Submitted to *Interfaces*
manuscript (Please, provide the mansucript number!)

Scheduling Crash Tests at Ford Motor Company

Daniel Reich
Research and Advanced Engineering, Ford Motor Company, Dearborn, Michigan, {dreich8@ford.com}

Yuhui Shi, Marina Epelman, Amy Cohn
Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan

Ellen Barnes, Kirk Arthurs
Product Development Safety, Ford Motor Company, Dearborn, Michigan

Erica Klampfl
Research and Advanced Engineering, Ford Motor Company, Dearborn, Michigan

This paper considers the problem of scheduling crash tests for new vehicle programs at Ford. We describe the development of a comprehensive web-based system that automates time-consuming scheduling analyses through mathematical optimization, while also institutionalizing expert knowledge about the engineering complexities of crash testing. The system enables engineers and managers to specify and consider multiple scheduling scenarios by relying on efficient interfaces for specifying problem instances and the underlying optimization model and solution algorithms.

Key words: Automotive, Scheduling, Decision Support System, Integer Programming
History: Semi-finalist paper for the 2015 Daniel H. Wagner Prize for Excellence in Operations Research Practice

---

Ford has performed over 20,000 crash tests since 1954, with a steep increase in recent years due to more product launches and new or additional tests required by safety regulations. These factors greatly increase the number of crash tests required each year and complexity of planning and scheduling. Prototypes built for product development testing can cost in excess of $200K each, because many of the parts and the full-vehicle prototypes are hand-made and highly customized. Commonly, a vehicle program corresponding to development for a product launch requires over 50 vehicles just for crash testing, so maximum utilization is critical for balancing engineering resources and program timing.

Maximizing utilization is not a simple matter. Crashes are destructive, and only certain tests can be combined together on the same prototype vehicle. Program milestones and

staggered prototype vehicle delivery form a difficult timing problem. The different configurations of vehicles offered (e.g., types of powertrain, trim, body-style) are an additional source of complexity, because a comprehensive testing plan matches crash tests to specific configurations.

Until recently, crash test plans were developed manually using pen and paper and Excel spreadsheets. This process was tedious, and constructing a test plan could take several days or weeks. The schedule produced was highly dependent on the knowledge and experience of the individual planning engineer. Determining if the crash schedule was optimal in terms of the number of vehicles needed was nearly impossible. Reacting to delays and program changes required extensive rework of the test plan.

We developed a completely custom-made crash test scheduling system that transformed a labor-intensive process relying on high levels of expertise, to one that automates time-consuming scheduling analyses through mathematical optimization, while also institutionalizing expert knowledge. Instead of a more traditional assignment-based modeling approach, we developed an integer programming model using composite variables representing sequences of tests to be performed on a single prototype vehicle. We are the first to apply this modeling technique to problems that combine the features of both bin packing and complex sequential job scheduling (in the literature to the best of our knowledge). We also describe a column-generation algorithm for solving our formulation, with a pricing problem structured specifically for crash test scheduling.

Our system produces optimized schedules in seconds or minutes (depending on program complexity and size). Engineers can use the time saved to run what-if scenarios including double-shifting prep time, working weekends and/or holidays, and introducing flexibility for vehicle specifications. This gives planning engineers and program management concrete data on resource requirements and potential areas of flexibility.

A key to delivering our technology and maximizing its benefits was designing the system for ease of use with engineers' current working document formats. We developed interfaces that ensure data quality and completeness. We designed our optimization package to run on Ford's High Performance Computing Center servers, while integrating it into a user-friendly web application with a backend database. In the following sections, we introduce crash test specifications, discuss the complexities associated with scheduling them, and present our web application along with the underlying optimization model and solution algorithms.

Crash Test Scheduling Problem Description

In its essence, safety test scheduling for a vehicle program involves deciding which prototype vehicles will be used for which of the crash tests, and when each test is going to be performed. A high-quality schedule would achieve high utilization of prototypes, while observing test specifications and rules for combining multiple tests on the same prototype, and following timing targets for completion of each test.

To illustrate the complexities involved in defining and solving each test scheduling instance, we begin with an example of (a portion of) a typical schedule generated for a vehicle program at Ford.

The actual execution of a crash test takes only seconds; however, preparing the prototype vehicle may take days or even weeks, depending on the readiness of the prototype when it is delivered to the safety organization. Initial preparation work, such as changing or adding parts, is carried out at one facility. Typically, a few days before a crash is executed, the prototype is transferred to the crash barrier location, where it is then instrumented with sensors, crash dummies, ballasts and any other required components.

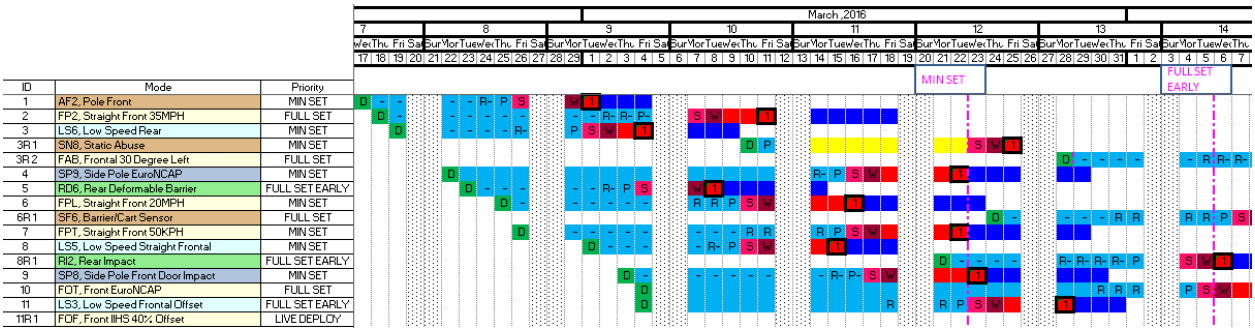


Figure 1 Gantt chart with a color-coded crash schedule.

Figure 1 shows a small example crash test schedule Gantt chart. Each row contains a single crash test with detailed timing information represented by color-coding. The day each prototype is delivered for the test is marked with a green "D". The prototype is then prepared during the days marked in light blue. The majority of tests are crashed at the barrier, in which case the vehicle may require signoff (red "S"), walk-around (red "W"), turnaround time in queue (solid red), the actual crash (red "1") and post-crash analysis time (dark blue). (A few "abuse"-type tests do not require crashing the vehicle into a barrier and have a simpler structure.)

In addition to the timing of each test, the Gantt chart in Figure 1 provides information on the utilization of each prototype: the first column contains prototype vehicle number, with "R1" appended to represent the first rehit (second crash), "R2" — to represent the second rehit (third crash), etc. For example, prototype number 8, which is scheduled to be delivered to the safety department on March 1st, will first be used for a low speed front collision test, followed by a higher speed rear impact test.

While the Gantt chart provides fine detail for executing a crash plan, it is less well-suited for viewing the plan's efficiency at a high level. The average number of crashes per vehicle, known as the rehit ratio, is a key metric of a schedule. The crash test sequence document, shown in Figure 2, is designed to highlight vehicle utilization and the associated rehit ratio. Each line represents a vehicle, with its sequence of crashes listed across the columns. Yet another method for summarizing assignments of tests to prototypes is the so-called "crash tree" illustrated in Figure 3. Here, the rows of the table correspond to tests, the columns — to specifications of prototypes used for each test, and the notations in the table indicate test-to-prototype assignment along with position of each test in the rehit sequence. Safety engineers would construct the coarse plan as shown in the crash sequence document and the crash tree, before starting on the finer timing details included in the Gantt chart.

Vehicle ID	Delivery Date	Driveline	Engine	Driver/Fuel Filler	Test 1	Test 2	Test 3
1	2016-02-17	4x4	Diesel	LEFT/LEFT	Pole Front [AF2]		
2	2016-02-18	4x4	Any	LEFT/LEFT	Straight Front 35MPH [FP2]		
3	2016-02-19	4x2	Gas	LEFT/LEFT	Low Speed Rear [LS6]	Static Abuse [SN8]	Frontal 30 Degree Left [FAB]
4	2016-02-22	4x4	Gas	LEFT/LEFT	Side Pole EuroNCAP [SP9]		
5	2016-02-23	4x4	Diesel	LEFT/LEFT	Rear Deformable Barrier [RD6]		
6	2016-02-25	4x2	Diesel	LEFT/LEFT	Straight Front 20MPH [FPL]	Barrier/Cart Sensor [SF6]	
7	2016-02-26	4x4	Gas	LEFT/LEFT	Straight Front 50KPH [FPT]		
8	2016-03-01	4x4	Gas	LEFT/LEFT	Low Speed Straight Frontal [LS5]	Rear Impact [RI2]	
9	2016-03-03	4x4	Any	LEFT/LEFT	Side Pole Front Door Impact [SP8]		
10	2016-03-04	4x2	Diesel	LEFT/LEFT	Front EuroNCAP [FOT]		
11	2016-03-04	4x4	Gas	LEFT/LEFT	Low Speed Frontal Offset [LS3]	Front IIHS 40% Offset [FOF]	

Figure 2 **Crash sequence document provides a compact summary of rehit strategies, highlighting crash test combinations on vehicles.**

Until recently, safety engineers constructed all three documents manually (including the detailed timing information in the Gantt chart above), using a standard Excel template. As we discuss in the next section, we have designed the database of our web-based support system to capture all the timing information associated with each test, allowing for automatic generation of the Gantt chart once the test assignment and rehit sequencing decisions have been made.

		Gas 4x2 L/L	Gas 4x4 L/L	Diesel 4x2 L/L	Diesel 4x4 L/L	4x4 L/L
Front Barrier	FPL,Straight Front 20MPH			6 (6R)		
	FPT,Straight Front 50KPH		7			
	FP2,Straight Front 35MPH					2
	FAB,Frontal 30 Degree Left	3RR				
	FOF,Front IIHS 40% Offset		11R			
	FOT,Front EuroNCAP			10		
Sensor	SF6,Barrier/Cart Sensor			6R		
	SN8,Static Abuse	3R (3RR)				
	AF2,Pole Front				1	
Low Speed	LS3,Low Speed Frontal Offset		11 (11R)			
	LS6,Low Speed Rear	3 (3R)				
	LS5,Low Speed Straight Frontal		8 (8R)			
Rear	RI2,Rear Impact		8R			
	RD6,Rear Deformable Barrier				5	
Side	SP8,Side Pole Front Door Impact					9
	SP9,Side Pole EuroNCAP		4			

Figure 3 **The "crash tree" specifies assignment of tests to specific prototype vehicles, along with the relevant specifications of the prototypes**

Engineering expertise about expected physical outcomes of crashes is needed to create test schedules that use rehits to increase vehicle utilization. Indeed, some pairs of tests are incompatible, i.e., cannot be performed on the same vehicle in a particular order, for one of the following reasons:

- If scheduled together on a particular prototype vehicle, the tests cannot both meet their timing targets. This may be the result of the duration of the test or may be caused by the late scheduled delivery of the vehicle.
- Tests require different vehicle specifications, e.g., one requires a model with 2 wheel drive and the other requires a 4 wheel drive model.
- The damage from the first crash test leaves the prototype vehicle with insufficient structural integrity to conduct the second crash test.

The first source of incompatibility can be detected based on the information about test duration and prototypes delivery schedule, while identifying the second and especially the third one requires a high level of engineering expertise. In the next section, we describe the web-based application we developed at Ford for specifying and solving test scheduling problem instances that captures this knowledge systematically.

The Test Planning Scheduler Support System (TP3S)

In this section we describe the Test Planning Scheduler Support System (TP3S) — a web-based application we developed for use by Ford engineers for crash test planning. This system has multiple benefits. It allows for systematic collection of information about crash test modes, timing, and rehit compatibility into a centralized database, institutionalizing expert knowledge. It also allows engineers to explore bottlenecks and potential improvements in test schedules by quickly specifying various testing scenarios. An optimal or near-optimal schedule for each scenario is generated in seconds or minutes by a custom-built optimization engine we have developed.

Test Management

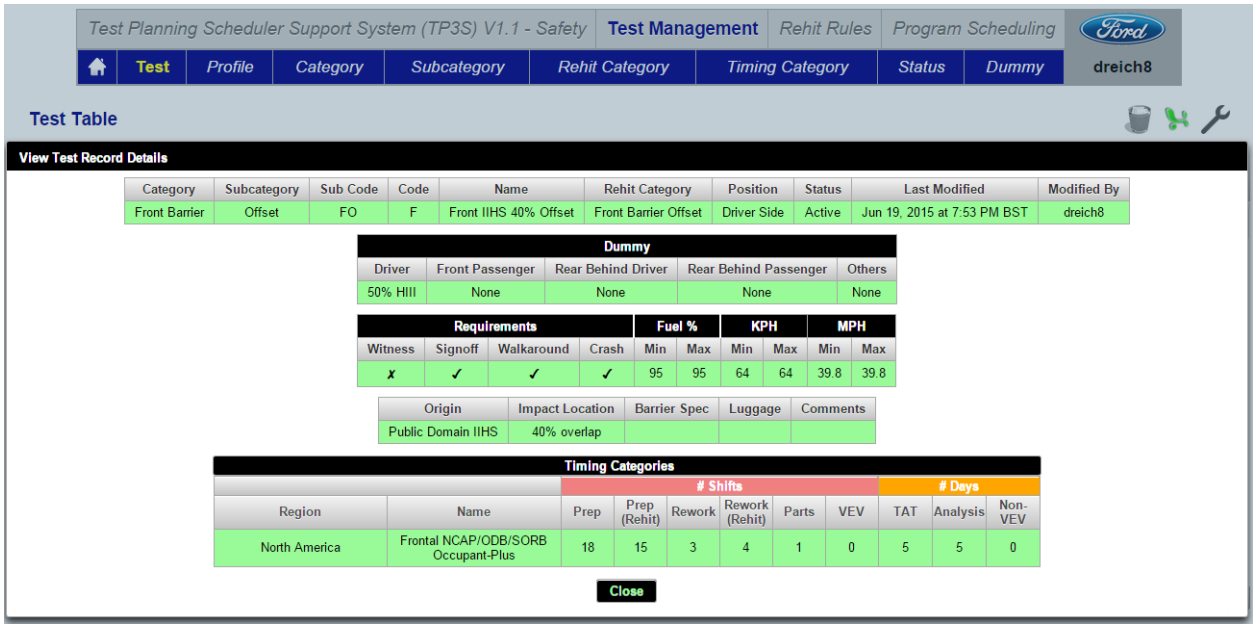


Figure 4 **Example of a crash test mode record from the TP3S web application, showing the requirements for an Insurance Institute for Highway Safety test.**

The first tab of the TP3S web interface is dedicated to test management, i.e., specifying and displaying information about each relevant crash test mode. For example, Figure 4 shows the application screen with requirements for an Insurance Institute for Highway Safety (IIHS) 40% front overlap test, which include a driver side crash impact position, a driver side HIII dummy sized to a 50th percentile male, an execution speed of 64 kilometers per hour (KPH), and the standard timing for preparing, executing and analyzing the test. To ensure consistency of the test data, our application includes screens to maintain and

view relational database information on categorizations, dummy types, and other aspects of the test, which are navigable through the blue menu bar.

Note that some timing information is expressed in days, while shifts are used for aspects of test timing that have flexibility to speed up execution by running multiple shifts per day. The impact of this flexibility can be explored in the scheduling process by considering what-if scenarios. Note also that the time allocated for a crash test may differ depending on whether it is the first test on the vehicle or not (i.e., a rehit). In particular, it is common for rehits to require less initial prep work, because some of the prep was already completed for a prior crash test; however, rework time may increase for repairs needed to restore the vehicle to prime condition.

Through the use of the TP3S application, we have built a database that currently contains over 100 unique crash test modes, a subset of which are run on each vehicle program.

Rehit Rules

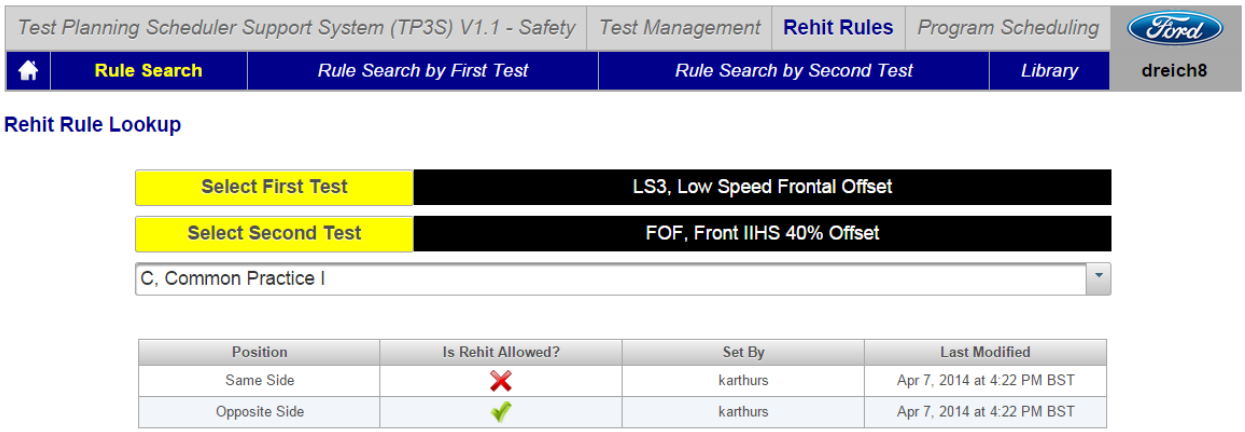


Figure 5 **Rehit rules lookup shows whether two crash tests can be run on the same prototype vehicle in the specified order. For tests that are off-center, rules are based on whether the tests are to be executed on the same or opposite sides of the prototype.**

Due to its high speed of 64 KPH, an IIHS 40% front overlap test damages a prototype vehicle severely enough so that it cannot generally be reused for other testing afterwards. However, it may be possible to conduct a less destructive crash prior to the IIHS test on the same prototype, if time permits and required specifications of the prototype are similar. The second tab of the TP3S application serves to display information on whether scheduling two crashes back-to-back on the same prototype is allowed, based on the rules

provided by a core group of safety engineers. Figure 5 shows an example: here, we can schedule a low speed front offset test, which runs at under 10 MPH, prior to the IIHS test, so long as it is done on the opposite (passenger) side of the vehicle. This combination also appears on prototype 11 in Figures 1, 2 and 3. When setting these rehit rules, engineers consider the location and extent of the damage expected from the first crash test conducted. If the damage is expected to be either insignificant or easily repairable, the combination is generally permitted; if not, the combination is prohibited.

In addition to the two crash test mode selections in Figure 5, a third dropdown allows the user to select a rehit rules library, which is designed for a specific vehicle platform and conservativeness measure. The vehicle platform is an important identifier because the same crash test mode causes different levels of damage on different sized vehicles, e.g., a Focus compact car versus an Explorer SUV. The conservativeness measure ("common practice" in the example shown) allows the engineer to take a more or less aggressive approach, with the understanding that more aggressive approaches may require additional resources later on. Specifically, if damage from an initial crash exceeds expectations, the rehit crash scheduled on the same vehicle may require an additional unbudgeted vehicle or additional parts may be required for repairs.

With over 100 unique test modes, the application currently contains over 10,000 rehit rules per vehicle platform and conservativeness measure. We developed a system for grouping test modes into crash categories to reduce the data entry burden on our core safety team. Categories capture position and impact type, so that the main distinguishing characteristic between test modes in a category is crash speed. Using this model allows core safety team members to set rules based on speed, e.g., any crash test mode in the "front barrier offset" category can precede any crash test mode in the "side pole" category, so long as the former test mode is run under a specified speed cutoff and the tests are conducted on opposite sides of the vehicle. A lower cutoff speed may be set for same side combinations. (If the cutoff is set to 0, no tests from these categories can be combined, and if the cutoff is set high enough, all pairings are allowed.)

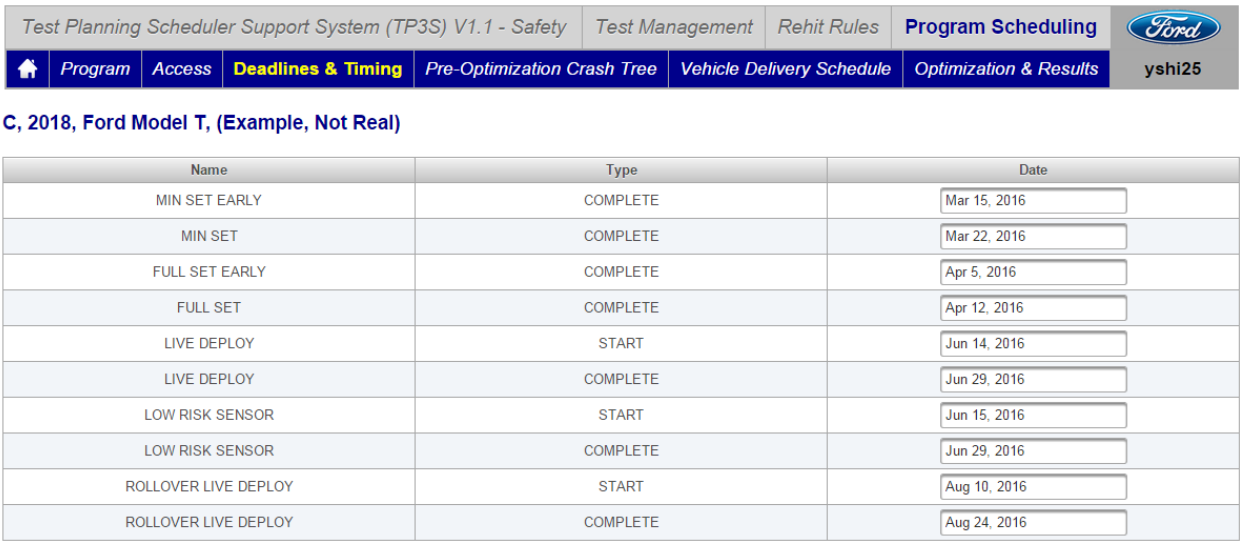
The number of crash categories is currently around 1/3 of the number of test modes, reducing the data entry burden by an order of magnitude. We further reduced this burden by designing the user interface to enable setting multiple rehit rules simultaneously, after selecting the category of the first test. We also provide deep-copy functionality that

replicates an entire rehit rules library, which can then be edited to increase or decrease the level of conservativeness with minimal effort.

The application contains additional screens, navigable through the blue menu bar in Figure 5, where engineers can view all tests allowed before or after a specified test for a selected vehicle platform and conservativeness measure.

Program Scheduling

The third tab of the TP3S application is dedicated to program scheduling, i.e., specification and scheduling of crash tests for each vehicle program (e.g., 2016 Ford Explorer). The primary scheduling responsibility for each program is assigned to a lead engineer in the safety department. This lead engineer is tasked with forming the initial plan well in advance of its execution to place orders for parts with long lead times. As the testing commencement date nears, the plans are typically updated several times to account for delays in part deliveries and changes in engineering designs.



Name	Type	Date
MIN SET EARLY	COMPLETE	Mar 15, 2016
MIN SET	COMPLETE	Mar 22, 2016
FULL SET EARLY	COMPLETE	Apr 5, 2016
FULL SET	COMPLETE	Apr 12, 2016
LIVE DEPLOY	START	Jun 14, 2016
LIVE DEPLOY	COMPLETE	Jun 29, 2016
LOW RISK SENSOR	START	Jun 15, 2016
LOW RISK SENSOR	COMPLETE	Jun 29, 2016
ROLLOVER LIVE DEPLOY	START	Aug 10, 2016
ROLLOVER LIVE DEPLOY	COMPLETE	Aug 24, 2016

Figure 6 Scheduling milestones for a program

The engineer's first step is to set dates for standard safety milestones, shown in Figure 6. Some milestones have only a target completion date, whereas others may also include a date before which tests cannot be executed. We designed the scheduling module to allow plans to be easily updated for program timing changes: the milestones listed are connected to tests, so that if dates associated with those milestones are updated, the new timing is automatically associated with all connected tests.

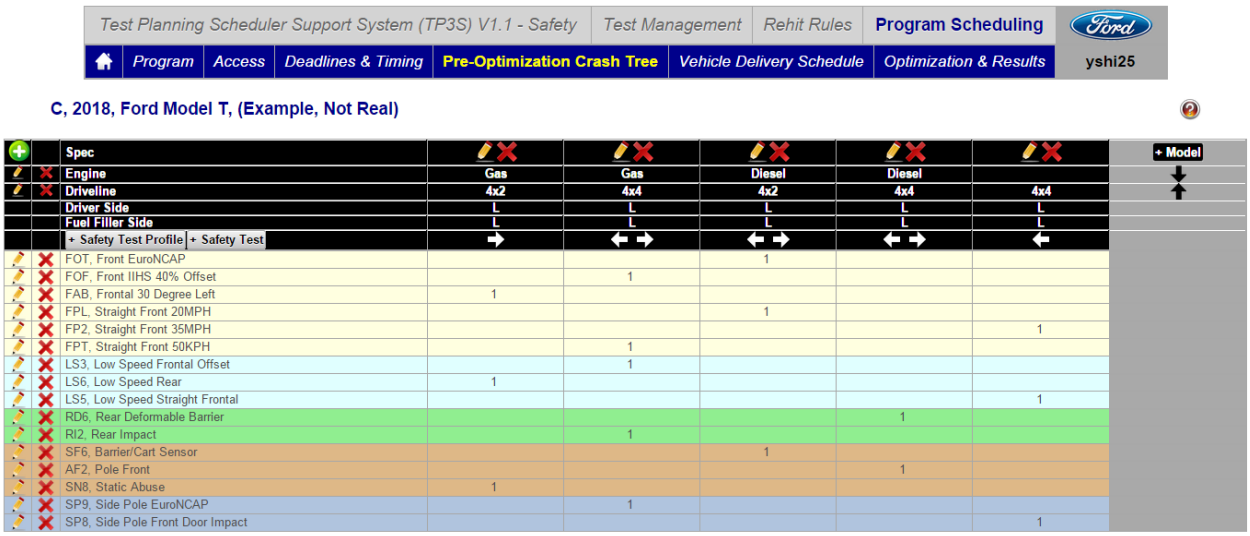


Figure 7 **Engineer enters test requirements, by connecting vehicle specifications to test modes and milestones.**

The engineer's next step is to connect test models and milestones with prototype vehicle specifications using the screen shown in 7. The engineer first constructs a list of vehicle specifications (e.g., engine type, driveline type, driver side location, and other vehicle characteristics relevant for some or all of the test modes) and lists available options for each specification (gas or diesel engine, 4x2 or 4x4 driveline, etc.). The engineer then constructs control models, which are combinations of those options, as columns. The required crash test modes are then added to the plan as rows; for convenience, predefined lists of crash modes, e.g., for a European car or a North America truck, can be imported and then modified by deleting or adding specific test modes. The engineer connects a test mode to a control model by clicking a cell within the matrix, which opens a dialog to select an associated timing milestone. For example, in Figure 7, the 1's in each column indicate required tests that need to be performed on the corresponding control model; note that the tests indicated in the last column can be performed on a vehicle with either engine type.

The prototype vehicles are used by all departments (safety, powertrain, electrical, etc.) within the Product Development organization. Thus, the lead safety engineer coordinates test schedules with a planner from Ford's centralized test planning group that oversees the distribution and sharing of prototypes. Based on these interactions, the engineer develops a schedule for expected deliveries of prototype vehicles to the safety department for crash testing. This schedule is input, tracked and updated using the screen shown in Figure 8.

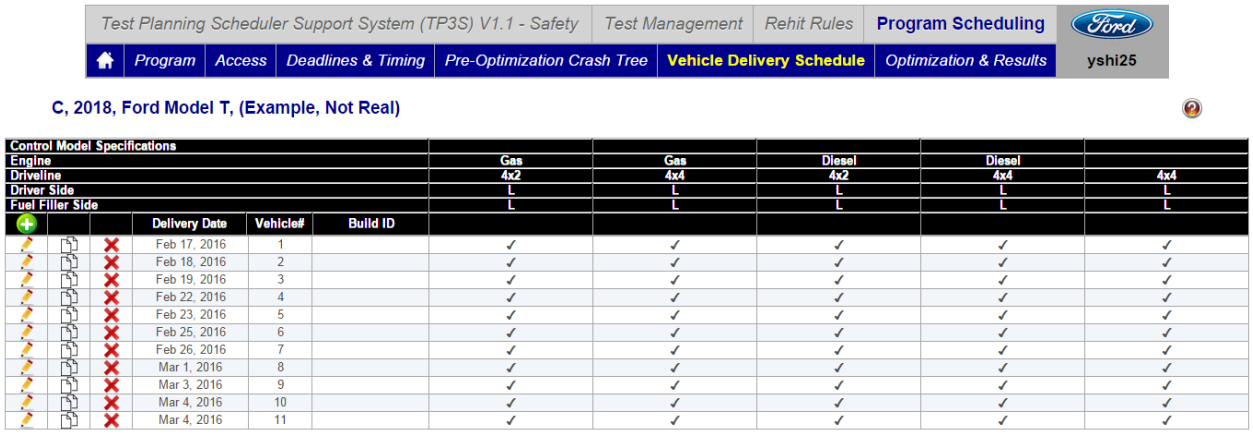


Figure 8 **The prototype vehicle delivery schedule is shown, along with the possible control models that could be delivered each date.**

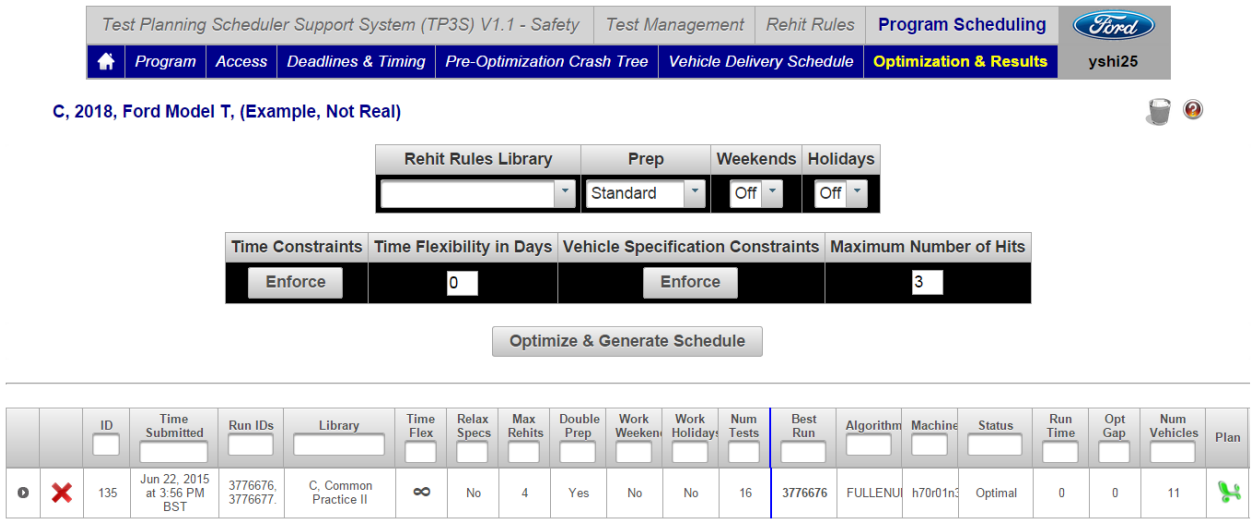


Figure 9 **Run optimizations and view results.**

When envisioning a decision support system for scheduling, one of our foremost goals was to enable what-if scenario analysis through which lead engineers could identify the main program constraints. For example, running a scenario with a couple of weeks of additional time would allow engineers to examine whether timing constrains vehicle utilization. Running a scenario with relaxed vehicle specifications, such as engine selection, would allow engineers to quantify the relationship between engineering design and resources required for testing.

Figure 9 shows the application screen for running optimizations, with options for what-if scenarios. When an optimization job is submitted, the application connects to Ford's

High Performance Computing Center and passes a single database ID, which allows all the information associated with the vehicle program to be retrieved by our executable Java scheduling program. This design of decoupling the web interface used for transactional data functions and the scheduling optimization program allows engineers to run many scenarios in parallel. When the scheduling program terminates, the results are stored in the database and are automatically retrieved by the web application. An Excel spreadsheet is downloadable from the screen in Figure 9 containing the scheduling documents from Figures 1, 2, and 3.

While a lead engineer is responsible for each program, plans are discussed and reviewed with fellow engineers, supervisors and managers. These reviews aim to ensure plans stay within budget, are executed on time, and are maximally efficient. By implementing several permissions levels in the application, we have allowed the lead engineer to construct and edit program information, grant read-only permission to fellow engineers, and have work viewable by management as a default.

Integer Programming Formulation of the Test Scheduling Problem

An important aspect of our TP3S application is the modeling and algorithmic techniques utilized "under the hood" to perform each test plan optimization program and scenario requested by engineers through the screen illustrated in Figure 9 . We first review the relevant literature for modeling and solving related optimization problems, then introduce our notation and integer linear optimization model, and discuss the algorithms we developed. Details are provided in the Appendix.

Related Research

Mathematically, the crash test scheduling problem can be viewed as a hybrid of the classic bin packing and parallel machine scheduling problems.

In the bin packing problem, items of different weights need to be packed into bins of limited capacities, and the minimum number of bins required is to be determined. There are extensive studies on this topic (Coffman Jr et al. 1996). A paper closely related to our research is Elhedhli et al. (2011). They consider a variation of the bin packing problem with conflicts between items, which means certain items cannot be packed into the same bin — a relationship similar to our rehit rules. They provide a set-partitioning formulation of the problem and propose a branch-and-price algorithm to solve it exactly, with the pricing problem formulated as a knapsack problem with conflicts.

In the parallel machine scheduling problem, time-sensitive tasks with given processing times need to be scheduled on a given set of machines, while minimizing a certain time-related additive criterion, such as make-span or penalty associated with exceeding timing targets. The literature on parallel machine scheduling has been developed over several decades and contains many useful models and algorithms. A comprehensive survey and comparison between different solution strategies can be found in two survey papers by Cheng and Sin (1990) and Potts and Strusevich (2009). The paper most relevant in our context is Chen and Powell (1999), in which the authors consider two different objectives: *total weighted completion time*, and *weighted number of tardy jobs*. The paper proposes a model in which each variable corresponds to a partial schedule on a machine, and a branch-and-price algorithm. However, they do not consider the conflict and precedence relations between different tasks, nor do they treat the number of machines as a decision.

In the context of crash test scheduling, there have been efforts to extend various theoretical methods and techniques to real-world applications. Chelst et al. (2001) use mathematical optimization to minimize the number of unique prototype vehicle configurations needed, given specification requirements of all individual tests. Bartels and Zimmermann (2009) consider building prototype vehicles as part of the decision and formulate the problem as a mixed integer programming model. However, their model can only deal with small problem instances; they propose heuristics to solve larger ones. Limtanyakul and Schwiegelshohn (2012) propose a hybrid model of using mixed integer programming (MILP) and Constraint Programming to schedule crash tests, where MILP is used as to estimate the demand for prototype vehicles and Constraint Programming — to find a complete feasible schedule; *no-good cuts* are generated to connect the two phases. However, these cuts are purely combinatorial and may cause slow convergence. Most recently, Reich et al. (2014) studied various analytical approaches, including both exact and heuristic methods, to solve the prototype vehicle scheduling problem. They discuss several implementation aspects specific to Ford's operation. However, their optimization methods are only capable of solving small-sized instances.

Our initial approach to the crash test scheduling problem was to model it using an assignment-type formulation, with binary variables used to indicate assignments of tests to prototype vehicles and sequencing of tests on each vehicle, and continuous variables reflecting starting times of test execution. Similar formulations are widely used and studied

in scheduling problems, and can be solved for instances of moderate size (Pinto and Grossmann 1995, Zhu and Heady 2000). However, our computational studies (as well as those by others) showed that for problem instances arising in our application, the formulation is inefficient and becomes intractable, even after addition of symmetry-breaking constraints and other model enhancements (see Reich et al. 2014). Our ultimate modeling approach, detailed below, utilized binary variables associated with *valid sequences* of tests; due to the excessive number of such variables in some problem instances, we use a delayed column generation algorithm in which the pricing problem is based on the earlier assignment-based formulation of the scheduling problem.

Problem Statement and Formulation

To formulate an integer linear optimization model for the test scheduling problem, we will use the following notation: Let $\mathcal{T}$ denote the set of all test modes specified for the vehicle program. Let $\mathbf{A} \in \Re^{|\mathcal{T}|\times|\mathcal{T}|}$ be the matrix that specifies precedence relationships among tests: its (t_1, t_2)th element $a_{t_1,t_2} = 1$ if test t_1 is allowed to be performed before test t_2 if they are performed on the same vehicle; $a_{t_1,t_2} = 0$ otherwise. (Matrix $\mathbf{A}$ is determined based on information provided in screens illustrated in Figures 5 and 7.) Let $r_t,\ p_t,\ d_t,$ $t \in \mathcal{T}$, denote the release time, processing time, and scheduling milestones of test $t \in \mathcal{T}$, respectively (see Figures 4 and 6). Finally, let $\mathcal{V}$ denote the set of prototype vehicles, with $q_v,\ v \in \mathcal{V}$ denoting the scheduled delivery time of vehicle $v \in \mathcal{V}$ (see Figure 8).

Formally, the Crash Test Scheduling Problem (CTSP) can be stated as follows: given the above inputs, determine the assignment of tests to prototype vehicles and starting time of each test, subject to conflict and precedence constraints captured in matrix $\mathbf{A}$ and timing constraints on test start dates and vehicle delivery times, so as to optimize vehicle utilization and adherence to timing targets.

As the straightforward assignment-based models for **CTSP** proved computationally inefficient, we use a set-partitioning formulation instead. Instead of dealing with individual tests, we consider valid sequences of tests (a sequence of tests is valid if all tests in the sequence can be performed in the specified order on a single prototype vehicle according to vehicle specification and rehit rules). Then, the problem is to identify a collection of sequences that cover all tests exactly once, i.e., partition the set $\mathcal{T}$. (See Chen and Powell (1999) for a discussion of the benefits of such formulations in parallel machine scheduling problems.)

Let Ω denote the set of all valid sequences, and let $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m \subset \mathcal{V}$ denote the groups of identical vehicles (i.e., those delivered on the same day and having identical specifications; if each vehicle is unique, $m = |\mathcal{V}|$). We have $\cup_{i=1}^{m} \mathcal{V}_i = \mathcal{V}$, and $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset, i \neq j$. Our formulation uses binary variables $\lambda_{\omega,i},\ \omega \in \Omega,\ i \in \{1, \ldots, m\}$, with $\lambda_{\omega,i} = 1$ indicating that the sequence of tests ω is assigned to a vehicle from group $\mathcal{V}_i$. Note that a variable $\lambda_{\omega,i}$ is only defined if all tests in sequence ω have the same vehicle specification requirements as those vehicles in group i, i.e., those tests can be actually performed on a vehicle from group i. Our optimization model is as follows:

$$\textbf{CTSP} \quad \text{minimize} \quad \sum_{\omega \in \Omega, i \in \{1,\ldots,m\}} c_{\omega,i} \lambda_{\omega,i} \tag{1}$$

$$\text{s.t.} \quad \sum_{\omega \in \Omega, i \in \{1,\ldots,m\}} \delta_{\omega,t} \lambda_{\omega,i} = 1, \quad \forall t \in T, \tag{2}$$

$$\sum_{\omega \in \Omega} \lambda_{\omega,i} \leq |\mathcal{V}_i|, \quad i = 1, \ldots, m, \tag{3}$$

$$\lambda_{\omega,i} \in \{0, 1\}.$$

In (1), $c_{\omega,i},\ \omega \in \Omega, i \in \{1, \ldots, m\}$ is the cost of assigning sequence ω to a vehicle from group $\mathcal{V}_i$, and is informed by the cost of using a vehicle from the group $\mathcal{V}_i$ and the timing associated with the assignment. Constraints (2) ensure that each test gets assigned, where $\delta_{\omega,t},\ \omega \in \Omega, t \in \mathcal{T}$ is a binary indicator such that $\delta_{\omega,t} = 1$ if sequence ω contains test t, and (3) are vehicle capacity constraints. For each group of vehicles i, there can be at most $|\mathcal{V}_i|$ sequences being assigned to that group, thus making sure that each vehicle is being used by at most one test sequence.

Solution Approach

Set-partitioning integer programming problems similar to **CTSP** typically have tight linear programming relaxations and other features that make them well-suited for standard branch-and-bound algorithm. However, a potential drawback of models built with *composite* variables, encompassing an entire sequence of decision, is their size. Indeed, the potential number of *sequences* of tests we would need to consider could be exponential in $|\mathcal{T}|$. Although the number of valid test sequences is limited by compatibility and timing considerations, in some of the instances that we have considered in our computational tests

of the model it exceeded the limit that CPLEX — our solver of choice — was capable of holding in memory. For that reason, we provide two optimization approaches within the TP3S application.

The first optimization approach uses a full enumeration of valid test sequences. If successful, this method is guaranteed to solve **CTSP** to optimality as the example in Figure 9 illustrates; however, in some instances this method exceeded allocated memory and terminated without finding a solution. The alternative approach applies delayed column generation to **CTSP**, thus avoiding enumerating all variables a priori (see Appendix for details).

Our computational experiments suggested that, while the full enumeration approach was often superior, occasionally it was unable to solve the problem due to memory limitations. The delayed column generation method, although slower, was able to scale more gracefully on larger problem instances, serving as reliable "backup" to the first method. These observations have been reinforced through the use of the optimization module of the TP3S application at Ford. Moreover, as larger sets of tests are considered in the future, the column generation methodology is likely to prove even more valuable.

Conclusion

Our work on developing the crash test scheduling application described in this paper began in late 2011. After several iterations of pilots and optimization model improvements, our first successful proof-of-concept in the 3rd quarter of 2012 eliminated the need for 2 prototype vehicles for a program. This achievement was realized through evaluation of what-if scenarios that highlighted opportunities to increase vehicle utilization by shifting tests to alternate, but acceptable control models. In this particular pilot, multiple engine choices were available and the key to reducing prototype vehicle costs by $500K was shifting test modes between V6 and V8 engine models.

Through several more pilots, we continued to enhance the integer programming optimization model and prove its robustness in producing high-quality crash plans. Our main point of emphasis then became designing and implementing a software application for engineers and management to use independently. We released the first version of the web application in summer 2014 to a select group of engineers. In early 2015, the application was fully released for use on all future programs. It is currently being shifted from research to IT production servers.

With continued work, we have plans to extend the capabilities of the Test Planning Scheduler Support System application to help facilities involved in crash testing coordinate and manage the requirements of multiple vehicle programs simultaneously. Our application is a first-of-its-kind within Ford's Product Development division and is already serving as an example for other groups within the company to follow.

Acknowledgments

We would like to thank many colleagues at Ford for their collaboration and contributions to this work, including Bryan Shiroda from Vehicle Evaluation & Verification, Matthew Schultz and David Kroos from the Java Center of Excellence, Bahij El Fadl and Joshua Catana from High Performance Computing, Chris Vinegar and Jim Cieslak from IT, and many engineers and managers from Product Development Safety. This work was supported by a grant from the Ford-University of Michigan Innovation Alliance.

Appendix. A Delayed Column Gereration Approach to CTSP

Recall that formulation (1) – (3) of **CTSP** defines a binary variable for each potential assignment of a valid test sequence to a prototype vehicle from a particular group. Here, we describe a delayed column generation heuristic for solving this problem to near-optimality; in our computational experiments, this method has proven useful in several instances that could not be handled by complete a priori specification of all variables.

Our method is similar to the branch-and-price algorithm proposed inBarnhart et al. (1998) to solve large scale combinatorial problems; however, rather than implementing a full branch-and-price algorithm, we simply solve the root-node partitioning problem using just the columns generated in the process of solving its LP relaxation; thus, our method is a heuristic.

Consider the LP relaxation of formulation **CTSP** obtained by replacing the integrality constraint $\lambda_{\omega,i} \in \{0,1\}$ by $\lambda_{\omega,i} \geq 0$, denoted by **LMP** (for Linear Master Problem). The upper-bound constraints $\lambda_{\omega,i} \leq 1$ are implicitly enforced by (2). As is typical in delayed column generation algorithms, we start with a subset of all variables of **LMP** and solve this restricted version of the original problem. Then, we continue adding variables that have potential to improve the objective function value of the restricted problem until the optimum is reached.

Let the initial subset of all variables be $\Omega' \subset \Omega$. Consider the Restricted Linear Master Problem:

$$\textbf{RLMP} \quad \text{minimize} \quad \sum_{\omega \in \Omega',\, i \in \{1,\ldots,m\}} c_{\omega,i}\lambda_{\omega,i} \tag{4}$$

$$\text{s.t.} \quad \sum_{\omega \in \Omega', i \in \{1,\ldots,m\}} \delta_{\omega,t}\lambda_{\omega,i} = 1, \quad \forall t \in T, \tag{5}$$

$$\sum_{\omega \in \Omega'} \lambda_{\omega,i} \leq |\mathcal{V}_i|, \quad i = 1,\ldots,m, \tag{6}$$

$$\lambda_{\omega,i} \geq 0 \quad \omega \in \Omega',\, i \in \{1,\ldots,m\}.$$

Let (π_t, ρ_i) be the optimal values of dual variables associated with sets of constraints (5) and (6), respectively. Then the reduced cost of a decision variable $\lambda_{\omega,i}$ is

$$\Gamma_{\omega,i} = c_{\omega,i} - \sum_{t\in\mathcal{T}} \delta_{\omega,t}\pi_t - \rho_i. \tag{7}$$

The associated pricing problem is defined to determine the sign of the optimal value of the problem

$$\textbf{PP} \quad \text{minimize}_{\omega\in\Omega,\, i\in\{0,\ldots,m\}} \Gamma_{\omega,i}; \tag{8}$$

if the optimal value is nonnegative, then the current solution to the **RLMP** is optimal for the full **LMP**, otherwise, if there exists some $\tilde{\omega}\in\Omega$ and $\tilde{i}\in\{1,\ldots,m\}$ such that $\tilde{\omega}\notin\Omega'$, and $\Gamma_{\tilde{\omega},\tilde{i}}<0$, then we conclude that variable $\lambda_{\tilde{\omega},\tilde{i}}$ should be added to **RLMP** to improve the solution.

Problem **PP** can be viewed as a search for a test sequence and vehicle combination with the smallest reduced cost. When it is possible to explicitly enumerate all valid test sequences, the minimization problem **PP** can be solved by "pricing out" each combination (note that, unlike when solving the full master problem, we don't need to hold the entire set of variables in memory; they can be created and evaluated one by one). When explicit enumeration is impossible or undesireable, we can formulate an integer linear program for solving **PP**. Indeed, in solving **PP**, given a vehicle group i, the only remaining decision is the composition of a sequence ω, namely indicators $\delta_{\omega,t}$ and the sequencing of all tests included.

Let $z_t,\ t\in\mathcal{T}$, be binary variables such that $z_t=1$ if test t is included in the sequence; $y_{t_1,t_2},\ t_1,t_2\in\mathcal{T}$, be binary variables such that $y_{t_1,t_2}=1$ if both tests t_1 and t_2 are included in the sequence and t_1 is performed before t_2 (not necessarily immediately); $s_t,\ t\in\mathcal{T}$, be continuous variables denoting the starting time of test t; finally, let $o_t,\ t\in\mathcal{T}$, be continuous variables that reflect penalties associated with the timing of each test. For vehicle group i, the pricing problem is

$$\textbf{PP-MILP}_i \quad \min \quad 1+\sum_{t\in\mathcal{T}} w_t o_t - \sum_{t\in\mathcal{T}} \pi_t z_t - \rho_i \tag{9}$$

$$\text{s.t.} \quad z_{t_1}+z_{t_2}-1 \le y_{t_1,t_2}+y_{t_2,t_1} \le \frac{1}{2}(z_{t_1}+z_{t_2}), \quad \forall t_1,t_2\in\mathcal{T}: t_1\neq t_2 \tag{10}$$

$$y_{t_1,t_2}=0, \quad \forall (t_1,t_2)\in\mathcal{T}: a_{t_1,t_2}=0 \tag{11}$$

$$s_{t_1}+p_{t_1} \le s_{t_2}+M(1-y_{t_1,t_2}), \quad \forall t_1,t_2\in\mathcal{T}: t_1\neq t_2 \tag{12}$$

$$s_t \ge r_t, \quad \forall t\in\mathcal{T} \tag{13}$$

$$s_t \ge q_i, \quad \forall t\in\mathcal{T} \tag{14}$$

$$s_t+p_t \le d_t+o_t, \quad \forall t\in\mathcal{T} \tag{15}$$

$$z_t\in\{0,1\},\ y_{t_1,t_2}\in\{0,1\},\ s_t\ge 0,\ o_t\ge 0. \tag{16}$$

Formulation **PP-MILP**$_i$ is based on the assignment formulation of the test scheduling problem; since it is restricted to only one vehicle group, it is significantly easier to solve. If the optimal value of the problem is negative, the variable $\lambda_{\omega,i}$, with ω corresponding to the sequence of tests identified by solving **PP-MILP**$_i$, can be added to the master problem, with $c_{\omega,i}=1+\sum_{t\in\omega} w_t o_t$.

Computational Results

Prior to deployment as part of the TP3S application at Ford, we tested both full enumeration and column-generation approaches to **CTSP** on several real problem instances on a personal computer. The data used in the computational experiments contained data from 7 past vehicle programs, with IDs 1 through 7. Table 1 provides high-level characteristics of those programs.

Table 1 Data characteristics of problem instances used in computational experiments

Instance ID	Num Vehicles	Num Tests	Density
1	36	46	0.90
1r	36	46	0.89
2	64	60	0.99
2r	64	60	0.88
3	95	64	0.94
3r	95	64	0.87
4	87	81	0.98
4r	87	81	0.79
5	15	18	0.86
5r	15	18	0.82
6	19	23	0.87
6r	19	23	0.84
7	18	12	0.96
7r	18	12	0.90

In Table 1, "Num Vehicles" is the number of available prototype vehicles that can be potentially used or budgeted in the program, providing an upper bound on the number of vehicles to be used in the schedule. "Num Tests" is the number of tests to be scheduled for each program. "Density" refers to the percentage of non-zeros in the matrix $\mathbf{A}$; the higher the density the more restrictive the test compatibility rules are, corresponding to a smaller number of valid test sequences.

In practice, we are often interested in solving instances where the vehicle specification requirements of test requests are relaxed. This is useful for engineers to measure the impact of engineering complexity on resources required for testing. Therefore, for each of the problem instances, we also considered a version where such specification constraints are relaxed. In Table 1, they are marked with "r" after the instance ID.

All methods were implemented in Java 1.7 and run on a laptop equipped with Intel Core i7, and 16GB memory. We use IBM ILOG CPLEX 12.6 to solve linear and mixed integer programs.

In the delayed column generation algorithm, we used branch-and-bound to solve the integer programming formulation of the pricing problem; we terminated the solution process if a sequence/vehicle group assignment variable was determined to have a negative reduced cost at 5% optimality gap. In order to ensure that an integer solution to the master problem can be found at the termination of our column generation procedure, at each iteration we generated several "sibling" sequences, in addition to the (near) optimal one: after solving the pricing problem with respect to all tests and vehicle groups and obtaining the sequence with the most negative reduced cost, we removed the tests that were included in this sequence and re-solved the pricing problem with respect to the remaining tests. This procedure was repeated until we have found a set of sequences that covers all tests.

Table 2 shows the runtime of the two algorithms on the above instances. The full enumeration algorithm solved most of the instances to optimality. However, we have two instances, 7r and 10r, where the number of variables exceeded available memory; we mark these cases as "−" in the table. The column generation algorithm is often slower, but it was able to handle all the instances, solving all but two of them to optimality (and coming within 3.5% and 0.3% of optimality for the remaining two instances).

Table 2 Run Time (in seconds) for full enumeration and column generation algorithms; a dash indicates that the algorithm did not terminate

Instnace ID	Full Enumeration	Column Generation
1	3.08	2.358
1r	3.82	2.352
2	1.83	34.283
2r	50.82	83.324
3	35.14	35.12
3r	-	47.701
4	3.70	17.874
4r	-	43.602
5	0.29	0.575
5r	0.34	0.942
6	0.39	1.829
6r	0.50	1.788
7	0.14	0.217
7r	0.19	0.285

References

Barnhart, Cynthia, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, Pamela H Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations research* **46**(3) 316–329.

Bartels, J-H, Jürgen Zimmermann. 2009. Scheduling tests in automotive R&D projects. *European Journal of Operational Research* **193**(3) 805–819.

Chelst, Kenneth, John Sidelko, Alex Przebienda, Jeffrey Lockledge, Dimitrios Mihailidis. 2001. Rightsizing and management of prototype vehicle testing at Ford Motor Company. *Interfaces* **31**(1) 91–107.

Chen, Zhi-Long, Warren B Powell. 1999. Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing* **11**(1) 78–94.

Cheng, TCE, CCS Sin. 1990. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research* **47**(3) 271–292.

Coffman Jr, Edward G, Michael R Garey, David S Johnson. 1996. Approximation algorithms for bin packing: a survey. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 46–93.

Elhedhli, Samir, Lingzi Li, Mariem Gzara, Joe Naoum-Sawaya. 2011. A branch-and-price algorithm for the bin packing problem with conflicts. *INFORMS Journal on Computing* **23**(3) 404–415.

Limtanyakul, Kamol, Uwe Schwiegelshohn. 2012. Improvements of constraint programming and hybrid methods for scheduling of tests on vehicle prototypes. *Constraints* **17**(2) 172–203.

Pinto, Jose M, Ignacio E Grossmann. 1995. A continuous time mixed integer linear programming model for short term scheduling of multistage batch plants. *Industrial & Engineering Chemistry Research* **34**(9) 3037–3051.

Potts, Chris N, Vitaly A Strusevich. 2009. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society* S41–S68.

Reich, Daniel, Yuhui Shi, Erica Klampfl, Marina Epelman, Amy Cohn. 2014. An analytical approach to prototype vehicle test scheduling. *Under review* .

Zhu, Zhiwei, Ronald B Heady. 2000. Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. *Computers & Industrial Engineering* **38**(2) 297–305.