ARTICLE IN PRESS

Omega ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Omega

journal homepage: www.elsevier.com/locate/omega

An analytical approach to prototype vehicle test scheduling☆

Yuhui Shi [a,*], Daniel Reich [b], Marina Epelman [a], Erica Klampfl [b], Amy Cohn [a]

[a] *Industrial & Operations Engineering, University of Michigan, Ann Arbor, MI 48109, United States*
[b] *Research & Advanced Engineering, Ford Motor Company, Dearborn, MI 48120, United States*

ARTICLE INFO

Article history:
Received 26 November 2014
Accepted 10 May 2016

Keywords:
Automobile industry
Scheduling
Integer programming
Heuristics

ABSTRACT

The test planning group within Ford's Product Development division develops schedules for building prototype vehicles and assigning them to departments in charge of different vehicle components, systems and aspects (e.g., powertrain, electrical, safety). These departments conduct tests at pre-production phases of each vehicle program (e.g., 2015 Ford Fusion, 2016 Ford Escape) to ensure the vehicles meet all requirements by the time they reach the production phase. Each prototype can cost in excess of $200 K because many of the parts and the prototypes themselves are hand-made and highly customized. Parts needed often require months of lead time, which constrains when vehicle builds can start. That, combined with inflexible deadlines for completing tests on those prototypes introduces significant time pressure, an unavoidable and challenging reality. One way to alleviate time pressure is to build more prototype vehicles; however, this would greatly increase the cost of each program. A more efficient way is to develop test plans with tight schedules that combine multiple tests on vehicles to fully utilize all available time. There are many challenges that need to be overcome in implementing this approach, including complex compatibility relationships between the tests and destructive nature of, e.g., crash tests. We introduce analytical approaches for obtaining efficient schedules to replace the tedious manual scheduling process engineers undertake for each program. Our models and algorithms save test planners' and engineers' time, increases their ability to quickly react to program changes, and save resources by ensuring maximal vehicle utilization.

1. Introduction

Ford's Product Development division is responsible for designing and testing new vehicles and readying them for production. Each vehicle program (e.g., 2015 Ford Fusion, 2016 Ford Escape) progresses through several consecutive stages: concept, design, development and testing, etc., before a new vehicle is manufactured on the assembly line. After the concept and design phases are completed, prototype vehicles are built and subjected to tests to ensure the new vehicle model meets all the design criteria. Each required test needs to be completed by its deadline to ensure adherence to the overall program timing. Test planners and engineers are tasked with scheduling all the tests, placing orders for parts to build the required prototype vehicles, scheduling the order of the builds (e.g., prototype vehicle with automatic transmission on day 1, one with manual transmission on day 2) and assigning the vehicles to departments in charge of tests for different vehicle components, systems, and aspects (e.g., powertrain, electrical, safety).

Each prototype built during the development and testing phases of a vehicle program can cost in excess of $200 K because many of the parts and the prototypes themselves are hand-made and highly customized. Parts needed often require months of lead time, which constrains when prototype vehicle builds can start. That, combined with inflexible deadlines for completion of tests on those vehicles, introduces significant time pressure, an unavoidable and challenging reality associated with maintaining the overall program timing. One way to alleviate time pressure is to build more vehicles, essentially decreasing competition between tests for available vehicle time; however, this would greatly increase the cost of each program. A more efficient way is to develop test plans with tight schedules that combine multiple tests on vehicles to maximally utilize available time. There are many challenges that need to be overcome in implementing this approach. For example, many tests are destructive (e.g., crash tests performed by the safety department), preventing scheduling further tests on the vehicle. Another complicating factor is that different tests may have different vehicle specification requirements; for example, one test may require a hybrid engine whereas

☆This manuscript was processed by Associate Editor W. Shen.
* Corresponding author.
E-mail addresses: yuhuishi@umich.edu (Y. Shi), dreich8@ford.com (D. Reich), mepelman@umich.edu (M. Epelman), eklampfl@ford.com (E. Klampfl), amycohn@umich.edu (A. Cohn).

http://dx.doi.org/10.1016/j.omega.2016.05.003
0305-0483/© 2016 Elsevier Ltd. All rights reserved.

another may require a conventional 4-cylinder I4 engine, prohibiting combinations of these tests on the same vehicle.

Prior to our work, test plans were exclusively developed manually using pen and paper and Excel spreadsheets. However, this process is tedious and constructing a test plan may take days, if not weeks. The schedule achieved may not be optimal in terms of the number of vehicles needed; moreover, when changes occur to deadlines and individual tests, manually editing the plan requires significant additional time and effort, and may lead to decreasing vehicle utilization. In this paper, we formally define the problem of obtaining optimized schedules (i.e., ones that minimize the number of vehicles used subject to all pertinent constraints) and introduce computational heuristics that replace the tedious manual scheduling process engineers undertake for each program. Automation saves test planners' and engineers' time, increases their ability to quickly react to program changes, and saves resources by providing schedules with high vehicle utilization.

In this paper, we describe the development and piloting of our schedule optimization models. We introduce the details of the scheduling problem, then provide an exact mathematical formulation, which turns out to have limited tractability. This leads to our practical heuristic algorithm that provides good feasible schedules. We present results from our first pilot and discuss ongoing efforts and goals of this project.

2. Literature review

Although certain aspects of the optimization problem addressed in this paper are specifically motivated by prototype vehicle test scheduling at Ford, it has some features of bin packing on the one hand, and parallel machine scheduling on the other, and can be viewed as an extension of both problems.

In the classic bin packing problem, a set of items with different sizes needs to be packed into bins of limited capacities, and the minimum number of bins required is to be determined. There are extensive studies of this problem (see, e.g., [1]). In our setting, determining the minimum number of vehicles needed to perform all tests is akin to bin packing with non-identical bins (vehicles), whose capacity reflects the time interval during which the vehicle is available, and with additional restrictions on the compatibility of items (tests) to be assigned to the same bin. A paper in this area most closely related to our research is [2]. In it, the authors consider a variation of the bin packing problem with conflicts between items. The authors provide a set-partitioning formulation of the problem and propose a branch-and-price algorithm to solve it exactly, with the pricing problem solved as a knapsack problem with conflicts. Our problem, however, is more complex, since tests assigned to the same vehicle need to be scheduled as well.

In the parallel machine scheduling problem, a set of time-sensitive tasks with associated processing times need to be scheduled on a given set of machines, while minimizing a certain criterion, usually time-related, such as make-span or total tardiness. The literature on parallel machine scheduling has developed over several decades and contains a variety of models and algorithms; comprehensive surveys and comparisons between different solution strategies can be found in [3–5]. Associating machines with vehicles and jobs with tests, one can see many similarities between test scheduling and certain types of machine scheduling problems. Indeed, in machine scheduling jobs often have release and due dates, and test compatibility and sequencing restrictions can be represented by including setup times between jobs, setting them to very high values for tests that cannot be performed together or in a particular order. However, our test scheduling problem has several features that make it unique in the scheduling literature. In particular, machines are usually assumed to be available throughout the scheduling process, whereas prototype vehicles are released gradually during testing. Moreover, while specification of which machines are capable of executing which jobs is considered in the literature, whether a prototype vehicle has the features needed for a particular test is determined by the *other* tests assigned to this vehicle (see Section 3 for details), making a priori specification impossible. Finally, the objective of minimizing the number of vehicles used is fairly uncommon in the scheduling literature. In light of the above, in our review of machine scheduling literature we will focus on the papers that aim to minimize the number of machines used. We also discuss representative papers which emphasize sequencing aspects of scheduling in the presence of precedence constraints or setup times, especially those that utilize heuristic algorithms similar to the Fit-and-Swap heuristic we propose in Section 5, to emphasize relevant results as well as elucidate the distinct features of our problem.

A small subset of machine scheduling literature focuses on a problem most closely related to ours, where the goal is to optimize some machine-related metrics, such as the cost of holding and using machines, rather than the traditional job-related time metrics. In addition to bin-packing resources, in the context of scheduling, a particularly relevant paper [6] studies the so called "Scheduling with Release times and Deadlines on a minimum number of Machines (SRDM)" problem, where each task has a duration and a time window for execution, and the number of machines required to perform all tasks on time remains a decision. The authors propose a polynomial algorithm for the special case when time windows of jobs are tight (namely, exactly 1 plus job duration). In the more general case, they propose an approximation method called Greedy-Best-Fit, which is a list scheduling algorithm that assigns jobs to the machine with the left-most available time slot. They prove that this heuristic is a 9-approximation algorithm in the special case of equal processing times. Reference [7] improves the approximation bound from 9 to 6 for the same special case. For another special case of common release dates, the authors of the latter paper propose another list scheduling algorithm, called Greedy-Increasing-Slack, which sorts and assigns jobs in reverse order of flexibility of shifting within its time window. This method has a constant approximation bound 2.

For a related vehicle routing problem, Reference [8] studies minimizing the number of trucks to satisfy customer loads, where each load has a time window during which it should be delivered. The authors propose a two-phase approach which uses a time-indexed formulation to form sequences of loads and later heuristically assigns them to trucks. Such an approach can solve up to instances with 34 loads and 10 trucks—smaller than the test scheduling instances for which we provide computational results.

Several other papers consider machine scheduling while minimizing the number of machines used, including [9–11]. However, they each make restrictive assumptions to guarantee that their proposed algorithms solve the problem exactly or with a guaranteed bound. For example, equal processing times and/or common release or due dates of jobs are often assumed in such papers. Interestingly, in [11] the authors also consider precedence constraints among different jobs across machines, where the start of job A cannot precede the completion of another job B (they do assume equal processing times). They propose polynomial algorithms that can solve this problem for special cases of precedence graph structures, such as trees and chains. Although precedence relationships may seem similar to a feature of the test scheduling problem, in the latter case, the precedence relations between two test are only relevant if they are assigned to the same vehicle.

In the more traditional context of scheduling (one where the number of available machines is specified a priori), one class of problems that are particularly relevant to ours is the setting with sequence-dependent setups. If we model the precedence relations between two tests, where test A cannot precede test B when assigned to the same machine, by making the setup time between test A and B arbitrarily large, we can view the problem as a special case of the sequence-dependent setup case. For example, References [12,13] model the problem of minimizing the earliness and tardiness in this setting using a Mixed Integer Linear Programming (MILP) formulation together with various strengthened constraints. However, they can only solve relatively small instances, with less than 10 jobs and 5 machines. Reference [14] studies another time-dependent setup problem, where setup time of each task is related to its starting time. The author proposes a time-indexed MILP formulation that can solve small instances with up to 50 tasks and 10 machines. Since tractability of MILP formulations is limited, another set of papers focuses on using heuristic methods to solve the sequence-dependent setup scheduling problem. Perhaps the most common approach is to use list scheduling methods motivated by the bin packing problem. Reference [15] considers the problem of minimizing the total weighted completion time where all jobs have common release dates. They propose a Best-Fit method, where the duration of each job is computed by also taking into consideration information about its setup time. This is achieved by adding, for example, the average setup time or minimum setup time to the job's duration. Then the jobs are ordered by longest processing time and assigned to machines where such assignment contributes least (with shortest processing time) to the objective function value. It should be noted that the intuition behind the Fit-and-Swap heuristic we propose in Section 5 may appear similar to such Best-Fit methods. While the heuristics have a greedy nature in common, their Best-Fit approach tends to evenly distribute all jobs onto given machines to optimize a time-based objective—an approach that would not be suitable if minimizing the number of machines used is desired. A number of papers emphasize the sequencing of jobs. For example, Reference [16] proposes several heuristics that combine the greedy approach to minimize makespan with TSP heuristics for job sequencing. Reference [17] uses the same idea, where the entire set of jobs is pre-ordered using some rules such as Earliest Due Data (EDD) on each machine. Duplicate jobs are gradually removed from each machine and those remaining are resequenced in a greedy way until a full schedule is found. As we already discussed, in the test scheduling problem, approaches that attempt to sequence the tests before assigning them to vehicles are not applicable.

There have also been extensive efforts to solve machine scheduling problems with other heuristic methods, such as metaheuristics, tabu search (for examples, see [18–23]). It is also common to combine several heuristics in order to obtain high-quality solutions. For example, Reference [24] considers parallel machine scheduling problems where jobs have types, and limited time is available for transitioning between two types of jobs on a single machine. The authors combine thresh-accepting methods, tabu lists and improvement procedures in order to obtain high-quality solutions, and show that such combinations can outperform methods that adopt only a single idea. However, in this paper we chose to focus on a simpler, less computationally demanding heuristic that proved more easily applicable in scheduling prototype vehicles at Ford.

In recent years, optimization models have been introduced specifically for test scheduling on prototype vehicles. Reference [25] focuses on determining the number of unique types of prototype vehicles needed for a program, given the specification requirements of all individual tests. This work, in collaboration with Ford, dates back to a time when planning was less process driven and resource constrained, so the models did not consider all the timing and compatibility requirements that are essential today. Reference [26] formulates the prototype vehicle test scheduling problem using mixed integer programming. While their model incorporates many of the same factors as the model we develop, it has limited computational tractability. It also differs in its assumptions around the vehicle build decision process; they introduce flexibility in determining the build schedule whereas the build schedule is inflexible in our model, which is aligned with the current operational process at Ford. Reference [27] proposes a hybrid model for scheduling prototype vehicle tests: integer programming is the first stage model that determines vehicle specification (e.g., hybrid or conventional engine, manual or automatic transmission) and test-to-vehicle assignments; constraint programming is the second stage model that determines timing and sequencing. While their approach is effective on smaller instances, their computational results demonstrate a lack of scalability. The first stage continues to provide candidate solutions for the second stage, but their constraint programming model cannot consistently reassign the tests to produce a feasible schedule.

3. Test scheduling problem

For each vehicle program, one engineer from each department (powertrain, safety, and electrical) is typically responsible for entering the department's testing requirements into a shared Excel file. These requirements include test durations, vehicle specification requirements, test severity and additional information. Once all the requirements have been gathered, a test planner begins developing the program schedule.

Three main decisions are required for scheduling:

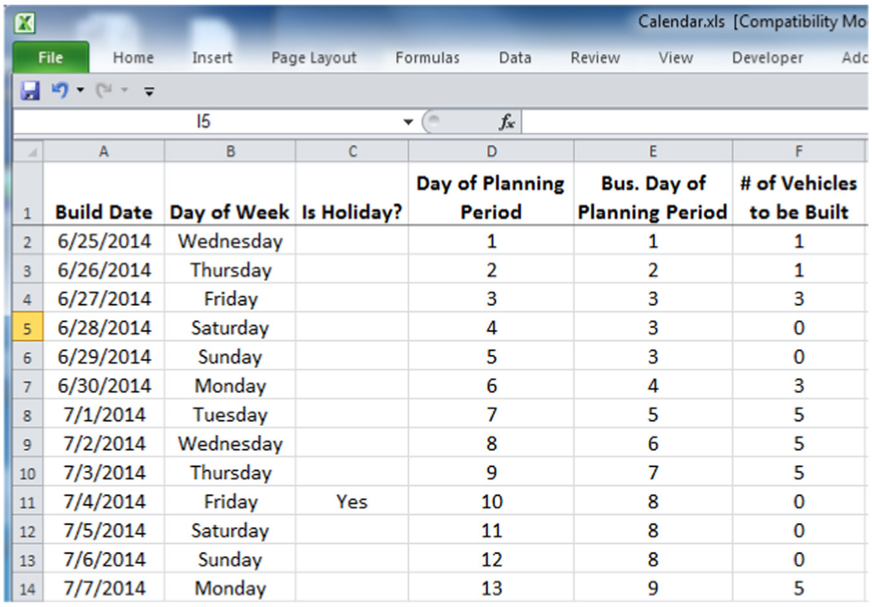
- How many prototype vehicles need to be built?
- What specifications are required for those vehicles (moonroof, manual transmission, and engine type)?
- Which tests are assigned to each vehicle?

Each vehicle is released on a given date during the testing horizon. These dates (i.e., the build schedule) are provided by the group in charge of building the prototypes and reflects its production capacity, whereas the exact vehicle specifications for each day may be flexible and decided as part of the scheduling process. The calendar template shown in Fig. 1 is used for inputting the build schedule. Each row corresponds to one day and includes the maximum number of vehicles that can potentially be delivered on that day. The fields are described in Table 1. The vehicle capacity (time) is the number of available business days from its release date to the end of the testing phase, which is set by the overall timeline for the vehicle program.

In addition to ensuring that schedules allocate sufficient amounts of time for each test to be completed by its individual deadline, the test planner must also ensure that compatibility relations between tests assigned to the same vehicle are not violated. For example, if one test requires a 6-cylinder V6 engine and another test requires a 4-cylinder I4 engine, they are not compatible with each other and cannot be assigned to the same vehicle. Another example is test severity, i.e., if one test is destructive, it may render a vehicle unusable for (certain kinds of) further testing.

Expertise on compatibility of tests currently resides with individuals, so we partnered with those experts to develop templates for storing their knowledge in standardized formats. During the scheduling process, these compatibility rules can be accessed systematically. While the actual compatibility rules are confidential, the matrix in Table 2 provides an illustrative example.

Notice that compatibility is not symmetric. In the example in Table 2, *B* may be performed after *A*, but not before. This is typical of testing requirements. For example, consider two crash tests: a



	A	B	C	D	E	F
1	Build Date	Day of Week	Is Holiday?	Day of Planning Period	Bus. Day of Planning Period	# of Vehicles to be Built
2	6/25/2014	Wednesday		1	1	1
3	6/26/2014	Thursday		2	2	1
4	6/27/2014	Friday		3	3	3
5	6/28/2014	Saturday		4	3	0
6	6/29/2014	Sunday		5	3	0
7	6/30/2014	Monday		6	4	3
8	7/1/2014	Tuesday		7	5	5
9	7/2/2014	Wednesday		8	6	5
10	7/3/2014	Thursday		9	7	5
11	7/4/2014	Friday	Yes	10	8	0
12	7/5/2014	Saturday		11	8	0
13	7/6/2014	Sunday		12	8	0
14	7/7/2014	Monday		13	9	5

Fig. 1. Calendar sheet describing the prototype vehicle build schedule.

Table 1
Description of fields in the calendar.

Release date: Date
Day of week: Su, M, Tu, W, Th, F, Sa
Is holiday: Is it a company holiday?
Day of planning period: # of days since the start of the planning period
Number of vehicles to be built: Max # of prototype vehicles delivered

Table 2
Test compatibility table; "F" indicates that the test in this row cannot be performed prior to the test in the column, "T" indicates otherwise.

	Test A	Test B
Test A	F	T
Test B	F	F

crash test at 2 miles per hour (MPH), e.g., a sensor test on the front bumper of a vehicle, and a 30 MPH structural test also striking the front bumper. Running the 2 MPH test first will not damage the bumper, so the 30 MPH test can follow. However, running the 30 MPH test first will render the bumper unusable for the sensor test to follow.

An example schedule is shown in Fig. 2.

4. Integer programming model

In this section, we introduce our original integer programming formulation for optimizing test plan schedules.

Sets and parameters

- T—set of tests
- V—set of available vehicles
- $E \subset T \times T$—set of incompatible ordered test pairs (e.g., in Table 2, $E = \{(A,A),(B,A),(B,B)\}$)
- p_t—duration of test t
- r_t—release date of test t
- d_t—deadline of test t
- q_v—delivery date of vehicle v

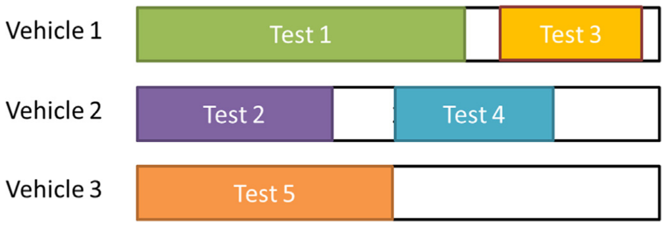


Fig. 2. A sample schedule.

Decisions

- u_v—binary; 1 if vehicle v is in the schedule, 0 otherwise
- $x_{t,v}$—binary; 1 if test t is assigned to vehicle v, 0 otherwise
- y_{t_1,t_2}—binary; 1 if test t_1 and test t_2 are performed on the same vehicle and t_1 precedes t_2, 0 otherwise
- s_t—continuous; starting time of test t.

Formulation

$$\min \sum_{v \in V} u_v \tag{1}$$

$$\text{s.t.} \quad x_{t,v} \le u_v \quad v \in V,\ t \in T \tag{2}$$

$$\sum_{v \in V} x_{t,v} = 1 \quad t \in T \tag{3}$$

$$s_{t_1} + p_{t_1} \le s_{t_2} + M(1 - y_{t_1,t_2}) \quad (t_1,t_2) \in T \times T : t_1 \ne t_2 \tag{4}$$

$$y_{t_1,t_2} + y_{t_2,t_1} \le 1 \quad (t_1,t_2) \in T \times T : t_1 \ne t_2 \tag{5}$$

$$x_{t_1,v} + x_{t_2,v} - 1 \le y_{t_1,t_2} + y_{t_2,t_1} \quad v \in V, \quad (t_1,t_2) \in T \times T : t_1 \ne t_2 \tag{6}$$

$$y_{t_1,t_2} = 0 \quad (t_1,t_2) \in E \tag{7}$$

$$s_t \ge r_t \quad t \in T \tag{8}$$

$$s_t \ge \sum_{v} q_v x_{t,v} \quad t \in T \tag{9}$$

$$s_t + p_t \le d_t \quad t \in T \tag{10}$$

$$x_t \in \{0,1\}, \quad y_{t_1,t_2} \in \{0,1\}, s_t \ge 0.$$

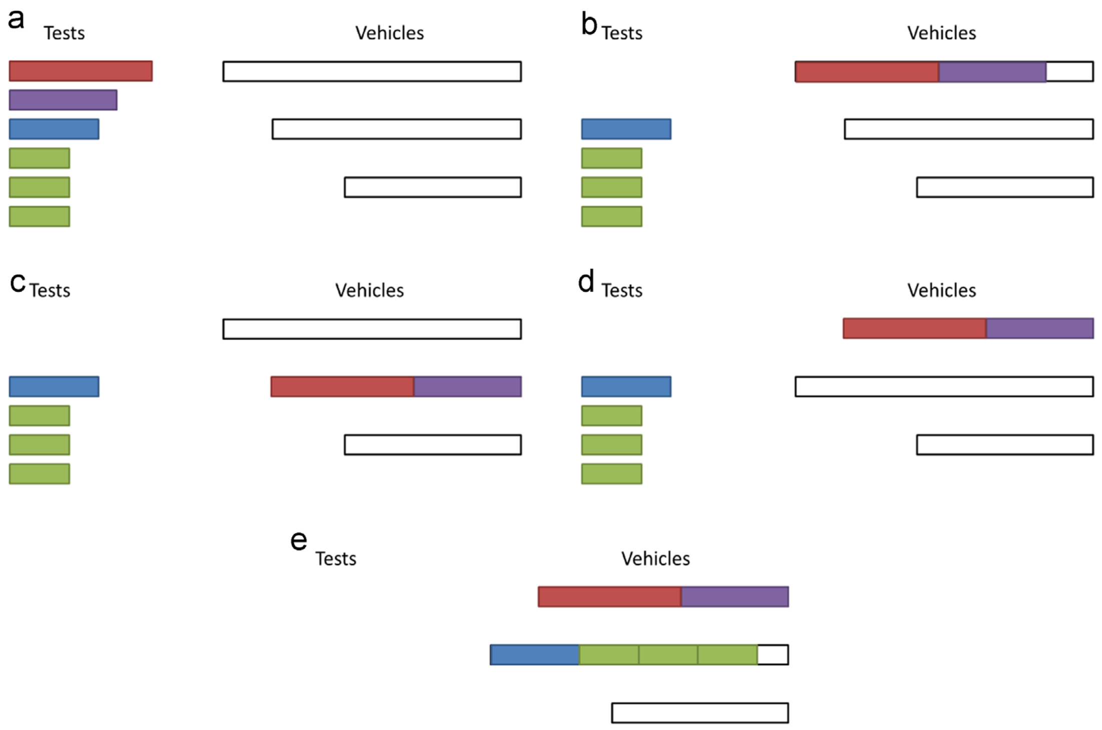


Fig. 3. Fit-and-Swap algorithm illustration.

In this formulation, objective (1) is to minimize total vehicle usage. Constraints (2) link the assignment decision variables x with the vehicle usage variables u. Constraints (3) ensure that each test gets assigned to exactly one vehicle. Constraints (4) and (5) enforce pairwise sequencing relationships for tests that are assigned to the same vehicle through the precedence disjunctive decision variables y, using a big-M formulation. Constraints (6) link those variables with the assignment variables. Constraints (7) enforce compatibility for tests assigned to the same vehicle. Constraints (8)–(10) ensure that each test is performed during its time window, and starts after the delivery of the vehicle to which it is assigned.

There are two problems with the above formulation that can lead to poor solvability: symmetry caused by identical vehicles (i.e., ones with the same release dates) and lack of tightness of the linear programming relaxation caused by the big-M constraints (4). To improve the formulation, we have included symmetry-breaking constraints: for any two identical vehicles with indices v_i and v_j where $v_i \le v_j$, we require that $u_{v_i} \le u_{v_j}$, and $x_{t,v_i} \ge x_{t,v_j}\ \ \forall t \in T$, i.e., we require that v_i is used before v_j, thus breaking the symmetry between two identical vehicles. Moreover, we set M equal to be the length of the planning period, starting from the delivery of the earliest vehicle to the latest deadline by which all tests need to be completed—the best available easily computable a priori value.

We tested the resulting formulation on instances of varying sizes with limited success. On smaller instances (ones consisting of only crash tests for a program), CPLEX was able to find optimal or near-optimal solutions within an hour. However, for any of the three problem instances (representative of a full program) reported in our computational experiments in Section 6, CPLEX was unable to produce even a feasible solution.

5. Heuristic algorithm

In this section, we introduce a heuristic and optimization subproblems that we can combine to obtain high-quality feasible solutions for larger problem instances.

5.1. *Fit-and-swap test scheduling heuristic*

We propose a greedy heuristic for generating a test schedule. The heuristic attempts to assign tests, in decreasing order of duration, to the vehicle with the largest capacity. Our method is similar to the First-Fit algorithm used in bin packing problems, and has a greedy flavor similar to methods used in [6,7,15], for their respective objectives; however, before assigning a test to a vehicle, we must consider compatibility, order restrictions, test time windows, and vehicle delivery date to determine if this assignment is feasible.

The algorithm, which we refer to as the *Fit-and-Swap* heuristic, is summarized in Algorithm 1 and illustrated in Fig. 3 on an instance with 3 vehicles and 6 tests. Procedurally, first we order the tests in decreasing order of their durations, and the vehicles in decreasing order of their (time) capacities, as shown in Fig. 3a. We then select the vehicle at the top of the list (one with largest capacity), and attempt to fill the time available on this vehicle with an ordered sequence of tests. We search through the test list in order until we find one that is compatible (in terms of vehicle specifications) with tests already assigned to this vehicle (if any), and has sufficiently short duration to fit in the (remaining) available time on the vehicle, as shown in Fig. 3b. If the vehicle already has some tests assigned, we solve the Ordering Satisfiability Problem (defined in Section 5.2) that determines whether there is a feasible order for combining the new test with already assigned ones, taking into account each test's time window and test sequencing restrictions. If not, this test cannot be assigned to the vehicle and the algorithm proceeds to the next test in the list.

When the end of the test list is reached, the identified set of tests is fixed, and the vehicle list is searched to find a vehicle with the smallest capacity that is sufficient for these tests (determined by solving the corresponding instances of the Ordering Satisfiability Problem). If such a vehicle differs from the current one, then the set of tests (possibly in a different order) is assigned to the former, and the latter is emptied, as shown in Fig. 3c. By swapping the vehicles, as in Fig. 3d, we may find opportunities for increasing vehicle utilization. For example, if the tests assigned to a vehicle with 100 available days only require 95 days, then we would assign this group of tests to a vehicle with 95 days, and the 100 day vehicle would remain available. The algorithm terminates when there are no unassigned tests, as shown in Fig. 3e, or there is insufficient remaining vehicle capacity to assign additional tests.

We observed that the current manual procedure for assigning tests is similar to this heuristic. Current practice already requires engineers to produce highly efficient schedules. Our goal for this work was to produce equivalent or slightly more efficient schedules, but do so significantly faster.

Algorithm 1. Fit-and-Swap algorithm.

```
Sort the vehicle list $V$ in decreasing order of vehicle capacity;
Sort the test list $T$ in decreasing order of test durations;
foreach $v \in V$ do
    foreach $t \in T$ do
        if $v$ is empty then
            if time available on $v$ is sufficient for $t$ then
                assign $t$ to $v$;
            end
        else
            if Ordering Satisfiability Problem is feasible then
                assign $t$ to $v$;
            end
        end
    end
    remove assigned tests from $T$;
    find $v' \in V$ with the smallest capacity, such that the set of tests on $v$
    can be reassigned to $v'$;
    if $v' \neq v$ then
        reassign all tests from $v$ to $v'$;
        swap locations of $v$ and $v'$ in $V$;
    end
end
```

5.2. Ordering satisfiability problem.

When attempting to add each new test to a vehicle within the Fit-and-Swap algorithm, we must be able to determine whether a feasible ordering exists, based both on compatibility relationships between tests (Table 2) and individual test release dates and deadlines. To do this efficiently, we solve the Ordering Satisfiability Problem, which can be seen as a significant simplification of the full integer programming model in Section 4, and is formulated as follows:

$$s_{t_1}+p_{t_1} \le s_{t_2}+M(1-y_{t_1,t_2}) \quad (t_1,t_2)\in \overline{T}\times\overline{T} : t_1 \neq t_2 \tag{11}$$

$$y_{t_1,t_2}+y_{t_2,t_1}=1 \quad (t_1,t_2)\in \overline{T}\times\overline{T} : t_1 \neq t_2 \tag{12}$$

$$y_{t_1,t_2}=0 \quad (t_1,t_2)\in E \tag{13}$$

$$s_t \ge r_t \quad t\in\overline{T} \tag{14}$$

$$s_t \ge q_v \quad t\in\overline{T} \tag{15}$$

$$s_t+p_t \le d_t \quad t\in\overline{T} \tag{16}$$

$$y_{t_1,t_2}\in\{0,1\}, s_t \ge 0.$$

Here, $\overline{T}$ is the candidate set of tests at the current iteration of the Fit-and-Swap algorithm, and v is the vehicle being filled. Unlike in the original integer program (1)–(10), which optimized vehicle usage, here a vehicle and a subset of assigned tests are pre-selected, making the Ordering Satisfiability Problem a feasibility problem. The only variables needed are the binary sequencing variables y_{t_1,t_2} and continuous start time variables s_t. Vehicle usage and test assignment constraints (2) and (3) are no longer required, and the original constraints (5), (6) and (9) reduce to (12) and (15), respectively.

Compared with the original IP in Section 4, which became intractable for larger problem instances, a typical instance of the Ordering Satisfiability Problem can be solved in under 1 s by an integer programming solver. With subset $\overline{T}$ containing no more than 20 pre-selected tests that are already assigned to a pre-selected vehicle, the problem size is limited to at most 400 binary variables, of which many are eliminated by constraints (13).

When all tests in the set $\overline{T}$ have the same release dates and deadlines, the ordering sub-problem can be reduced even further to the following linear constraint:

$$z_{t_2} \le z_{t_1} - 1 \quad (t_1, t_2) \in E, \tag{17}$$

where the values of continuous decision variables $z_t,\ t \in \overline{T}$, provide the order of the tests. Alternatively, one can formulate this special case as a problem of topologically sorting all nodes in a directed graph, where each test t corresponds to a node and directed arcs reflect ordering possibilities among compatible tests.

5.3. Integer programming models for grouping crash tests

Due to the destructive nature of crash tests, their scheduling is both particularly important and restricted. The test scheduling process in the safety department is focused on maximizing crash rehits—multiple crashes on the same vehicle—so that the fewest number of vehicles is destroyed, thereby increasing the supply of undamaged parts that can be made available for other purposes or programs.

The safety department is in the unique position of consistently being the last user of shared vehicles. While other groups' tests may be staggered throughout the schedule, crash tests are consolidated. Therefore, isolating and scheduling them separately is convenient, and does not significantly impact the optimality of the overall schedule. To implement this approach, we first aggregate crash tests into groups using an integer optimization model. These groups are then passed into the Fit-and-Swap algorithm, which schedules all the tests, treating the grouped crashes as a single test each. The computational results presented in the next section confirm the benefits of this pre-grouping approach, which consistently reduces the number of crashed vehicles.

One method to group crash tests is to formulate and solve an instance of the integer program (1)–(10) with the set T containing only crash tests. (We refer to this approach to crash test grouping as *grouping via full IP*.) However, obtaining solutions and proving optimality can still be computationally prohibitive, even for these smaller subproblems. Thus, we also consider a simplified approach to crash test grouping.

Algorithm 2. Group release time calculation algorithm.

```
$r_{t_G} := d_{t_n} - p_{t_n}$;
for $i = n-1 : 1$ do
    $r_{t_G} := \min\{d_{t_i}, r_{t_G}\} - p_{t_i}$
end
```

In practice, it is rare to have more than three crash tests executed on a single vehicle. Moreover, a balanced schedule is preferable; for example, it is generally better to have two vehicles with two crash tests each than three on one and one on the other. This motivated us to use the following matching-based model. (We refer to this approach as *grouping via matching*.)

Let $T_S \subset T$ be the set of all safety (i.e., crash) tests. The formulation below uses binary variables $w_{t_1,t_2,v}$ for all $t_1, t_2 \in T_S$ with $t_1 < t_2$ and all $v \in V$ to indicate whether t_1 and t_2 can be matched and executed on vehicle v. (We do not define variables w for $t_2 > t_1$ to avoid double-counting; however, test t_2 may be scheduled before t_1.)

$$\text{maximize} \quad \sum_{v \in V} \sum_{t_1,t_2: t_1 < t_2} w_{t_1,t_2,v} \tag{18}$$

$$\text{s.t.} \quad \sum_{v \in V} \left(\sum_{t_2: t_2 < t_1} w_{t_2,t_1,v} + \sum_{t_2: t_1 < t_2} w_{t_1,t_2,v} \right) \le 1, \quad t_1 \in T_S \tag{19}$$

$$\sum_{t_1,t_2: t_1 < t_2} w_{t_1,t_2,v} \le 1, \quad v \in V \tag{20}$$

$$w_{t_1,t_2,v} = 0, \quad \{t_1,t_2\} \in E_v, \quad v \in V \tag{21}$$

$$w_{t_1,t_2,v} \in \{0,1\}, \quad t_1,t_2 \in T_S: \quad t_1 < t_2, \quad v \in V$$

For each vehicle v, set E_v enumerates all pairs $\{t_1, t_2\}$ of safety tests that are incompatible based on specification, ordering, or vehicle-specific timing and deadline restrictions; we identify the set E_v in a pre-processing step. The objective (18) maximizes the number of matches among safety tests, while constraints (19) ensure that each test is combined with at most one other, and the pair of tests is assigned to at most one vehicle. Constrains (20) ensure that each vehicle is used at most once, and constraints (21) enforce compatibility requirements.

Once the number of pairings is maximized, we can search for triplets and quadruplets of tests by solving another instance of the matching problem on the set of previously identified pairs and remaining individual tests.

Each of the identified rehit grouping is passed to the Fit-and-Swap heuristic as an individual test with the following associated parameter values: for a grouping-based test $t_G = (t_1, \ldots, t_n)$, $p_{t_G} = \sum_{i=1}^{n} p_i$,

$$d_{t_G} = p_{t_G} + \min_{i=1,\ldots,n} \left\{ d_i - \sum_{k=1}^{i} p_k \right\},$$

and r_{t_G} computed by the following Algorithm 2:

Finally, we apply the Fit-and-Swap heuristic to schedule all tests in the program, treating the identified rehit groupings as single tests. Recall that the heuristic assigns longer tests first, which in effect gives scheduling priority to groups of safety tests, ensuring that all deadlines within the group are met. Note that the assignments of test groups to vehicles made in the grouping subroutines may differ from the final assignments generated by the Fit-and-Swap heuristic.

5.4. Test planning algorithm

The Test Planning Algorithm combines the heuristic and optimization subproblems presented above to obtain feasible solutions to the integer programming model in Section 4. The steps of the Test Planning Algorithm are summarized as follows:

- Read in the tests and create set T and crash test subset T_S.
- Read in the calendar (Fig. 1) and create vehicle set V.
- Read in compatibility rules and construct set E.
- (Optional) Formulate and solve a grouping IP on T_S and V (Section 5.3) and update T with grouped tests.

- Run the Fit-and-Swap algorithm on T and V and return the schedule.

6. Computational experiments

For our computational experiments, we tested instances from three recent vehicle programs at Ford. The total number of tests, and the number of safety tests, in each program are presented in Table 3. We attempted to solve these instances using the IP presented in Section 4 with the symmetry-breaking constraints added, but CPLEX was unable even to find a feasible solution within the 1 h time limit, also noted in Table 3.

Next, we applied the Fit-and-Swap heuristic algorithm from Section 5.1, without safety tests grouping, and with grouping via matching and via IP. Table 4 presents vehicle usage results for the three methods. The first important takeaway is the significant reduction in the number of vehicles crashed when using either of the grouping subroutines. On the three instances tested, both subroutines arrived at the same number of crashed vehicles, demonstrating the effectiveness of the simpler matching-based approach. The actual pairings though were different, evidenced by the differences in the overall solutions.

While Fit-and-Swap without grouping produced solutions inferior in all respects for two of the instances, for Instance 2 the total number of vehicles was actually slightly fewer. However, this would come at the expense of crashing 11 more vehicles, thereby destroying expensive resources.

We also report vehicle utilization, which is calculated as the ratio between the sum of capacities of all vehicles used in a schedule, and the total duration of all the tests in the program. This metric can be used to compare efficiency of test schedules in different programs, which may vary in size and therefore in the number of vehicles used. However, considered in isolation this metric may be misleading. As can be seen in Table 4, solutions with different numbers of vehicles can yield identical utilizations by selecting vehicles of different capacities. For this reason, maximizing utilization is not the primary objective, but it is still an important metric for assessing the schedule.

Table 5 summarizes the runtimes of the three approaches, separating runtimes for the subroutines and the Fit-and-Swap heuristics applied to the resulting test sets. The full IP has limited tractability even when used as a subroutine, while the matching IP is solved quite efficiently. The Fit-and-Swap heuristic is quick in all cases.

Table 3
Description of problem instances used in computational experiments. The full IP formulation (1)–(10) of Section 4 did not produce feasible solutions within 1 h time limit on any of these instances.

Instance ID	Number of tests	Number of safety tests	IP feasible solution?
Instance 1	501	82	No
Instance 2	474	49	No
Instance 3	434	14	No

Table 4
Vehicle usage results for the three methods.

Instance ID	Total number of vehicles	Number of vehicles crashed	Vehicle utilization
No crash test grouping subroutine			
Instance 1	109	57	0.871
Instance 2	143	43	0.946
Instance 3	112	13	0.923
Crash test grouping via matching			
Instance 1	96	52	0.941
Instance 2	144	32	0.946
Instance 3	110	11	0.937
Crash test grouping via full IP			
Instance 1	97	52[a]	0.939
Instance 2	145	32	0.945
Instance 3	110	11	0.937

[a] Note that for Instance 1 the full IP in the subroutine was not solved to optimality: after 60 min, the optimality gap was 7.7% the best feasible solution found in that time is reported.

7. Discussion

In 2012, we piloted our scheduling models and algorithms for the first time on a vehicle program with 108 prototype vehicles budgeted, with an average vehicle cost of around $250 K. The vehicles were scheduled to be delivered starting in May of 2013, at a rate of about 13/week, with a ramp-up period at the beginning. Engineers submitted over 500 test requests.

Practically speaking, producing an initial schedule with a utilization of 95%—based exactly on engineer-submitted test requests —exceeded expectations for our initial pilot. To further increase utilization, test planners typically negotiate timing with engineers as schedules are constructed. For example, if 7 days remain unused on one vehicle and the test planner has a 10 day test needing to be scheduled that is otherwise compatible with that vehicle, it may be added by slightly shortening time allocated for each of the tests assigned to that vehicle.

While the overall schedule produced successfully demonstrated proof-of-concept for automating time-consuming aspects of the scheduling process, it also highlighted gaps that need to be addressed in future work. Specifically, more low-level details on department-specific compatibility requirements need to be incorporated into the models. Building the information systems that collect and provide the data necessary for such detailed models will be a process that takes time and is refined as the system is used in more programs.

One of the major successes of our first pilot was in crash test planning. The matching integer programming model in Section 5.3 was effectively applied to develop a crash test rehit strategy that was estimated by the safety engineer on the program to save weeks of manual planning time. The safety engineer supplied a list of tests, including the corresponding vehicle specifications, time required and deadline for each test. The initial compatibility relationships provided by safety were updated and the integer programming model was iteratively rerun until both the accuracy and feasibility of the results were verified by a team of experts within the safety department. The resulting plan supplied an optimal rehit schedule given the original specifications.

Table 5
Runtimes, in seconds, of test grouping subroutines and Fit-and-Swap heuristics on the resulting test sets.

Instance ID	No subroutine	Matching		Full IP	
	Fit-and-Swap	Subroutine	Fit-and-Swap	Subroutine	Fit-and-Swap
Instance 1	8.5	59	6.4	833[a]	5.2
Instance 2	3.3	2	2.6	8	2.8
Instance 3	3.2	<1	2.8	<1	3.0

[a] Note that for Instance 1 the full IP in the subroutine was not solved to optimality: after 60 min, the optimality gap was 7.7%; the time reported reflects time until finding the best feasible solution, which was passed to the Fit-and-Swap heuristic.

Our next step was to consider multiple what-if scenarios. In particular, the vehicle specification constraints that are incorporated in the overall compatibility constraints in the model were relaxed. The results highlighted strategic opportunities for shifting tests between prototype vehicles with different types of powertrains. By adopting some of these modifications, the safety department reduced their vehicle count by 2 vehicles, thereby increasing the number of rehits and vehicle utilization. The resulting savings for the program was estimated to be $500 K.

This project began in late 2011. During the first few months, the team focused on understanding the current methods and processes for test planning, developing the mathematical models presented in this paper, and implementing a prototype software application in Java. Following our initial pilot in 2012, we gained management support to transition from the proof-of-concept phase to developing a system with embedded optimization to be independently run by engineers. We are currently exploring more sophisticated optimization algorithms (e.g., column generation) to improve solution quality. We are focusing our optimization efforts on developing refined models for individual departments, which will enable a bottom-up approach to prototype test scheduling. In future work, we are also interested in incorporating uncertainty into our models, e.g., variability in test duration and vehicle release dates, factors which will enable us to develop even more robust schedules.

Acknowledgment

We thank the reviewers and editors for their thorough and thoughtful feedback that has certainly helped us improve this paper. We thank our colleagues in Product Development at Ford for their collaboration and contributions. This work was supported by a grant from the Ford-University of Michigan Innovation Alliance.

References

[1] Coffman EG, Jr., Garey MR, Johnson DS. Approximation algorithms for bin packing: a survey. In: Approximation algorithms for NP-hard problems. Boston, US: PWS Publishing Co.; 1996. p. 46–93.
[2] Elhedhli S, Li L, Gzara M, Naoum-Sawaya J. A branch-and-price algorithm for the bin packing problem with conflicts. INFORMS Journal on Computing 2011;23(3):404–15.
[3] Cheng T, Sin C. A state-of-the-art review of parallel-machine scheduling research. European Journal of Operational Research 1990;47(3):271–92.
[4] Potts CN, Strusevich VA. Fifty years of scheduling: a survey of milestones. Journal of the Operational Research Society 2009:S41–68.
[5] Sterna M. A survey of scheduling problems with late work criteria. Omega 2011;39(2):120–9.
[6] Cieliebak M, Erlebach T, Hennecke F, Weber B, Widmayer P. Scheduling with release times and deadlines on a minimum number of machines. New York City, US: Springer; 2004.
[7] Yu G, Zhang G. Scheduling with a minimum number of machines. Operations Research Letters 2009;37(2):97–101.
[8] Lee S, Turner J, Daskin MS, Homem-De-Mello T, Smilowitz K. Improving fleet utilization for carriers by interval scheduling. European Journal of Operational Research 2012;218(1):261–9.
[9] Kravchenko SA, Werner F. Minimizing the number of machines for scheduling jobs with equal processing times. European Journal of Operational Research 2009;199(2):595–600.
[10] Alidaee B, Li H. Parallel machine selection and job scheduling to minimize sum of machine holding cost, total machine time costs, and total tardiness costs. IEEE Transactions on Automation Science and Engineering 2014;11(1):294–301.
[11] Finke G, Lemaire P, Proth JM, Queyranne M. Minimizing the number of machines for minimum length schedules. European Journal of Operational Research 2009;199(3):702–5.
[12] Zhu Z, Heady RB. Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. Computers & Industrial Engineering 2000;38(2):297–305.
[13] Balakrishnan N, Kanet JJ, Sridharan SV. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. Computers and Operations Research 1999;26(2):127–41.
[14] Sawik T. An integer programming approach to scheduling in a contaminated area. Omega 2010;38(3):179–91.
[15] Kim DW, Na DG, Chen FF. Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. Robotics and Computer-Integrated Manufacturing 2003;19(1–2):173–81.
[16] Kurz ME, Askin RG. Heuristic scheduling of parallel machines with sequence-dependent set-up times. International Journal of Production Research 2001;39(16):3747–69.
[17] Heady RB, Zhu Z. Minimizing the sum of job earliness and tardiness in a multimachine system. International Journal of Production Research 1998;36(6):1619–32.
[18] Cao D, Chen M, Wan G. Parallel machine selection and job scheduling to minimize machine cost and job tardiness. Computers and Operations Research 2005;32(8):1995–2012.
[19] Radhakrishnan S, Ventura JA. Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. International Journal of Production Research 2000;38(10):2233–52.
[20] Liu C. A hybrid genetic algorithm to minimize total tardiness for unrelated parallel machine scheduling with precedence constraints. Mathematical Problems in Engineering 2013;2013:1–11.
[21] Sivrikaya-Srifoglu F, Ulusoy G. Parallel machine scheduling with earliness and tardiness penalties. Computers and Operations Research 1999;26(8):773–87.
[22] Lin Y-K, Hsieh F-Y. Unrelated parallel machine scheduling with setup times and ready times. International Journal of Production Research 2014;52(4):1200–14.
[23] Rabadi G, Moraga R, Al-Salem A. Heuristics for the unrelated parallel machine scheduling problem with setup times. Journal of Intelligent Manufacturing 2006;17:85–97.
[24] Chen J-F, Wu T-H. Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. Omega 2006;34(1):81–9.
[25] Chelst K, Sidelko J, Przebienda A, Lockledge J, Mihailidis D. Rightsizing and management of prototype vehicle testing at Ford Motor Company. Interfaces 2001;31(1):91–107.
[26] Bartels J-H, Zimmermann J. Scheduling tests in automotive R&D projects. European Journal of Operational Research 2009;193(3):805–19.
[27] Limtanyakul K, Schwiegelshohn U. Improvements of constraint programming and hybrid methods for scheduling of tests on vehicle prototypes. Constraints 2012;17(2):172–203.