOUND PROCEDURE FOR THE CJSP

We propose a branch and bound procedure for the CJSP that relies on two major pillars: the consistency of a graph to derive bounds on the shift events and Howard's algorithm to calculate the cycle time and to evaluate a node. The following paragraph first establishes the underlying theoretic concepts before we describe the structure of the algorithm in the second subsection and demonstrate the algorithm through the use of the example introduced in chapter 2.3 in the third subsection.

3.1 Theoretic Concepts

A graph G is said to be consistent if and only if every circuit C in G has a height equal or bigger than 1. The height of a circuit is defined as the sum of the heights of all arcs on the circuit. This implies that we sum up heights in case of uniform arcs and event shifts in case of disjunctive arcs. A new disjunctive arc that is appended to the graph creates at least one circuit and may not violate the consistency of the graph if the solution is to remain feasible. Therefore, each disjunctive arc (i, j) has a clear lower bound on its shift event K_{ij} which may not be undercut to guarantee the consistency of the graph.

We can easily show this concept in our example. Assuming that the tasks 2 and 4 are processed on the same machine, two disjunctive arcs - one from 2 to 4 and the other from 4 to 2 - are required to determine a precedence between the two nodes. Let us focus on the disjunctive arc from 2 to 4. This arc induces a new cycle $C = (2, 4, 5, 6, 1, 2)$. Concerning the height of the cycle, we know that $H(C) = K_{24} + 0 + 0 + 2 + 0 \geq 1$ needs to hold for the solution to be feasible. This can be rewritten to $K_{24} \geq -1$. It leads to the following theorem.

Theorem 2. Consider a CJSP where tasks i and j are performed onto the same machine. We can derive a lower bound denoted K_{ij}^- for the event shift K_{ij} such that

$$K_{ij}^- = 1 - \min\{H(\mu) | \mu \text{ path from } j \text{ to } i \text{ in } G\}. \tag{3.1}$$

Proof. A feasible solution requires a height at least equal to 1 for each circuit in the graph. Therefore, $H(C) \geq 1$ for all circuits C in G. When a shift event for arc (i, j) is on the circuit, we obtain $K_{ij} + \{H(\mu) | \mu \text{ path from } j \text{ to } i \text{ in } G\} \geq 1$, which can be rearranged to $K_{ij} \geq 1 - \{H(\mu) | \mu \text{ path from } j \text{ to } i \text{ in } G\}$. Since this inequality has to hold for all paths from j to i in G, we must consider the minimum path.

Corollary 3. Since we know that $K_{ij} + K_{ji} = 1$, we can derive an interval for each variable K_{ij} as follows:

$$K_{ij}^- \leq K_{ij} \leq 1 - K_{ji}^- \tag{3.2}$$

To allow for an efficient way to compute bounds on all K_{ij}, we construct a height matrix B of size $n \times n$. The elements of the matrix B_{ij} are initialized with the heights H_{ij} of all uniform arcs and the shift events K_{ij} of all already determined disjunctive arcs. Subsequently, we employ the algorithm from Floyd and Warshall as for example described in Cormen et al. (2009) which runs in $O(n^3)$ to find all pairs' shortest paths in B. Using the height matrix, the lower bound on a shift event K_{ij} equals $K_{ij}^- = 1 - B_{ji}$.

Since updating the height matrix involves the algorithm of Floyd and Warshall and hence might slow the procedure down, we suggest two versions of the algorithm. Version one updates the bounds on the shift events in every node on the search tree and hence does not examine feasibility of the solution. Version two computes bounds only once when the algorithm is initialized but checks the feasibility of each solution with Howard's algorithm when evaluating the node.

Furthermore, we can derive bounds on the cycle time from the general problem (see section 2.1).

Theorem 4. Consider a CJSP, we have the following lower bound denoted α^- for the cycle time.

$$\alpha^- = \max\{\lambda_u, \lambda_s\}. \tag{3.3}$$

in which λ_u is the solution of the GBCSP induced by the uniform constraints and λ_s is defined as follows

$$\lambda_s = \max_{m \in \mathcal{M}} \{\sum p_i | M(i) = m\}.$$

Proof. It is obvious to remark that the relaxed version of the CJSP (*i.e.* without resource constraints) leads to the GBCSP and it is not possible to obtain a lower cycle time than λ_u. The second bound is the sum of all processing times of tasks belonging to a same machine and we obviously have

$$t_j(k+1) \geq \sum_{M(i)=M(j)} p_i + t_j(k).$$

Since all tasks have the same cycle time, even if they do not belong to the same machine, we deduce that the cycle time of the CJSP cannot be smaller than the maximal sum of processing times per machine.

3.2 Structure of the Branch and Bound Procedure

We now outline our branch and bound procedure version one and subsequently show the differences of version two. The algorithm can be divided into two parts, namely the head and the body. Whereas the head initializes the procedure, the body houses the main loop.

The head consists of the method *initialize()*, which determines the upper bound on the cycle time by summing up the processing time of all tasks. The first node in the search tree is created and Howard's algorithm is employed to determine its cycle time and check the feasibility of the initial solution. This means that the graph consisting only of uniform arcs is tested on its consistency and the cycle time for the GBCSP is computed. The lower bound on the cycle time is derived from theorem 4. If the initial node is feasible - and the graph is consistent - we push this node on the node stack and determine the shift event lower bounds in two steps. First, we set all B_{ij} to H_{ij} if an arc (i, j) exists and assign ∞ to all other elements in B. Second, we employ the algorithm from Floyd and Warshall to find the shortest paths in B.

In the body, the algorithm repeats a set of operations until the stack is empty or until the lower bound equals the upper bound on the cycle time. The method *checkSum()* determines both shift events $K_{ij} = K_{ij}^-$ and $K_{ji} = K_{ji}^-$ as in Hanen's procedure if the sum of their lower bounds equals one. If all shift events K_{ij} have been determined, a complete selection prevails. In this case, the upper bound on the cycle time is updated with the current α if it is lower than α^+. If the cycle time is higher than the upper bound on the cycle time, we can discard this node. In case of no complete selection, a new K_{ij} is selected according to the *branchingRule()*. This method selects an undetermined K_{ij} for which the sum of the lower bounds on K_{ij} and K_{ji} is maximal, which induces the minimal number of child nodes and hence leads to the smallest possible search tree. We then call the method *branch()* which creates a new child node for each integer in the interval $[K_{ij}^-, 1 - K_{ji}^-]$ for the selected K_{ij}, assigns an integer in this interval to the respective child node, and puts the nodes into the node container N. The subsequent subroutine *evaluateNodes()* updates the lower bounds on the shift events by inserting the respective shift event K_{ij} into the height matrix B at B_{ij} and by calling the Floyd and Warshall algorithm to update the matrix once again. The second step of the method *evaluateNodes()* is to employ Howard's algorithm to update the cycle time. Finally, we call the procedure *nodeSelection()* which pushes all child nodes on the stack in an order defined by the node selection rule. The node selection rule dictates, that the node with the lowest cycle time is selected first. As a tie breaker in case of identical cycle times, the minimum index of a solution can be used. The outlined procedure can also be seen in figure 2.

Input: $G = (T, E)$, assignment of tasks to machines
Output: α
`//Head`
$S_0 \leftarrow$ *Initialize()*
`//Body`
while *nodestack* $\neq$ *EMPTY and* $\alpha^- \neq \alpha^+$ **do**
  $S \leftarrow$ upper most node from nodestack
  checkSum(S)
  if $S(\alpha) < \alpha^+$ **then**
    if *S is a complete selection* **then**
      $\alpha^+ \leftarrow S(\alpha)$
    else
      S(selected K_{ij}) $\leftarrow$ *branchingRule(S)*
      $N \leftarrow$ *branch(S)*
      evaluateNodes(N)
      nodestack $\leftarrow$*nodeSelection(N)*
return α^+

Fig. 2. The new procedure

Version two of the algorithm implements two changes in the method *evaluateNodes()*. First, the height matrix B is not updated for each child node but only initialized as in version one. This leads to the second adjustment: the feasibility of each child node needs to be checked, which is achieved with the procedure of Howard.

3.3 Example

In this subsection, we provide a concrete picture of our procedure version one by solving the example from chapter 2.3.

The method *initialize()* computes the sum of all processing times to 14 and saves this value as α^+. It then creates the first node S_0 which represents the uniform graph, calculates the solution of the GBCSP to 5 and also confirms that the uniform graph is consistent. It then determines the sum of processing times of tasks performed on machines 1 and 2 to $5+2$ and $4+3$, respectively. Given the sum of processing times per machine and the solution of the GBCSP, the lower bound on the cycle time can be computed to $\alpha^- = \max\{5; 7\} = 7$. Before the initial node is pushed on the node stack, the lower bounds on the shift events K_{24}, K_{42}, K_{35} and K_{53} and calculated. For this, we employ the height matrix B which can be seen in table 2.

Table 2. The initial height matrix B

1	0	0	0	0	0
2	1	0	2	2	0
2	2	1	2	2	0
2	2	2	1	0	0
2	2	2	2	1	0
2	2	2	2	2	1

For K_{24} we know that $K_{24}^- = 1 - B_{42} = 1 - 2 = -1$. Similarly, we obtain $K_{42}^- = -1$, $K_{35}^- = -1$, and $K_{53}^- = 1$.

In the body, we select S_0 and delete it from the node stack. The method *checkSum()* does not affect the solution since neither $K_{24}^- + K_{42}^- = -2$ nor $K_{35}^- + K_{53}^- = 0$ equals 1. The cycle time of S_0 is 7 and below α^+ with 14. Hence, we further fathom this node. Since no complete selection prevails, a shift event is determined. The method *branchingRule()* selects the shift event K_{35} over K_{24} because the interval for the shift event K_{24} is $[-1;0]$ compared to $[-1;2]$. For each integer in this interval, the method *branch()* creates one child node which inherits the solution of its parent plus the new pair of disjunctive arcs. In our example, we create two child nodes $S1$ and $S2$ with the values $K_{35} = -1$ and $K_{35} = 0$, respectively and the corresponding values for $K_{53} = 2$ and $K_{53} = 1$, respectively. For both nodes, the method *evaluateNodes()* calculates the cycle time with Howard to 7 and 9 and updates the lower bounds on K_{24} and K_{42}. For $S2$, the lower bound on K_{24} increases from -1 to 0 since B_{42} now equals 1. The first iteration of the loop finally calls the procedure *nodeSelection()* to select the child node with the lowest cycle time - in our example $S1$.

The second iteration continues with $S1$. Again, *checkSum()* does not alter the solution and since $7 \leq 14$ and no complete selection prevails, we select the last remaining shift event K_{24} to branch on. Afterwards, the method *branch()* creates four child nodes $S3, S4, S5$, and $S6$ with the values $-1, 0, 1$, and 1 for K_{24} and *evaluateNodes()* computes their cycle times $11, 7, 7$, and 10. *nodeSelection()* selects $S4$. In the third iteration, we obtain a complete selection with a cycle time below α^+ and hence update α^+ to 7. Afterwards, the procedure stops because $\alpha^- = \alpha^+$ and the optimal cycle time 7 is found. Figure 3 shows the search tree for the branch and bound procedure.

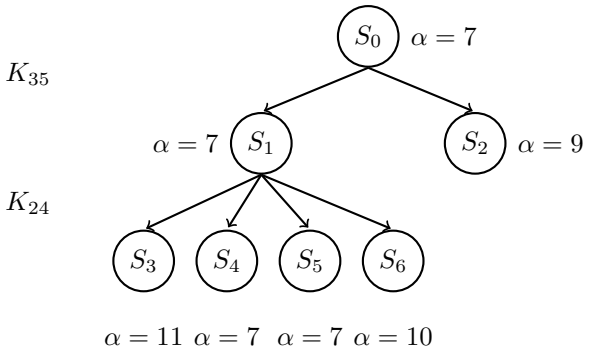

Fig. 3. Example search tree

4. COMPARISON OF SOLUTION PROCEDURES

To evaluate performance, we have tested the two versions of our procedure using randomly generated cyclic job shop problems and compare the results with the performance of the procedure from Hanen and the MILP from Hanen and Brucker in the following subsections.

4.1 Comparison of Different Branch and Bound Concepts

We have compared four implementations of branch and bound procedures to solve the CJSP: our procedure version one, our procedure version two, Hanen's procedure with the algorithm from Howard and the MILP formulation which is solved with CPLEX. We briefly contrast the four former approaches in table 3.

Table 3. Overview over implementations to solve the CJSP without CPLEX

Procedure	Our's		Hanen's
Version	Version 1	Version 2	
Algorithm for GBCSP	Howard	Howard	Howard
Concept of bounds	Consistent graph, updated for each node	Consistent graph, checked with Howard	Amplitude
Discarding of nodes	$\alpha \geq \alpha^+$	$\alpha \geq \alpha^+$, node unfeasible	Positive circuit amplitude
End of procedure	Node stack empty, $\alpha^- = \alpha^+$	Node stack empty, $\alpha^- = \alpha^+$	Node stack empty
Node selection	Lowest α	Lowest α	Based on circuit amplitude
Branching rule	Smallest interval	Smallest interval	Based on circuit amplitude

In addition to the observations in table 3, our algorithm performs a depth-first search and computes the cycle time for every node. Hanen's approach also pursues a depth-first approach but calculates the critical circuit of the GBCSP only when a complete selection is reached.

4.2 Technical Specifications

The testing of the four implementations has been conducted on a computer with a AMD Athlon 64 x2 3800+ processor, 1.7 GB RAM and a Linux Fedora 9 operating system. The CJSP test instances were randomly generated. They are characterized by the number of jobs, number of tasks and number of machines and follow the layout of the example: each job is connected to one dummy start node and one dummy end node. Moreover, the two dummy nodes are connected by a uniform arc of processing time 0 and height 2. We tested six different types of CJSP instances, each consisting of ten problems, which can be seen in table 4.

Table 4. Overview over problem types

Type	1	2	3	4	5	6
Number of jobs	2	2	5	5	10	8
Number of tasks	10	20	10	20	20	30
Number of machines	2	5	2	5	2	4

4.3 Numerical Test Results

In this subsection we present results of the numerical test of all four implementations. For each of the six problem types, ten problem instances have been solved. We have limited the running time of each implementation to six seconds and then noted the number of problems that could be solved within this time interval. The results are shown in table 5.

Table 5. Numerical test results

	1	2	3	4	5	6
Our procedure version 1	10	10	10	10	9	9
Our procedure version 2	10	10	10	10	3	5
MILP	10	10	9	8	0	0
Hanen's procedure	10	10	9	5	0	0

4.4 Interpretation of numerical results

The results can be interpreted in two ways. First, all implementations are compared with each other, and we can distinct between small problems (type 1 and 2), medium problems (type 3 and 4) and larger problems (type 5 and 6). Second, we discuss the different performance of the two versions of our procedure.

For small problems (type 1 and 2), the four implementations perform similarly well and solve all 10 problem instances within six seconds. Hence, no significant differences can be observed. The picture changes for medium problems, where our procedure still solves all problem instances, the MILP and CPLEX perform slightly worse and Hanen's procedure even only solves half the number of all problems of type 4. This trend continues in case of larger problems, where our procedure still shows superior performance. However, both the MILP and Hanen's procedure underperform significantly as they cannot solve a single problem in under six seconds.

The two versions of our procedure behave equally well for small and medium problems. In case of larger problems, the two versions differ considerably. Obviously, updating the height matrix is superior to letting Howard's algorithm examine the feasibility of a node. Furthermore, less nodes need to be examined in case of version one, as the shift event lower bounds can only increase during the course of the procedure and hence the interval and the number of child nodes decreases.

5. CONCLUSION

In this paper, we have proposed a new procedure to solve the CJSP. Its two major characteristics are the node evaluation with Howard's algorithm and bounds based on the consistency of a generic graph. We have implemented two versions of the new procedure that pursue different bounding approaches and have compared them with two other implementations of solution procedures for the CJSP: First, the branch and bound algorithm from Hanen with Howard's algorithm and second an MILP formulation solved with CPLEX. The numerical tests demonstrated that our procedure performs as good as CPLEX and as Hanen's procedure for small problem instances. For bigger instances, it clearly outperforms Hanen's procedure and partly also CPLEX.

Although the results are promising, the new procedure still has potential for improvement and should be subject to future research to achieve even better results. One area is the node evaluation function of the new procedure. Currently, it uses Howard's algorithm to determine the lower bound of a subproblem but neglects disjunctive arcs that are still to be appended to the graph. In addition to improving the procedure itself, further computational experiments could be conducted. Our procedure stands to be tested for larger problem instances, which might trigger a large set of disjunctive arcs. To be able to cope with many disjunctive arcs, either a depth-first approach or a preselection of disjunctive arcs via a heuristic might be a solution. Also, a benchmarking of the new procedure with the original version of Hanen's procedure incorporating the algorithm from Gondran and Minoux should be conducted to obtain additional performance comparisons.

REFERENCES

Baccelli, F., Cohen, G., Olsder, G.J., and Quadrat, J.P. (1992). *Synchronization and Linearity*. Wiley.

Brucker, P. and Kampmeyer, T. (2008). A general model for cyclic machine scheduling problems. *Discrete Applied Mathematics*, 156(13), 2561 – 2572.

Carlier, J. and Chrétienne, P. (1988). *Problèmes d'Ordonnancement: modélisation, complexité, algorithmes*. Masson, Paris.

Cochet-Terrasson, J., Cohen, G., Gaubert, S., Gettrick, M.M., and Quadrat, J.P. (1998). Numerical computation of spectral elements in max-plus algebra. In *Proceedings of the IFAC Conference on System Structure and Control*. Nantes.

Cohen, G., Dubois, D., Quadrat, J.P., and Viot, M. (1985). A linear system theoretic view of discrete event processes and its use for performance evaluation in manufacturing. *IEEE Trans. on Automatic Control*, AC–30, 210–220.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. (2009). *Introduction to Algorithms*. The MIT Press.

Dasdan, A. and Gupta, R.K. (1998). Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17(10).

Gondran, M. and Minoux, M. (1995). *Graphes et algorithmes, 3e édition revue et augmentée*. Eyrolles.

Hanen, C. (1994). Study of a np-hard cyclic scheduling problem: The recurrent job-shop. *European Journal of Operational Research*, 72(1), 82 – 101.

Hanen, C. and Munier, A. (1995a). *Scheduling Theory and Its Applications*, chapter Cyclic Scheduling on Parallel Processors: An Overview. Wiley.

Hanen, C. and Munier, A. (1995b). A study of the cyclic scheduling problem on parallel processors. *Discrete Applied Mathematics*, 57, 167–192.

Hillion, H.P. and Proth, J.M. (1989). Performance evaluation of job-shop systems using timed event graphs. *IEEE Transaction on Automatic Control*, 34(1), 3–9.

Houssin, L. (2011). Cyclic jobshop problem and (max,plus) algebra. In *Proceedings of IFAC WC*. Milan, Italy.

Howard, R. (1960). *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology.

Karp, R.M. (1978). A characterization of the minimum cycle mean in a digraph. *Discrete Math*, 23.

Kats, V. and Levner, E. (1997). A strongly polynomial algorithm for no-wait cyclic robotic flowshop scheduling. *Operations Research Letters*, 21, 171–179.

Pinedo, M. (2005). *Planning and Scheduling in Manufacturing and Services*. Springer.