Ph. D. Paweł Sitek
Technical University of Kielce
Ph. D. Julian Jakubowski
University of Zielona Góra

APPLICATION OF CONSTRAINT LOGIC PROGRAMMING TO OPTIMIZATION OF MAKESPAN IN THE FLOW PRODUCTION LINE

A mathematical model of makespan optimization in the divided flow production line is presented. The methodology of Constraint Logic Programming to optimization of makespan in the above environment has been considered. Multiple examples illustrate the concept proposed.

ZASTOSOWANIE PROGRAMOWANIA W LOGICE OGRANICZEŃ DO OPTYMALIZACJI CYKLU LINII PRODUKCYJNEJ

A mathematical model of makespan optimization in the divided flow production line is presented. The methodology of Constraint Logic Programming to optimization of makespan in the above environment has been considered. Multiple examples illustrate the concept proposed.

1. INTRODUCTION

Shop scheduling has been researched in many varieties. The basic shop scheduling model consists of **machines** and **jobs** each of which consists of a set of **operations**. Each operation has an associated machine on which it has to be processed for a given length of time. The processing times of operations of a job cannot overlap. Each machine can process at most one operation at the given time. In the basic models are **m** machines and **n** jobs. The processing time of an operation of job **j** on machine **i** is denoted by $\mathbf{p_{ij}}$ and $\mathbf{p_{max}} = \mathbf{max}\ \mathbf{p_{ij}}$. The three well-studied models are the open shop, flow shop and job shop problems. In an open shop problem, the operations of a job can be performed in any order. In a job shop problem the operations must be processed in a specific, job-depended order. A flow shop is a special case of a job shop in which each job has exactly m operations- one per machine. And also the order in which they must be processed is the same for all the jobs. The problem is to minimize makespan. Makespan is the overall length, of the schedule with the above constraints [1]. All above mentioned problems are strongly NP-hard. For the flow shop problem, the case where there are more than two machines is strongly NP-hard., although the two machines version is polynomial solvable [2].

In this paper we present the problem which belongs to the flow shop production environment. There is a flow production line which can be dynamically divided into sections designated to concurrent processing different products. The production flow through machines belonging to the section is synchronized. The set of products which can be manufactured in a given line depends on tools the line is equipped with. Each feasible set of products processed concurrently in the line has been named the production variant. This is a common manufacturing environment for repetitive production in a small or medium sized manufacturing company. Moreover, we present two computational philosophies in illustrative examples for the above problem.

2. DESCRIPTION OF THE PROBLEM OF OPTIMIZATION OF MAKESPAN IN THE FLOW PRODUCTION LINE

In this paper we will consider a production system (see fig.1) which consists of a flow production line, which is composed of N identical workstations (machines). The line can be dynamically divided into sections which execute operations on their products $j \in J$ and each machine (section) is equipped with tools in the same time (setup time).

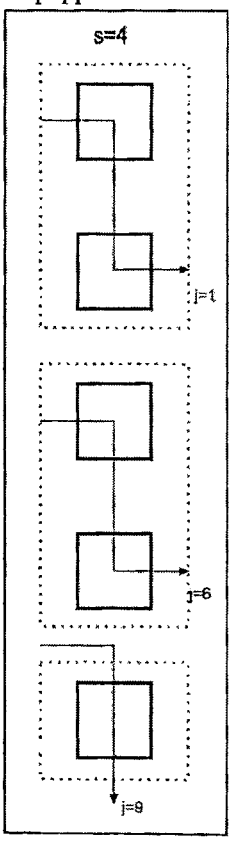

Fig. 1. Flow production line in variant s=4. The line is divided into three sections for products j=1, j=6, j=9 (see example 1).

The problem for this particular case is how to allocate products to production variants $s \in S$ of the line to minimize makespan and cover requirements for the system products Zj.

A Better allowance of products to variants assures:

- Better utilization of machines, shortening of time of production orders execution
- Smaller numbers of production variants which involves smaller frequency of set-ups.

3.MATHEMATICAL MODEL OF OPTIMIZATION OF MAKESPAN IN THE FLOW PRODUCTION LINE

In the system of a flow production line (fig.1) products $j \in J$ are produced, where J is a set of all products. Total production from all production variants of the line should cover requirements for the system products. This dependence can be expressed as equations (1),

$$\sum_{s=1}^{S} z_{sj} = Z_j \qquad \text{for } j \in J, \qquad (1)$$

where:

Z_j –quantity of order of the product j,

z_{sj} – planned quantity of the product j from the variant s of line, $j \in J$, $s = 1..S$.

Processing the product demands definite number of operations. Every operation is executed on the single machine. It is obvious that the total number of simultaneously executed operations may not be greater than number of machines in the production line. Thus,

$$\sum_{j \in J} x_{sj} K_j \le N, \qquad \text{for } s = 1..S \qquad (2)$$

where

$$x_{sj} = \begin{cases} 1, \text{ if product } j \text{ is produced in variant } s \text{ of line,} \\ 0, \text{ otherwise,} \end{cases}$$

for $j \in J$, $s = 1..S$

K_j – number of operations of the product $j \in J$, as well as number of machines in the section producing the product,

N – number of machines in the line production line.

The run time t_s for the variant s of the production line may not be smaller than the run time of any product processed in this variant (fig.2).This condition can be expressed as inequality:

$$p_j z_{sj} \le t_s, \qquad \text{for } j \in J, s = 1..S, \tag{3}$$

where

p_j– production pace of the product j. It is equal to the longest operation time of the product $j \in J$.

$$y_s = \begin{cases} 1, \text{ if the variant h of the production line really exists,} \\ 0, \text{ otherwise,} \end{cases}$$

$$\text{for } s = 1..S$$

The binary decision variables y_s are introduced to the model because the number of variants for a given line is unknown and S is its assumed upper bound.

If the product j is not allocated to the variant s of the production line, that is to say when $x_{sj}=0$, then the proper quantity z_{sj} should be equal to 0. Similarly, if variant s does not exist for the production line, that is to say when $y_s=0$, then its run time t_s equals 0.

These rules are equivalent to conditions (5) and (6),

$$p_j z_{sj} \le x_{sj} T_d, \qquad \text{for } j \in J, s = 1..S \tag{5}$$

$$t_s \le y_s T_d, \qquad \text{for } s = 1..S \tag{6}$$

$$T_d \ge \sum_{j=1}^{J} (p_j Z_j + \tau) \tag{7}$$

Obviously, binary decision variables must satisfy the following constraints (8)..(10)

$$x_{sj} \in \{0,1\}, \qquad \text{for } j \in J, s = 1..S \tag{8}$$

$$y_s \in \{0,1\} \qquad \text{for } s = 1..S \tag{9}$$

$$y_s \ge x_{sj}, \qquad \text{for } j \in J, s = 1..S \tag{10}$$

$$y_s \le \sum_{j \in J} x_{sj}, \qquad \text{for } s = 1..S, \tag{11}$$

$$f: C_{max} = \sum_{s=1}^{S} (t_s + y_s \tau), \tag{12}$$

where

τ – setup time, which is assumed, for simplicity, to be identical for all variants

The goal function of this problem is the makespan (12). The problem of optimization makespan in flow production line which can be divided into sections is to minimize f under constraints from (1) to (11). Thus, this is an integer programming problem.

4. THE ILLUSTRATIVE EXAMPLES

The mathematical model (1) .. (12) is the integer programming model (classical OR model) than can be implemented in any computational environment without loss of cohesion.

In this section, two illustrative examples are presented. The exemplified flow production line consists of five identical workstations (machines) (fig. 1). The main difference between examples is the size (tab.1).

Table 1 The size of the example

Example	Number of		
	Products j	Constraints	Variables (Integers)
example_1	10	126	127 (55)
example_2	26	467	468 (216)

In *example_1* the line should produce 10 products in the maximum 10 production variants (this is upper bound of the number of production variants which is equal to the number of products). The quantity of orders of the products in *example_1* is the following $Z_j=\{180, 18, 14, 14, 20, 20, 18, 18, 18, 14, 20\}$. The other data for this example p_j, K_j are in the table 2. The setup time is identical for all variants $\tau=10$. The result of optimization is the makespan $f=480$. For this optimal solution the proper number of variants, run times t_s, planned quantities z_{sj} are shown in table 2.

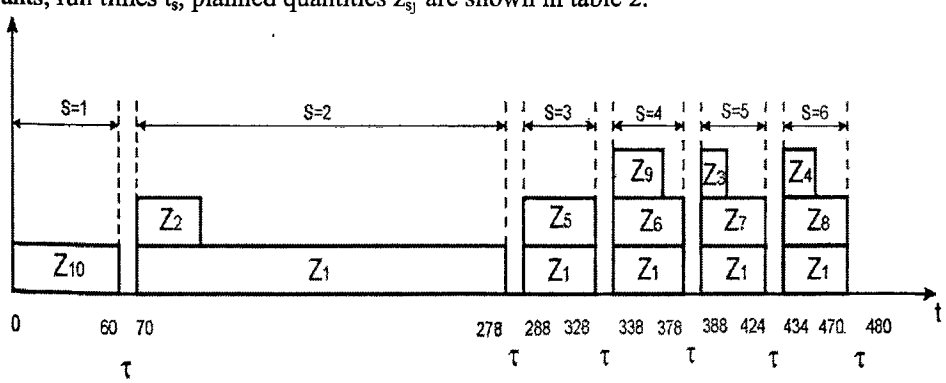

Fig. 2 Gantt's chart of the optimal schedule for example_1

The Gantt's chart of the schedule for the optimal makespan C_{max}=480 in *example_1* is shown in fig. 2. There are six production variants. In each variant from one to three products are produced (tab.4 and fig.2).

Table 2 Data and results for example_1

s					t_s
1	j	10			60
	p_j	3			
	z_{sj}	20			
	K_j	4			
	$p_j * z_{sj}$	60			
2	j	1	2		208
	p_j	2	2		
	z_{sj}	104	18		
	K_j	2	2		
	$p_j * z_{sj}$	208	36		
3	j	1	5		40
	p_j	2	2		
	z_{sj}	20	20		
	K_j	2	2		
	$p_j * z_{sj}$	40	40		
4	j	1	6	9	40
	p_j	2	2	1	
	z_{sj}	20	20	14	
	K_j	2	2	1	
	$p_j * z_{sj}$	40	40	14	
5	j	1	3	7	36
	p_j	2	1	2	
	z_{sj}	18	14	18	
	K_j	2	1	2	
	$p_j * z_{sj}$	36	14	36	
6	j	1	4	8	36
	p_j	2	1	2	
	z_{sj}	18	14	18	
	K_j	2	1	2	
	$p_j * z_{sj}$	36	14	36	

In *example_2* the production line should produce 26 products in the maximum 26 production variants (this is upper bound of the number of production variants which is equal to the number of products). The quantity of orders of the products is the following Z_j ={2,1,3,3,10,10,9,9,7,7,7,7,7,10,10,2,9,9,7,7,2,7,7,3,3,3}.The quantity of order in this example is expressed in the number of batches. The batch size carries out one hundred. The other data i.e. p_j, K_j are in the table 2. The setup time is identical for all variants τ=1. The result of optimization is the makespan f=126. Number of variants, run times t_s, planned quantities z_{sj} for the optimal solution are shown in table 4.

Table 4 Data and results for example_2

s						t_s
1	j	26	25	24	21	3
	p_j	1	1	1	1	
	z_{sj}	3	3	3	2	
	K_j	1	1	1	1	
	p_j* z_{sj}	3	3	3	2	
2	j	17	20	22	23	18
	p_j	2	1	1	1	
	z_{sj}	9	7	7	7	
	K_j	2	1	1	1	
	p_j* z_{sj}	18	7	7	7	
3	j	14	15	19		20
	p_j	2	2	1		
	z_{sj}	10	10	7		
	K_j	2	2	1		
	p_j* z_{sj}	20	20	7		
4	j	13	18			28
	p_j	4	1			
	z_{sj}	7	9			
	K_j	4	1			
	p_j* z_{sj}	28	9			
5	j	10	11	12	16	7
	p_j	1	1	1	2	
	z_{sj}	7	7	7	2	
	K_j	1	1	1	2	
	p_j* z_{sj}	7	7	7	4	

6	j	7	8	9		18
	p_j	2	2	1		
	z_{sj}	9	9	7		
	K_j	2	2	1		
	$p_j * z_{sj}$	18	18	7		
7	j	4	5	6		20
	p_j	1	2	2		
	z_{sj}	3	10	10		
	K_j	1	2	2		
	$p_j * z_{sj}$	3	20	20		
8	j	1	2	3		4
	p_j	2	2	1		
	z_{sj}	2	1	3		
	K_j	2	2	1		
	$p_j * z_{sj}$	4	2	3		

In both optimum solutions we can notice better utilization of machines and a smaller number of productive variants (tab. 3).

Table 3 Number of variants and the proper utilization of machines

Example	Number of variants (upper bound)	Number of variants (after optimization)	Utilization of machines	
example_1	10	6	s=1	4
			s=2	4
			s=3	4
			s=4	5
			s=5	5
			s=6	5
example_2	26	8	s=1	4
			s=1	5
			s=1	5
			s=1	5
			s=1	5
			s=1	5
			s=1	5
			s=1	5

The considered examples of optimization of makespan in divided flow production line are problems of integer programming. Therefore the Branch-and-Bound method was applied at first to solve them. Any Branch-and-Bound (B&B) algorithm consists of two basic procedures: branching or partitioning the feasible solution into some number of subsets and bounding or estimating the optimal value of the objective (goal) function on these subsets. The B&B algorithm is implemented in many commercial and freeware computational systems. One of them is the LINGO system which consists of the modelling language and optimizer. LINGO is a simple tool for performing complex and powerful tasks [3].

For optimization of *example_1* LINGO system was used. The result of optimization $f:C_{max}$=480 was obtained. Unfortunately, It was not possible to use LINGO system for *example_2*. The calculation time was too long. Calculations were interrupted after 12 hours. So it was necessary to examine an alternative method of optimization. In view of this that considered problem possesses constraints and we have some experience with this kind of problems [4] therefore the alternative optimization method, namely the CLP (constraint logic programming) was applied.

CLP may be defined as a body of techniques used for solving problems with constraints. Constraint Logic Programming (CLP) as a synthesis of programming in Logic (Prolog), constraint solving methods and optimization methods. The essence of CLP: modelling combinatorial, continuous and mixed decision problems with the help of constraints and logical relations, solving combinatorial, continuous and mixed decision problems by analysing and propagating the constraints [5].

The main idea of CLP is:

- Problems to be solved are modelled using elementary logic, in a way that turns the model into a part of the problem-solving program.
- Exploring constraints, which should be satisfied by the solutions, generates solutions.

Using Constraint Logic Programming (CLP) for solving the optimization problem (1)..(12) its constraints (1) .. (11) and the goal function (12) may be directly introduced to the problem declaration which is equivalent to the source code of the program.

For optimization presented problem we used the CLP language - CHIP (Constraint Handling in Prolog).

Using CHIP language we obtained the result of optimization for *example_2* (f=126).

Time of calculations in both approaches is shown in the table 5. It resulted from the version of both tools. CHIP language had to be started on the older computer (PII, 300 MHz, RAM 64 MB) under operating system DOS whereas the system LINGO was started on the computer (PIV, 1,4 GHz, RAM 512 MB) under Windows XP.

Time of calculation was about one hour (see tab.3).

Table 5 Time of calculation

Example	*f*: C_{max}	LINGO	CHIP
Example 1	480	600s	900s
Example 2	126	> 40 000s	3900s

In the above computational experiments we only compared capability and efficiency of two different computational environments: LINGO – integer programming and CLP – constraint logic programming for the particular example. CLP framework could be implemented not only as an optimization method but as a modeling method.

5. CONCLUSIONS

The methodologies of propagation of constraints with Branch-and-Bound in CLP and only Branch-and-Bound in Integer programming (LINGO) for optimization of makespan in the divided flow production line are considered. Its objective is to provide a computer-implemented model of the above presented problem. Additionally, the article introduces comparison of two computational environments: CLP and Integer Programming in commercial solver (LINGO). There are two different computational philosophies. In the commercial solver one should transform a mathematical model to a suitable form using the language of modelling. Then solver uses the implemented algorithms and methods try to solve it. During the transformation the some aspects of the problem could be lost. The idea underlying in CLP is that constraints can be used to represent the problem, to solve it. In the presented problem the CLP approach is more effective for the greater size examples.

An effective CLP framework was implemented in a simple flow shop production line environment for rather small companies. The extension to the whole flow shop and open shop problems is a subject of our currently conducted research. Constraint Logic Programming framework will be implemented not only as an optimization method but as a modelling method and the method for searching feasible solutions.

REFERENCES

1. Janiak A.: *Wybrane problemy i algorytmy szeregowania zadań i rozdziału zasobów.* Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1999.
2. Johnson S.M.: *Optima two- and tree stage production schedule with setup times included.* Naval Research Logistic Quarterly, 1:61-68, 1954.
3. *LINGO-the modeling language and optimizer.* LINDO SYSTEMS INC. Chicago, 1995.
4. Sitek P. *Optymalizacja uzbrojenia maszyn w systemie nadążnego sterowania produkcją,* Rozprawa doktorska, Gliwice, 2000.
5. Niederliński A. : *Constraint Logic Programming – From Prolog to CHIP.* Proceedings of the Workshop on Constraint Programming for Decision and Control, Gliwice, 1999, pp.27-34.