constraint programming, decision support, scheduling

Paweł SITEK[*], Jarosław WIKAREK[**], Zbigniew Banaszak[***]

CONSTRAINT-BASED SCHEDULING IN OZ/MOZART

Abstract

In this paper we present how scheduling problems can be solved in the concurrent constraint programming (CCP) language Oz. Oz is the first high-level constraint language, which offers programmable search, modularity, expressiveness and an interface to invent new constraints using C++. As an example of scheduling problem the job-shop scheduling problem with manpower resources representation was considered. The objective discussed here is the minimization of the makespan. Allocating manpower to the jobs is more complicated than allocating machines, as different jobs and operations require different manpower. This problem is strongly NP-hard. We proposed the constraint programming (CP) based approach to decision support in this environment. The most important features of CP are declarative problem modeling, allowing a clear separation between the problem statement (variables and constraints) and the resolution of the problem.

1. INTRODUCTION

Scheduling is the process of designing a procedure for a particular objective, specifying the sequence or time for each item in this procedure. Typical scheduling problems are railway time-tabling, the scheduling of project, production (job-shop and flow-shop) and medical man shifts in a hospital. Scheduling application examples in computer systems are in FMS (Flexible Manufacturing Systems), robot activity scheduling, scheduling in networks and hard real-time scheduling. Furthermore, there is a number of related problems, e.g., resource allocation in a job-shop scheduling.

The scheduling problems with the constrained resources availability is a very interesting research area for SME (Small and Medium-Sized Enterprises). They commonly use external resources, which can have decisive influence regardless the overall workability of the schedule. One such limiting resource is manpower. Manpower resources often vary with time due to holidays and sickness, more thorough representation is therefore required in the job-shop

[*] dr inż. Jarosław Wikarek, E-mail: j.wikarek@tu.kielce.pl
[**] dr inż. Paweł Sitek, E-mail: sitek@tu.kielce.pl
Technical University of Kielce, Department of Electric Engineering, Automation and Computer Science, Control and Management Systems Section, 1000-PP 7, Kielce, Poland
[***] Prof. dr hab. inż. Zbigniew Banaszak, E-mail: banaszak@tu.koszalin.pl
Department of Computer Science and Management, Technical University of Koszalin, Sniadeckich 2, 75-453 Koszalin, Poland

environment. Some papers [1],[2],[3] deal with general issues concerning the modeling and resource-constrained project scheduling. Authors present a survey of models and algorithms for the discrete-continuous project-scheduling. The heuristic methods are proposed. In this paper we present the job-shop scheduling problems with manpower resources representation. The objective considered here is the minimization of the makespan. This problem is strongly NP-hard. Classical approaches to these problems, as practiced in the management science and operations research (OR) fields, are characterized by a reduction of the problem into a mathematical programming formulation, and subsequent solution by formal algorithms methods in software tools. These approaches face the difficulty that the problem quickly becomes computationally infeasible for real sized problems, due to a large number of decision variables and large search space generated by the model. An additional disadvantage is that such models do not directly reflect the natural structure of the domain, making it difficult to employ domain specific knowledge to reduce search. Recent developments in artificial intelligence (AI) are making way for the use of knowledge-based techniques for solving the classical scheduling problems. AI provides a large number of tools and techniques such as informed search methods, high level programming environments, knowledge representation schemes. The aim of the paper is to present the constraint programming (CP) modeling framework as well as to illustrate its application to decision making in the case of job-shop scheduling problems with manpower resources representation. The possible approach based on CP/CCP paradigm assumes a possibility of adjusting the viable ways of problem specification, its programming language implementation as well as a searching strategy. We propose Oz/MOZART as a platform to modeling and solving scheduling problems.

2. JOB-SHOP SCHEDULING PROBLEMS

The classical job-shop scheduling problem is defined as in [4]. There are $j \in J$ jobs to be processed through $s \in S$ machines. Each job must pass through each machine exactly once. The processing of a job on a machine is called an operation $o \in O_j$ and requires a duration called the processing time p_{jos}. Technological constraints demand that each job should be processed through machines in a specific order. Each job should have a release time and deadline. The general problem in the above environment is to find a sequence in which jobs pass through the machines, which is compatible with the technological constraints and optimal with some performance criterion (very often makespan C_{max}). A variant of the described problem is the flow-shop scheduling problem, which arises if all jobs share the same processing order.

Job-shop scheduling problem is prototypical problem for a number of problems arising in several disciplines. Apart from other problems in production scheduling such as assembly line balancing and flexible manufacturing systems, job-shop scheduling has a close correspondence to project scheduling, time-tabling of lectures, etc.

In the classic job-shop scheduling problem resources outside the schedule are not taken into account, but it is a serious problem in decision making process, in practical scheduling problems. External resources can have radical influence whether or not the overall schedule is workable. In practice there is a number of other factors that must be taken into account in a typical job-shop, the manpower resources being the most important. Solving the job-shop scheduling problem with manpower resources one can support decision making process and answer basic support questions (Fig.5):

- If we have limited manpower resources, how long will this work take?
- If we have unlimited manpower resources, how long will this work take?

- How much manpower will we allocate in order to meet our delivery schedule?
- Can we use some of our excess manpower to start the production of another production without impact on our promised delivery of the current production?

2.1 Illustrative example

Let us consider the job-shop scheduling example, where $j \in \{a,b,c\}$ jobs are to be processed through $s \in \{s1,s2, s3\}$ machines. Each job must pass through each machine exactly once. The processing of a job *a* on machine *s1* is an operation *a* (denoted *aa*), on machine *s2* is an operation *b* (denoted *ab*), on machine *s3* is an operation *c* (denoted *ac*) etc. Technical requirements demand that each job should be processed in a specific order. For the purpose of this example the order for each job can be stated as follow *j=a, {aa, ab, ac}; j=b, {bb, ba, bc}; j=c, {ca, cb, cc}*. Processing time p_{ab} is a duration of the processing operation *b* in job *a*. There are processing times for the illustrative example: $p_{aa}=4$, $p_{ab}=3$, $p_{ac}=4$, $p_{ba}=5$, $p_{bb}=4$, $p_{bc}=4$, $p_{ca}=3$, $p_{cb}=4$, $p_{cc}=6$. The problem in this example is to find a sequence in which jobs *j=a, j=b, j=c* pass through the machines *s1, s2, s3* which satisfy technological constraints and are optimal with makespan. At first, in this example, manpower resources were not taken into account The result is the optimal schedule (Fig.1). Then the manpower requirement for the schedule calculated in the previous example was taken into account (Fig. 2). For each operation, except for last of each job, the manpower was two workers, for the last operation of each job manpower was one worker. The basic question is if we have limited manpower resources (for instance 3 workers) and how long will this work take?

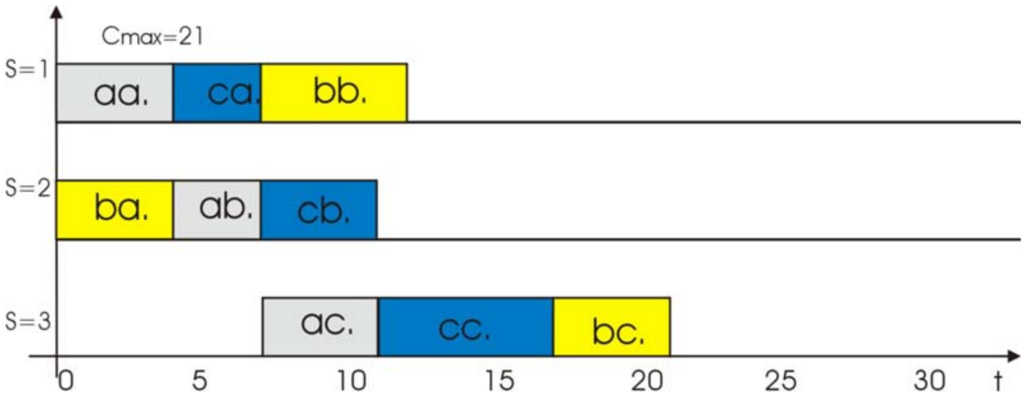


Fig.1. Classic Gantt chart for optimal schedule of illustrative example (C_{max}=21)

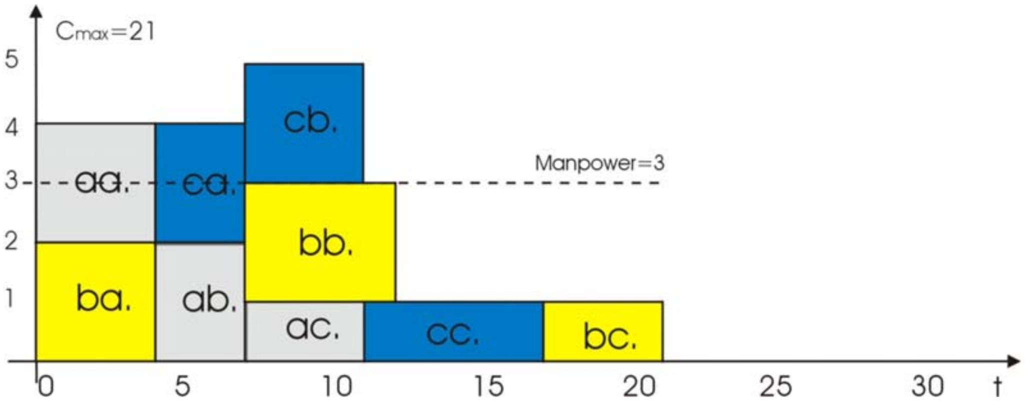


Fig.2. Optimal schedule for illustrative example (C_{max}=21, manpower requirements are from 1 to 5)

The answer to the above question is shown in Fig.4 and Fig.5. There are optimal schedules for the illustrative example with manpower limit equal to 3 workers. The manpower requirements are from 3 to 1 and the makespan is 27. This information is very important for decision process in SME. The next question is how to support the decision making process in this environment? The answer is constraint programming approach. A CP-based modeling and solving framework provides a good platform for consistency checking between the production orders completion requirements (makespan) and a company's resources and capability offered (machines, manpower, etc.).

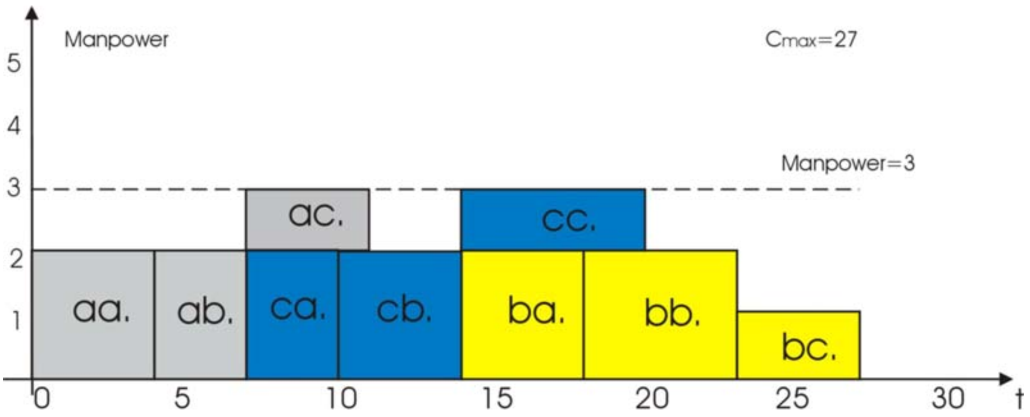


Fig.3. Optimal schedule for illustrative example (C_{max}=27, manpower requirements are from 1 to 3)

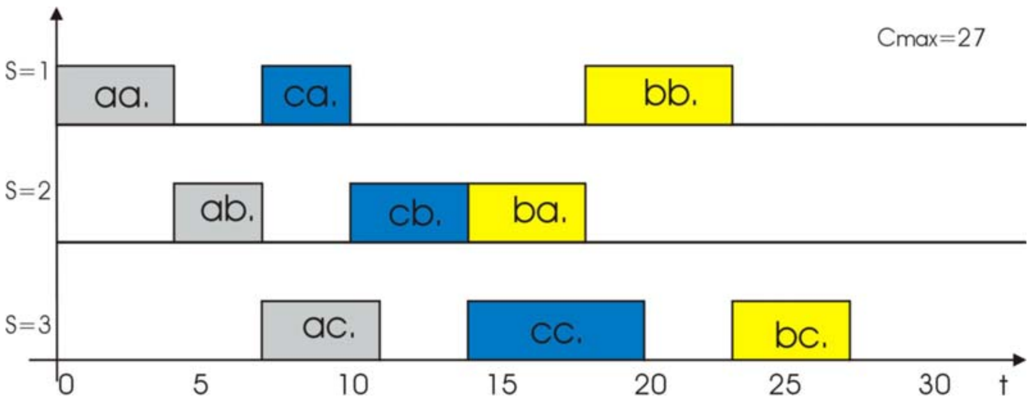


Fig.4. Classic Gantt chart for optimal schedule of illustrative example (C_{max}=21)

3. CP-APPROACH TO DECISION SUPPORT IN JOB-SHOP SCHEDULING

There are growing needs for decision support tools capable of assisting a decision maker in the scheduling problems in SME. The possible approach to decision support is based constraint programming. Constraint Programming (CP) is a problem solving method that was developed out of Logic Programming and Artificial Intelligence. The diversity of scheduling problems, the existence of many specific constraints (precedence, resource, capacity, etc.) in each problem and the efficient constraint based scheduling algorithms [2] make constraint programming a method of choice for the resolution of complex practical problems. In constraint programming approach to decision support in scheduling problems (fig.5), the problem to be solved is represented in terms of decision variables and constraints on these variables. In brief, constraint programming could be defined as a programming method based on the main principles mentioned below:

- The problem to be solved is explicitly represented in terms of decision variables and constraints on these variables. There are clearly separation between the problem statement and the algorithm to solve them.
- A constrained-based definition of the problem to be solved and a set of decision, is in itself translated into constraints, a deductive process referred to as constraint propagation is used to propagate the consequences of the constrains.
- The constraint propagation process takes place each time a new decision is made. This process is separated from the decision making algorithm.
- The overall constraint propagation process results from the combination of several local processes, each of which is associated with a particular constraint.

In constraint programming approach to decision support in scheduling problems the problem to be solved is represented in terms of decision variables and constraints on these variables. Depending on the particular applications, the variables of scheduling problems (job-shop, flow-shop, open-shop, and project shop) can be the following:

- The start time and the end time of each operation;
- The set of resources assigned to each operation (if this set is not fixed);

- The capacity of a resource that is assigned to an operation (e.g. the number of workers from a given team assigned to operation);
- The duration of an operation.

The constraints of a scheduling problem include:

- Temporal and precedence constraints;
- Resource constraints;
- Capacity constraints.

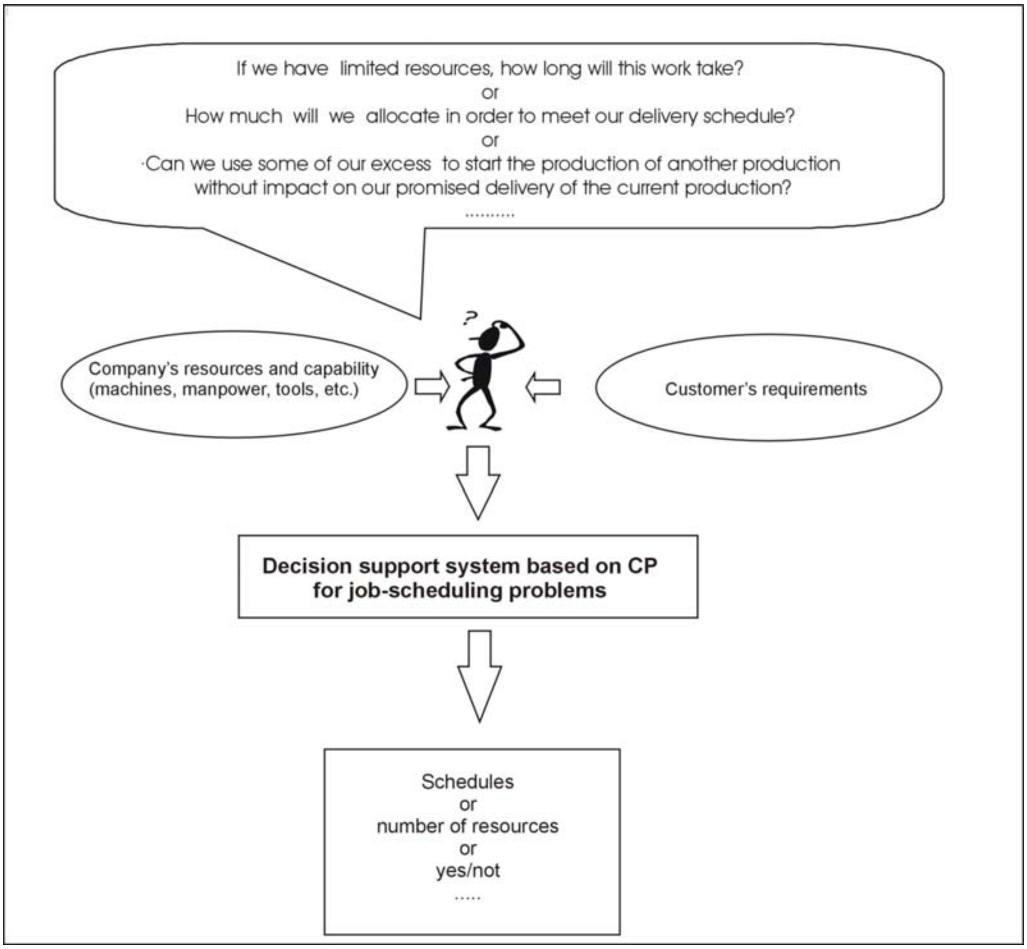

If we have limited resources, how long will this work take?
or
How much will we allocate in order to meet our delivery schedule?
or
Can we use some of our excess to start the production of another production without impact on our promised delivery of the current production?
....
?
Company's resources and capability (machines, manpower, tools, etc.)
Customer's requirements
Decision support system based on CP for job-scheduling problems
Schedules
or
number of resources
or
yes/not
.....

Fig.5. Decision support system based on constraint programming for job-shop scheduling problems

Programming methodology proposed takes into account the constraints imposed by a programmer experience and capability (possible problem statement), by a set of available software tools (CP languages) and a set of searching strategies (build-in the software tools or the proposed by programmers) (Fig.6)

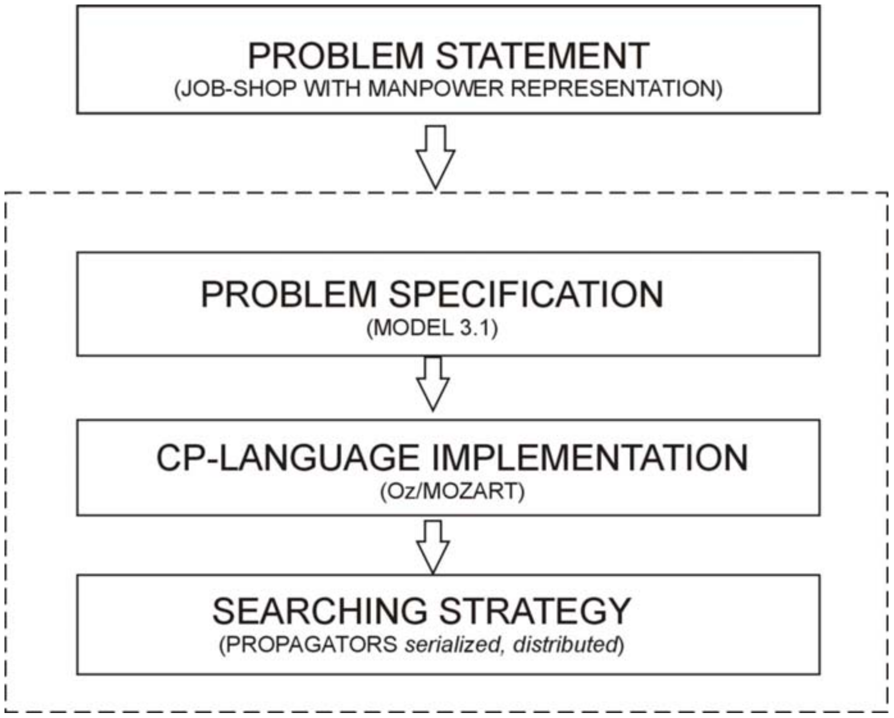

PROBLEM STATEMENT
(JOB-SHOP WITH MANPOWER REPRESENTATION)
PROBLEM SPECIFICATION
(MODEL 3.1)
CP-LANGUAGE IMPLEMENTATION
(Oz/MOZART)
SEARCHING STRATEGY
(PROPAGATORS *serialized, distributed*)

Fig. 6 Stages of CP-based programming methodology [10]

3.1 Constraint programming modeling-CP model for job-shop scheduling with manpower resources representation

The problem model in CP contains declarations of all decision variables, including their domains. Additionally, a set of constraints called *conceptual constraints* imposed on those variables is determined. Another type of constraints, *actual constraints*, is used for increasing the effectiveness of searching for a problem solution. Frequently, such constraints are provided by the CP software producers as ready predicates attached to the system. While modeling problems in CP environment, variables 0-1, which are common in the MP – Mathematic Programming environment, can be used. A knowledge of the specificity of the problem variables and how they relate to one another makes the process more effective. Thus we can say that in CP the problem is viewed through its variables, the approach, which is different from that in MP modeling. Here, the problem is considered as a whole, the logic is modeled through allocation variables and transformed into linear constraints. Furthermore, MP does not make use of the variables' characteristics and relations at the stage of modeling. Basic problem solving techniques in CP are: constraint propagation, a variety of variables distribution methods, backtracking, etc. The knowledge both on the CP implementation environment, described in Chapter 4, as well as on the job-shop system was used while building the model. The job-shop system makes it possible to arrange freely particular jobs and operations on the machines, whereas within a given job it defines a fixed order of operations. Allocation of operations to machines is also defined. A knowledge of data structures and relations between them helps determine decision variable X_{jo} as a moment which begins the operation *o* of job *j* (in MP it would be X_{josr}), and two dependent variables U_{jots}, K_{jor} (after specifying the X_{jo} their values are determined automatically – domain propagation), which in the MP environment would have a form of independent variables.

Tab.1. Decision variables

X_{jo}	*a decision variable, which means the moment the operation begins* $o \in O_j$, *for job* $j \in J$, *on machine* $s \in S$, *using resource* $r \in R$;
U_{jots}	$U_{jots} = \begin{cases} 1 \ \textit{when at the moment t, the operation o of the job j is performed on} \\ \textit{machine s} \\ 0 \ \textit{in all other cases} \end{cases}$
K_{jotr}	$K_{jotr} = \begin{cases} A_{j,o,r} \ \textit{when at the moment t the operation of the job j is performed} \\ \textit{using the resource type r} \\ 0 \ \textit{in all other cases} \end{cases}$

Basic constraints occurring in the job-shop scheduling and resource allocation are (1) sequence constraint, which enables the subsequent operation to start after the previous one has ended, (2) the constraint which prevents simultaneous access to a machine (at the *t* moment only one operation can be performed on machine *s*, and (3) limited resources in the system, necessary to perform further jobs. Constraints (2) and (3) are declarative and have a form of a loop.

$$X_{jo}+p_{jo} <= X_{j,o+1} \text{ for } j \in J \text{ i } o \in \{O_j - \text{final operation}\} \tag{1}$$

Where: p_{jo} – period of time operation *o* is performed for job *j* on machine *s*

(2)

```
For $s \in S$
    For $t = 1..T$
        $\sum_{j \in J} \sum_{o \in O_j} U_{jots} \le 1$
```

(3)

```
For t=1..T
    For $r \in R$
        $\sum_{j \in J} \sum_{o \in o_j} K_{j,o,t,r} \le N(r)$
```

where: $N(r)$	number of accessible *r-type* resources
A_{jor}	number of *r-type* resources necessary to perform operation *o* for job *j*

4. IMPLEMENTATION OF CONSTRAINT-BASED SCHEDULING IN OZ/MOZART

We propose Oz [6] as a platform to decision support in job-shop scheduling which integrates algorithms form OR and CP to achieve a combination of a high-level constraint language with efficient OR techniques. Oz is a concurrent constraint language providing for object-oriented, functional and constraint programming. The unique advantages of Oz are [7]:

- **Expressiveness**. Different language paradigms allow a natural high-level modeling of the job-shop scheduling problem. This is a way for rapid testing of different models.
- **Programmable Search**. The user can program his own search strategies. The search is completely separated from the reduction of the search space achieved by constraint propagation.
- **Modularity**. By expressing of different algorithms as constraints, they can be combined and interact in one environment.
- **Openness**. Due to the use of a C++ interface, new constraints can be implemented efficiently by the programmer and used like any Oz procedure.

For the purpose of the implementing of model (1) .. (3) the MOZART package environment and programming language Oz were used. The implementation of the model in Oz had several stages. In the first stage appropriate data structures were proposed. In order to ensure scaling properties, which in an important factor due to the implementation flexibility and easiness in carrying out calculation experiments, list structures for both parameters and decision variables were used (Fig. 7). The next step involved programming the model's constraints, which, due to the declarativity of Oz/MOZART environment, became the part of the program, which solved them. The manpower resource requirements for each operation can be interpreted as a next parameter of an operation (Fig. 8). Our experiences in applying language Oz to the above type of problems [8] induced us to search for more effective solutions in implementation of search algorithms. Therefore we used propagators what does Oz deliver for stronger propagation employed for capacity constraints. A unary resource is one that cannot be shared by two activities; i.e., as soon as an activity requires a resource, for a time interval, no other activity can use it during the same time interval. Unary resources can be used to model a variety of applications. A typical example of a unary resource is a machine in a job-shop scheduling application.

For unary resources Oz provides two propagators employing edge-finding to implement capacity constraints. The propagator *Schedule.serialize* is an improved version of an algorithm described in [9]. A single propagation step has complexity $O(n^2)$ where n is the number of jobs the propagator is reasoning on, i. e. the number of jobs on the resource considered by the propagator. Because the propagator runs until propagation reaches a fixed-point, we have the overall complexity of $O(k*n^3)$ when k is the size of the largest domain of a job's start time (at most $k*n$ values can be deleted from the domains of job variables).

The propagator *Schedule.taskIntervals* provides stronger propagation than *Schedule.serialize*. While a single propagation step has complexity $O(n^3)$, the overall complexity is $O(k*n^4)$.

```
%NAME pe RESERVED
declare W
fun {W}
   NUMBER_Of_Workers=10
   Jobs=[a b c d e f g h x j ]           %List of jobs
   Operations=[a b c d e f g h x j ]     %List of operations
%table of processing times
   Processing_Times=[ 62 24 25 84 47 38 82 93 24 66
                        47 97 92 22 93 29 56 80 78 67
                        45 46 22 26 38 69 40 33 75 96
                        85 76 68 88 36 75 56 35 77 85
                        60 20 25 63 81 52 30 98 54 86
                        87 73 51 95 65 86 22 58 80 65
                        81 53 57 71 81 43 26 54 58 69
                        20 86 21 79 62 34 27 81 30 46
                        68 66 98 86 66 56 82 95 47 78
                        30 50 34 58 77 34 84 40 46 44
                   0 ]
%table of manpower requirements for each operation
   Manpower_requirements=  [1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                               1 1 1 1 1 1 1 1 1 1
                        0 ]
%table of allocation operations of each Job to machines
   Machines = machines(
           0:[ ah bx cf dx ef fx gf hg xf ja]
           1:[ aj be ca dh eg ff gj hh xh je]
           2:[ ag bb cd dg ej fd gb hf xj jd]
           3:[ ad bh ch de ed fa gx hj xg jb]
           4:[ ae bf cg da ee fe gg ha xx jh]
           5:[ ac ba cj dc eh fc ga hc xc jf]
           6:[ af bj cc df ex fg gd hb xb jj]
           7:[ aa bg cb dj ec fj gc hx xe jc]
           8:[ ab bc cx db ea fh gh hd xd jg]
           9:[ ax bd ce dd eb fb ge he xa jx])
```

Fig.7. Data structures in Oz for Example_3 (MT10)

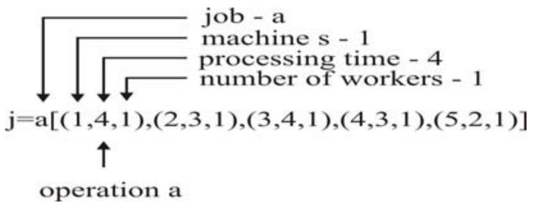


Fig.8. Description of job data structure

Tab.2. Computational examples

Example_1	**Example_2**

j∈{a,b,c,d,e}
o∈{a,b,c,d,e}
s∈{s1,s2,s3,s4,s5}
	j∈{a,b,c,d,e}
	o∈{a,b,c,d,e}
	s∈{s1,s2,s3,s4,s5}

j=a [(1,4,1), (2,3,1), (3,4,1), (4,3,1), (5,2,1)]	j=a [(1,4,2), (2,3,1), (3,4,1), (4,3,1), (5,2,1)]
j=b [(5,4,1), (4,5,1), (3,4,1),(2,3,1), (1,4,1)]	j=b [(5,4,1), (4,5,1), (3,4,1),(2,3,1), (1,4,2)]
j=c [(1,3,1), (5,4,1), (2,8,1), (3,3,1), (4,6,1)]	j=c [(1,3,2), (5,4,1), (2,8,1), (3,3,1), (4,6,1)]
j=d [(1,8,1), (2,4,1), (3,4,1), (4,3,1), (5,4,1)]	j=d [(1,8,2), (2,4,1), (3,4,1), (4,3,1), (5,4,1)]
j=e [(5,3,1), (4,8,1), (3,4,1), (2,3,1), (1,6,1)]	j=e [(5,3,1), (4,8,1), (3,4,1), (2,3,1), (1,6,2)]

Example_3

j∈{a,b,c,d,e,f}
o∈{a,b,c,d,e,f}
s∈{s1,s2,s3,s4,s5,s6}

j=a [(1,4,2), (2,3,1), (3,4,1), (4,3,1), (5,2,1),(6,3,2)]
j=b [(5,4,1), (4,5,1), (3,4,1),(2,3,1), (1,4,2),(6,4,1)]
j=c [(1,3,2), (5,4,1), (2,8,1), (3,3,1), (4,6,1),(6,3,2)]
j=d [(1,8,2), (2,4,1), (3,4,1), (4,3,1), (5,4,1),(6,4,1)]
j=e [(5,3,1), (4,8,1), (3,4,1), (2,3,1), (1,6,2),(6,5,2)]
j=f [(5,3,1), (4,8,1), (3,4,1), (2,3,1), (1,6,2),(6,4,1)]

Example_3 (MT10)

j∈{a,b,c,d,e, f,g,h,x,j}
o∈{a,b,c,d,e,f,g,h,x,j}
s∈{s0,s1,s2,s3,s4, s5, s6, s7, s8, s9}

j=a [(7,62,1) (8,24,1) (5,25,1) (3,84,1) (4,47,1) (6,38,1) (2,82,1) (0,93,1) (9,24,1) (1,66,1)]
j=b [(5,47,1) (2,97,1) (8,92,1) (9,22,1) (1,93,1) (4,29,1) (7,56,1) (3,80,1) (0,78,1) (6,67,1)]
j=c [(1,45,1) (7,46,1) (6,22,1) (2,26,1) (9,38,1) (0,69,1) (4,40.1) (3,33.1) (0,75,1) (5,96,1)]
j=d [(4,85,1) (8,76,1) (5,68,1) (9,88,1) (3,36,1) (6,75,1) (2,56,1) (1,35,1) (0,77,1) (7,85,1)]
j=e [(8,60,1) (9,20,1) (7,25,1) (3,63,1) (4,81,1) (0,52,1) (1,30,1) (5,98,1) (6,54,1) (2,86,1)]
j=f [(3,87,1) (9,73,1) (5,51,1) (2,95,1) (4,65,1) (1,86,1) (6,22,1) (8,58,1) (0,80,1) (7,65,1)]
j=g [(5,81,1) (2,53,1) (7,57,1) (6,71,1) (9,81,1) (0,43,1) (4,26,1) (8,54,1) (3,58,1) (1,69,1)]
j=h [(4,20,1) (6,86,1) (5,21,1) (8,79,1) (9,62,1) (2,34,1) (0,27,1) (1,81,1) (7,30,1) (3,46,1)]
j=x [(9,68,1) (6,66,1) (5,98,1) (8,86,1) (7,66,1) (0,56,1) (3,82,1) (1,95,1) (4,47,1) (2,78,1)]
j=j [(0,30,1) (3,50,1) (7,34,1) (2,58,1) (1,77,1) (5,34,1) (8,84,1) (4,40,1) (9,46,1) (6,44,1)]

Tab.3. Computational results

Example	Manpower	C_{max}	Processing times (ms)
Example_1	∞	33	< 100
Example_1	5	33	< 100
Example_1	3	37	< 100
Example_2	∞	33	< 100
Example_2	5	33	< 100
Example_2	3	44	< 100
Example_3	∞	47	109
Example_3	6	47	145
Example_3	4	53	134453
Example_4 (MT10)	∞	943	2333
Example_4 (MT10)	7	945	106903

Tab.4. Computational results (different propagators)

Example	propagator	Manpower	C_{max}	Processing times (ms)
Example_3	*Schedule.serialize*	10	47	109
Example_3	*FD.distribute*	10	47	1060
Example_3	*Schedule.serialize*	6	47	145
Example_3	*FD.distribute*	6	47	1450
Example_3	*Schedule.serialize*	4	53	134453
Example_3	*FD.distribute*	4	----	---------

After the complete implementation of the model (3.1) into Oz/MOZART environment computation experiments were carried out. The parameters of computational examples are presented in table 2. For each of the examples optimal solution was searched for due to the makespan – C_{max} (optimal schedule). Computation experiments were carried out with the different number of manpower resource. The results of calculation (makespans, processing times) are presented in table 3. The experiments were started on the computer PIV 1,4 GHz, RAM 512 under Windows XP. The received schedules for **Example_3** with different number of manpower resource as Gantt's charts are shown in Figs. 9-16.

To estimate effectiveness of searching strategies for implementation of the model (3.1) different searching strategies were used. First, **Example_3** was solved by using *Schedule.serialize* secondly *FD.distribute.* The results of calculation are shown in table 4. The analysis of the obtained results leads to the conclusion that propagator *Schedule.serialize* makes the calculating process significantly faster through a stronger propagation. Figures 17 and 19 present the parameters and the way both propagators were solved.

To visualize searching strategies, Oz Explorer was used. Oz Explorer is a visual constraint programming tool for Oz. The Explores uses the search tree as its central metaphor. The user can interactively explore the search tree which is visualized as it is explored. Visible nodes carry information on the corresponding constraints. The Explorer can be used with any search problem, the problem does need to be changed, annotated or modified in any way. First insights into the structure of the problem can be gained from visualization of the search tree: how are solutions distributed, how many solutions exist, how large are the parts of the tree explored before finding a solution. The full search tree for *Schedule.serialize* consists of 147 choice nodes and 8 solution nodes (see Fig.16). However, the full search tree for *FD.distribute* consists of 2169 choice nodes and 8 solution nodes (see Fig.18).

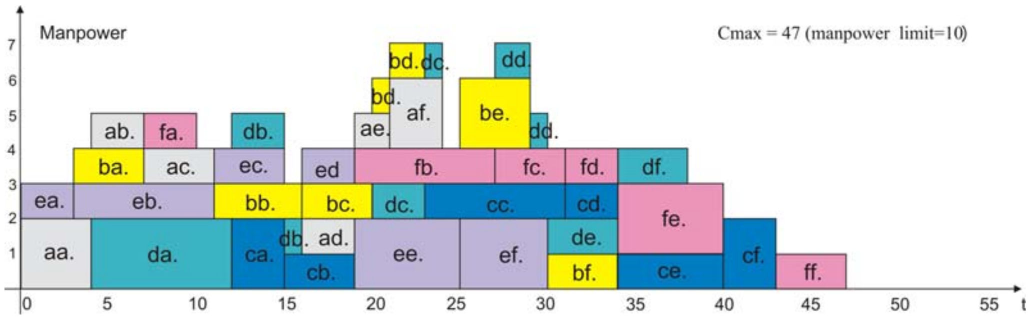


Fig.9. Optimal schedule for Example_3 (C_{max}=47, manpower requirements are from 1 to 7)

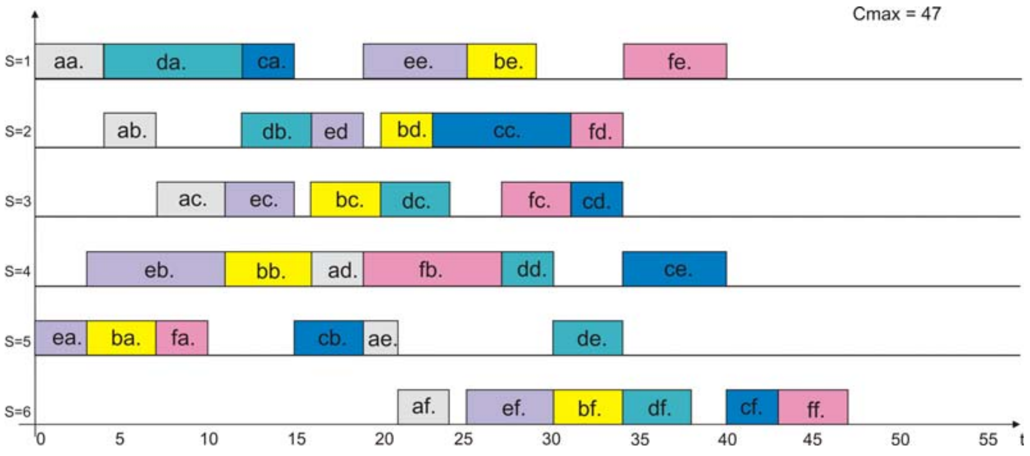


Fig.10. Gantt chart for optimal schedule of example from fig.9 (C_{max}=47)

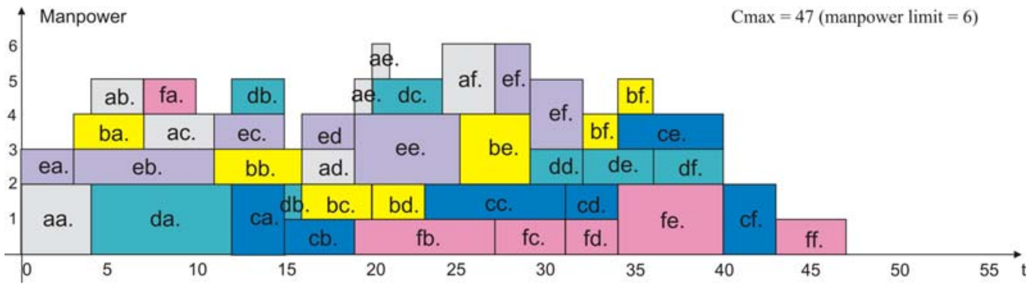


Fig.11. Optimal schedule for Example_3 (C_{max}=47, manpower requirements are from 1 to 6)

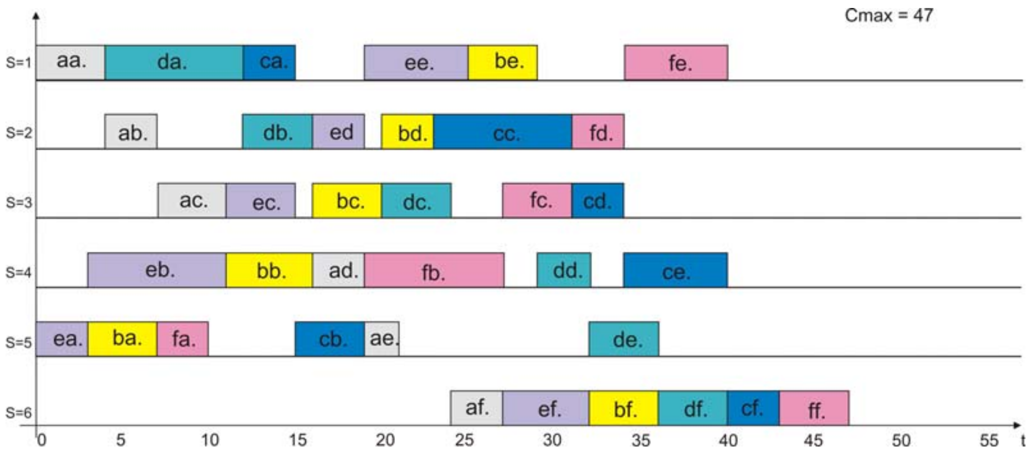


Fig.12. Gantt chart for optimal schedule of example from fig. 11 (C_{max}=47)

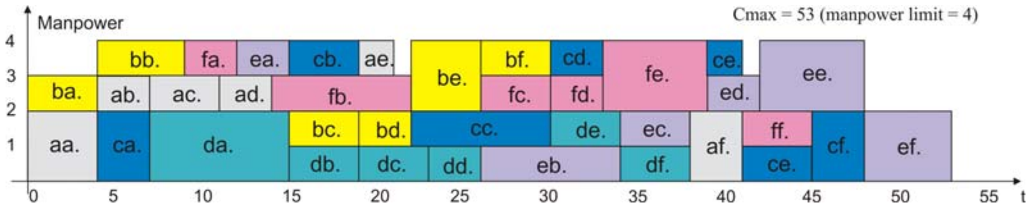


Fig.13. Optimal schedule for Example_3 (C_{max}=53, manpower requirements are from 3 to 4)

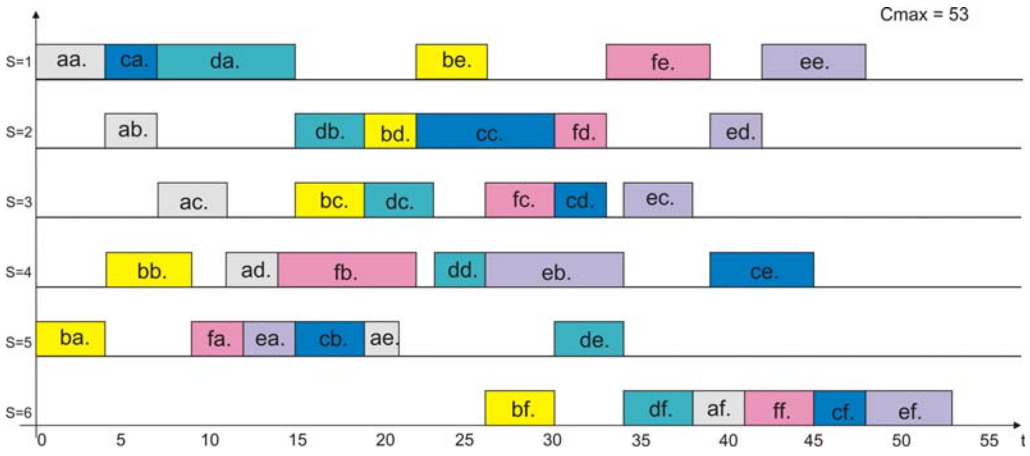


Fig.14. Gantt chart for optimal schedule of example from fig.13 (C_{max}=53)

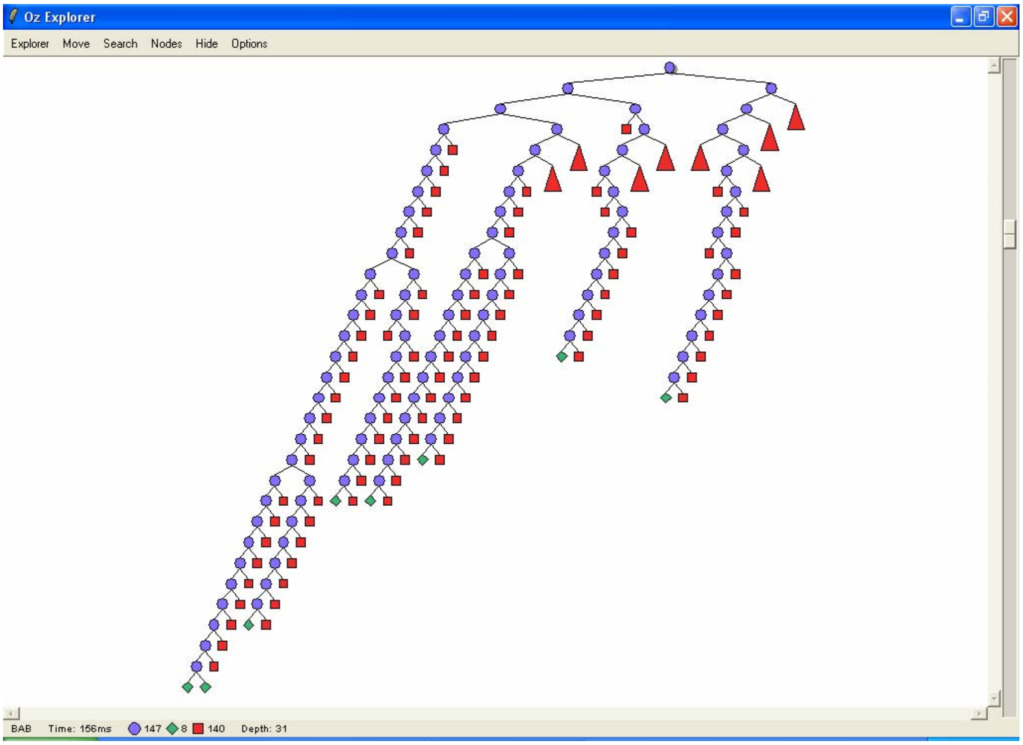
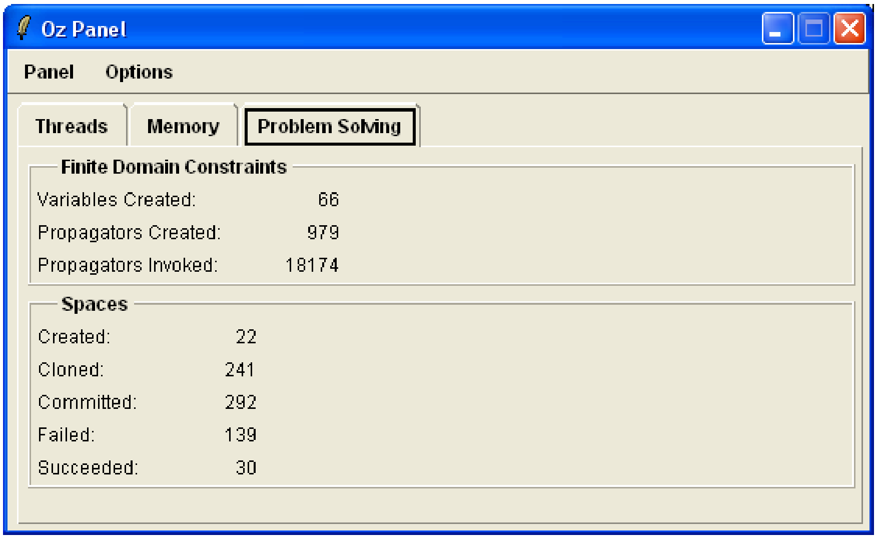

Oz Explorer
Explorer Move Search Nodes Hide Options
BAB Time: 156ms ● 147 ◆ 8 ■ 140 Depth: 31

Fig.16. A search tree for Example_3 (manpower =6) using *Schedule.serialize*



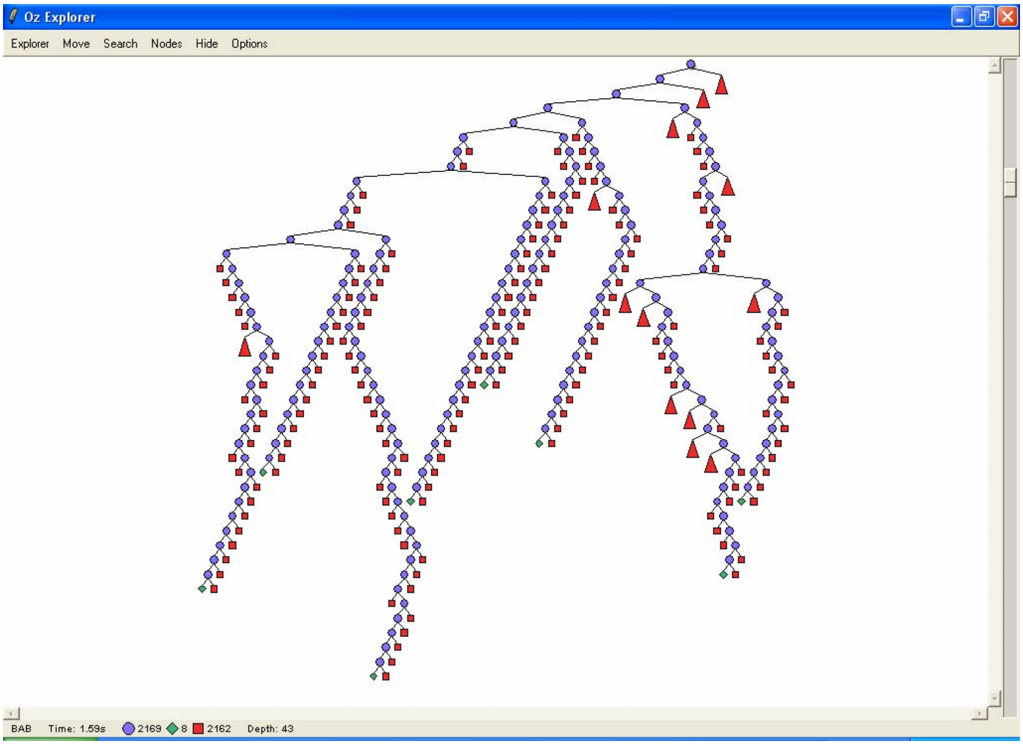
Finite Domain Constraints

```
Variables Created:        66
Propagators Created:     979
Propagators Invoked:   18174
```

Spaces

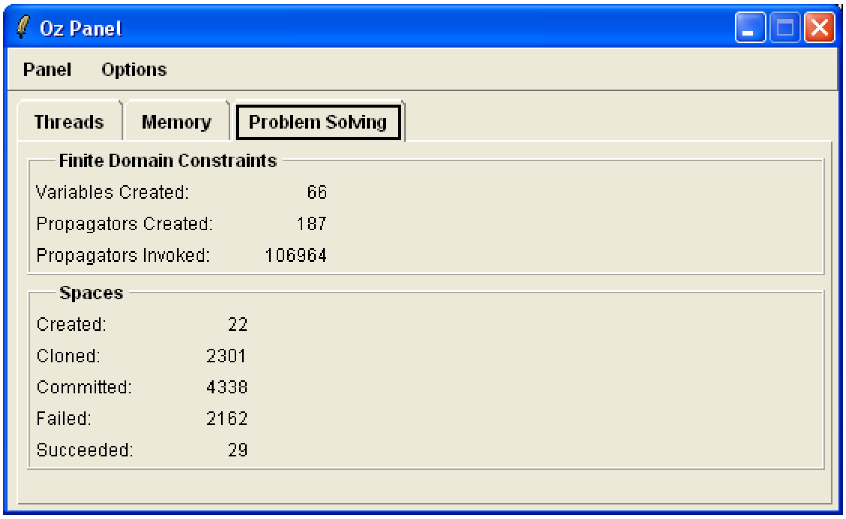
```
Created:        22
Cloned:        241
Committed:     292
Failed:        139
Succeeded:      30
```

Fig.17. Problem solving parameters for search tree from fig.16



BAB Time: 1.59s ● 2169 ◆ 8 🟥 2162 Depth: 43

Fig.18. A search tree for Example_3 (manpower =6) using *FD.distribute*



Finite Domain Constraints

```
Variables Created:          66
Propagators Created:       187
Propagators Invoked:    106964
```

Spaces

```
Created:            22
Cloned:           2301
Committed:        4338
Failed:           2162
Succeeded:          29
```

Fig.19. Problem solving parameters for search tree from fig.18

5. CONCLUSIONS

Allocation manpower to the jobs is more complicated than allocating machines, as different jobs and operations require different manpower. Therefore, CP approach is unusually interesting when referred to generally understood scheduling problems with manpower or other external resources. The constraint propagation reduces significantly the domains of decision variables of the modeled problems. This feature, together with backtracking-based methods, makes CP methodology very effective.

In addition, the CP implementation environment, e.g. Oz/MOZART possesses very strong propagation strategies (propagators: *Schedule.taskIntervals, Schedule.serialize*). The CP methodology presented here seems to be a promising alternative for commercially available tools based on other technologies. The proposed approach can be considered as a contribution to job-shop scheduling problems with manpower resources applied in SME where this kind of resources can have influence on production and delivery schedules. That is especially important in the context of cheap, fast and user friendly decision support in SME. The CP-tools fulfill the need of intelligent production management structures and can be based successfully in cases of job-shop scheduling problems with external resources.

Further work will concentrate on the generalization of the proposed approach and tools for all scheduling problems and resources allocation in SMEs, considering time constraints of their availability. The proposed approach seems to be a viable alternative option for supporting quite a number of decision making processes in SMEs.

REFERENCES

[1] JÓZEFOWSKA J.: *Discrete-continuous project scheduling,* Project Driven Manufacturing, WNT, Warszawa 2003, pp. 7-22.

[2] JÓZEFOWSKA J., RÓŻYCKI R., WĘGLARZ J.: *Uncertainty in discrete-continuous project scheduling* Project Driven Manufacturing, WNT, Warszawa 2003, pp. 23-33.

[3] MIKA M., WALIGÓRA G., WĘGLARZ J.: *Metaheuristic approach to the multi-mode resource-constrained project scheduling problem with discounted cash flows and progress payments,* WNT, Warszawa 2003, pp 34-53.

[4] FRENCH S.: *Sequencing and Scheduling: An Introduction to the Mathematics of Job-shop*. Chichester:Ellis Horwood, 1982.

[5] PAPE C.Le.: *Three Mechanisms for Managing Resource Constraints in a Library for Constraint-Based Scheduling* Proceedings INRIA/IEEE Conference on Emerging Technologies and Factory Automation, 1995.

[6] www.m**oz**art-**oz**.org/

[7] WÜRTZ J.: *Constraint-Based Scheduling in Oz,*. Operations Research Proceedings 1996, Berlin, Heidelberg, New York, Springer-Verlag, pp. 218-223.

[8] SITEK P., BANASZAK Z., MUSZYŃSKI W.: *Application of CLP to Prototyping Shop Orders in Small and Medium-Sized Enterprises* MMAR 11th IEEE International Conference on Methods and Models in Automation and Robotics, Międzyzdroje 2005, pp 1091–1096.

[9] MARTIN P., SHMOYS D. B.: *A new approach to computing optimal schedules for the job shop scheduling problem.* In International Conference on Integer Programming and Combinatorial Optimization, Vancouver, pp. 1996, 389-403.

[10] BANASZAK Z., JÓZEFCZYK J.: *Towards dedicated decision support tools: CLP-based approach.* Applied Computer Science and Production Management. Vol.1, No 1, 2005, pp. 161-180.