https://doi.org/10.26552/com.C.2013.1.39-43

Ludmila Janosikova – Tomas Hreben *

MATHEMATICAL PROGRAMMING VS. CONSTRAINT PROGRAMMING FOR SCHEDULING PROBLEMS

This paper focuses on a classical scheduling problem known as the job-shop scheduling problem which is one of the most difficult problems in combinatorial optimisation. The paper presents two solution techniques, namely mathematical programming and constraint programming and compares their computational efficiency on benchmark problems. In addition, the experience with scheduling trains in a passenger railway station is presented. The computational experiments proved that the mathematical programming approach outperforms constraint programming with respect to the quality of the solution.

Keywords: *Mathematical programming, constraint programming, scheduling, job-shop scheduling problem.*

1. Introduction

In general, the scheduling problem consists of positioning resource-demanding activities over time in such a way that side constraints are respected and an objective is minimised. Problems like this arise in diverse areas including production planning, civil engineering, computer science, logistics, transportation, and public service systems.

This paper focuses on a classical scheduling problem known as the job-shop scheduling problem which is one of the most difficult problems in combinatorial optimisation. The paper presents two solution techniques, namely mathematical programming (MP) and constraint programming (CP) and compares their computational efficiency. The concluding section deals with a more specific scheduling problem arising in transportation practice.

In the job-shop scheduling problem we are given a set of jobs and a set of machines. The job consists of a chain of operations. Each operation needs to be processed during a specified time period on a given machine. Once an operation has begun execution, it cannot be pre-empted by another operation. Each machine can process at most one operation at a time. The goal is to find such a schedule (i.e. an allocation of the operations to time intervals on the machines) which minimises the value of the given objective function. The most popular objective function is the makespan, i.e. the total time until all jobs are completed.

Although the basic job-shop scheduling problem is simple to understand, it is one of the most intractable optimisation problems. Therefore, it has attracted attention of many researchers since the 1950s. As a result the problem is well documented and thoroughly studied in the available literature (e.g. [1, 2, 3, 4, 5]). A detailed survey of scheduling techniques can be found in [6]. However, the problem remains attractive due to its computational complexity and still provides a good ground for new algorithmic ideas. At the same time it serves as a starting point for more practically relevant models.

2. Mathematical programming model

We begin by specifying the job-shop problem more precisely. The specification and the following MP model are adopted from [2].

As the input, we have a finite set J of n jobs and a finite set M of m machines. For each job $j \in J$ we are given a permutation (π_{j1}, π_{j2}, ..., π_{jm}) of the machines, which represents the processing order of j through the machines. Thus the first operation of j must be processed on π_{j1}, the second operation on π_{j2}, etc. Also for each job j and machine a we are given a nonnegative integer tjα, the processing time of j on a. The goal is to find a schedule of J on M, that is, an allocation of the operations to time intervals on the machines. It means that the output is the start time of each operation on a given machine. Since the operation cannot be interrupted, the end time is determined by the start time plus the processing time. Let the time, when the execution of job j starts on machine a, be represented by continuous variable x_{ja}. The objective is to minimise the makespan, i.e. the maximum of the completion times of all jobs. In accordance with the literature, let C_{max} denote the makespan. The schedule must respect two types of constraints. The precedence constraints reflect the order of operations of one job and say that the operation on machine π_{jk} needs to be finished before the next operation on machine π_{jk+1} can start ($k = 1$, ..., $m - 1$). The precedence can be expressed by the equations

* **Ludmila Janosikova, Tomas Hreben**
Department of Transportation Networks, Faculty of Management Science and Informatics, University of Zilina, Slovakia,
E-mail: Ludmila.Janosikova@fri.uniza.sk

$$x_{j\pi_{jk}} + t_{j\pi_{jk}} \le x_{j\pi_{j,k+1}} \quad \text{for all } j \in J \text{ and } k = 1, ..., m-1 \qquad (1)$$

The disjunctive constraints express that each machine can process only one operation at a time, i.e. one of the following relations holds:

$$x_{ia} \ge x_{ja} + t_{ja} \text{ or } x_{ja} \ge x_{ia} + t_{ia} \quad \text{for all } i, j \in J, i < j$$

$$\text{and } a \in \text{M} \qquad (2)$$

In a mathematical programming model, the "either-or" constraint (2) is transformed to two linear constraints introducing a binary variable y^a_{ij} for each machine and each couple of jobs. The interpretation is that y^a_{ij} is 1 if *i* precedes *j* on machine *a*, and 0 if *j* precedes *i*.

The resulting formulation of the model is as follows:

$$\textit{minimise } C_{\max} \qquad (3)$$

$$\textit{subject to } x_{j\pi_{jk}} + t_{j\pi_{jk}} \le x_{j\pi_{j,k+1}} \quad \forall j \in J, k = 1, ..., m-1 \qquad (4)$$

$$x_{ia} \ge x_{ja} + t_{ja} - Ky^a_{ij} \quad \forall i, j \in J, i < j, \forall a \in M \qquad (5)$$

$$x_{ja} \ge x_{ia} + t_{ia} - K(1 - y^a_{ij}) \quad \forall i, j \in J, i < j, \forall a \in M \qquad (6)$$

$$C_{\max} \ge x_{j\pi_{jm}} + t_{j\pi_{jm}} \quad \forall j \in J \qquad (7)$$

$$x_{ja} \ge 0 \quad \forall j \in J, a \in M \qquad (8)$$

$$y^a_{ij} \in \{0,1\} \quad \forall i, j \in J, i < j, \forall a \in M \qquad (9)$$

The objective (3) minimises the makespan. Constraints (4) are the precedence constraints. Constraints (5) and (6) are linear forms of the disjunctive constraints (2). *K* is a large constant (e.g. the sum of the processing times of all operations). Constraints (7) define the makespan as the maximum of the end times of the last operations of all jobs. Constraints (8) and (9) specify the definition domains of the variables.

3. Constraint programming model

Constraint programming is an alternative approach to combinatorial optimisation. It is a software technology that attempts to reduce the gap between the high-level description of an optimisation problem and the computer algorithm implemented to solve it. Constraint programming allows for representing many problems in a way which is very close to a natural language description. The problem to be solved must be formulated as a *constraint satisfaction problem* (CSP) first. In case we are interested in finding a solution, which is the best with respect to some criterion, we associate CSP with an objective function that we want to minimise or maximise. This leads to a modification of a CSP that we call a *constraint optimisation problem*.

A constraint satisfaction problem is defined by:
- a set of decision variables,
- for each variable, a set (or a range) of possible values called its *domain*,
- a set of constraints on these variables.

Informally, a constraint on a sequence of variables is a relation on their domains. It can be viewed as a requirement that states which combinations of values from the variable domains are admitted. In constraint programming, constraints are used actively to deduce infeasible values and delete them from the domains of variables. This mechanism is called *constraint propagation*. It represents the core of constraint programming systems. Each constraint computes impossible values for its variables and informs other constraints. This process continues as long as new deductions are made. Constraint propagation is associated with *tree search techniques* in order to find solutions or prove optimality. Each node and each decision will induce constraint propagation automatically. Many specific and efficient algorithms are used in this propagation, but do not need to be known by the end-user.

In this section we formulate the job-shop scheduling problem as a constraint optimisation problem. As a modelling language we use the extended version of the Mosel language, which is embedded into the general optimisation software tool *Xpress Optimization Suite. Xpress* includes module *Kalis* [7] for defining and solving constraint programming problems. *Kalis* extends the Mosel modelling language with new features. Some of them are designed especially for scheduling problems, e.g. objects modelling operations and resources (machines, raw material etc.). When working with these scheduling objects it is often sufficient to state the objects and their properties, such as processing times or resource use; the necessary constraint relations are set up automatically by *Xpress-Kalis* (referred to as implicit constraints).

To make the CP model of the job-shop problem simpler, we first slightly change the definition of the processing time. Suppose that the processing time is related to the order of the operation within a given job, not to the machine at which it is executed. So the processing time of the *k*-th operation of job *j* will be denoted by d_{jk} instead of the symbol $t_{j\pi_{jk}}$ used in the MP model.

In the CP model formulation, every operation is represented by an object $task_{jk}$ ($j \in J, k = 1, ..., m$) of type *cptask*, which encapsulates three discrete variables:
- *start* representing the start time of the operation,
- *end* representing the completion time of the operation,
- *duration* representing the processing time of the operation.

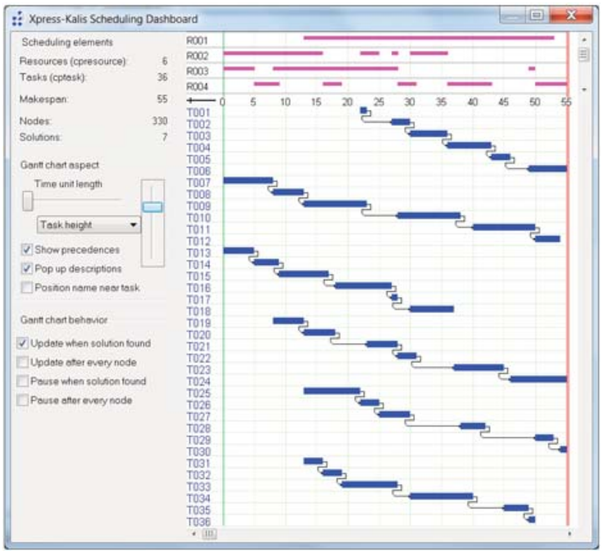
Variables *start* and *end* have the same domain {0, ..., *K*}, *duration* is fixed to the processing time d_{jk}. *Xpress-Kalis* implicitly states the relation between the start, duration, and completion time of an operation: $task_{jk}.end = task_{jk}.start + task_{jk}.duration$.

Machines are represented by objects res_k ($k = 1, ..., m$) of type *cpresource*. Every machine has unary capacity. This means that at most one operation may be processed at any one time. The disjunction between the jobs is also established implicitly.

The following model formulates the job-shop scheduling problem.

```
declarations
task: array(J,1..m) of cptask
    res: array(1..m) of cpresource
    makespan: cpvar
    L: cpvarlist
end-declarations
K := sum(i in J, j in 1..m) d(i,j)
forall(k in 1..m) set_resource_attributes(res(k),
KALIS_UNARY_RESOURCE, 1)
forall(j in J) do
    forall(k in 1..m) do
    set_task_attributes(task(j,k), d(j,k), res(pi(j,k)))
    0 <= getstart(task(j,k)); getstart(task(j,k)) <= K
    0 <= getend(task(j,k)); getend(task(j,k)) <= K
    end-do
    L += getend(task(j,m))
end-do
forall(j in J, k in 1..m-1) setsuccessors(task(j,k), {task(j,k+1)})
makespan = maximum(L)
```

Xpress Optimization Suite offers a visual development environment *Xpress-IVE* that make easy to implement and debug the model and display the solution. For illustration, Fig. 1 shows the Gantt chart display of the solution created by *IVE*. Above the Gantt chart we can see the resource usage display.



Scheduling elements
Resources (cpresource): 6
Tasks (cptask): 36
Makespan: 55
Nodes: 330
Solutions: 7

Gantt chart aspect
[✓] Show precedences
[✓] Pop up descriptions
[ ] Position name near task

Gantt chart behavior
[✓] Update when solution found
[ ] Update after every node
[ ] Pause when solution found
[ ] Pause after every node

Fig. 1 Solution display in Xpress-IVE (problem FT 6)

4. Experiments with models

We tested the two approaches on the benchmark problems available from the Operations Research Library [8]. The benchmark problems are formulated by various authors:

- ABZ 5, ABZ 6, and ABZ 9 are from [1],
- FT 6, FT 10, and FT 20 are from [3],
- LA 1 – LA 40 are from [4].

The MP model was implemented using two general optimisation software tools: *Xpress Optimization Suite* and *Gurobi Optimizer* [9]. The CP model was implemented in *Xpress Optimization Suite*. The goal of the experiments was to compare computational efficiency of the implementations, i.e. CPU time needed to prove the optimality. The computational time was limited to an hour. If the solver did not finish in the limited time, the best solution found so far was recorded. The experiments were performed on a personal computer equipped with the Intel Core i7 processor with 1.60 GHz and 8 GB of RAM.

The results of the experiments are reported in Table 1. A summary comparison of the approaches is in Table 2. The MP approach outperforms CP with respect to the quality of the solution. Between the two MP solvers tested, Gurobi wins from both points of view, the solution quality and the CPU time. Generally Xpress-Kalis was not able to achieve so good solutions as the MP solvers, however it runs quickly especially on the problems with small number of machines.

5. Conclusions

The results of the computational experiments with the job-shop problem suggest that constraint programming could be a successful method for solving complex scheduling problem arising in practice. To verify this hypothesis, we investigated the problem of routing and scheduling trains at a passenger railway station.

The problem of routing and scheduling trains at a station is a subproblem of the generation of a timetable for a railway company. The generation of a timetable is a hierarchical process. At the first stage, a preliminary timetable for the whole network is proposed. In this phase, a macroscopic viewpoint at the railway network is applied. Stations are considered as black boxes. Capacity limits of particular stations and the movement of trains inside the stations are not taken into account. Then, at the second stage, a microscopic viewpoint related to stations is applied. At every station, the network timetable is checked whether it is feasible with respect to capacity, safety and train operators' preferences. To prove the feasibility, detailed routes and schedules for the trains are generated. If desired arrival and departure times are not feasible at the microscopic level, the process returns to the first stage, where the timetable must be adjusted.

To design routes and schedule for the trains at a station, the following partial issues are subject to decision-making process. For each train,

- a platform track must be specified at which the train should arrive; the platform track assignment determines the route, on which the train approaches from an in-line (or from a depot) to the platform, or departs from the platform to an out-line (or to a depot),

Table 1

Computational comparison									
Problem	No. of jobs	No. of mach.	Opt. makespan	Mathematical programming				Constraint programming Xpress-Kalis	
				Xpress		Gurobi			
				Makespan	CPU time [s]	Makespan	CPU time [s]	Makespan	CPU time [s]
ABZ 5	10	10	1234	1239	3600.00	1234	21.11	1234	921.01
ABZ 6	10	10	943	943	13.90	943	3.85	943	40.25
ABZ 9	20	15	678	852	3600.00	742	3600.00	1842	3600.00
FT 6	6	6	55	55	0.80	55	0.27	55	0.08
FT 10	10	10	930	950	3600.00	930	473.30	930	3600.00
FT 20	20	5	1165	1347	3600.00	1177	3600.00	1265	3600.00
LA 1	10	5	666	666	598.98	666	5.51	666	0.30
LA 2	10	5	655	655	92.42	655	4.91	655	0.51
LA 3	10	5	597	597	144.40	597	3.46	597	0.40
LA 4	10	5	590	590	83.23	590	1.83	590	0.29
LA 5	10	5	593	593	3600.00	593	16.38	593	0.25
LA 6	15	5	926	926	3600.00	926	3600.00	926	3.20
LA 7	15	5	890	890	3600.00	890	3600.00	890	2.20
LA 8	15	5	863	863	3600.00	863	3600.00	863	0.35
LA 9	15	5	951	951	3600.00	951	3600.00	951	3.78
LA 10	15	5	958	958	3600.00	958	3600.00	958	1.35
LA 11	20	5	1222	1222	3600.00	1222	3600.00	1222	4.83
LA 12	20	5	1039	1044	3600.00	1039	3600.00	1039	1.05
LA 13	20	5	1150	1150	3600.00	1150	3600.00	1281	3600.00
LA 14	20	5	1292	1292	3600.00	1292	3600.00	1292	5.71
LA 15	20	5	1207	1264	3600.00	1207	3600.00	1207	76.70
LA 16	10	10	945	945	882.19	945	6.77	945	7.60
LA 17	10	10	784	784	1293.82	784	9.64	784	0.80
LA 18	10	10	848	848	15.97	848	7.47	848	28.11
LA 19	10	10	842	842	268.20	842	10.84	855	3600.00
LA 20	10	10	902	902	42.89	902	3.31	911	3600.00
LA 21	15	10	1046	1167	3600.00	1064	3600.00	1104	3600.00
LA 22	15	10	927	994	3600.00	927	3600.00	1082	3600.00
LA 23	15	10	1032	1088	3600.00	1032	3600.00	1034	3600.00
LA 24	15	10	935	1043	3600.00	938	3600.00	1050	3600.00
LA 25	15	10	977	1083	3600.00	977	1944.78	1323	3600.00
LA 31	30	15	1784	2244	3600.00	1784	3600.00	7110	3600.00
LA 40	15	15	1222	1360	3600.00	1243	3600.00	1383	3600.00

- arrival time at the platform and departure time from the platform need to be determined.

The objective is to meet the time and space requirements specified by the railway operators as much as possible. It means the schedule should minimise the deviations from desired arrival and departure times and maximise the preferences of the trains for platforms.

A mixed integer programming, multiple criteria model was proposed within the previous research project [10]. The model was verified by using the real data of Prague main station and the

Table 2

Combined MRE[a] and CPU values						
	Mathematical programming				Constraint programming Xpress-Kalis	
	Xpress		Gurobi			
	MRE [%]	CPU time [s]	MRE [%]	CPU time [s]	MRE [%]	CPU time [s]
Suma	132.75	82636.80	14.23	67313.43	576.09	47898.76
Mean	4.02	2504.15	0.43	2039.80	17.46	1451.48
Standard deviation	7.19	1592.04	1.68	1767.60	58.80	1766.11

[a] Mean relative error (MRE) = (Makespan achieved-Optimum)/Optimum*100

timetable valid for the years 2004/2005. Because of the computational complexity, the problem was solved using the Local Branching metaheuristics. Afterwards the problem was formulated as a constraint optimisation problem. In this formulation the tasks correspond to trains and the resources correspond to platform tracks. The model is not trivial because many structural constraints have to be respected. A part of these constraints model safety rules for train movements. If two trains are on conflicting routes we must ensure that there is at least a required minimum headway (time interval) between them, for safety and signalling reasons. In the model the headways are represented by setup times between tasks. Another complication stems from the fact that a train may usually travel to several platforms, i.e. a task is not assigned unambiguously to a resource. This leads to a scheduling model with alternative resources.

The CP model of the considered problem was formulated but its solution did not live up to expectations. The time limit for the solver run was set to half an hour (the same total computation time limit was used for the Local Branching metaheuristics in the previous study). The solution achieved within this time limit was several times worse than the best solution found by the Local Branching metaheuristics. To conclude, according to our experience the MP approach seems to be a better choice for complex scheduling problems arising in transportation practice.

Acknowledgement

This research was supported by the Scientific Grant Agency of the Ministry of Education of the Slovak Republic and the Slovak Academy of Sciences under project VEGA 1/0296/12 "Public service systems with fair access to service" and by the Slovak Research and Development Agency under project APVV-0760-11 "Designing of Fair Service Systems on Transportation Networks".

References

[1] ADAMS, J., BALAS, E., ZAWACK, D.: The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science* 34 (1988), pp. 391-401.
[2] APPLEGATE, D., COOK, W.: A Computational Study of the Job-shop Scheduling Problem. *ORSA J. of Computing* 3(2), 1991, pp. 149-156.
[3] FISHER, H., THOMPSON, G. L.: *Probabilistic Learning Vombinations of Local Job-shop Scheduling Rules.* J. F. Muth, G. L. Thompson (eds.), Industrial Scheduling, Prentice Hall, Englewood Cliffs, New Jersey, 1963, pp. 225-251.
[4] LAWRENCE, S.: *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques* (Supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.
[5] VAESSENS, R. J. M., AARTS, E. H. L., LENSTRA, J. K.: Job Shop Scheduling by Local Search. *INFORMS J. on Computing* 8, 1996, pp. 302-317.
[6] JAIN, A. S., MEERAN, S.: Deterministic Job-shop Scheduling: Past, Present and Future, *European J. of Operational Research* 113 (1999), pp. 390-434.
[7] *Xpress-Kalis Reference Manual* [online]. Available from: http://www.fico.com [Accessed 10 October 2011].
[8] *OR-Library* [online]. Available from: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html [Accessed 19 June 2012].
[9] *Gurobi Optimizer 5.0* [online]. Available from: http://www.gurobi.com [Accessed 19 June 2012].
[10] JANOSIKOVA, L., KREMPL, M.: Routing and Scheduling Trains at a Passenger Railway Station. *Proc. of the 16th Intern. Sci. Conference Quantitative Methods in Economics,* Bratislava, 2012. Bratislava : Ekonom, 2012, pp. 108-114.