Deploying a Schedule Optimization Tool for Vehicle Testing

Jeremy Ludwig, Annaka Kalton, and **Robert Richards**
Stottler Henke Associates, Inc.
San Mateo, California
{ludwig, kalton, richards} @ stottlerhenke.com

Brian Bautsch Craig Markusic, and **Cyndi Jones**
Honda R&D Americas, Inc.
Raymond, OH
{ CMarkusic, Bbautsch, CJones } @ oh.hra.com

Abstract

Whenever an auto manufacturer refreshes an existing car or truck model or builds a new one, the model will undergo hundreds if not thousands of tests before the factory line and tooling is finished and vehicle production begins. These tests are generally carried out on expensive, custom-made prototype vehicles because the new factory lines for the model do not exist yet. The work presented in this paper describes how an existing intelligent scheduling software framework was modified to include domain-specific heuristics used in the vehicle test planning process. The result of this work is a scheduling tool that optimizes the overall given test schedule in order to complete the work in a given time window while *minimizing* the total number of vehicles required for the test schedule. The tool was validated on the largest testing schedule for an updated vehicle to date. This model exceeded the capabilities of the existing manual scheduling process but was successfully handled by the tool. Additionally the tool was expanded to better integrate it with existing processes and to make it easier for new users to create and optimize testing schedules.

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 48 km/h dynamic rollover and 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 an ultra-luxury 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 of 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.

To summarize, the constraints placed on creating a valid schedule in this domain are:

- **Temporal**: Tests must be scheduled between the project start and end date; each test has duration and an optional start date and an optional end date.
- **Calendar**: Tests can only be scheduled during working shifts; tests cannot be scheduled on holidays.
- **Ordering**: Tests can optionally be assigned to follow either immediately after another test or sometime after another test.
- **Resource**: Each test can only be scheduled on certain vehicle types; tests may optionally be required to use the exact same vehicle as another test; tests may require personnel to be available; and tests may require facilities to be available.
- **Build Pitch**: Vehicles are not available for tests until the date they are created; creation dates follow a given build pitch schedule with additional constraints.
- **Exclusive**: Test indicated as exclusive must be the first test on the selected vehicle.

- **Destructive**: Tests indicated as destructive must be the last test on the selected vehicle.

The work presented in this paper describes how Aurora, an existing intelligent scheduling software framework (Kalton, 2006), 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. The resulting domain-specific scheduler is named Hotshot.

The end product of this work is a deployed system that automatically creates a valid schedule from a set of constraints provided by the planner. The created test schedule will complete the work in a given time window and observe all of the scheduling constraints. The schedule optimization process includes determining which vehicle types are built and the order in which they are built and *minimizes* the total number of vehicles required for the entire test schedule.

Results from the deployed system are presented from applying the system 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. Prior work reported elsewhere does include a direct comparison between Hotshot and the previous scheduling process with a 12% reduction in number of vehicles required (Ludwig et al., 2014).

In the remainder of this paper, we first discuss related work. Following this we describe the Aurora scheduling framework and summarize changes made to create the domain-specific Hotshot scheduling tool, focusing on the features added to support the transition from prototype to deployed system. The methods and results sections contains the details of how the deployed system was validated by creating one of the largest test schedules for a single vehicle model to date. Finally, we present future work in the conclusion.

The primary contributions of this case study are describing the customization of an existing general scheduling framework to solve a specialized and highly constrained problem and discussing the requirements included in deploying a scheduling system that both supports novice planners and integrates with existing processes.

Background and Related Work

The current version of the software extends the prototype Hotshot system (Ludwig et al., 2014), which demonstrated the ability to generate a valid schedule with a significant reduction in the number of vehicles required relative to the existing planning process. The deployed version of Hotshot includes a number of significant improvements to the initial prototype.

Schwindt & Zimmerman (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 (2012, 2007). 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 (2007) 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 (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.

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 (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 (e.g., Richards, 2015; Richards, 2010a; Richards, 2010b). 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 (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, including the Hotshot prototype, in a number of 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.

Scheduling Framework

Aurora was designed to be a highly flexible and easily customizable scheduling system. It is composed of a number of components that can be plugged in and matched to gain different results. The scheduling system permits arbitrary flexibility by allowing a developer to specify what components to use for different parts of scheduling. Aurora has been successfully applied in a multitude of domains, including medical, manufacturing, and aerospace. (Richards, 2015; Richards, 2010a; Richards, 2010b). The steps in the scheduling process are described in detail below. All configurable elements are shown in bold. Elements that were modified for the test vehicle domain will be discussed further in later sections.

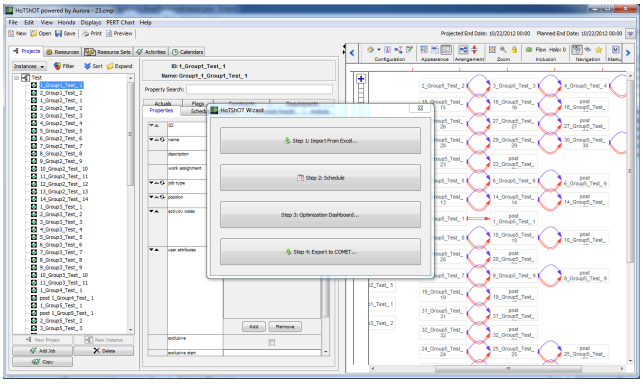

Figure 1. Aurora scheduling user interface.

Additionally, Aurora includes a default user interface that is further customized for each domain. As shown in Figure 1, a domain specific wizard to walk the user through the scheduling process is shown in the center. Behind this are parts of the standard interface: list of tasks (left), information on the selected task (center), and a network diagram (right) showing the ordering constraints between tasks. Other views include resources and calendars as well as various reports and graphs.

Scheduling Process

Schedule Initialization

1. Aurora undoes any previous post-processing (to get back to the "true" schedule result state) and applies the **Preprocessor** to the schedule information.
2. Aurora uses the **Queue Initializer** to set up the queue that will be used to run the scheduling loop. A standard Queue Initializer puts some or all of the schedulable elements—activities, flows, and resources—onto the queue.
3. The queue uses the **Prioritizer** to determine the priority of each element. Depending on the execution strategy, these priorities may be used to periodically sort the queue or to schedule the element with the highest priority at each stage.
4. The Schedule Coordinator triggers the scheduling of the elements on the queue by starting the Scheduling Loop.

Scheduling loop

1. A schedulable element (task, project, or resource) asks the **Scheduler** to schedule it.
2. The Scheduler calls constraint propagation on the schedulable so as to be sure that all of its requirements and restrictions are up to date.
3. The Scheduler looks at the element, considers any **Scheduling Method** that is associated with it (e.g., Forward, Backward). A Scheduling Method determines how the system goes about trying to schedule an element. The Scheduler also selects which **Quality Criterion** to associate with the selected scheduling method; the Quality Criterion determines what makes an assignment "good."
4. The Scheduler calls the Schedule Method on the schedulable. The process depends a great deal on the Schedule Method, but the result is that the schedulable element is assigned to a time window and has resources selected to satisfy any resource requirements. It also returns a list of the conflicts resulting from the given assignment.
5. The Scheduler calls constraint propagation on the schedulable (again) in order to update all of the neighbors so that they are appropriately restricted by the newly scheduled element. This process may result in additional

conflicts; if so, these are added to the list of conflicts from scheduling.
6. The Scheduler adds the conflicts to the **Conflict Manager** and asks the manager to attempt to resolve those conflicts.

Schedule Finalization
1. When the queue is empty, Aurora goes through a final conflict management step, this time at the global level.
2. Aurora calls the **Postprocessor** on the schedule, so that any additional analysis may be done before Aurora returns the schedule results.
3. Aurora sends the schedule results to the GUI for display.

Domain-Specific Customization

Two different types of modifications were made to the Aurora framework to create the Hotshot tool. First, the user interface front end was modified to import the testing model, display and edit domain-specific properties, perform the optimization to minimize the number of required vehicles, and to search for additional resources that could be added to shorten the project schedule. Second, components in the scheduling back end were updated specifically for this domain.

User Interface Customization
There are seven features added to the general scheduling user interface that are specific to the vehicle test domain: import an Excel model of the testing problem, edit the build pitch, edit the vehicles and build order, determine how to handle irregular tasks, minimize the number of vehicles required, search for additional resources that could be used to shorten the schedule, and export the schedule to a client-specific format. Each of these features will be described in greater detail, highlighting new features and changes made to existing features as the user base of the system grew.

The starting point of the Aurora customization for the vehicle testing domain is importing the testing tasks, task constraints, calendars, resources (vehicles, vehicle types, personnel, facilities), and build pitch information from a set of Excel spreadsheets. These Excel spreadsheets represent a model of the overall testing problem. Once imported, the general user interface supports graphically viewing and editing the model elements such as tasks, resource requirements, resources, resource sets, constraints, and calendars. For the deployment, the primary additions were in model error checking. In practice, multiple team leaders specify portions of the overall model. This leads to situations where models are defined that are logically impossible to solve. One example is when there are more days of work indicated than there are days of vehicles available given the build pitch and end date. The deployed system looks for common mistakes made with the prototype and alerts the user to the problem using familiar terms that they can easily understand.

The Build Pitch (Figure 2) dialog was added for viewing and editing the general build pitch per week (number of vehicles that can be built) as well as a maximum for each vehicle type per week. For example, 10 test vehicles per week can be built, but only 5 all-wheel-drive can be built in a week. Additionally, each vehicle type has a first available date, which determines the earliest date that a vehicle of that type could be available. The Manage Vehicles dialog is related to Build Pitch, allowing the user to edit the vehicles to be built and their build order. The build order is assigned automatically during the optimization process. Build dates are assigned based on the build order, moving from 1 to n and selecting the first available date that meets two criteria: number of vehicles/week is not exceeded and vehicle type per week is not exceeded.

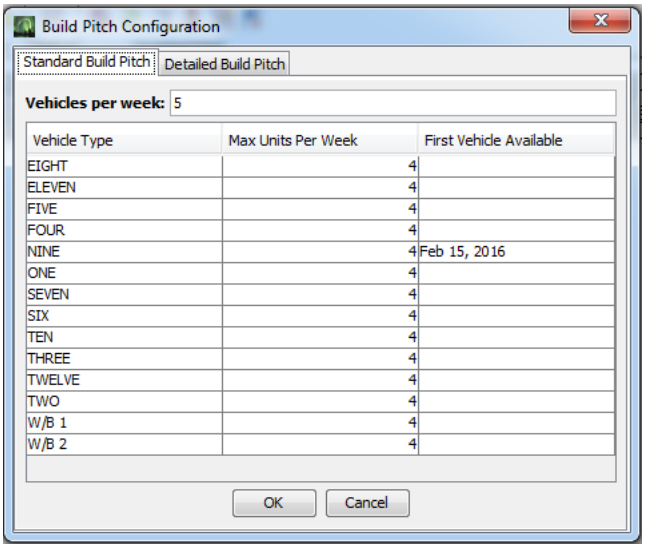

Figure 2. Build pitch configuration.

While the general build pitch can be used at the start of the scheduling process, part-way through the test process a more detailed build pitch is released that gives specific dates in which vehicles will be available, for example, 3 vehicles on Monday the 1st. Later in the test process, the information is even more detailed, with specific dates given to each test vehicle to be built. The deployed version of Hotshot supports the existing planning process by including all three of these modes and updating the schedule as the model transitions from most general to most specific build pitch.

Greater use of the system also indicated that an additional dialog to handle tasks with irregular dates was needed. The Set Task End Dates dialog was added to support tasks that were allowed to extend beyond the

desired project end date. Examples of this include tasks with a given end date later than the project end date, individual tasks that are longer than the entire project duration, and chains of tasks linked with ordering constraints such that the combined duration exceeds the entire project duration. In all these cases, the user determines if the entire project should be extended or if the task(s) will be allowed to complete at a date past the project end date.

The Optimization Dashboard (Figure 3) is used to minimize the number of vehicles required to schedule the testing tasks. In Hotshot, optimization is accomplished by using the scheduling engine to perform a search through the space of schedules to find a valid schedule that requires fewer vehicles. The Optimization Dashboard helps visualize the setup and search process for the user. The upper left area of Figure 3 summarizes the present state of the current schedule, showing the number of vehicles required, the number of destructive and exclusive tasks, the utilization of vehicles in the testing schedule, and the actual project end date relative to the initial. The upper right shows the current status of optimization, which will change once the Start button is pressed. This portion of the dialog also provides an estimate of how long the remaining optimization will take. The central portion of the dialog contains the five parts of the optimization process. Buttons for starting and controlling the optimization are found along the bottom of the dialog.

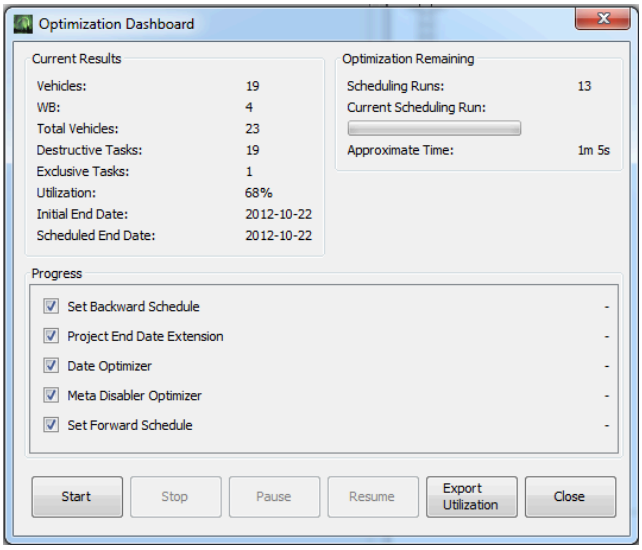

Figure 3. Optimization dashboard.

There are five steps in the optimization process. The first three steps prepare the scheduling model for search, the fourth step carries out the search, and the final step returns the schedule to the end user format:

1. **Set Backward Schedule.** Mark all tasks to be backward scheduled. This means that the schedule will be created from the end of the project to the beginning, with all tasks scheduled as close to their late end dates as possible.
2. **Project End Date Extension.** Users of the prototype often ran into problems when the desired model was too ambitious for a solution to be found. This optimization step was added to correct for overly optimistic end dates. The end date extension attempts to schedule the project with all of the constraints except build pitch. If conflicts are found, it extends the project end date to try to fix the issues for the user. If conflicts still exist, optimization is aborted. At this point the planner will need to refer to the conflicts to fix the model issues.
3. **Date Optimizer.** Once the tasks are backward scheduled, assign build order based on the earliest dates tasks are assigned to vehicles. That is, if the first task assigned to Vehicle A starts on Jan 15 and the first task assigned to Vehicle B starts on Jan 18, then A will come before B in the build order. The heuristic is that vehicles that are needed earlier should be built earlier. Note that this optimizer greatly reduces the amount of time available to test vehicles built later in the schedule. Due to the same issue with ambitious test models as seen in step 2., this optimizer will also attempt to extend the project end date if conflicts are found once build dates are applied.
4. **Meta Disabler Optimizer.** This step uses the scheduling engine as part of a greedy search for valid schedules that require fewer vehicles. Starting with the vehicles created last in the schedule, temporarily disable the vehicle and use the scheduling engine to try to create a schedule without the vehicle. If this succeeds, permanently delete the vehicle. If this fails, restore the vehicle and continue. As vehicles are removed, the Date Optimizer is used to re-order the build dates of the remaining vehicles.
5. **Set Forward Schedule.** Finally each task is returned to forward schedule mode and re-scheduled so that all tasks try to schedule as close to the project start date as possible. This is the preferred output format for downstream processes that make use of the schedule created by Hotshot.

The Resource Analysis dialog was developed to provide guidance to new users on how they could improve the schedule, either by reducing the number of vehicles required or by shortening the project duration. Starting from an optimized schedule, the Resource Analysis dialog carries out a meta-search process. For each resource (vehicle type, personnel, or facility), the dialog will perform the optimization process as if another of that resource were available. The user is shown the effect of adding each resource individually to the schedule in terms of project end date and number of cars required. This serves as a starting point for discussion on what-if scenarios for improving the schedule.

The final feature is aimed at making the scheduling results easier to use as part of the larger process of

planning for and carrying out tests. Aurora contains a variety of highly customizable displays such as the plot shown in Figure 4. In this figure the actual vehicle and tasks have been simplified and obfuscated. Vehicles are shown on the x-axis and time on the y-axis. For example, SIX~001 is the first vehicle instance of type SIX. It has the task 9_Group2_Test_9 assigned to it from August 9 to October 10. The light-purple cell on August 8 to the left of the task indicates the vehicle is not yet available for scheduling, visualizing the build pitch. The plot has also been customized to color code tasks by group and to indicate destructive tasks with a yellow tag in the upper right corner. However, an existing format of scheduling results is already in use as part of the vehicle testing process. The deployed version of Hotshot includes custom export capabilities to inject the Hotshot results into the existing, proprietary system.

Scheduling Component Customization

The main change for the deployed scheduling system was to support irregular tasks that are allowed to complete outside of the project start and end dates.

The scheduling model is based on a hierarchical structure with flows and tasks. Flows represent high-level projects made up of individual tasks that must be scheduled. One of the key constraints on flows and tasks is the *late end date*. Late end date represents that last possible date the project or task can be completed. Under normal conditions, the late end date of the flow will constrain the late end date of any of the tasks that make up the flow. For example, Task A has a defined late end date of 12/31/2015. Task A is part of Flow "Test Project," which has a late end date of 12/01/2015. The **Preprocessor** synchronizes the end dates of the tasks and the flow that contains the tasks. So in the above example, Task A will be updated to have a late end date of 12/01/2015 because it is constrained by the flow end date. In real-world conditions, the flow late end date describes when the bulk of the tasks need to be finished by, but there are some tasks that can be safely finished after the project is considered complete. To support this, tasks can be marked as *override project end date*. The updated **Preprocessor** does not change the late end dates of these tasks.

The Hotshot prototype component customization focused on three central areas: scheduling direction maintenance, special handling for vehicle testing's unusual requirements, and more standard heuristic tailoring for the domain. In this paper we focus on the **Prioritizer** component, which was not described in detail previously. See Ludwig et al. (2014) for the customization of other scheduling components: **Preprocessor**, **Scheduler quality criteria**, **Scheduler**, and **Post-processor**.

The **Prioritizer** uses a cascading series of heuristics to determine which activities should be scheduled earlier in the process. In general, if "difficult" activities in the given domain are scheduled earlier in the process, it tends to avoid subsequent conflicts in the schedule that would be difficult to repair. The heuristic prioritizer considers each heuristic in turn, until it can differentiate between two prospective activities. If two activities tie on a given heuristic (e.g. both are "exclusive use" activities), the prioritizer will consider the next heuristic ("long task"), and the next, until it can break the tie. The primary heuristics in this domain are:

- **Exclusive task**: Prefer to schedule activities that must have exclusive use of a vehicle earlier in the process.
- **Long task**: Schedule the long activities early in the process and fill in with short activities.

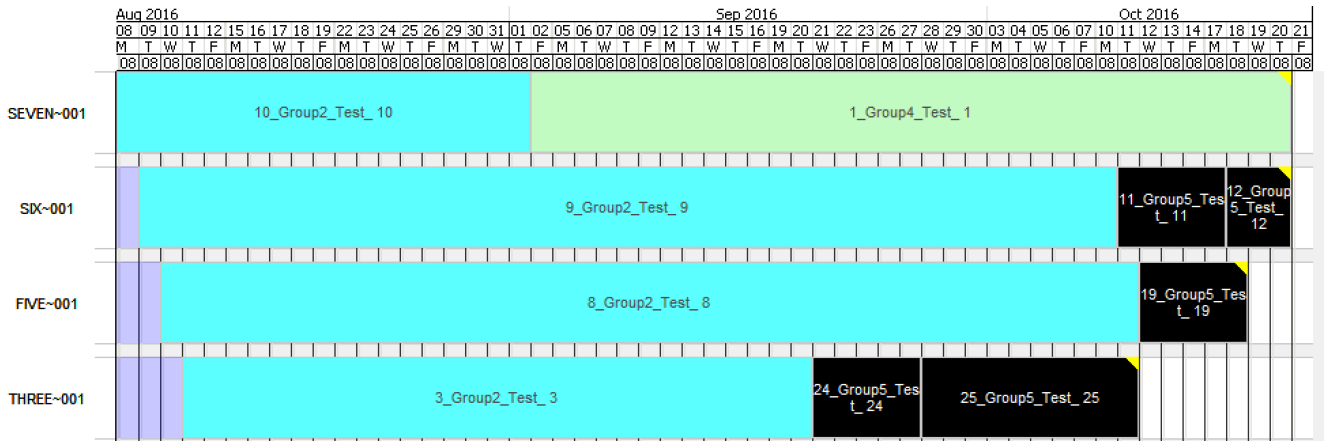

Figure 4. Aurora assignment of tasks to vehicles over time. For example, SIX~001 is the first instance of a vehicle of type SIX. It is available for tasks starting August 9th. The first task assigned to this vehicle is 9_Group2_Test_9.

- **Destructive task**: Some tests involve destroying the vehicle. This prevents any activities from subsequently making use of the vehicle, so it is important to place the destructive tasks - as late in the schedule as possible - early in the scheduling process.
- **Tight window**: This reflects the fact that activities with a short window of opportunity tend to be harder to place than those with a long window of opportunity. In this case, the "tightness" reflects the difference between task duration and the projected window size.
- **End based**: Schedule tasks that must be completed first earlier in the process for the forward-schedule phase.
- **Load-based**: Prefer activities with fewer vehicle options and/or more competition for those vehicles.
- **Subsequent duration**: Considers the amount of follow-on work after the current activity, based on ordering constraints.

Methods

Based on the prototype results, Hotshot was immediately put to work on a large project that had several challenging constraints from the start. The first challenge was that the factory in charge of this project had capacity issues and was not able to build enough vehicles to satisfy all of the testing and development requirements. As a result, a second factory was sourced to make up the shortage of vehicles the first factory was unable to produce. Developing an optimized test schedule manually for this type of build—one that included two build locations, two separate build pitches, and two different timelines—had not been attempted previously.

To tackle this challenge, the project leader followed a divide and conquer approach. Instead of treating this project as one very large schedule, it was divided into two medium-sized schedules that were individually optimized. To carry this out, the project leader separated the exclusive tasks from the non-exclusive tasks. All exclusive tasks were to be scheduled and optimized at the second factory while the non-exclusive tasks were to be scheduled and optimized at the first factory.

The second challenge centered on negotiations to create schedules that worked for both the testing team and the factories building the test vehicles. The schedule that was using the first factory, which was to build vehicles for the non-exclusive tasks, was created first. Ongoing negotiations took place with this factory with regard to build timing and build pitch. This factory had several assignments it was balancing and had to make changes and requests in real time during the schedule optimization process. These requests were fed back to the project team, which utilized Hotshot to update the test schedule. Most of the change requests involved when to build certain vehicles and how many vehicles to build per week. The project team was able to honor the factory's requests as well as counter-propose options that would help further optimize vehicles and schedules.

After the non-exclusive tasks' schedule was created, the exclusive schedule was started. The second factory was an in-house fabrication department that had different requirements and constraints than those of the first factory. However, the same process of creating a test schedule was used in this case. Build pitch and build timing were considered when performing the optimizations and schedule creation.

Results

In the end, the team was able to create an optimized schedule with a larger number of vehicles (50 – 150) that met all of the non-exclusive test groups' needs. The second exclusive task schedule was created using a smaller number (10 – 50) of test vehicles. In total, a large number of vehicles were needed to satisfy the project requirements and testing needs, drawing from 30 – 50 different vehicle types. This included 4042 days of testing, with over 340 testing tasks. These tasks were constrained by the completion of preceding tasks, by requiring the use of the same resources as preceding tasks, and by the availability of vehicles, personnel, and testing facilities. In addition to successfully scheduling a suite of tests that would have been very difficult previously, Hotshot also supported the negotiation process and minimized the number of test vehicles required.

Working with the initial prototype, planners demonstrated the ability of generating a schedule in under two minutes, as opposed to this task requiring days of labor. This capability enabled the planners to generate numerous "what-if" scenarios. Planners could quantify the effect of compressing or extending the schedule in terms of how many cars would be required. Planners also demonstrated the effects that steeper and shallower build pitches have on the number of cars required for a given set of tasks and project end dates. Planners were also able to negotiate about the vehicle types required by tests. For example, a vehicle type requested by one test but not usable by other tests stands out in plot as a vehicle with very low utilization. The planner can then go back to the person in charge of the test and see if a more commonly used vehicle type could be substituted. The ability to quickly run "what-if" scenarios held true with the larger models as well. The ability to quickly examine these types

of effects enabled a more efficient negotiation process to take place between the test and vehicle production teams than during previous challenging projects.

Hotshot was also used to minimize the number of test vehicles required for this large project. Unfortunately, there is no direct comparison to the previous method because no manual model was attempted given the complexity of the test schedule. The only estimate we can give for the number of vehicles saved is based on the combined judgment from several members of the project team. They estimated that Hotshot created at least six fewer vehicles than would have been created with the previous method. This represents a 6% reduction in the number of vehicles required and a significant cost savings in the millions of dollars.

Note that this estimate was purposely conservative. The prototype version (Ludwig et al., 2014) demonstrated a 12% reduction in vehicles when directly compared with the manually created schedule on a much smaller model.

Conclusion

This paper described a complex, real-world scheduling problem in automotive vehicle testing prototype management. To address this problem, we added domain-specific heuristics to a general intelligent scheduling software framework to create the custom Hotshot scheduling software.

Hotshot helped solve a very complex scheduling challenge in the presented use case. Solving this challenge with the previous, manual method would have been almost impossible. As deployed, Hotshot enabled the schedules to be created in an efficient manner while also building fewer vehicles than the manual method would have needed.

Due to the reduction in required vehicles, this use case also demonstrates the cost-effective development of a customized scheduling system. The savings from the reduced vehicles alone in the presented use case greatly outweighs development cost, and additional savings are generated with each new project. Hotshot has already saved the end user millions of dollars in prototype costs while increasing transparency of the entire process from the implementation level to the executive level.

Ongoing work is aimed at scaling Hotshot, and its optimization capabilities, to multiple simultaneous projects. Currently, Hotshot is used to optimize a single project. The functionality around build pitch allows constraints caused by vehicle availability for multiple projects to be factored into the schedule. However, the schedule does not take into account delays that could be introduced due to conflicts in using limited personnel and testing facilities for different projects being run at the same time. The next step in development will assist planners in creating a combined schedule for all of the active testing projects at any given time.

References

Bartels, J.-H., & Zimmermann, J. (2007). 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. Ph.D. diss., Robotics Institute and Graduate School of Industrial Administration, Carnegie Mellon university, Pittsburgh, PA.

Framiñan, 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.

Kalton, A. (2006). Applying an Intelligent Reconfigurable Scheduling System to Large-Scale Production Scheduling. *International Conference on Automated Planning & Scheduling (ICAPS) 2006*. Ambleside, The English Lake District, U.K. June 6-10, 2006.

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

Limtanyakul, K., & Schwiegelshohn, U. (2007). Scheduling tests on vehicle prototypes using constraint programming. In *Proceedings of the 3rd multidisciplinary international scheduling conference: Theory and applications* (pp. 336–343).

Ludwig, J., A. Kalton, R. Richards, B. Bautsch, C. Markusic, J. Schumacher (2014). A Schedule Optimization Tool for Destructive and Non-Destructive Vehicle Tests. *Proceedings of the Twenty-Sixth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI 2014)*

Richards, R. (2010a). Critical Chain: Short-Duration Tasks & Intelligent Scheduling in e.g., Medical, Manufacturing & Maintenance. *Proceedings of the 2010 Continuous Process Improvement (CPI) Symposium.* Cal State University, Channel Islands. August 19-20, 2010.

Richards, R. (2010b). *Enhancing Resource-Leveling via Intelligent Scheduling: Turnaround & Aerospace Applications Demonstrating 25%+ Flow-Time Reduction. 2010 PMI College of Scheduling Conference PMICOS. Calgary, Canada. May 2-5, 2010.*

Richards, R. (2015). Packaging Line Scheduling Optimization. *Pharmaceutical Manufacturing Vol 14 no 8 pp 13-15, Oct 2015.*

Schwindt, C. & Zimmermann, J. (Eds.). (2015). *Handbook on Project Management and Scheduling Vol. 2*. Springer International Publishing.

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