

Scheduling Tests on Vehicle Prototypes using Constraint Programming

Kamol Limtanyakul, Uwe Schwiegelshohn

Robotics Research Institute, Dortmund University, 44227 Dortmund, Germany,
{kamol.limtanyakul, uwe.schwiegelshohn}@udo.edu

In this paper, we address the problem of scheduling tests on vehicle prototypes using Constraint Programming. The problem originates in the automotive industry where a manufacturer must perform several hundreds of tests on prototypes before starting mass production of a vehicle. Each test must be allocated to an appropriate prototype with respect to hardware requirements. Further, it is necessary to observe various test dependencies. The manufacturer is interested in reducing the number of prototypes to save test costs and in minimizing the makespan in order to start the production as early as possible.

We formulate the problem using Constraint Programming with the makespan being the primary objective. The number of prototypes is a parameter which must be specified before starting the optimization procedure. Using exemplary input data given from a car manufacturer, we solve this single objective optimization problem several times for different values of our parameter to determine the relation between the number of prototypes and the corresponding makespan. We are able to either find the optimal solution or at least a good feasible solution within a reasonable computational time even for our largest problem size comprising about five hundreds tests and more than one hundred prototypes.

Keywords: Applications, Constraint Logic Programming, Machine Scheduling

1 Introduction

In the automobile industry, a manufacturer must typically perform several hundred tests on vehicle prototypes before a vehicle series goes into production. These prototypes are handmade and expensive (between 0.5 to 1.5 Million Euro each). Most prototypes can be used for several tests if the tests are arranged in an appropriate order. For instance, any crash test must obviously be the last test on this prototype. This leads to a parallel machine scheduling problem with prototypes and tests being machines and jobs, respectively. Therefore, we must determine the allocation of tests to prototypes and the appropriate sequence for all tests allocated to a prototype. Ignoring cost differences between different versions of prototypes, the problem has two major objectives as we want to minimize the number of prototypes required to perform all tests and the makespan of the schedule in order to start the production as early as possible.

Further, there are many additional constraints. For instance, each prototype is a combination of different kinds of components, like engine or gear box. Due to various incompatibilities between those components, this produces in practice up to 600 possible variants with different costs and production times. Of course, we can only allocate a test to a prototype if this prototype satisfies the component requirements of this test. For instance, while a prototype with a gasoline engine is not suited to execute a diesel engine test it does not matter which kind of engine is used to perform a brake system test. However note that we do not yet consider how to configure a proper prototype variant as this requires further technical information to match various components, for instance. The list of possible variants is previously determined by the manufacturer and can be

used several times. Our scheduling algorithm must find the minimum number of prototypes and their corresponding variants from the given list to ensure that all tests can be performed.

We must also take into account temporal restrictions like release dates and due dates. For instance, consider a driving test in winter. As the complete testing process typically takes more than one year, we can obtain appropriate conditions for the driving test by using appropriate release and due dates. Also the manufacturer may use due dates to ensure that some critical tests are completed early enough to allow a repetition in case of a failure.

Most companies have a special shop that produces the prototypes. Due to the limited capacity of this shop, the prototypes are sequentially manufactured resulting in an availability time for each prototype. Following the manufacturing, a prototype requires an initial set-up process whose duration depends on the selected variant of the prototype. For instance, a prototype for a corrosion test must be painted while this is not necessary for executing a crash test. Hence, the scheduler must ensure that tests can start only when the assigned prototype is constructed and sufficiently prepared for the test. A valid schedule must obey further constraints. For instance, a brake system test must be completed before starting a long-run test. Further, the same prototype cannot be used for two long-run tests. Finally, a brake system test and a suspension system test must be executed on the same prototype in order to verify the co-ordination of both systems together.

We use Constraint Programming (CP) to formulate our scheduling problem. Further, we realize that the number of prototypes is discrete while the makespan is (almost) continuous. Therefore, we apply the classical approach of parameterization, that is, we fix the number of prototypes and minimize the makespan with respect to all constraints. Initially, we assume a large number of prototypes such that the makespan only depends on the temporal constraints. Then we repeatedly reduce the number of prototypes and solve the problem again. This produces a relation between the number of prototypes and the makespan.

This paper is organized as follows. Section 2 provides background information about previous work and other relevant projects. Then we formally define our problem and introduce appropriate notations in Section 3. The CP model of our problem is explained in Section 4. The computational results in Section 5 show the capability of our method to handle various sizes of this problem within a reasonable time. Finally, we end with a brief conclusion and an outline of future work.

2 Previous Work

To solve this problem, Scheffermann et al. [9] have developed a heuristic approach based on specific problem knowledge and on some intuitive ideas. They improve the quality of the result by applying statistical methods to adjust tuning parameters. This approach is used as a support tool to help the manufacturer make decisions in a real working environment. However, it is still difficult to find the appropriate parameters that lead to a good result. A deterministic method may be more suitable.

Mixed-Integer Linear Programming (MILP) is a well known method to solve many combinatorial optimization problems. Although it has been used for several decades, many researchers report that frequently MILP alone is not an effective tool to solve large-scale scheduling problems for parallel machines with release and due dates. For instance, Hooker [4] and Sadykov and Wolsey [8] compared the performance of several methods, including the MILP approach, to minimize makespan and allocation cost. Their results showed that MILP can handle small problems well but that it is not appropriate for cases dealing with many jobs like our problem.

Constraint Programming (CP) is an alternative deterministic approach, see Baptiste et al. [1]. It has originally been developed for computer science applications. CP consists of two main parts:

a modeling part which generates a set of constraints that must be satisfied and a search part that describes how to search for solutions. CP will start by searching for a solution that satisfies all constraints. This part is known as Constraint Satisfaction Problem. Basically, CP applies a mechanism called constraint propagation to reduce the domain of variables. However, that is usually not enough to obtain a feasible solution. The solver must still enumerate through the remaining set of variables. Further, the reduction and the search parts must simultaneously interact to determine a feasible solution. In the next step, CP adds a new constraint to ensure that the new objective is strictly better than the current value. As a result, the solution will be improved until it reaches the optimal value.

CP is superior to MILP in expressing constraints that are not limited to linear inequalities. Hence, some kinds of constraints, like disjunctive constraints, can be formulated more naturally and solved more effectively. Also, machine eligibility constraints can obviously be represented with the help of logic expressions, like if...else, instead of using many complex linear inequalities.

CP can solve a problem efficiently if constraints propagate well and tighten the objective value. Therefore, minimization of the makespan is well suited as the last completion time of all job directly leads to a lower bound of the objective value. For minimization of a sum of the set-up times or the allocation cost, any bounds of the cost function do not indicate directly which term of the sum should be reduced, see Danna and Pape [3].

In addition to the previous work [9], there are a few similar projects related to vehicle tests in the automotive industry. Zakarian [10] developed an analysis model and a decision support tool to evaluate the performance of product validation and test plans for General Motors Truck Groups. His work concentrates on stochastic scheduling. He models uncertainties associated with processing times of tests and product failures in order to determine the trade-off between the number of vehicles used in the validation plan and percentage of completed tests. First, several heuristic rules are applied to generate initial schedules based on the probability function. Then, the performance of obtained schedules is measured by a Markov model and simulation approaches.

Lockledge et al. [6] applied a multi-stage mathematical programming model to optimize the fleet of prototypes for Ford Motor Company. In the first stage, they determine the number of required variants subject to component requirements of all tests. Then, the second model solves the problem for a minimum number of cars of each variant such that all tests can be executed before their due dates. It is possible to apply MILP in this case because there are few different values of due dates instead of arbitrary values. The problem reduces to an assignment problem of tests to one of these time slots. This obviously simplifies the model and reduces the number of integer variables. This method is not applicable to our case as the values of our due dates are widely spread over the whole time scale. Moreover, the model will become rather complicated, if we want to consider other temporal constraints like release dates and availability times.

Bartels and Zimmermann [2] used time-indexed decision variables to formulate a MILP model which can directly minimize the number of built prototypes. As only small size problems can be solved optimally, they further developed an heuristic approach to cope with real-life problems. Tests are successively scheduled according to the given priority while the number of prototypes only increases when there is still a test which cannot be allocated on the existing prototypes.

3 Problem Description and Notations

After informally explaining the problem in the Section 1, we now introduce the notations used in the rest of the paper. These notations are based on Pinedo [7] whenever possible.

V , I , and J are the sets of prototype variants, prototypes, and tests, respectively, with $|V| = l$,

$|I| = m$, and $|J| = n$. Each prototype $i \in I$ corresponds to variant $v_i \in V$. $M_j \subseteq V$ is the set of prototype variants that can perform test $j \in J$.

Each test $j \in J$ has a processing time p_j , a release date r_j , and a due date d_j . As all tests are executed without interruption, the completion time C_j of test j must obey $r_j + p_j \leq C_j \leq d_j$. Further, we consider the following relations between two different tests $j, k \in J$:

- $j \prec k$ iff test j must be completed before test k is started.
- $j \sim k$ iff tests j and k must be executed on the same prototype.
- $j \succ k$ iff tests j and k must not be executed on the same prototype.

Finally, $J_{Last} \subseteq J$ is the set of tests that must not be followed by any other test on the same prototype.

Further, we must formulate the availability time of a prototype. As already mentioned, we use a simple model for this purpose. This model considers manufacturing time of a prototype, the set-up time of a prototype variant and a potential delivery delay of a component of this prototype. In general, prototype $i \in I$ is not available before time a_i which is independent of v_i . An additional time s_v is required to set-up variant $v \in V$. Occasionally, there may be a delay for the delivery of a component such that this delay cannot be compensated within prototype manufacturing. In this case, we assume that the delay time y_v of prototype variant v dominates the manufacturing of the prototype and includes the integration of this component into the already existing prototype. Therefore, a test cannot be executed on prototype i before time $\max\{a_i + s_{v_i}, y_{v_i}\}$. Finally, the objective of this scheduling problem is to minimize the makespan $C_{max} = \max_{j \in J} \{C_j\}$.

To employ the common notation of scheduling problems, we consider prototypes and tests as machines and jobs, respectively. Using the standard scheduling framework, this problem can be classified as an extension of scheduling identical machines in parallel with machine eligibility constraints since a job can be performed with constant processing time on any machine (prototype) which has suitable characteristics (variant). Therefore, the notation $P_m|r_j, d_j, M_j, prec|C_{max}$ represents our problem. However, we must be aware that this notation does not include all facets of the problem like the selection of a variant for each machine (prototype). This selection determines the machine eligibility restrictions, M_j . Further, we must consider application-specific conditions like performing two tests on the same prototype ($j \sim k$). The problem is NP-complete in the strong sense, as it is a generalization of $P_m|prec|C_{max}$, see Pinedo [7].

4 CP Formulation

It is quite common in CP to define a resource constraint for a single machine in a general scheduling problem by using the term **cumulative**($\vec{t}, \vec{p}, \vec{c}, C$), where $\vec{t} = [t_1, \dots, t_n]$, $\vec{p} = [p_1, \dots, p_n]$, $\vec{c} = [c_1, \dots, c_n]$ are the vectors of starting time, processing time, consumption rate of jobs, respectively, and C is the capacity of the machine. The constraint is satisfied if the condition $\sum_{j \in J_t} c_j \leq C$ holds for all time instances t in the valid time frame, where $J_t = \{j \in J | t_j \leq t \leq t_j + p_j\}$ is the set of tasks that are in process at time t . Therefore, the total consumption of all jobs $j \in J_t$ does not exceed the capacity C .

In this model, we consider each prototype i as an unary resource with capacity $C = 1$ as it can perform at most one test at a time. Its corresponding variant is represented by variable $v_i \in V \forall i \in I$. There are also other possibilities to model the resources of this problem. For instance, we may consider a group of prototypes with the same variant as a single cumulative resource with

unknown capacity (the number of prototypes). However, this approach requires CP solvers which can directly define the capacity of a cumulative resource by using a variable.

Further, we have $c_j = 1, \forall j \in J$ as each test requires a single prototype. The starting time t_j of a test j must be in the interval $[r_j, \dots, d_j - p_j]$ to obey the release date and due date constraints. Finally, $x_j \in I, \forall j \in J$ represents the allocation of test j to a prototype. This leads to the following CP formulation:

Minimize C_{max}

subject to

$$C_{max} \geq t_j + p_j, \quad \forall j \in J \quad (1)$$

$$t_j \geq r_j, \quad \forall j \in J \quad (2)$$

$$t_j + p_j \leq d_j, \quad \forall j \in J \quad (3)$$

$$\text{cumulative}(t_j | x_j = i, p_j | x_j = i, c_j = 1 | x_j = i, 1), \forall i \in I \quad (4)$$

$$v_i \notin M_j \Rightarrow x_j \neq i, \quad \forall i \in I, \forall j \in J \quad (5)$$

$$x_j = i \Rightarrow t_j \geq a_i + s_{v_i}, \quad \forall i \in I, \forall j \in J \quad (6)$$

$$x_j = i \Rightarrow t_j \geq y_{v_i}, \quad \forall i \in I, \forall j \in J \quad (7)$$

$$t_j + p_j \leq t_k, \quad \forall j \prec k, j, k \in J \quad (8)$$

$$x_j = x_k, \quad \forall j \sim k, j, k \in J \quad (9)$$

$$x_j \neq x_k, \quad \forall j \succ k, j, k \in J \quad (10)$$

$$x_j = x_k \Rightarrow t_j \geq t_k + p_k, \quad \forall j \in J_{Last}, \forall k \in J, j \neq k. \quad (11)$$

Constraint (1) ensures that the makespan is not smaller than the completion time of any test. Constraint (2) and Constraint (3) state that tests must be performed between their respective release and due dates. Constraint (4) is a resource constraint with the term $(t_j | x_j = i)$ denoting the tuple of starting times for tests that are assigned to prototype i . The constraint prohibits the concurrent execution of two jobs on the same machine. Constraint (5) prevents that a test is assigned to a prototype whose variant does not belong to the eligibility set of the test.

Constraint (6) and Constraint (7) ensure that a test on machine i can neither start before the availability of the prototype according to our model. Remember that both set-up time s_{v_i} and component available time y_{v_i} depend on a value of variable v_i , which is unknown before optimization. It is another benefit of the CP approach that variables can index parameter arrays.

Precedence constraints are represented in Constraint (8). Constraint (9) makes sure that any pair of tests (j, k) with $j \sim k$ will be executed on the same machine. Further, tests j and k with $j \succ k$ must be processed on different machines which is achieved by Constraint (10). Finally, Constraint (11) ensures that test $j \in J_{Last}$ will be the last job executed on the machine to which it is allocated. Some of the constraints are very specific to our problem. But due to its flexibility, the CP model can be adapted rather easily to problems using a similar parallel machine environment.

5 Computational Results

To demonstrate our approach, we apply the CP model together with data obtained from a real-life test scenario. There are four data sets of different sizes from about 40 tests to almost 500 tests. Table 1 shows other details of those data sets, like the number of variants and the number of tests

with precedence constraints. It can be seen that up to 10% of all tests may be subject to special constraints. In the large data set involving several hundred tests, the impact of those constraints may be considerable. Therefore, it is difficult to later verify and correct a schedule that has been obtained by neglecting these requirements.

Table 1: Detailed information about the given data sets.

Data	n	l	$ \{j \in J j \prec k\} $	$ \{j, k \in J j \sim k\} $	$ \{j, k \in J j \succ k\} $	$ J_{Last} $
1	41	38	1	2	2	1
2	100	3	1	2	2	-
3	231	9	3	6	6	3
4	486	663	1	54	2	1

We use OPL Studio 3.7 [5] to formulate and solve the CP model introduced in Section 4. In addition to the standard CP functions, OPL also provides a specific scheduler module that includes several functions for solving scheduling problems. Further, it has good constraint propagation techniques and efficient searching algorithms. The numbers of variables and constraints presented in Table 2 correspond to the model formulated in OPL. However, these numbers are not directly related to the difficulty of the optimization problem. We only provide them to give an impression of the problem size and their variations during the calculations. All computations were performed on a Pentium IV, 3.0 GHz, and 1 GB main memory.

Table 2 shows the computational results. For each data set and a given number of prototypes, we minimize the makespan. Initially, we may set m to a large value which means we try to construct as many prototypes as time allows in order to accommodate all tests. In some cases, this procedure does not lead to a feasible solution. For instance, prototypes may be available too late if there are many tests with very early due dates. However, it can suggest the conflicts should arise from temporal constraints rather than from a limited number of prototypes.

After obtained the results, we reduce the number of prototypes step-by-step to see how it affects to the makespan. This procedure is repeated until it becomes infeasible to find a solution or the calculation cannot terminate within a given time limit. We specify this time limit to be 1 hour for Data 1–3 and 6 hours for Data 4. For example in Data 1, we start from $m = 14$ and obtain a makespan of 330 days. If we restrict ourselves to only 5 prototypes we still get the same optimal makespan. However, if we set $m = 4$ the problem becomes infeasible as it has not enough prototypes to execute all tests with respect to all constraints.

Provided there is enough time, the solver either achieves an optimal solution for this number of prototypes or proves that the problem is infeasible. But the required computation time grows quickly when the problem size becomes larger. Due to limitations of computation time in practice, the solver may not be able to prove the infeasibility of the problem. Even if it finds a feasible solution we cannot be sure about its optimality as long as there are still unexplored nodes in the search tree. The results show that for all data sets, the optimal solution can be achieved when the number of given prototype is large enough. If the number of available prototypes decreases it is getting more difficult to find any feasible solution or prove optimality, although the numbers of variables and constraints decrease. However, if the number of prototypes is sufficiently reduced the solver can again prove infeasibility within the limit time because the search space has shrunk substantially. For instance, the calculation for Data 3 shows that the optimal solution can be obtained when $m = 20$. For $m = 18$ or $m = 19$, feasible solutions with the slight increase of the makespan are found after one hour of computation time. But, it has an unknown gap in the range

of $9 \leq m \leq 17$ prototypes. We cannot determine whether the problem can be solved or not for a number of this set.

Further, it should be noted that an optimal value of the makespan in step m can be regarded as a lower bound of the makespan in the next step $m' = m - 1$. This may reduce the search space as constraints are tighter. However, this idea does not work effectively here since in each case where an optimal solution is found the makespan is actually determined by the earliest completion time ($r_j + p_j$) of a test with a very long processing time. Normally during constraint propagation, a lower bound of the makespan is already constrained to be greater than or equal to the earliest completion times of all jobs. Therefore, it does not help CP when we try to include the same condition.

Table 2: Minimizing the makespan for the given number of prototypes.

Data	m	#Variables	#Constraints	Total Time (sec)	Makespan (day)
1	4	314	517	0.04	Infeasible
1	5	361	605	0.09	330 ^a
1	14	784	1,397	0.18	330 ^a
2	3	621	722	3.02	Infeasible
2	4	727	829	_b	_b
2	5	833	936	0.19	312 ^a
2	34	3,907	4,039	1.14	312 ^a
3	8	2,592	8,383	1,006.35	Infeasible
3	9	2,829	9,315	_b	_b
3	17	4,725	16,771	_b	_b
3	18	4,962	17,703	10.26 ^c	381
3	19	5,199	18,635	13.45 ^c	379
3	20	5,436	19,567	16.48	376 ^a
3	77	18,945	72,691	91.87	376 ^a
4	111	56,214	114,392	_b	_b
4	112	56,707	115,405	261.88 ^c	517
4	113	57,200	116,418	319.57 ^c	501
4	118	59,665	121,483	471.55 ^c	481
4	123	62,130	126,548	832.21 ^c	413
4	128	64,595	131,613	13,917.50	381 ^a
4	162	81,357	166,055	21,430.75	381 ^a

^a Optimal solution

^b No result within the time limit

^c Computation time of the achieved solution

6 Conclusion

We have presented a scheduling problem from automobile industry where several hundreds of tests must be performed on vehicle prototypes. We have used the CP method to formulate the problem with the objective to minimize the makespan and applied it to solve real data sets. CP allows us to naturally express requirements like resource constraints and machine eligibility sets and then to efficiently solve the problem.

We are able to obtain good solutions for our scheduling problem within a reasonable computation time. Either the optimal or good feasible solutions are found even for the largest size problem which comprises about five hundreds tests and one hundred prototypes. The optimal makespan solution

can always be achieved when sufficient prototypes are available. However, it is more difficult to find an optimal solution if the number of machines is reduced. But in most cases, the solver manages to obtain good feasible solutions with the slight increase of makespan. If the number of given prototype is too low then we can prove that the problem becomes infeasible.

It is still quite hard and takes a very long time to exactly determine the minimum number of required prototypes. This is especially true for the data sets comprising several hundreds tests. In future, we are looking to determine better lower bound values. This may help a user to decide whether it is worth trying to further reduce the number of prototypes. Moreover, we will implement a more specific search strategy for this problem in order to improve the efficiency of constraint programming. In particular, it is interesting how we can use solutions obtained from solving previous iterations in finding an optimal or at least better feasible solution.

References

- [1] P. Baptiste, C.L. Pape and W. Nuijten (2001), *Constraint-Based Scheduling: applying constraint programming to scheduling problems*, Kluwer.
- [2] J.-H. Bartels and J. Zimmermann (2005), Scheduling Tests in Automotive R&D Projects, *Operations Research Proceedings 2005*, Springer, volume 11, 661 – 666.
- [3] E. Danna and C.L. Pape (2004), *Constraint and Integer Programming: Toward a Unified Methodology*, chapter Two generic schemes for efficient and robust cooperative algorithms, 33 – 58, Kluwer.
- [4] J. Hooker (2005), A Hybrid Method for the Planning and Scheduling, *Constraints* **10**(4), 385 – 401.
- [5] ILOG S.A. (2003), *ILOG OPL Studio 3.7 Language Manual*.
- [6] J. Lockledge, D. Mihailidis, J. Sidelko, and K. Chelst (2002), Prototype fleet optimization model, *Journal of the Operational Research Society* **53**, 833 – 841.
- [7] M. Pinedo (2002), *Scheduling-Theory, Algorithm and Systems*, Prentice Hall, 2nd edition.
- [8] R. Sadykov and L. Wolsey (2006), Integer programming and constraint programming in solving a multi-machine assignment scheduling problem with deadlines and release dates, *INFORMS Journal on Computing* **18**, 209 – 217.
- [9] R. Scheffermann, U. Clausen, and A. Preusser (2005), Test-scheduling on vehicle prototypes in the automotive industry, In *Proceedings of Sixth Asia-Pacific Industrial Engineering and Management Systems (APIEMS)*, volume 11, 1817 – 1830, Manila.
- [10] A. Zakarian (2000), Performance analysis of product validation and test plans, Annual Progress Report, Center for Engineering Education and Practice, University of Michigan-Dearborn.