Lessons Learned from a Schedule Optimization Tool for Prototype Vehicle Testing

Jeremy Ludwig[1], Annaka Kalton[2], Robert Richards[3], Brian Bautsch[4], Luke Walker[5]
Stottler Henke Associates, Inc.,[1,2,3]
Honda Development & Manufacturing of America, LLC [4,5]
{ludwig[1], kalton[2], richards[3]} @ stottlerhenke.com {bbautsch[4], lwalker[5]}@na.honda.com

Abstract

This industrial case study describes the customization of an existing general scheduling framework that generates solutions for the specialized and highly constrained problem of prototype vehicle test scheduling. In addition to creating optimized solutions, the deployed scheduling system both supports novice planners and integrates with existing processes. Going beyond the prior work, this case study focuses on the challenges encountered, updated implementation, and lessons learned from six years of operational use.

Introduction

Vehicle testing is an essential part of building new cars and trucks. Whether an auto manufacturer refreshes an existing model or builds a new one, the model will undergo hundreds if not thousands of tests. Some tests are exciting, such as a 56 km/h frontal flat barrier crash test measuring the impact on the crash-test dummies. Other tests are not quite as sensational but still important, like testing the heating and air conditioning system.

What these tests have in common is that they are generally carried out on hand-built prototype vehicles because the new factory lines for the models do not exist yet. These vehicles can each cost as much as a Bentley or Lamborghini, which results in pressure to reduce the number of vehicles. There are two additional complications with the test vehicles. First, the hand-built vehicles take time to build and are not all available at once, but instead become available throughout the testing process based on the *build pitch* of the test vehicles. An example of this is one new test vehicle being made available each weekday. Second, there are many particular types of a model, and each test might require a particular type or any of a set of types (e.g., any all-wheel-drive vehicle). There may be dozens of types of a particular vehicle model to choose from, varying by frame, market, drivetrain, and trim.

At the same time, market forces dictate when new or refreshed models must be released. The result is additional pressure to complete testing by certain dates so model production can begin.

Finally, testing personnel and facilities are limited resources. For example, it would be desirable to schedule all the crash tests at the very end of the project so other tests could be carried out on those vehicles first. However, there aren't enough crash labs or personnel to support this so the crashes must be staggered throughout the project.

This case study builds on prior work (Ludwig et al., 2016) that describes how an existing intelligent scheduling software framework was modified to include domain-specific algorithms and heuristics used in the vehicle test planning process.

The framework combines graph analysis techniques with heuristic scheduling techniques to quickly produce an effective schedule based on a defined set of activities, precedence, and resource requirements. These heuristics are tuned on a domain-specific basis to ensure a high-quality schedule for a given domain. An outer optimization loop embeds this tuned scheduling system. The heuristic-guided optimization incrementally attempts to remove vehicles, scheduling in between, to determine whether a lower vehicle count is feasible.

The resulting domain-specific scheduler is named Hotshot. The end product of this work is a deployed system that automatically generates a valid schedule from a set of constraints provided by the planner. The generated test schedule will complete the work in a given project time window and enforce all of the scheduling constraints if possible.

The schedule optimization process includes determining which vehicle types are built and the order in which they are built while *minimizing* the total number of vehicles required for the entire test schedule. Results using the deployed system were presented as part of Ludwig et al. (2016), where Hotshot was applied to a large-scale testing effort for a vehicle model update. This effort was not considered manageable using the existing manual scheduling process, so there

is no direct comparison to the pre-existing scheduling process.

As this system has been in continuous use since the initial publication, this case study focuses on the challenges encountered, updated implementation, and lessons learned from six years of operational use. Related work is presented first, followed by operational challenges, updated implementation, and a concluding section.

Related Work

The current version of the software extends prior work on the Hotshot system (Ludwig et al., 2014, 2016), which demonstrated the ability to generate a valid testing schedule with a significant reduction in the number of vehicles required relative to the existing planning process.

Schwindt & Zimmerman (Schwindt & Zimmermann, 2015) provide a thorough review of related work aimed at creating test schedules that respect testing constraints and minimize the number of prototype vehicles required.

The work presented in this paper is most similar to that of Limtanyakul and Schwiegelshohn (Limtanyakul & Schwiegelshohn, 2007, 2012). They use constraint programming to solve nearly the same problem of creating a test schedule for prototype vehicles. Both papers work towards a valid test schedule that meets the same scheduling constraints described previously (temporal, resource, ordering, build pitch, etc.), minimizes how many vehicles are built, determines the vehicle types to build, and determines the order in which the prototypes should be built according to a build pitch.

Bartels and Zimmerman (Bartels & Zimmermann, 2009) also worked on the problem of scheduling tests on prototype vehicles meeting temporal, resource, and ordering constraints while minimizing the number of vehicles required. Initially they use a mixed integer linear program model for smaller schedules, moving to a heuristic scheduling method to find solutions for larger schedules. They found that dynamic, multi-pass heuristics produced the best results. These are the same type of prioritization heuristics used in Aurora.

Zakarian (Zakarian, 2010) took a different approach in their prototype scheduling work for General Motors. They focused on developing a scheduling and decision support tool that considers the uncertainty in the test process, such as duration of tests, possibility of failure, and prototype availability. The tool helps users trade off between competing goals such as completing the tests according to schedule, quality of testing, and number of prototype vehicles required. Similar to their work, Aurora will highlight conflicted tests that cannot be scheduled because of insufficient resource availability in the given time frame.

Work done for Ford (Shi et al., 2017) on prototype vehicle test scheduling is also highly related, not only in the scheduling constraints but also in how engineers from each test department define the tests required in Excel spreadsheets. Their work reports good results from their pilot program using a *Fit-and-Swap* heuristic algorithm combined with an integer programming model for grouping crash tests on the same vehicle.

Glos et. al., (Glos et al., 2022) solve a similar problem as part of the BMW Quantum Computing Challenge "Optimizing the Production of Test Vehicles", though their particular problem focuses on configuration of test vehicles. Their approach is to formulate the schedule model as a satisfiability problem and then use hybrid constrained quantum annealing to minimize the number of vehicles required for testing. Their approach found results similar to classical solvers but required more time than classical solvers.

One primary difference from previous research is that our work focuses on domain specific customization of a general-purpose scheduling framework already in use in other applications. A scheduling framework takes advantage of the large degree of commonality among the scheduling processes required by different domains, while still accommodating their significant difference. This is accomplished by breaking parts of the scheduling process into discrete components that can easily be replaced and interchanged for new domains.

Framinan and Ruiz (Framinan & Ruiz, 2010) present a design for a general scheduling framework for manufacturing. Aurora, used in our work, is one example of an implemented scheduling framework (Kalton, 2006). Aurora distills the various operations involved in most scheduling problems into reconfigurable modules that can be exchanged, substituted, adapted, and extended to accommodate new domains (Ludwig et al., 2017). The OZONE Scheduling Framework (Smith et al., 1996) is another example of a system that provides the basis of a scheduling solution through a hierarchical model of components to be extended and evolved by end-developers. Becker (Becker, 1998) describes the validation of the OZONE concept through its application to a diverse set of real-world problems, such as transportation logistics and resource-constrained project scheduling.

Another difference from existing research is that the scope of the work presented in this paper extends beyond the prior work in several ways. The work presented in this paper is part of a deployed system that includes visualization, analysis, and integration with existing processes; is currently in use by novice planners; includes methods to identify and automatically resolve common types of modeling errors created by novice planners; and includes methods to transition the testing schedule from planning stage to execution phase.

Operational Challenges

Challenge 1: Manually Entered Request Data

The test information and corresponding vehicle requirements are entered manually into individual Excel spreadsheets by many department project managers and then combined on import. This data entry process results in several pitfalls, including poorly formatted data, missing data, and data that is logically incompatible with other information in the request.

For example, a master spreadsheet indicates which vehicle types are available for the current test cycle. Individual departments independently enter information about the tests they need to perform, and which vehicle types would work for those tests. The department information can be correctly formatted and accurate, but if there is no overlap between their required vehicles and the available vehicles, they have effectively defined a test that cannot be satisfied.

Checks for these types of data issues have been added incrementally. In each case the verification process now includes a default data repair and notification for the user. The goal with this challenge is to allow the user to proceed without manually repairing the data, while still warning them that some of the results may be problematic.

Challenge 2: Model Consistency Complications

There are a range of model issues that are not data entry issues, but instead reflect a collision between the defined model constraints and reality. This category of issue cannot readily be identified as part of the import process. They only become apparent when the model is scheduled or analyzed. A few examples include:

The number of vehicles required to support testing cannot be built with the specified build pitch, within the bounds defined for the project. In this case, the user needs to extend the project bounds, increase the build rate, or eliminate test duration.

A series of inter-constrained tests will not fit within the bounds defined by the earliest possible availability of a vehicle type option and the project end. In this case, the test series would be treated as an exception to the project end (effectively violating the target end), but the user would be informed of the override.

A series of tests are supposed to be performed on the same vehicle, but the actual referenced vehicle types are incompatible.

Test 1 requires vehicle type A or B.

Test 2 requires vehicle type B or C.

Test 3 requires vehicle type C.

This set of requirements may pass a preliminary check, but on further analysis, the intersection of (A | B), (B | C), and C is empty.

As with challenge 1, the goal is to ensure that the user can get preliminary results as quickly as possible, while still being notified of the model issues. When possible, the model is repaired with a default strategy and the user is notified of the change. Otherwise, the user must address the model issue in the excel spreadsheets before continuing.

Challenge 3: Shifting User Goals

In the original Hotshot implementation, the dominant use case was for the user to define the desired vehicle types, and the maximum number of vehicles of each type. Within those constraints, Hotshot would then attempt to find a solution involving as few individual vehicles as possible. However, more recently the focus has shifted to determining whether a defined test set can be accomplished with a given overall number of vehicles.

On the surface, this seems like a very similar problem, but it has some fundamental differences. The two most significant differences are that vehicle types can be added (an operation that was not supported in the original implementation), and the optimization target is different (meeting a total number of types, while taking secondary criteria into account).

This shift in focus also reflects a wish to be able to check whether the target vehicle count is attainable quickly, easily, and with rough data. This reflects both the common data issues noted in the first two challenges, as well as the human tendency to over-ask. That is, the first set of requests is usually too demanding and so the user needs a fast analysis to determine whether to follow up with the department project managers to reduce and fine-tune their demands.

Updated Implementation

The original implementation, which was based on heuristic iterative optimization using repeated scheduling cycles to reduce vehicles, was computationally intensive, and could take many minutes for a large model. Conflicts caused by data issues discussed in challenges 1 and 2 also prevented some aspects of the optimization from working properly, since normally it uses the presence or absence of conflicts to determine whether the vehicle reduction is feasible. As such, it is not well suited to quickly answering the new question of whether the test requests are feasible with the target vehicle count, with data that may be preliminary or problematic.

The original implementation's strength is that it was based on a more general scheduling system, and so it could handle novel situations in the data, in many cases without any code modification or tailoring. However, it could not truly take advantage of the dominant structure of the domain. Also, it could not take shortcuts to produce the sort of analysis that would allow the user to produce a rough but reasonable schedule with dirty data.

To answer this new need, without losing the generality presented by the original implementation, we implemented a pre-optimization analyzer. Its goal is to take advantage of the problem's structure to quickly give a preview of likely vehicle utilization. Although it will miss edge cases that the full optimizer can cover, it can provide the user with a vehicle utilization preview in seconds instead of in many minutes. Achieving this goal is a four-step process.

Combine Related Tasks

A number of the tests are series of tests that are supposed to be performed sequentially on the same vehicle. Combining many tests into a single meta-test chunk makes reasoning significantly easier. An example of this is shown in Figure 1, where ten tests are combined into a single test for Vehicle 2.

Construct a Graph of Task Chunks and Vehicle Types

This graph-based model allows each task chunk to know what vehicle type(s) it could use, and each vehicle type to know which task(s) want to use it. The graph-based structure makes it easy to move tests from one vehicle type to another. The process of producing this graph also provides useful statistics about overall type-based utilization that can help guide the assignment process. An example of this is shown in Figure 2.

Iteratively Assign and Refine Vehicle Instances

The algorithm then iteratively adds test tasks to vehicle instances, starting with the longest and/or most restrictive test tasks. Tasks may be added to vehicle instances that are already in use, or they may require a vehicle to be added to the build list. As tasks are added, the algorithm checks for issues and bottlenecks, and may potentially reallocate an earlier task, change the vehicle build order (moving test time from one vehicle to another), or make other common, minor model fixes.

The graph structure makes these changes computationally inexpensive and allows the model to remain fluid. This makes it easier to perform localized optimization at different points in the process and makes it easier to pursue secondary goals (e.g., keep a vehicle for a specific department, rather than switching among three departments).

Preview and Reporting

When the analysis is complete, the schedule model is updated to show the version of the schedule that the analyzer produced. This is then used to prime the regular scheduling process, which allows the regular scheduling algorithm to give the same outcome if there are no significant edge cases missed by the analyzer. It also provides the user with detailed utilization information.

This reporting makes it easier for the user to determine whether there is significant flexibility in the schedule. For example, if they were hoping to use 75 vehicles, and the analysis produced a preview scheduling using 76 vehicles, all of which are at 80% load, it is unlikely that 75 vehicles is actually feasible without adjusting the test parameters.

However, they might then look at the vehicle statistics, and see that one department has requested 400 days of test time. If that is double what they used in the last test cycle, then the user can go back and double-check the data with the department. The objective throughout is to allow the user to check the feasibility of their goal quickly, easily, and robustly.

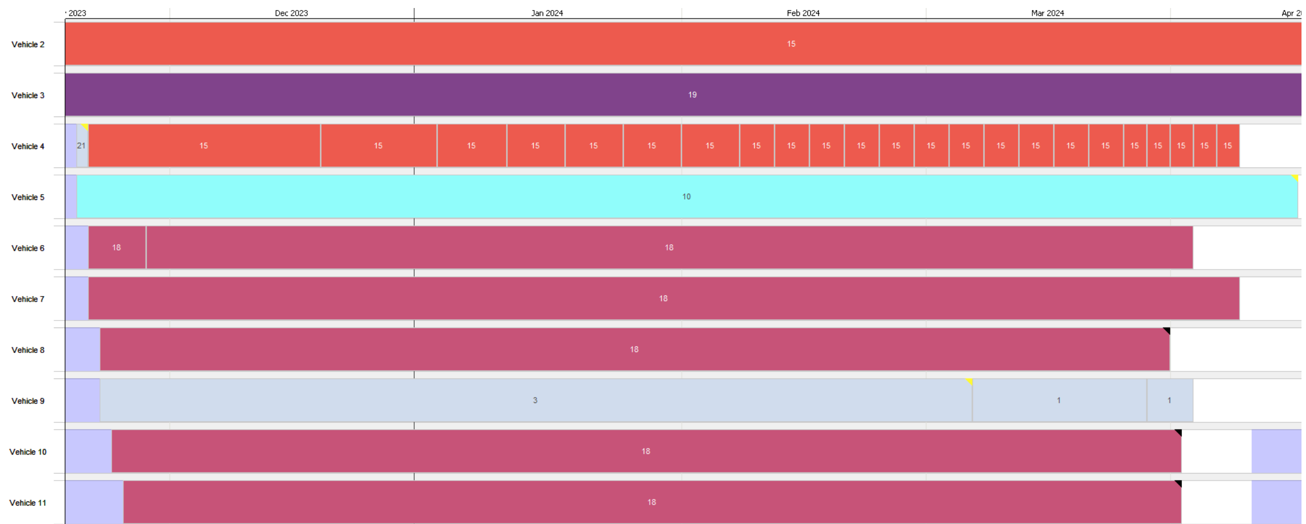

Figure 1. Combining individual tasks into chunks. Colors note departments.

Figure 2. Striping indicates viable vehicle types for each task. Tasks with just a corner tag can only use one vehicle type, multiple corner tags indicate greater flexibility.

Conclusion

The primary lesson learned over the last few years is that the project team greatly underestimated the difficulties that departments would have in using the Excel input format. Even with a plethora of data checks embedded in the Excel file, and even more checks during the import process, it is simply too easy to create the ill-defined scheduling models described in Challenges 1 and 2. This is especially true in a task like this that is not done very often so no expertise is developed. The result was a burden on the scheduler, who needs to work with each department to update their portion of the schedule. The updated solution that highlights the issues, repairs as best as possible, and lets the scheduler get on with their job of creating a rough estimate is a significant usability improvement.

Challenge 3, shifting user goals, is not a lesson learned so much as another example of an old adage – the only constant is change. In this case, the modular architecture of the Aurora scheduling framework allowed us to pivot and respond to the client's needs while limiting the scope of the required changes.

In the short term, the initial scheduling and optimization process is still used to ensure all edge cases are covered. Our ongoing work focuses on adding the more common of those edge cases to the analysis step, making follow-on scheduling less and less necessary.

References

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

Becker, M. A. (1998). *Reconfigurable Architectures for Mixed-initiative Planning and Scheduling*. Carnegie Mellon University.

Framinan, J. M., & Ruiz, R. (2010). Architecture of manufacturing scheduling systems: Literature review and an integrated proposal. *European Journal of Operational Research*, *205*(2), 237–246. https://doi.org/10.1016/j.ejor.2009.09.026

Glos, A., Kundu, A., & Salehi, Ö. (2022). Optimizing the Production of Test Vehicles using Hybrid Constrained Quantum Annealing. *ArXiv Preprint ArXiv:2203.15421*.

Kalton, A. (2006). Applying an Intelligent Reconfigurable Scheduling System to Large-Scale Production Scheduling. *International Conference on Automated Planning & Scheduling (ICAPS) 2006*. International Conference on Automated Planning & Scheduling (ICAPS) 2006, Ambleside, The English Lake District, U.K.

Limtanyakul, K., & Schwiegelshohn, U. (2007). Scheduling tests on vehicle prototypes using constraint programming. *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications*, 336–343.

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

Ludwig, J., Kalton, A., Richards, R., Bautsch, B., Markusic, C., & Jones, C. (2016). Deploying a Schedule Optimization Tool for Vehicle Testing. *Proceedings of the 10th Scheduling and Planning Applications WoRKshop (SPARK)*, 44–51.

Ludwig, J., Kalton, A., Richards, R., Bautsch, B., Markusic, C., & Schumacher, J. (2014). A Schedule Optimization Tool for Destructive and Non-destructive Vehicle Tests. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2998–3003. http://dl.acm.org/citation.cfm?id=2892753.2892967

Ludwig, J., Richards, R., Kalton, A., & Stottler, D. (2017). Applying a heuristic-based scheduling framework in manufacturing, service, and communication domains. *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 1–4. https://doi.org/10.1109/SMC.2017.8122568

Schwindt, C., & Zimmermann, J. (2015). *Handbook on project management and scheduling vol. 1*. Springer.

Shi, Y., Reich, D., Epelman, M., Klampfl, E., & Cohn, A. (2017). An analytical approach to prototype vehicle test scheduling. *Omega*, *67*, 168–176.

Smith, S. F., Lassila, O., & Becker, M. (1996). Configurable, Mixed-Initiative Systems for Planning and Scheduling. In A. Tate (Ed.), *Advanced Planning Technology*. AAAI Press.

Zakarian, A. (2010). A methodology for the performance analysis of product validation and test plans. *International Journal of Product Development*, *10*(4), 369–392.