Optimization Models and Algorithms for Prototype Vehicle Test Scheduling

by

Yuhui Shi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in The University of Michigan
2017

Doctoral Committee:

Associate Professor Marina A. Epelman, Co-chair
Associate Professor Amy E.M. Cohn, Co-chair
Associate Professor Amitabh Sinha
Dr. Daniel Reich, Ford Motor Company

Yuhui Shi

yuhuishi@umich.edu

ORCID ID: 0000-0001-5763-2445

ACKNOWLEDGEMENTS

First, I would like to express the deepest appreciations to my advisors Professors Amy Cohn and Marina Epelman. They went above and beyond in guiding and supporting me through my four-year doctorate study, and addressed great flexibility in allowing me exploring various interesting ideas.

I also would like to thank Professor Amitabh Sinha for being on my committee. I have had inspiring discussions with him during my dissertation work. I always enjoyed the short meetings with him and was surprised at the amount of topics we covered during the 10-minute meetings.

In addition, I would like to express my gratitude to the generous support and founding I received from Ford Motor Company to allow me to study the interesting problems they encountered during their business operations. Especially, I would like to extend my thanks to my colleagues at Ford, Doctors Daniel Reich, Jae-Young Jung, Yan Fu and Erica Klampfl. Pursing a doctorate degree while working significant amount of time for a company in the industry is not a easy task. Without your help and support, this cannot be possible. I enjoyed the summers I spent at Ford working with you smart people.

Finally, I would like to thank my wife Yihuan for her persistent support during my life.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Optimization Models and Algorithms for Prototype Vehicle Test Scheduling

by

Yuhui Shi

Chair: Marina Epelman, Amy Cohn

Automotive makers conduct a series of tests at pre-production phases of each new vehicle model development program. The main goal of those tests is to ensure that the vehicle models meet all design requirements by the time they reach the production phase. These tests target different vehicle components or functions, such as powertrain systems, electrical systems, safety aspects, etc. However, one big issue is that the cost of the resources, mainly prototype vehicles, invested in the testing process is exceedingly expensive. An individual prototype vehicle can cost over 5 times its counterpart's price in the commercial market because many of the parts and the prototype vehicles themselves are highly customized and produced in small batches. Parts needed often require months of lead time, which constrains when vehicle builds can start. That, combined with inflexible time-window constraints for completing tests on those prototypes introduces significant time pressure, an unavoidable and challenging reality. What makes the problem even more difficult is that in addition to the prototype vehicle resources, there are other constrained supporting resources involved during the execution of those tests, such as testing facilities, instruments and equipment like cameras and sensors, human-power availability, etc.

An efficient way to conquer the problem is to develop test plans with tight schedules that combine multiple tests on vehicles to fully utilize all available time while balancing the loads of other supporting resources. 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.

In this thesis, we show how to mathematically model these test scheduling problems as optimization problems. We develop corresponding solution approaches that enable quick generation of an efficient schedule to execute all tests while respecting all constraints. Our models and algorithms save test planners' and engineers' time, increase their ability to quickly react to program changes, and save resources by ensuring maximal vehicle utilization.

CHAPTER I

Introduction

1.1 Background

Product Development division at the Ford Motor Company is responsible for designing and testing new vehicles and readying them for production. Each vehicle program (e.g., 2020 Ford Fusion, 2018 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 as much as 5 times more than its counterparts seen on the market 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. Additionally, there are other capacitated resources that constrain the throughput of the testing process, such as testing facilities, instruments like sensors and cameras, human-power availability, etc. Therefore, even if there are enough vehicles, the maximum number of tests that can be performed concurrently is subject to the availability of those resources.

An efficient way to reduce prototype usage is to develop test plans with tight schedules that combine multiple tests on vehicles to maximally utilize available time while balancing the load of other supporting resources. 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 same vehicle. Another complicating factor is that different tests may have different vehicle specification requirements; for example, one test may require a hybrid engine whereas another may require a conventional 4-cylinder I4 engine, prohibiting combinations of these tests on the same vehicle.

Prior to our work, test plans at Ford 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, individual tests requirements, etc., manually editing the plan requires significant additional time and effort, and may lead to decreasing vehicle utilization. In this thesis, 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 algorithms 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.

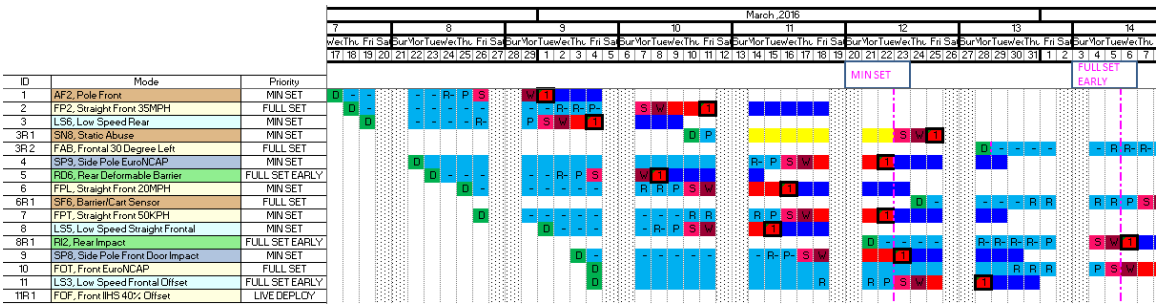
1.2 Problem description

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

To illustrate the complexities involved in defining and solving each test scheduling instance, we begin with an example of a typical schedule generated specifically for the safety testing lab for a vehicle program at Ford. This may be viewed as part of a full schedule for a vehicle program because tests from other department, such as powertrain or aerodynamics, are not showed here.

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

Figure 1.1 shows a small example schedule Gantt chart for a series of crash tests. Each row contains a single crash test with detailed timing information represented by character- and color-coding. The day each prototype is delivered for the test is marked



ID	Mode	Priority
1	AF2, Pole Front	MIN SET
2	FP2, Straight Front 35MPH	FULL SET
3	LS6, Low Speed Rear	MIN SET
3R1	SN8, Static Abuse	MIN SET
3R2	FAB, Frontal 30 Degree Left	FULL SET
4	SP9, Side Pole EuroNCAP	MIN SET
5	RD6, Rear Deformable Barrier	FULL SET EARLY
6	FPL, Straight Front 20MPH	MIN SET
6R1	SF6, Barrier/Cart Sensor	FULL SET
7	FPT, Straight Front 50KPH	MIN SET
8	LS5, Low Speed Straight Frontal	MIN SET
8R1	RI2, Rear Impact	FULL SET EARLY
9	SP8, Side Pole Front Door Impact	MIN SET
10	FOT, Front EuroNCAP	FULL SET
11	LS3, Low Speed Frontal Offset	FULL SET EARLY
11R1	FOF, Front IIHS 40% Offset	LIVE DEPLOY

Figure 1.1: The Gantt chart provides a character- and color-coded crash schedule.

with a "D." The prototype is then prepared during the following days. The majority of tests are performed at the crash barrier, where the actual crashes happen, in which case the vehicle may require sign-off ("S"), walk-around ("W"), turnaround time in queue, the actual crash ("1") and post-crash analysis time. (A few "abuse"-type tests do not require crashing the vehicle into a barrier and have a simpler structure.)

In addition to the timing of each test, the Gantt chart in Figure 1.1 provides information on the utilization of each prototype: the first column contains prototype vehicle number, with "R1" appended to represent the first rehit (second crash) on this prototype, "R2" — to represent the second rehit (third crash), etc. For example, prototype number 8, which is scheduled to be delivered to the safety department on March 1st, will first be used for a low speed front collision test, followed by a higher speed rear impact test.

Although the Gantt chart provides fine details for executing a crash plan, it is less well-suited for viewing the plan's efficiency at a high level. The average number of crashes per vehicle, known as the rehit ratio, is a key metric of a schedule. The crash-test sequence document, shown in Figure 1.2, is designed to highlight vehicle utilization and the associated rehit ratio. Each line represents a vehicle, with its sequence of crashes listed across the columns. Yet another method for summarizing assignments of tests to prototypes is the so-called "crash tree" illustrated in Figure 1.3.

Here, the rows of the table correspond to tests, the columns — to specifications of prototypes used for each test, and the notations in the table indicate test-to-prototype assignment and the position of each test in the rehit sequence. Safety engineers would construct the coarse plan as shown in the crash sequence document and the crash tree, before starting on the finer timing details included in the Gantt chart.

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

Figure 1.2: The crash sequence document provides a compact summary of rehit strategies, highlighting test combinations on vehicles.

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

Figure 1.3: The crash tree specifies assignment of tests to specific prototype vehicles and the relevant specifications of the prototypes

Until recently, test planners constructed all these documents manually (including the detailed timing information in the Gantt chart above), using a standard Excel

template. As we discuss in the following chapters, we have designed a web-based support system for automated scheduling with a database that is implemented so as to capture all the timing information associated with each test, allowing for automatic generation of the Gantt chart once the test assignment and rehit sequencing decisions have been made.

1.2.1 The General Test Scheduling Problem (GTSP)

In this section, we discuss the common settings of the General Test Scheduling Problem (GTSP) and introduce corresponding notation. In the following chapters, we will introduce variations of the GTSP that each have special features.

In the GTSP, we are given a set of tests T and a set of available vehicles V. For each test $t \in T$, duration p_t, release date r_t, and due date d_t are specified. For each vehicle $v \in V$, release date q_v is also specified; tests that are assigned to a vehicle cannot begin until its release date.

The goal is to plan a schedule to execute all tests in a way that minimizes an objective function that combines the number of prototype vehicles used and a time-related penalty function reflecting the "tardiness" of tests completed after their due dates. In the schedule, we need to decide: (1) the assignments of tests to prototype vehicles, (2) the sequencing of tests that are assigned to the same vehicle, and (3) the execution interval of each test on the vehicle.

In the final schedule, it is possible that we use only a subset of vehicles V. The set V represents a "budget" of vehicles available for tests, and it is often set rather conservatively. The optimal schedule often uses fewer vehicles than are given; however, if it turns out that there does not exist a feasible schedule within the current vehicle budget, a testing planner will always go back to the upper level management and renegotiate the budget.

There are several constraints we need to consider in planning a schedule.

Temporal constraints Each test $t \in T$ takes amount of time p_t to complete, and execution of test t cannot start before the test's release date r_t. If the due date d_t serves as a hard constraint, i.e., a deadline, then the test must be completed before d_t. However, in some cases the due dates are treated as soft constraints by associating a time-related penalty with the test missing its deadline. Throughout the thesis, we use a penalty based on tardiness: if test t starts execution at time s_t, then it is completed at time $s_t + p_t$, and its tardiness is $\max\{0, s_t + p_t - d_t\}$. In our models, we use a mixture of approaches: there is a fixed "grace period," within which tardy tests are penalized for missing their due dates; if completion time of a test exceeds the grace period, the schedule is considered infeasible.

In addition, the vehicles that are used to execute the tests are not all available through the entire planning horizon. The reason is that they are built in small batches and with a lot of highly customized parts that require long lead times. Therefore, the number of such prototype vehicles that can be built each week is limited. They are delivered for tests gradually, and for each vehicle v, release time q_v is specified. As a result, all tests that are assigned to vehicle v need to start after the vehicle is available, i.e., after time q_v.

Specification compatibility When considering assigning more than one test to the same vehicle, we need to consider several factors that might prevent these tests from sharing a vehicle.

First, if combining two tests on a particular vehicle makes it impossible for one or both of the tests to meet their due dates (with grace period, if appropriate) due to the timing specifications of the tests and vehicle release, this combined assignment is invalid.

Secondly, tests usually require specific parts configurations of vehicles. For example, the deformation of a vehicle in a perpendicular frontal crash test is sensitive to

the layout under its engine hood. Therefore, when developing a new vehicle model that can be equipped with different engine types, such as 4-cylinder or V6 engines, test planners want to perform perpendicular frontal crash tests on both of the vehicle trims to ensure the safety standards on both designs. Other types of tests may also have requirements on body type, electrical system, even the material of the seats. Therefore, if two tests require different configurations of vehicles, it may be impossible to perform them on a same vehicle. However, there are cases when, even though tests require different configurations, e.g., different types of tires on the vehicle, it is relatively easy to re-configure the vehicle (change the tires) to make the sharing possible.

Even if two tests require compatible vehicle configurations, we still need to consider the sequencing restrictions when deciding the order among multiple tests. We discuss the details of such restrictions next.

Sequencing restrictions Many of the tests are destructive and impact certain vehicle areas or functions afterwards. For example, a lot of high-impact safety crash tests totally disable the vehicle after the crash happens. Therefore, after those tests are completed, the vehicle is unusable for other tests. Even some tests that are not as destructive as a crash test still impact systems and structures of the vehicle, which might influence the accuracy of any following tests. So whenever multiple tests are assigned to the same vehicle, the order of executing them plays an important role in getting reliable and accurate test results.

For example, if two tests, t_i and t_j, are performed on the same vehicle, and t_i is a non-destructive test while t_j is a high-impact crash test as described above, it is important to ensure t_i is performed before t_j, because it is possible that the vehicle will have to be discarded after test t_j.

In the example above, we say an order $t_i \rightarrow t_j$ is permissible while the order

$t_j \rightarrow t_i$ is not. The relation between two tests that determines whether they can be performed in a particular order on the same vehicle is called a rehit rule. Notice that a rehit rule is usually independent of vehicle configuration requirements of tests and only depends on what type of tests they are.

Specification compatibility and sequencing restrictions can be captured in a $|T| \times |T|$ binary matrix, denoted by $\mathbf{A}$. An element a_{ij} of $\mathbf{A}$ equals 1 if tests t_i and t_j have compatible vehicle configuration requirements and performing t_i prior to t_j on the same vehicle is allowed by the rehit rules, and $a_{ij} = 0$ if these tests cannot be combined, or combined in this order. Notice that having $a_{ij} = 1$ does not necessarily means that $a_{ji} = 1$ since rehit rules are not always symmetric, as illustrated in the above example of combining non-destructive and destructive tests. However, if $a_{ij} = 0$ because t_i and t_j have different vehicle configuration requirements, then we also have $a_{ji} = 0$.

Another interesting property of matrix $\mathbf{A}$ is that permissible orders do not have transitive property, i.e., having permissible orders $t_i \rightarrow t_j$ and $t_j \rightarrow t_k$ does not necessarily mean that $t_i \rightarrow t_k$ is permissible. This lack of transitivity makes it essential to check for a topological order among *all* tests when more than 2 tests are assigned to the same vehicle. For example, it may be the case that combining 3 tests is impossible, even if any pair of them is compatible in a particular order.

Determination of specification compatibilities and sequencing restrictions is highly reliant on the engineering expertise of test engineers. Procedures for collecting this information are described in Chapter III.

Support resources capacity In many instances, in addition to prototype vehicles, other supporting resources are involved in the execution of a test. For example, most of the crash tests at Ford are performed at a crash barrier facility where controllable barriers are instrumented to simulate different impact angles and areas of crashes.

The number of these barriers is limited and barriers take a long time to set up before a crash. Therefore, the number of crashes that can be performed at the facility during a single day is subject to a capacity constraint. Same is often true for other types of resources, such as human labor. Sometimes, a vehicle is ready for testing in terms of physical preparation work but still waits in the garage simply because there are not enough engineers to initialize the testing process on that vehicle. Therefore, it is critical to take capacities of these supporting resources in every time period into account when scheduling the tests.

This issue is very common when it comes to concurrent scheduling of multiple vehicle programs. Because Ford is developing multiple new vehicle models each year, including pickup trucks, compact cars, sedans, and SUVs, these vehicle programs are competing for supporting resources if their testing periods overlap with each other. Therefore, it is essential to consider such constraints when dealing with this variation of the problem.

1.3 Thesis outline

In the following section, we will review the relevant literature.

In each of the following chapters we will discuss a variation of the general test scheduling problem (GTSP).

In Chapter II we discuss models and algorithms for scheduling of general tests from a wide range of departments, where a large portion of the tests are non-destructive. As a result, sharing a vehicle across departments for different testing purposes is common and actually encouraged to achieve better utilization of expensive resources. We show that formulating the problem as a general mixed integer linear program is not sufficient due to large sizes of problem instances in this setting. Even identifying a good feasible solution is exceedingly difficult. Therefore, motivated by the list scheduling algorithm used to solve bin-packing problems, we propose a variation of

the algorithm that can handle the unique constraints and complexities of GTSP. In particular, we discuss implementation details of the algorithm and propose a lower bounding technique that can evaluate the quality of solutions obtained by heuristics. This chapter is based on *Reich et al.* (2016).

One of the lessons learned from our work in Chapter II is that a good schedule for the safety crash tests is critical in reducing the number of prototype vehicles used by the entire vehicle program. The destructive nature of crash tests means that once a prototype vehicle is delivered for safety testing, it is unlikely that it will be used for other testing purposes afterward. In addition, because of the high precision requirements on crash test outcomes, it is also unlikely that crash tests would share prototype vehicles with other tests (like powertrain system tests). Therefore, scheduling crash tests is rather self-contained.

In Chapter III we focus on the scheduling of safety crash tests. Because we are dealing with a subset of all tests considered in Chapter II, more detailed models and algorithms are proposed. Again, we show that modeling the crash test scheduling problem (CTSP) as an MILP is not sufficient to solve it. Alternatively, we model it as a set-partitioning model that contains a large number of variables, and propose a delayed column generation algorithm that can obtain high quality solutions. We discuss various implementation details that speed up the algorithm, and suggest an exact solution approach based on a branch-and-price algorithm. A shortened version of some of the material in this chapter has appeared in *Shi et al.* (2017).

In Chapter IV, we study a variation of the CTSP, where prototype vehicles are not the only resource required to execute a crash test, but other types of supporting resources, such as equipments, manpower, and testing facilities also play a role in the scheduling. Based on the model proposed in Chapter III, we extend the set-partitioning formulation to a richer context that can handle the additional resource constrains. The solution approach also fits into the framework of delayed column

generation, but alternative methods for solving the pricing problem are proposed.

Finally, in chapter V, we summarize our results and discuss the impact of our work.

1.4 Literature review

Although certain aspects of the optimization problem addressed in this thesis 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., *Coffman Jr et al.* 1996). 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 *Elhedhli et al.* (2011). 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 com-

parisons between different solution strategies can be found in *Cheng and Sin* (1990), *Potts and Strusevich* (2009), and *Sterna* (2011). 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 1.2 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 2.3, 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 *Cieliebak et al.* (2004) 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. *Yu and Zhang* (2009) 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, *Lee et al.* (2012) 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 instances with up to 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 *Kravchenko and Werner* (2009); *Alidaee and Li* (2014); *Finke et al.* (2009). 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 *Finke et al.* (2009) 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 the precedence relationship between two tests is 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 is particularly relevant 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 tests A and B arbitrarily large, we can view the problem as a special case of the sequence-dependent setup case. For example, *Zhu and Heady* (2000) and *Balakrishnan et al.* (1999) 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. *Sawik* (2010) studies another sequence-dependent setup time problem, where setup time of each task is related to its starting time. He proposes a time-indexed MILP formulation that can be solved for small instances with up to 50 tasks and 10 machines. Since tractability of MILP formulations is limited, another set of papers focuses on heuristic methods. Perhaps the most common approach is to use list scheduling methods motivated by the bin packing problem. *Kim et al.* (2003) 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 2.3 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, *Kurz and Askin* (2001) proposes several heuristics that combine the greedy approach to minimize makespan with TSP heuristics for job sequencing. *Heady and Zhu* (1998) 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 re-sequenced 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 meta-heuristics or tabu search (for examples, see *Cao et al.* 2005; *Radhakrishnan and Ventura* 2000; *Liu* 2013; *Sivrikaya-rifolu and Ulusoy* 1999; *Lin and Hsieh* 2014; *Rabadi et al.* 2006). It is also common to combine several heuristics in order to obtain high-quality solutions. For example, *Chen and Wu* (2006) consider parallel machine scheduling problems where jobs have types, and limited time is available for transitioning between two types of jobs on a single machine. They combine thresh-accepting methods, tabu lists, and improvement procedures in order to obtain high-quality solutions. They showed that such combinations can outperform methods that adopt only a single idea. However, Chapter II we chose to focus on a simpler, less computationally demanding heuristic that proved more easily applicable specifically for the GTSP.

In recent years, optimization models have been introduced specifically for test scheduling on prototype vehicles. *Chelst et al.* (2001) 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. *Bartels and Zimmermann* (2009) 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. *Limtanyakul and Schwiegelshohn* (2012) 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.

In addition to the papers discussed above, we will cover more relevant papers when we proceed to each specific chapter.

CHAPTER II

Scheduling of general tests

2.1 Introduction

In this chapter, we discuss the General Scheduling Planning Problem (GTSP) for general tests during new vehicle development stages. During the development stage of a new vehicle model, prototype vehicles of that model need to be tested for a wide variety of design aspects, such as powertrain, safety, aerodynamics, etc. However, as we discussed in Section 1.1, the prototypes are extremely expensive. To reduce the number of vehicle prototypes required for testing, sharing the vehicles among different tests is encouraged and helps improve utilization of expensive resources.

The size of the portfolio of tests is often large for this version of the scheduling problem. Therefore, it is extremely difficult to solve a problem instance to optimality. We first introduce an Mixed Integer Linear Programming (MILP) formulation of the scheduling problem as a reference. However, the formulation suffers from several computational issues and is generally extremely challenging to solve. We propose an alternative — a modified list scheduling heuristic motivated by the bin packing problem, that can handle the constrains and other specifics of our problem.

At the beginning of each vehicle program, one engineer from each department (powertrain, safety, electrical, etc.) 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, engine type, etc.)?
- Which tests are assigned to each vehicle?

In addition to ensuring that schedules allocate sufficient amounts of time for each test to be completed by its individual due date, 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 by constructing a $|T| \times |T|$ binary matrix $\mathbf{A}$. While the actual compatibility rules are confidential, the sub-matrix of $\mathbf{A}$ in Table 2.1 provides an illustrative example.

	Test i	Test j
Test i	0	1
Test j	0	0

Table 2.1: Test compatibility table; "0" indicates that the test in this row cannot be performed prior to the test in the column, "1" indicates otherwise.

Notice that compatibility is not symmetric. In the example in Table 2.1, t_j may be performed after t_i, but not before. This is typical of testing requirements. For example, consider two crash tests: a 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.

2.2 Mixed integer formulation

In this section, we introduce our original MILP formulation for the GTSP.

We define binary decision variables $u_v,\ v \in V$ such that $u_v = 1$ means that vehicle v is being used, i.e., some tests are assigned to the vehicle; $u_v = 0$ otherwise. We define binary decision variables $x_{t,v},\ t \in T, v \in V$ to indicate the assignments of tests to vehicles, where $x_{t,v} = 1$ if we assign test t to vehicle v; $x_{t,v} = 0$ otherwise. Binary decision variables $y_{t_i,t_j},\ t_i, t_j \in T : t_i \neq t_j$ indicate the relative order between two tests t_i and t_j if they are one the same vehicle: $y_{t_i,t_j} = 1$ if t_i and t_j are assigned to the same vehicle and t_i precedes t_j; $y_{t_i,t_j} = 0$ otherwise. Notice that t_i is not necessarily conducted immediately before t_j, just some time prior to t_j, if $y_{t_i,t_j} = 1$. Finally, we define continuous decision variables $s_t,\ t \in T$ as the starting times of tests.

The formulation is as follows:

$$\min \quad \sum_{v \in V} u_v \tag{2.1}$$

$$\text{s.t. } x_{t,v} \leq u_v \qquad v \in V,\ t \in T \tag{2.2}$$

$$\sum_{v \in V} x_{t,v} = 1 \qquad t \in T \tag{2.3}$$

$$s_{t_i} + p_{t_i} \leq s_{t_j} + M(1 - y_{t_i,t_j}) \qquad (t_i, t_j) \in T \times T : t_i \neq t_j \tag{2.4}$$

$$y_{t_i,t_j} + y_{t_j,t_i} \leq 1 \qquad (t_i, t_j) \in T \times T : t_i \neq t_j \tag{2.5}$$

$$x_{t_i,v} + x_{t_j,v} - 1 \leq y_{t_i,t_j} + y_{t_j,t_i} \qquad v \in V,\ (t_i, t_j) \in T \times T : t_i \neq t_j \tag{2.6}$$

$$y_{t_i,t_j} = 0 \qquad t_i, t_j \in T : t_i \neq t_j, a_{t_i,t_j} = 0 \tag{2.7}$$

$$s_t \geq r_t \qquad t \in T \tag{2.8}$$

$$s_t \geq \sum_{v} q_v x_{t,v} \qquad t \in T \tag{2.9}$$

$$s_t + p_t \leq d_t \qquad t \in T \tag{2.10}$$

$$x_t \in \{0, 1\}, y_{t_i,t_j} \in \{0, 1\}, s_t \geq 0. \tag{2.11}$$

In this formulation, objective (2.1) is to minimize total vehicle usage by summing up binary indicators $u_v,\ \forall v \in V$. Constraints (2.2) link the assignment decision variables x with the vehicle usage variables u: if some tests are assigned to a vehicle, then its usage indicator u_v must be 1. Constraints (2.3) ensure that each test gets assigned to exactly one vehicle. Constraints (2.4) and (2.5) 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, where the value of M can be set equal to the length of the planning horizon. Constraints (2.6) link those variables with the assignment variables. Constraints (2.7) enforce compatibility for tests assigned to the same vehicle. Constraints (2.8), (2.9), and (2.10) ensure that each test is performed during its time window, and starts after the delivery of the

vehicle to which it is assigned (in this problem setting, tests must be completed by their specified due dates, which are treated as "hard" deadlines).

This general formulation can be applied to many other scheduling applications. However, there are two problems with the above formulation that can lead to poor tractability: 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 disjunctive constraints (2.4). To improve the formulation, we have included symmetry-breaking constraints: for any two identical vehicles with indices v_i and v_j where $v_i \leq v_j$, we require that $u_{v_i} \leq u_{v_j}$, and $x_{t,v_i} \geq x_{t,v_j}$ $\forall t \in T$, i.e., we require that v_i is used before v_j. Moreover, we set M equal to 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 implemented all algorithms using Java and used CPLEX 12.6 as our LP and MIP solver, and ran the experiments on a desktop platform with Xeon E1241 processor and 32 GB RAM. We tested formulation (2.1)–(2.11) 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 2.4, CPLEX was unable to produce even a feasible solution.

2.3 Heuristic algorithm

Due to large numbers of tests that need to be scheduled, it is extremely difficult to solve a typical instance of the GTSP by an exact method. In this section, we introduce a heuristic and optimization subproblems that we can combine to obtain high-quality feasible solutions for larger problem instances.

2.3.1 Fit-and-Swap test scheduling heuristic

The GTSP can be viewed as a generalization of the classical bin-packing problem, which motivated us to propose a variant of the list scheduling (allocation) algorithms, which are widely used to solve bin-packing problems, extended for the GTSP.

In the bin packing problem, objects of different sizes must be packed into a finite number of bins or containers in a way that minimizes the number of bins used. The problem is a combinatorial NP-hard problem which can be viewed as a special case of GTSP, thus easily proving that GTSP is also NP-hard. To see that, we let all vehicles release at a same time 0, i.e., $q_v = 0,\ \forall v \in V$. Additionally, we set release times of tests and deadlines to be the same, too, i.e., $r_t = 0$ and $d_t = d,\ \forall t \in T$. In this case, the duration of test t, p_t, serves as the item size in the bin packing problem and d is the capacity of each bin.

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 *Cieliebak et al.* (2004), *Yu and Zhang* (2009), and *Kim et al.* (2003) for their respective objectives; however, before assigning a test to a vehicle, we must consider compatibility, sequencing restrictions, test time windows, and vehicle delivery date to determine if this assignment is feasible.

The algorithm, to which we refer as the *Fit-and-Swap* heuristic, is summarized in Algorithm 1 and illustrated in Figure 2.1 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 Figure 2.1(a). 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 Figure 2.1(b). If the vehicle already has some tests assigned, we solve the Ordering Satisfiability Problem (defined in Section 2.3.2) 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 Figure 2.1(c). By swapping the vehicles, as in Figure 2.1(d), 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 Figure 2.1(e), 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.

2.3.2 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

```
Sort the vehicle list V in decreasing order of vehicle capacity;
Sort the test list T in decreasing order of test durations;
foreach v ∈ V do
    foreach t ∈ 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′ ∈ V with the smallest capacity, such that the set of tests on v can
     be reassigned to v′;
    if v′ ≠ v then
        reassign all tests from v to v′;
        swap locations of v and v′ in V;
    end
end
```

Algorithm 1: Fit-and-Swap algorithm

on compatibility relationships between tests (Table 2.1) 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

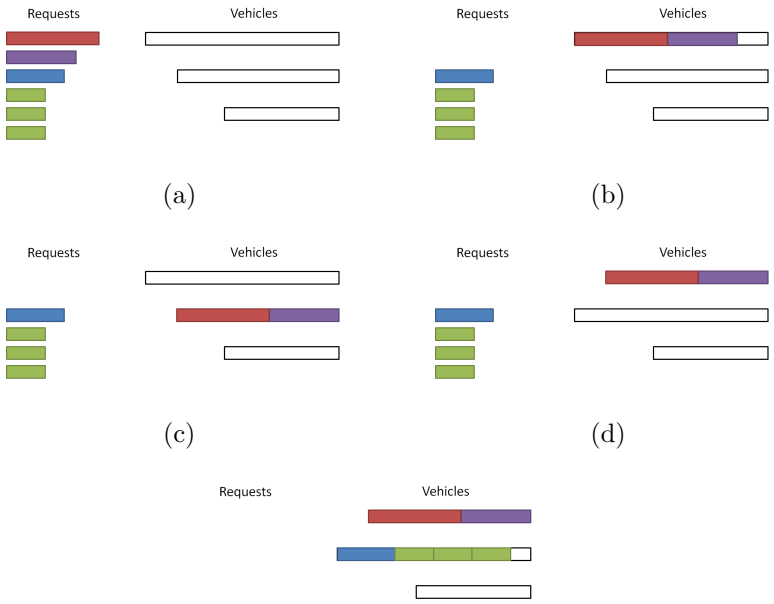


Figure 2.1: Fit-and-Swap algorithm illustration

model in Section 2.2, and is formulated as follows:

$$s_{t_1} + p_{t_1} \leq s_{t_2} + M(1 - y_{t_1,t_2}) \qquad (t_1, t_2) \in \bar{T} \times \bar{T} : t_1 \neq t_2 \qquad (2.12)$$

$$y_{t_1,t_2} + y_{t_2,t_1} = 1 \qquad (t_1, t_2) \in \bar{T} \times \bar{T} : t_1 \neq t_2 \qquad (2.13)$$

$$y_{t_1,t_2} = 0 \qquad (t_1, t_2) \in E \qquad (2.14)$$

$$s_t \geq r_t \qquad t \in \bar{T} \qquad (2.15)$$

$$s_t \geq q_v \qquad t \in \bar{T} \qquad (2.16)$$

$$s_t + p_t \leq d_t \qquad t \in \bar{T} \qquad (2.17)$$

$$y_{t_1,t_2} \in \{0, 1\}, s_t \geq 0. \qquad (2.18)$$

Here, $\bar{T}$ 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 (2.1)–(2.11)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 (2.2) and (2.3) are no longer required, and the original constraints (2.5), (2.6), and (2.9) reduce to (2.13) and (2.16), respectively.

Compared with the original IP in Section 2.2, which became intractable for larger problem instances, a typical instance of the Ordering Satisfiability Problem can be solved in under 1 second by an integer programming solver. With subset $\bar{T}$ 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, many of which are eliminated by constraints (2.14).

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

$$z_{t_2} \leq z_{t_1} - 1 \qquad\qquad (t_1, t_2) \in E, \tag{2.19}$$

where the values of continuous decision variables $z_t,\ t \in \bar{T}$, 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.

2.3.3 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 (2.1)–(2.11) 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.

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.

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.)

$$\text{maximize} \sum_{v \in V} \sum_{t_1, t_2 : t_1 < t_2} w_{t_1, t_2, v} \tag{2.20}$$

$$\text{s.t.} \sum_{v \in V} \left(\sum_{t_2 : t_2 < t_1} w_{t_2, t_1, v} + \sum_{t_2 : t_1 < t_2} w_{t_1, t_2, v} \right) \leq 1, \ t_1 \in T_S \tag{2.21}$$

$$\sum_{t_1, t_2 : t_1 < t_2} w_{t_1, t_2, v} \leq 1, \ v \in V \tag{2.22}$$

$$w_{t_1, t_2, v} = 0, \ \{t_1, t_2\} \in E_v, \ v \in V \tag{2.23}$$

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

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 (2.20) maximizes the number of matches among safety tests, while constraints (2.21) ensure that each test is combined with at most one other, and the pair of tests is assigned to at most one vehicle. Constrains (2.22) ensure that each vehicle is used at most once, and constraints (2.23) 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. (We refer to this approach as *grouping via matching.*)

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:

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

Algorithm 2: Group release time calculation algorithm

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.

2.3.4 Test planning algorithm summary

The Test Planning Algorithm combines the heuristic and optimization subproblems presented above to obtain feasible solutions to the integer programming model in Section 2.2. The steps of the Test Planning Algorithm are summarized as follows:

1. Read in the tests and create set T and crash test subset T_S.
2. Read in the vehicle release schedule and create vehicle set V.
3. Read in compatibility rules and construct set E.
4. (Optional) Formulate and solve a grouping IP on T_S and V (Section 2.3.3) and update T with grouped tests.
5. Run the Fit-and-Swap algorithm on T and V and return the schedule.

2.4 Numerical results

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 2.2. We attempted to solve these instances using the IP presented in Section 2.2 with the symmetry-breaking constraints added, but CPLEX was unable even to find a feasible solution within the 1 hour time limit, which is also noted in Table 2.2.

Table 2.2: Description of problem instances used in computational experiments. The full IP formulation of Section 2.2 did not produce feasible solutions within 1 hour time limit on any of these instances.

	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

Next, we applied the Fit-and-Swap heuristic algorithm from Section 2.3.1, without safety tests grouping, and with grouping via matching and via IP. Table 2.3 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 smaller. 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 2.3, 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 2.3: Vehicle usage results for the three methods. Note that for Instance 1 the full IP in the subroutine was not solved to optimality: after 60 minutes, the optimality gap was 7.7%; the best feasible solution found in that time is reported.

	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*	0.939
Instance 2	145	32	0.945
Instance 3	110	11	0.937

Table 2.4 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 2.4: Runtimes, in seconds, of test grouping subroutines and Fit-and-Swap heuristics on the resulting test sets. *Note that for Instance 1 the full IP in the subroutine was not solved to optimality: after 60 minutes, 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.

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

CHAPTER III

Scheduling of safety crash tests

3.1 Introduction

From Chapter II, we have seen the importance of an efficient schedule of crash tests to the overall vehicle usage in all the tests for the entire program. The main reason is that crash tests are destructive and can render a vehicle unusable for any further tests. Therefore, crash tests tend to use dedicated vehicles not only to reduce the number of vehicles destroyed but also to improve crash result accuracy by isolating them from other departments' tests. In chapter II, we have tested solving the MILP formulation (2.1)–(2.11) of the GTSP just for crash tests. However, computational experiments indicate that it is difficult to obtain a provably high-quality solution due to the challenges we described in Section 2.2. As an alternative, we proposed a matching based method by observing that it is rare to have more than 2 crash tests on a single vehicle.

Compared to existing practices, this matching algorithm performed fairly well for most of the vehicle programs the safety organization is dealing with, since performing two tests (i.e., one rehit) is a widely used strategy. However, not all safety tests are destructive or severely destructive. For example, there are tests that are testing mainly sensors and deployment of airbags that can be triggered electronically without performing an actual crash. In those cases, it is possible that more than 2 safety tests

are performed on a vehicle. Hence, in this chapter, we seek a more accurate model than the matching method we proposed in Section 2.3.3. Additionally, we consider a more flexible treatment of test due dates.

3.1.1 Background

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

Maximizing utilization is not a simple matter. Crashes are destructive, and only certain tests can be combined together on the same prototype vehicle. Program milestones and staggered prototype vehicle delivery form a difficult timing problem. The different configurations of vehicles offered (e.g., types of powertrain, trim, bodystyle) are an additional source of complexity, because a comprehensive testing plan matches crash tests to specific configurations.

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

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

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

A key to delivering our technology and maximizing its benefits was designing the system for ease of use with engineers' current working document formats. We developed interfaces that ensure data quality and completeness. We designed our optimization package to run on Ford's High Performance Computing Center servers, while integrating it into a user-friendly web application with a backend database.

The success of the Test Planning Scheduler Support System (TP3S) application is evidenced by the rapid technology adoption within the safety group of product development. After a series of pilots and limited releases, the application was rolled out for use on all future programs. As of summer 2015, TP3S has been used to develop crash-test plans for the following vehicle programs: Ford SuperDuty, C-Max, Fiesta,

EcoSport, Mustang, Fusion, and Edge; and Lincoln MKC and MKX. In addition to a host of benefits, from improved planning processes and information infrastructure to valuable time saved by engineers, the use of TP3S enables increased utilization of testing resources, especially prototype vehicles, thus allowing more tests to be performed under the same resource consideration and imposing a higher safety standard within the company.

In the following sections, we introduce crash-test specifications, discuss the complexities associated with scheduling them, and present our web application along with the underlying optimization model and solution algorithms. We conclude by describing the evolution of the TP3S application and its impact at Ford.

3.2 System overview

In this section we describe TP3S — a web-based application we developed for use by Ford engineers for crash-test planning. This system has multiple benefits. It allows for systematic collection of information about crash-test modes,[1] timing, and rehit compatibility into a centralized database, institutionalizing expert knowledge. It also allows engineers to explore bottlenecks and potential improvements in test schedules by quickly specifying various testing scenarios. An optimal or near-optimal schedule for each scenario is generated in seconds or minutes by a custom-built optimization engine we developed.

3.2.1 Test management module

The first tab of the TP3S web interface is dedicated to test management, i.e., specifying and displaying information about each relevant crash-test mode. For example, Figure 3.1 shows the application screen with requirements for an Insurance Institute

[1] A test mode is an abstraction of types of crash tests, which is independent of vehicle configuration. For example, crashing the front side of a vehicle at a perpendicular angle at 35 mph into a barrier is a common test mode.

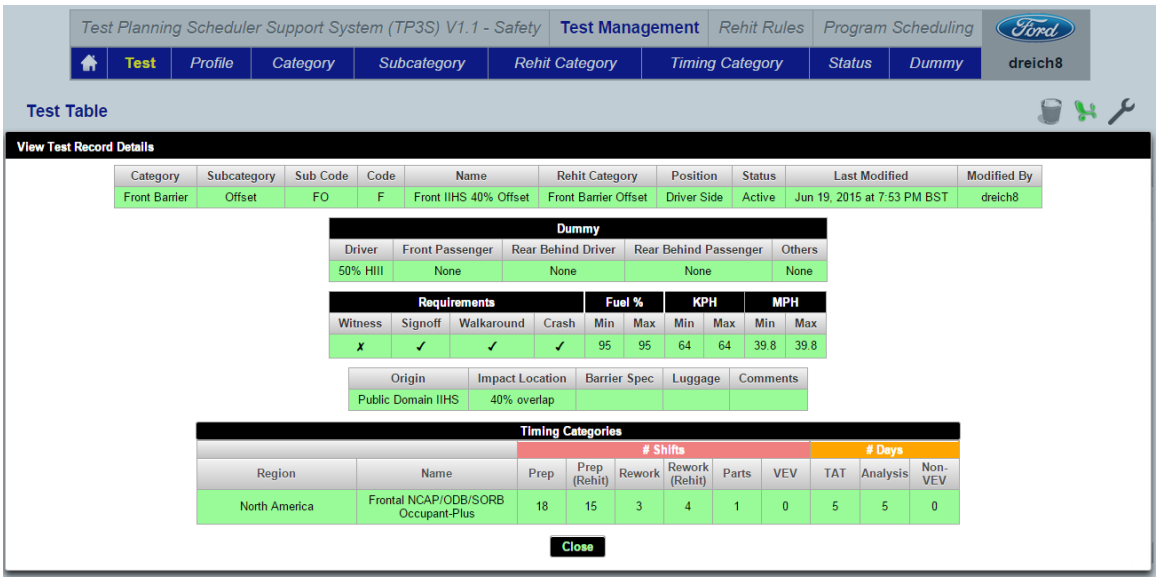


Figure 3.1: Example of a crash-test mode record from the TP3S web application, showing the requirements for an Insurance Institute for Highway Safety test.

for Highway Safety (IIHS) 40 percent front-overlap test, which include a driver-side crash-impact position, a driver-side dummy sized to a 50th percentile male, an execution speed of 64 kilometers per hour (KPH), and the standard timing for preparing, executing, and analyzing the test. To ensure consistency of the test data, our application includes screens to maintain and view relational database information on categorizations, dummy types, and other aspects of the test, which are navigable through the submenu bar.

Note that some timing information is expressed in days, while shifts are used for aspects of test timing that have flexibility to speed up execution by running multiple shifts per day. The impact of this flexibility can be explored in the scheduling process by considering what-if scenarios. Note also that the time allocated for a crash test may differ depending on whether it is the first test on the vehicle or a rehit. In particular, it is common for rehits to require less initial prep work, because some of the prep was already completed for a prior crash test; however, rework time may

increase for repairs needed to restore the vehicle to prime condition.

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

3.2.2 Rehit rules module

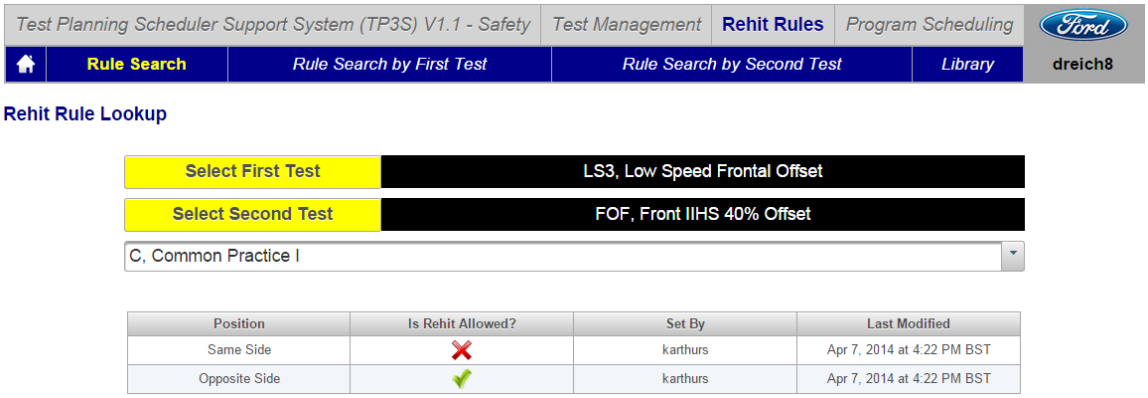


Figure 3.2: Rehit rules lookup shows whether two crash tests can be run on the same prototype vehicle in the specified order. For tests that are off-center, rules are based on whether the tests are to be executed on the same or opposite sides of the prototype.

As a result of its high speed of 64 KPH, an IIHS 40 percent front-overlap test damages a prototype vehicle severely enough so that it cannot generally be reused for other testing. However, it may be possible to conduct a less destructive crash prior to the IIHS test on the same prototype, if time permits and required specifications of the prototype are similar. The second tab of the TP3S application serves to display information on whether scheduling two crashes back-to-back on the same prototype is allowed, based on the rules provided by a core group of safety engineers. Figure 3.2 shows an example: here, we can schedule a low-speed front-offset test, which runs at under 10 MPH, prior to the IIHS test, so long as it is done on the opposite (passenger) side of the vehicle. This combination also appears on prototype 11 in Figures 1.1, 1.2

and 1.3. When setting these rehit rules, engineers consider the location and extent of the damage expected from the first crash test conducted. If the damage is expected to be either insignificant or easily repairable, the combination is generally permitted; if not, the combination is prohibited.

In addition to the two crash-test mode selections in Figure 3.2, a third dropdown allows the user to select a rehit rules library, which is designed for a specific vehicle platform and conservativeness measure. The vehicle platform is an important identifier because the same crash-test mode causes different levels of damage on different sized vehicles, e.g., a Focus compact car versus an Explorer SUV. The conservativeness measure ("common practice" in the example shown) allows the engineer to take a more or less aggressive approach, with the understanding that more aggressive approaches may require additional resources later on. Specifically, if damage from an initial crash exceeds expectations, the rehit crash scheduled on the same vehicle may require an additional unbudgeted vehicle or additional parts may be required for repairs.

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

The number of crash categories is currently around one third of the number of test modes, reducing the data entry burden by an order of magnitude. We further reduced this burden by designing the user interface to enable setting multiple rehit rules simultaneously, after selecting the category of the first test. We also provide deep-copy functionality that replicates an entire rehit rules library, which can then be edited to increase or decrease the level of conservativeness with minimal effort.

The application contains additional screens, navigable through the submenu bar in Figure 3.2, where engineers can view all tests allowed before or after a specified test for a selected vehicle platform and conservativeness measure.

3.2.3 Program scheduling module

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

The engineer's first step is to set dates for standard safety milestones, shown in Figure 3.3. Some milestones have only a due dates, whereas others may also include a date before which tests cannot be executed. We designed the scheduling module to allow plans to be easily updated for program timing changes: the milestones listed are connected to tests, so that if dates associated with those milestones are updated, the new timing is automatically associated with all connected tests.

The engineer's next step is to connect test models and milestones with prototype vehicle specifications using the screen shown in Figure 3.4. The engineer first con-

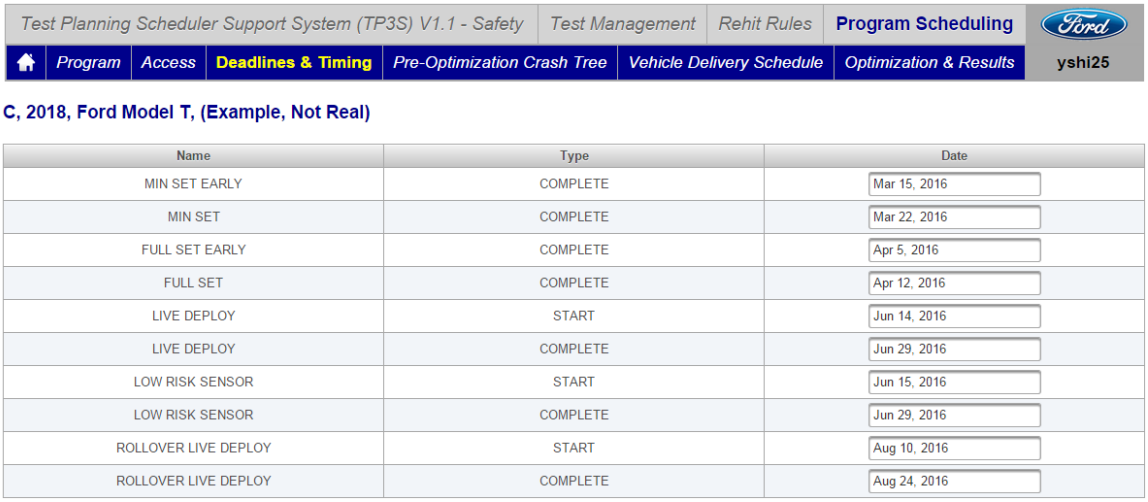


Figure 3.3: The user interface allows dates to be selected for scheduling milestones in a program.

structs a list of vehicle specifications (e.g., engine type, driveline type, driver side location, and other vehicle characteristics relevant for some or all of the test modes) and lists available options for each specification (gas or diesel engine, 4x2 or 4x4 driveline, etc.). The engineer then constructs control models, which are the combinations of those options, as columns. The required crash-test modes are then added to the plan as rows; for convenience, predefined lists of crash modes, e.g., for a European car or a North American truck, can be imported and then modified by deleting or adding specific test modes. The engineer connects a test mode to a control model by clicking a cell within the matrix, which opens a dialog to select an associated timing milestone. For example, in Figure 3.4, the 1's in each column indicate required tests that need to be performed on the corresponding control model; note that the tests indicated in the last column can be performed on a vehicle with either engine type.

The prototype vehicles are used by all departments (safety, powertrain, electrical, etc.) within the Product Development organization. Thus, the lead safety engineer coordinates test schedules with a planner from Ford's centralized test planning group that oversees the distribution and sharing of prototypes. Based on these interactions,

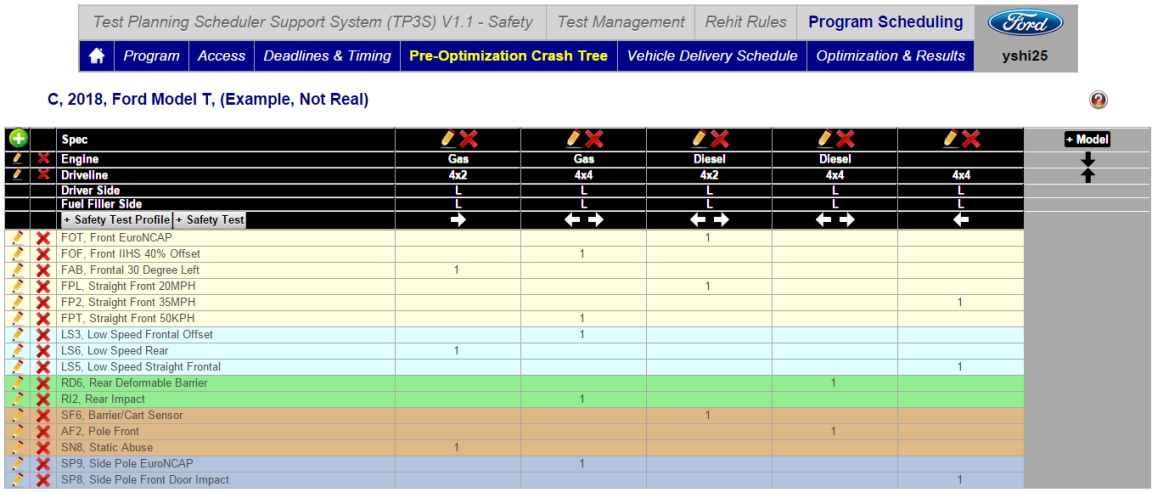


Figure 3.4: An engineer enters test requirements, by connecting vehicle specifications to test modes and milestones.

the engineer develops a schedule for expected deliveries of prototype vehicles to the safety department for crash testing. This schedule is input, tracked and updated using the screen shown in Figure 3.5.

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

Figure 3.6 shows the application screen for running optimizations, with options for what-if scenarios. When an optimization job is submitted, the application connects to Ford's High Performance Computing Center and passes a single database ID, which allows all the information associated with the vehicle program to be retrieved by our executable Java scheduling program. This design of decoupling the web interface used for transactional data functions and the scheduling optimization program

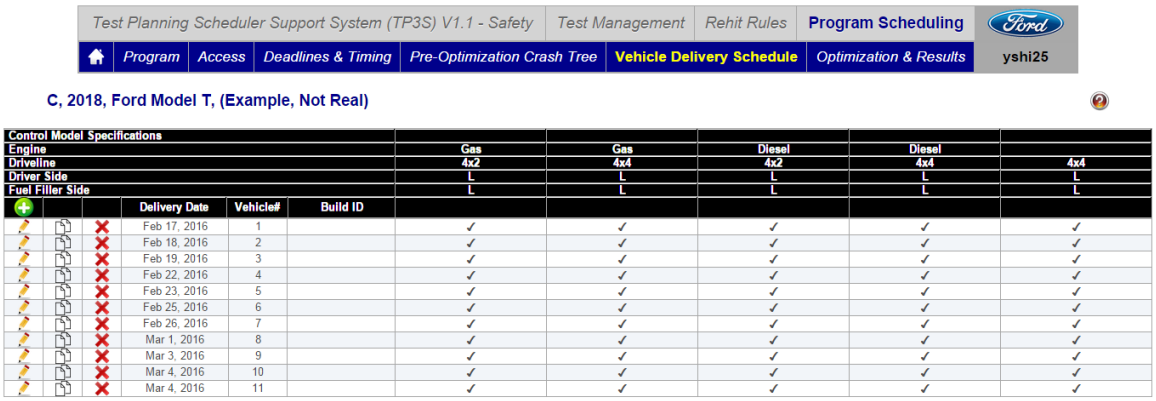


Figure 3.5: The prototype vehicle delivery schedule is shown, along with the possible control models that could be delivered each date.

allows engineers to run multiple scenarios in parallel. When the scheduling program terminates, the results are stored in the database and are automatically retrieved by the web application. An Excel spreadsheet is downloadable from the screen in Figure 3.6 containing the scheduling documents from Figures 1.1, 1.2, and 1.3.

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

3.3 Models and solution approaches

In this and the following sections, we discuss the models and algorithms that power the back-end scheduler of the TP3S system. In Section 2.2, we proposed an MILP for solving the GTSP. The MILP model of Crash Test Scheduling Problem (CTSP) is the same — we make a distinction since in this chapter all tests are crash tests. As a result, instance sizes of CTSP tend to be smaller. However, it is still difficult

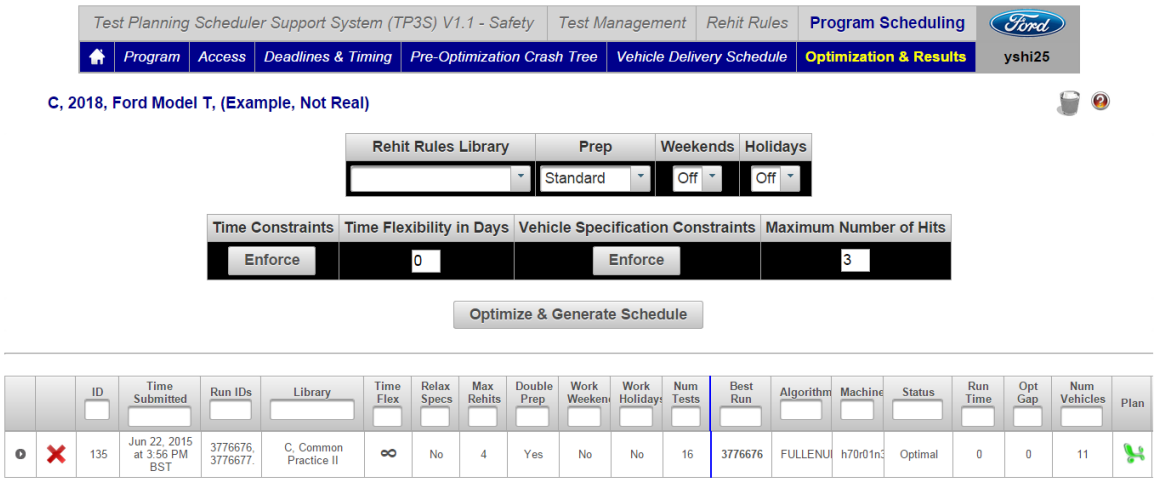


Figure 3.6: Run optimizations and view results.

to solve the MILP formulation within a reasonable optimality gap. Alternatively, we propose a set-partitioning formulation that is equivalent to the MILP and propose a delayed column generation approach to solve the problem.

3.3.1 A review of delayed column generation methods

There are many papers using column generation method to solve large-scale discrete optimization problems. *Gilmore and Gomory* (1961, 1963) proposed using a linear programming technique to solve large-scale cutting stock problems. In the cutting stock problems, there are demands on rolls of metal or paper with different width requirement. The way to produce those rolls with different widths is to cut a large roll into smaller ones. The goal is to minimize the usage of large rolls while satisfying the demand. The problem is formulated as a set-covering formulation by defining cutting patters for a large roll and then deciding the subset of patterns to use in the production. As there manny possible patterns patterns, the authors introduced a delayed column generation algorithm that can solve the linear relaxation of the original problem to optimality. The algorithm does not include all variables at once

but rather generates them dynamically by solving a pricing sub-problem. Specifically, the pricing sub-problem in the cutting stock problem can be solved as a knapsack problem within pseudo-polynomial time. At the last step of the solution process, a rounding scheme is used to obtain a feasible integer solution to the original problem based on the optimal solution of the linear relaxation.

There are many papers that apply the column generation idea in other problem settings. We ask readers to refer to two comprehensive review papers, *Desrosiers and Lübbecke* (2005); *Lübbecke and Desrosiers* (2005) on this topic. Notably, column generation is extremely popular in solving scheduling and routing problems, such as in *Taillard* (1999); *Ribeiro and Soumis* (1994); *Lavoie et al.* (1988).

A key issue to resolve in the development of a successful delayed column generation algorithm is the ease of solving the pricing problem to generate new variables to include in the master problem. For example, in the cutting stock problem, the pricing problem is solved as a knapsack problem. In the vehicle routing problems, the pricing problem is usually formulated as finding a shortest path in a graph. In both cases, efficient (pseudo)-polynomial algorithms (Dynamic Programming, Dijkstra's) can be applied. However, in the case of crash test scheduling, because of the presence of time window constraints, compatibility, and precedence relations between tests, solving the pricing problem is a nontrivial task. In the following sections, we shall cover how we conquer this computational challenge.

Another commonly discussed topic when developing a delayed column generation algorithm is how to obtain an integer solution, since delayed column generation only solves linear relaxations of integer programs. Three approaches are commonly used: (1) we can solve the linear relaxation of the original problem and gather the columns generated in the process, and then re-introduce integrality constraints and solve a restricted version of the original problem; (2) similarly, we can solve the linear relaxation to optimality, and then use some approximation algorithm such as rounding

to obtain a feasible integer solution based on the optimal solution of the linear relaxation, as we have seen in the case of solving cutting stock problems; (3) we can implement a branch-and-bound algorithm for the original integer program, using delayed column generation to solve linear relaxations of problems at each node of the branch-and-bound tree. The latter strategy is known as the branch-and-price algorithm, and is discussed in *Barnhart et al.* (1998); *Savelsbergh* (1997); *Mehrotra and Trick* (1996). In this chapter, we consider and test each of the 3 strategies.

In the following sections, we will discuss how to apply delayed column generation to CTSP.

3.3.2 Set-partitioning formulation of CTSP

MILP formulation (2.1)–(2.11) approaches the CTSP by deciding the assignments of tests to vehicles, and then sequencing tests that are assigned to each vehicle. Following the ideas of the papers mentioned above, we propose an alternative way of modeling the scheduling problem by using *composite variables*, namely, aggregating multiple tests into a scheduling unit. We define a sequence of tests ω as an ordered list of tests that are to be performed on the same vehicle. Notice that for this sequence to be *valid* all tests in this sequence should have compatible vehicle specifications and should be executable in the specified order, i.e., for any ordered pair of tests t_i and t_j that are contained in ω, we should have $a_{ij} = 1$.

Assume a sequence ω contains k tests, $t_1, t_2, \ldots, t_k$, where t_1 is the first test to be performed in that sequence. When assigned to a vehicle v, we can assume t_1 will start execution right after the vehicle is released at time q_v. Similarly, we can argue that all the following tests should start execution right after their respective predecessors have completed, to minimize the tardiness incurred by missing the due dates,[2] unless

[2]In this chapter, we do not consider any additional supporting resources required for testing, and thus this assumption is without loss of generality. In the following chapter, this assumption is no longer made.

a test is not released until a later time. Therefore, the starting time $s_{t_i}, i = 1, \ldots, k$ of each test can be computed as

$$s_{t_i} = \begin{cases} \min\{r_{t_i}, q_v\} & i = 1 \\ \min\{r_{t_i}, s_{t_{i-1}} + p_{t_{i-1}}\} & i = 2, \ldots, k. \end{cases} \tag{3.1}$$

Hence, the cost of assigning sequence ω to vehicle v is $c_{\omega,v} \equiv \sum_{i=1}^{k}(s_{t_i} + p_{t_i} - d_{t_i})^+ + c'_v$, where the first term sums up the total tardiness of all tests on this vehicle (assumed to have unit cost per unit of time), and c'_v is the (fixed) cost of using v, expressed in appropriate units.

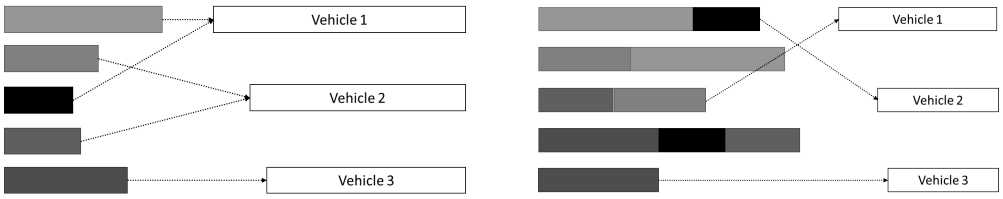


Figure 3.7: Assigning individual tests to vehicles versus assigning test sequences to vehicles

Now the scheduling problem reduces to deciding the assignment of sequences to vehicles to complete all tests as showed in Figure 3.7. Let T denote the set of all test modes specified for a vehicle program and let Ω denote the set of all valid sequences. Our formulation uses binary variables $\lambda_{\omega,v},\ \omega \in \Omega,\ v \in V$, with $\lambda_{\omega,v} = 1$ indicating that the sequence of tests ω is assigned to vehicle v. Note that a variable $\lambda_{\omega,v}$ is only defined if all tests in sequence ω have the same vehicle specification requirements as the vehicle they are assigned to, i.e., those tests can be actually performed on vehicle v. Our optimization model, which we termed **CTSP-MP** (for Master Problem) is as follows:

$$\textbf{CTSP-MP} \quad \text{minimize} \quad \sum_{\omega \in \Omega, v \in V} c_{\omega,v} \lambda_{\omega,v} \tag{3.2}$$

$$\text{s.t.} \quad \sum_{\omega \in \Omega : t \in \omega, v \in V} \lambda_{\omega,v} = 1, \quad t \in T, \tag{3.3}$$

$$\sum_{\omega \in \Omega} \lambda_{\omega,v} \leq 1, \quad v \in V, \tag{3.4}$$

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

In objective (3.2), $c_{\omega,v},\ \omega \in \Omega, v \in V$ is the cost of assigning sequence ω to a vehicle v, and is informed by the cost of using the vehicle and the timing associated with the assignment. Constraints (3.3) ensure that each test gets assigned, where the notation $t \in \omega$ means that t is contained in the sequence ω. Constraints (3.4) are vehicle capacity constraints: for each vehicle v, there can be at most 1 sequence assigned to it.

Set-partitioning integer programming problems similar to **CTSP-MP** typically have tight linear programming relaxations and other features that make them well-suited for the standard branch-and-bound algorithm. However, a potential drawback of models built with composite variables, encompassing an entire sequence of decisions, is their size. Indeed, the potential number of *test sequences* we would need to consider could scale exponentially with the number of tests. Although the number of valid test sequences is limited by compatibility and timing considerations, in some of the instances that we consider in our computational tests, the model exceeded the limit that CPLEX (our solver of choice) was capable of holding in memory. For that reason, we provide two optimization approaches within the TP3S application.

The first optimization approach uses a full enumeration of valid test sequences. If successful, this method is guaranteed to solve **CTSP-MP** to optimality as the example in Figure 3.6 illustrates; however, in some instances this method exceeded

allocated memory and terminated without finding a solution. The alternative approach applies delayed column generation to **CTSP-MP**, thus avoiding enumerating all variables a priori.

Our computational experiments suggested that, while the full enumeration approach was often superior, occasionally it was unable to solve the problem because of memory limitations. The delayed column generation method, although slower, was able to scale more gracefully on larger problem instances, serving as a reliable backup to the first method. These observations have been reinforced through the use of the optimization module of the TP3S application at Ford. As larger test sets are considered in the future, the column generation methodology is likely to prove even more valuable. Therefore, in the following sections, we mainly focus on the description of the delayed column generation algorithm and various techniques for improving its performance.

3.4 Delayed column generation algorithm

We introduce a delayed column generation heuristic for solving **CTSP-MP** to near-optimality; in our computational experiments, this method proved effective in several instances that could not be handled by full-enumeration. Our method is similar to the branch-and-price algorithm proposed in *Barnhart et al.* (1998) for large scale combinatorial problems. Rather than implementing a full branch-and-price algorithm, we first provide a heuristic that solves the root-node partitioning problem using only the columns generated in solving its LP relaxation (branch-and-price implementation is discussed in Section 3.5).

Consider the LP relaxation of formulation **CTSP-MP** obtained by replacing the integrality constraint $\lambda_{\omega,i} \in \{0,1\}$ by $\lambda_{\omega,i} \geq 0$, denoted by **CTSP-LMP** (for Linear Master Problem). The upper-bound constraints $\lambda_{\omega,i} \leq 1$ are implicitly enforced by (3.3). As is typical in delayed column generation algorithms, we start with a subset

of all variables of **CTSP-LMP** and solve this restricted version (**CTSP-RLMP**) of the original problem. Then, we continue adding variables that have the potential to improve the objective function value of the restricted problem until the optimum is reached.

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

$$\textbf{CTSP-RLMP} \quad \text{minimize} \quad \sum_{\omega \in \omega',\, i \in \{1,\ldots,m\}} c_{\omega,v} \lambda_{\omega,v} \tag{3.5}$$

$$\text{s.t.} \quad \sum_{\omega \in \omega' : t \in \omega,v} \lambda_{\omega,v} \geq 1, \quad t \in T, \tag{3.6}$$

$$\sum_{\omega \in \omega'} \lambda_{\omega,v} \leq 1, v \in V, \tag{3.7}$$

$$\lambda_{\omega,v} \geq 0 \quad \omega \in \omega', v \in V.$$

Notice that we replaced equality (3.3) with inequality in (3.6). This is a valid substitution because whenever we find that a test is covered more than once in an optimal schedule, we can always remove duplicates from all but one sequences to make the equality relation hold; the objective function value would not deteriorate, but would not improve either, by optimality. One advantage of doing the substitution is that the dual variables associated with this set of constraints will always be nonnegative.

Let (π, ρ) be the optimal values of dual variables associated with sets of constraints (3.6) and (3.7), respectively. Then the reduced cost of a decision variable $\lambda_{\omega,v}$ is

$$\Gamma_{\omega,v} = c_{\omega,v} - \sum_{t \in T} \delta_{\omega,t} \pi_t - \rho_v. \tag{3.8}$$

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

of the problem

$$\textbf{CTSP-PP} \quad \text{minimize}_{\omega \in \Omega,\, v \in V} \Gamma_{\omega,v}; \tag{3.9}$$

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

By repeating this process, at some iteration, **CTSP-PP** cannot identify further variables that have negative reduced cost. Then, we claim that we have solved **CTSP-RLMP** to optimality. If the optimal solution λ^{RLMP} to **CTSP-LRMP** does not involve fractional values, then we can also claim that we have found the optimal integer solution λ^{RLMP} to **CTSP-RMP** and **CTSP-MP**. Otherwise, we reintroduce the integrality constraints $\lambda \in \{0, 1\}$ and solve **CTSP-RMP** using all the variables we have generated in the column generation process to find a feasible integer solution to **CTSP-MP**. An illustration of the solution process is shown in Figure 3.8

Problem **CTSP-PP** can be viewed as a search for a test sequence and vehicle combination with the smallest reduced cost. When it is possible to explicitly enumerate all valid test sequences, the minimization problem **PP** can be solved by "pricing out" each combination (note that, unlike when solving the full master problem, we don't need to hold the entire set of variables in memory; they can be created and evaluated one by one). When explicit enumeration is impossible or undesirable, we can formulate an integer linear program for solving **CTSP-PP**. We approach **CTSP-PP** by minimizing $\Gamma_{\omega,v}$ over ω separately for each $v \in V$.

Recall that $\mathbf{A} \in \Re^{|T| \times |T|}$ is the matrix that specifies precedence relationships among tests: its (i, j)th element $a_{ij} = 1$ if test t_i is allowed to be performed before test t_j if they are performed on the same vehicle; $a_{ij} = 0$ otherwise. (Matrix $\mathbf{A}$ is

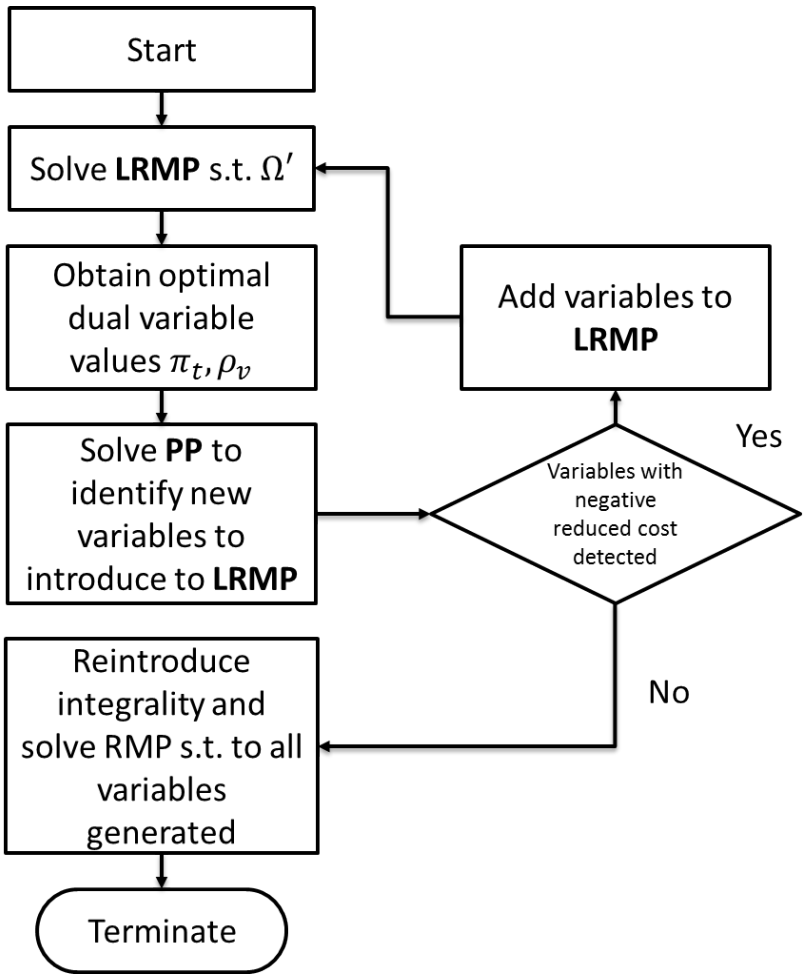


Figure 3.8: A flow chart of the delayed column generation approach

determined based on information provided in screens illustrated in Figures 3.2 and 3.4.) As usual, let $r_t,\ p_t,\ d_t,\ t \in T$, denote the release time, processing time, and due date of test $t \in T$, respectively (see Figures 3.1 and 3.3), and $q_v,\ v \in V$ — the scheduled delivery time of vehicle $v \in V$ (see Figure 3.5).

Let $x_t,\ t \in T$, be binary variables such that $x_t = 1$ if test t is included in the sequence; $y_{t_i,t_j},\ t_i, t_j \in T$, be binary variables such that $y_{t_i,t_j} = 1$ if both tests t_i and t_j are included in the sequence and t_i is performed before t_j (not necessarily immediately); $s_t,\ t \in T$, be continuous variables denoting the starting time of test t; finally, let $o_t,\ t \in T$, be continuous variables that the tardiness of each test. If we consider assigning a sequence to vehicle v, then the corresponding pricing problem

just consists of selecting tests and determining their timing:

CTSP-PP-MILP$_v$

$$\min \quad 1 + \sum_{t \in T} w_t o_t - \sum_{t \in T} \pi_t x_t - \rho_v \tag{3.10}$$

$$\text{s.t.} \quad x_{t_i} + x_{t_j} - 1 \leq y_{t_i,t_j} + y_{t_j,t_i} \leq \frac{1}{2}(x_{t_i} + x_{t_j}), \quad t_i, t_j \in T : t_i \neq t_j \tag{3.11}$$

$$y_{t_i,t_j} = 0, \quad t_i, t_j \in T : a_{t_i,t_j} = 0 \tag{3.12}$$

$$s_{t_i} + p_{t_i} \leq s_{t_j} + M(1 - y_{t_i,t_j}), \quad t_i, t_j \in T : t_i \neq t_j \tag{3.13}$$

$$s_t \geq r_t, \quad t \in T \tag{3.14}$$

$$s_t \geq q_v, \quad t \in T \tag{3.15}$$

$$s_t + p_t \leq d_t + o_t, \quad t \in T \tag{3.16}$$

$$x_t \in \{0,1\},\ y_{t_i,t_j} \in \{0,1\},\ s_t \geq 0,\ o_t \geq 0. \tag{3.17}$$

Formulation **CTSP-PP-MILP**$_v$ is based on the assignment formulation of the test scheduling problem; since it is restricted to only one vehicle, it is significantly easier to solve. The constraints have similar definitions as their counterparts in (2.1)–(2.11). If the optimal value of the problem is negative, the variable $\lambda_{\omega,v}$, with ω corresponding to the sequence of tests identified by solving **CTSP-PP-MILP**$_v$, can be added to the master problem, with $c_{\omega,v} = 1 + \sum_{t \in \omega} w_t o_t$.

In the following sections, we discuss techniques that can be applied to improve computational performance of this column generation approach.

3.4.1 Aggregating the vehicle capacity constraints

One way to strengthen the set-partitioning formulation is by aggregating the vehicle capacity constraints (3.4). This aggregation has multiple advantages: (1) it reduces the number of variables in the formulation; (2) it eliminates symmetry in the

branching tree; (3) it reduces the complexity of the pricing problem.

In the vehicle set V, there are often vehicles that are identical in terms of release dates and, possibly, configurations (if they are not universal models). Suppose two vehicles v_i and v_j are identical in the above sense. Then our formulation of the master problem contains two constraints in the set of (3.4), namely $\sum_{\omega\in\Omega}\lambda_{\omega,v_i} \leq 1$ and $\sum_{\omega\in\Omega}\lambda_{\omega,v_j} \leq 1$. Suppose we solve set-partitioning formulation **CTSP-MP** using branch-and-bound algorithm, and at some iteration, we have a fractional solution where, say, $\lambda_{\omega,v_i} = 0.5,\ \lambda_{\omega,v_j} = 0$. In this case, we will enforce branching constraints on the fractional value of λ_{ω,v_i} by setting either $\lambda_{\omega,v_i} = 1$ or 0. In the latter branch, we may encounter another fractional solution where $\lambda_{\omega,v_i} = 0,\ \lambda_{\omega,v_j} = 0.5$. This solution satisfies the branching constraint we imposed; however, it is effectively identical to the previous fractional solution we have seen. In other words, by interchanging v_i with v_j, the solver does not improve the integrality of the fractional solution. This type of symmetry is due to the similarity of vehicles v_i and v_j, and can be made worse by presence of multiple identical vehicles.

Motivated by this example, we propose to aggregate each subset of identical vehicles. Specifically, we partition the entire set of vehicles V into smaller subsets $V_1, V_2, \ldots, V_m$ such that a subset V_i contains vehicles that have the same release date and configuration. Let V denote the set of prototype vehicles and let $V_1, V_2, \ldots, V_m \subset V$ denote the groups of identical vehicles (i.e., those delivered on the same day and having identical specifications; if each vehicle is unique, $m = |V|$). We have $\cup_{i=1}^{m} V_i = V$, and $V_i \cap V_j = \emptyset, i \neq j$.

Instead of assigning sequences ω to individual vehicles, we will assign them to vehicle subsets in the set-partitioning formulation, i.e., we replace (3.4) with

$$\sum_{\omega\in\Omega}\lambda_{\omega,i} \leq |V_i|, \quad i = 1, \ldots, m.$$

This reduces the total number of variables from $|\Omega| \times |V|$ to $|\Omega| \times m$, where $m \leq |V|$. Moreover, the symmetry caused by identical vehicles will be eliminated. In addition, instead of solving a pricing problem **CTSP-PP-MILP**$_v$ for each vehicle, we can solve it for each vehicle group V_i.

3.4.2 Offline pricing algorithm

In small and moderate-sized problem instances, it may be possible to enumerate all valid sequences in the set Ω. However, we may not want to solve the full master problemtextbfCTSP-M. For example, even when the memory of the computer can hold the entire collection of variable objects, some solvers still have limits on their memory usage. Moreover, the full formulation may be slow to solve even when memory is not the bottleneck. Delayed column generation was introduced to overcome these issues; however, solving the pricing problem **CTSP-PP-MILP** to optimality can sometimes be very expensive, even more expensive than explicitly evaluating the reduced cost of every valid column. In this section, we propose an alternative way of solving the pricing problem by an explicit scan over the entire collection of valid sequences Ω.

First, we show how to calculate the reduced cost given the optimal dual variable values of **CTSP-LRMP** for a variable $\lambda_{\omega,v}$. Suppose ω contains k tests $t_1, t_2, \ldots, t_k$. Recall that we can assume, without loss of optimality, that each test starts as soon as it's available (released) or the previous test in the sequence is completed, whichever comes later. Then the starting times $s_{t_i},\ i = 1, \ldots, k$ can be computed as

$$s_{t_i} = \begin{cases} \min\{r_{t_i}, q_v\} & i = 1 \\ \min\{r_{t_i}, s_{t_{i-1}} + p_{t_{i-1}}\} & i = 2, \ldots, k. \end{cases}$$

Next, we can compute individual tardiness contributions $o_{t_i} = [s_{t_i} + p_{t_i} - d_{t_i}]^+$ of each

test t_i in the sequence. With that, the reduced cost of $\lambda_{\omega,v}$ is

$$\Gamma_{\omega,v} = c'_v + \sum_{i=1}^{k} w_{t_i} o_{t_i} - \sum_{i=1}^{k} \pi_{t_i} - \rho_v.$$

Notice that the computation of the reduced cost for a given variable involves just simple arithmetics. We can apply the formula to the entire collection of columns and then do a scan over the collection to get the column with most negative reduced cost. The complexity is dependent on the maximal number of tests a sequence can contain. Suppose that number is K, then the complexity is $O(|T|^K)$. If we wish to return more than one column at each iteration, e.g., we want the top 5% of all variables which have the most negative reduced costs to be returned, we can sort the entire collection of variables in increasing order of their reduced costs; in this case, the complexity is $O(K|T|^K \log |T|)$.

This evaluation process can be easily parallelized to achieve a perfect linear speed-up due to the independence of the computation between variables. For example, if H processors are available, we can partition the entire set of variables $\lambda_{\omega,v},\ \lambda \in \Omega, v \in V$ into H subsets of equal or almost equal size. Then we apply the formula to every variable in each of the H subsets and scan over the subsets to get the column with most negative reduced cost locally. This process can be executed in parallel as showed in Figure 3.9

Afterwards, we aggregate the best local columns to return the best global column. The complexity is $O(|T|^K/H)$.

If sorting is required, we can first sort the subsets locally after the reduced costs are computed, then combine them into a globally sorted list of variables in linear time.

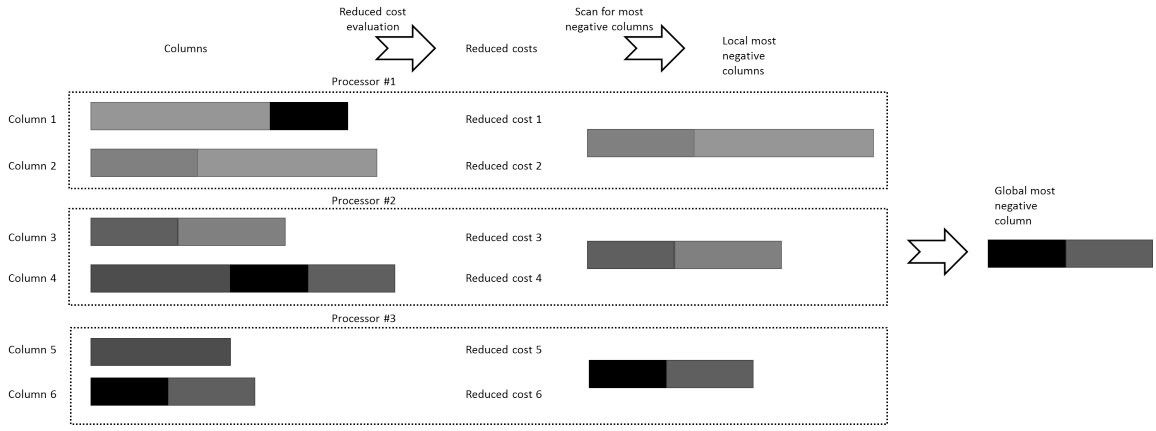


Figure 3.9: Parallel offline pricing algorithm for 6 sequences using 3 processors

3.4.3 Column dominance

Even if we can enumerate all possible columns and store them in computer memory, it is essential to reduce the set of possible columns as much as possible, to speed up the pricing operation as described in Section 3.4.2. It is also useful to remove dominated columns from the master problem after we solve its linear relaxation and proceed to solve the integer version (i.e., one with restored integrality constraints.)

Suppose we have two different sequences ω' and ω'', and for any $t \in \omega'$, we also have $t \in \omega'$, i.e., the tests contained in ω' are a subset of the tests contained in ω''. We say that ω' is dominated by ω'' when assigned to vehicle group i if $c_{\omega',i} = c_{\omega'',i}$. In other words, if variable $\lambda_{\omega'',i}$ is included in the RMP, variable $\lambda_{\omega',i}$ will never be active in the RMP because we can always substitute $\lambda_{\omega',i}$ with $\lambda_{\omega'',i}$ without increasing the objective function value while covering more tests.

If for two different sequences ω' and ω'', ω' is dominated by ω'' when assigned to any vehicle group i, $i = 1, \ldots, m$, then we say sequence ω' is dominated by ω''. We can remove any variables involving using sequence ω' whenever a variable using ω'' is used.

3.4.4 Strategies to generate compatible columns

Notice that we do not always need to solve the pricing problem **CTSP-PP** to optimality. Indeed, any column with negative reduced column can be added to **CTSP-LRMP** to improve the objective function value. Therefore, one strategy for generating columns is to add all columns found while solving the pricing problem that have negative reduced costs, or are otherwise deemed beneficial to the overall solution, to the **CTSP-LRMP**. This is exceedingly handy if we adopt the offline pricing strategy as described in Section 3.4.2 by storing all columns with negative reduced column while scanning over the entire collection of them. However, this strategy may add too many useless columns that will never be used in the **CTSP-LRMP**, which would consume extra memory and make the size of the **CTSP-LRMP** excessive.

On the other hand, if we adopt an overly parsimonious approach to choosing columns to add to **CTSP-LRMP** in each iteration, we may encounter a feasibility issue in the last step of our algorithm. In particular, when we attempt to find an *integer* solution to the restricted master problem, the instance that uses only the columns generated in the process of solving the linear relaxation of the master problem may be infeasible once integrality restrictions are restored.

In light of the above, we can adopt a strategy which balances the added complexity of adding too many columns and the benefit of returning multiple columns at each iteration. Specifically, we return a set of columns that can produce a full integer feasible solution to the **CTSP-LRMP**. Such columns together satisfy the constraints (3.3) and (3.4). We adopt a very simple heuristic algorithm to generate such collections of columns.

We first return the column with the most negative reduced cost to the **CTSP-LRMP**. After that, we remove all tests that are contained in that column and the vehicle used from the pricing problem data. Then, we re-solve the pricing problem for the column with the smallest reduced cost using remaining tests and vehicles. We

repeat that procedure until we cover the entire set of tests.

If we solve the pricing problem as an MILP, i.e., **CTSP-PP-MILP**, this procedure involves solving the MILP repeatedly. However, if we adopt the strategy of offline pricing described in section 3.4.2, we can achieve the desired result with just one scan over the entire sorted collection of available columns with complexity $O(K|T|^{K+1}\log|T|)$ if we assume $|V| \sim |K|$.

Computational experiments have showed that by generating such a set of compatible columns at each iteration, we can significantly improve the convergence speed of the delayed column generation algorithm for solving the **CTSP-LRMP** (even though the amount of time it takes to do the pricing increases), as well as ensure feasibility of its integer counterpart in the last step of the algorithm.

3.5 Branch-and-price method

Delayed column generation algorithm described in Section 3.4 is an efficient way to solve large-scale linear programming problems. However, it can only solve the linear relaxation of the original set-partitioning formulation to optimality. If we simply reintroduce integrality constraints on $\lambda_{\omega,v}$ and solve the resulting IP using only columns that were generated in the process of solving **CTSP-LRMP**, we are not guaranteed to find an optimal solution to the original problem.

In order to find the optimal solution to the original set-partitioning formulation **CTSP-MP**, where λ's are binary variables, we need to implement a branch-and-bound algorithm, specifically, a branch-and-price algorithm, since column generation will be necessary to solve linear relaxations of problems at the nodes of the branch-and-bound tree. This section describes the details of our implementation of branch-and-price.

3.5.1 Branching rules

Recall that, in the branch-and-price algorithm, a linear relaxation of an integer program at each node of the branch-and-bound tree will be solved via delayed column generation. Therefore, branching rules must be designed in a way that maintains applicability of the column generation scheme, and in particular, the pricing subproblem that we developed.

To illustrate the issue at hand, suppose we adopt the traditional branching approach: if the solution to the LP relaxation at a node is non-integer, find a fractional variable $\lambda_{\omega,v} \in (0,1)$ and create two branches by adding constraints $\lambda_{\tilde{\omega},\tilde{v}} = 1$ and $\lambda_{\tilde{\omega},\tilde{v}} = 0$, respectively.

In the first branch, the sequence $\tilde{\omega}$ is always assigned to vehicle $\tilde{v}$. We can remove the tests contained in sequence $\tilde{\omega}$ from the test portfolio T and vehicle $\tilde{v}$ from the vehicle set V, and apply delayed column generation to the set-partitioning formulation with the remaining tests and vehicles.

In the second branch, we forbid the assignment of sequence $\tilde{\omega}$ to vehicle v, and we need to respect this branching constraint when generating columns by solving the pricing problem. However, we do not have control over what types of columns will be generated when we solve the integer programming formulation of the pricing problem, so if it turns out the column with the most negative reduced cost does not satisfy the branching constraint, we need to somehow find another column which does not use the current test sequence and vehicle assignment while optimizing the reduced cost among all remaining columns. As we go deeper into the branching tree and add more branching constraints of this type, keeping track and enforcing these constraints in subsequent pricing problems will become increasingly complex. We can try enforcing these hierarchies of constraints by introducing cuts, but these can fundamentally change the structure of the pricing problem and may cause more computational challenges. This issue is well-recognized in the literature on branch-

and-price methods. A common way to resolve this difficulty is to use alternative branching rules; the general idea is, rather than branching on the variables in the set-partitioning formulation, to branch on appropriately defined auxiliary variables.

For the problem at hand, we propose a branching rule based on the integrality of the following auxiliary variables:

$$y^v_{t_i,t_j} = \sum_{\omega\in\Omega:\ t_i,t_j\in\omega,\ t_i\to t_j} \lambda_{\omega,v}, \tag{3.18}$$

defined for every vehicle v and every pair of tests t_i and t_j, and

$$x^v_t = \sum_{\omega\in\Omega:\ t\in\omega} \lambda_{\omega,v}, \tag{3.19}$$

defined for each vehicle v and test t.

Let $\lambda_{\omega,v},\ \omega \in \Omega, v \in V$ be an optimal solution of the linear relaxation of the master problem **CTSP-MP** defined in (3.2)–(3.4). We can show that integrality of y's and x's defined in (3.18) and (3.19) is necessary and sufficient for integrality of λ's.

Necessity follows trivially, since a sum of integers is also integer.

We will establish sufficiency by contradiction: suppose x's and y's are all integers, but there exist $\omega' \in \Omega$ and $v' \in V$ such that $\lambda_{\omega',v'} \in (0,1)$. Suppose $t_i \in \omega'$. In order to satisfy constraint (3.3) for t_i, there must exist a different variable $\lambda_{\omega'',v''} \in (0,1)$, with $t_i \in \omega''$. Consider the following two cases:

- Case 1: $v' = v''$. Since there cannot be any duplicate columns generated by the column generation method, we must have $\omega' \neq \omega''$. Therefore, we must have one of the following two sub-cases:
  - Case 1a: $\exists t_j$ such that $t_j \in \omega',\ t_j \notin \omega''$;
  - Case 1b: $\exists t_j$ such that $t_j \notin \omega',\ t_j \in \omega''$.

In either of the two cases, we have

$$\begin{aligned}1 &= \sum_{v} \sum_{\omega: t_i \in \omega} \lambda_{\omega,v} \geq \sum_{\omega: t_i \in \omega} \lambda_{\omega,v'} = \sum_{\omega: t_i \in \omega, t_j \in \omega} \lambda_{\omega,v'} + \sum_{\omega: t_i \in \omega, t_j \notin \omega} \lambda_{\omega,v'} \\ &> \sum_{\omega: t_i \in \omega, t_j \in \omega} \lambda_{\omega,v'} = \sum_{\omega: t_i, t_j \in \omega; t_i \to t_j} \lambda_{\omega,v'} + \sum_{\omega: t_i, t_j \in \omega; t_j \to t_i} \lambda_{\omega,v'} \\ &= y^{v'}_{t_i,t_j} + y^{v'}_{t_j,t_i} > 0.\end{aligned}$$

The first strict inequality follows since in either case 1a or 1b, there exists some $\lambda^{v'}_{\bar{\omega}} > 0$, where $t_i \in \bar{\omega},\ t_j \notin \bar{\omega}$. The second strict inequality follows because in either case 1a or 1b, there exists some $\lambda^{v'}_{\hat{\omega}} > 0$, where $t_i, t_j \in \hat{\omega}$. Therefore $\sum_{\omega: t_i, t_j \in \omega} \lambda_{\omega,v'} > 0$.

However, $y^{v'}_{t_i,t_j}$ and $y^{v'}_{t_j,t_i}$ are integers, and their sum cannot be strictly between 0 and 1, leading to a contradiction.

- Case 2: $v' \neq v''$. We have

$$1 = \sum_{v} \sum_{\omega: t_i \in \omega} \lambda_{\omega,v} > \sum_{\omega: t_i \in \omega} \lambda_{\omega,v'} = x^{v'}_{t_i} > 0.$$

The first strict inequality follows since, for $t_i \in \omega''$, $\lambda_{\omega'',v''} > 0$. The second strict inequality follows since $\lambda_{\omega',v'} > 0$ for $t_i \in \omega'$. However, $x^{v'}_{t_i}$ is an integer, which leads to a contradiction.

In summary, if all x's and y's are integers, then all λ's must be integers, too.

We also make a note of the fact that if λ's are feasible for the linear relaxation of (3.2)–(3.4), then each of the auxiliary variables has a value in the interval [0, 1].

Next, we show that by imposing branching constraints on x and y variables, we do not add more complexity to the pricing problem used for delayed column generation at a node in the branching tree. If at some node in the tree we identify that $y^{v}_{t_i,t_j}$ is fractional for some tests t_i and t_j and vehicle v, we can create two branches by

imposing constraints $y^v_{t_i,t_j} = 1$ and $y^v_{t_i,t_j} = 0$.

- Forcing $y^v_{t_i,t_j} = 1$ is equivalent to making the following modifications to the pricing problems:
  - in pricing problems on vehicle v, use both t_i and t_j, and t_i precedes t_j;
  - in pricing problems on vehicles other than v, since neither t_i nor t_j can be assigned to them, delete them from the set of available tests.
- Forcing $y^v_{t_i,t_j} = 0$ is equivalent to adding a new compatibility rule that t_i cannot precede t_j on vehicle v; other pricing problems remain unchanged.

If we identify that x^v_t is fractional for some test t and vehicle v, we can create two branches where

- forcing $x^v_t = 1$ is equivalent to forcing test t to be assigned to vehicle v in the corresponding pricing problem;
- forcing $x^v_t = 0$ is equivalent to forbidding assignment of test t to vehicle v in the corresponding pricing problem.

If we formulate the pricing problem as an MILP as described in Section 3.4, the above requirements can be enforced by appropriate fixing of values of variables in the model.

3.5.2 Other implementation details

3.5.2.1 Finding best incumbent

In order to evaluate the optimality gap early in the branching tree, we need an incumbent, i.e., a feasible integer solution, to provide the upper bound on the set-partitioning formulation. Then, whenever we are solving a pending node and get an integer solution, we can update the best incumbent if the current integer solution

provides a better objective function value. However, the branch-and-bound process may not generate an initial integer solution for quite some time.

Alternatively, we can use heuristics to find feasible integer solutions at branching nodes. Section 3.4 has provided us a way of doing so by solving an integer version of the set-partitioning formulation using subsets of columns. However, doing this at each node is rather computationally expensive. Therefore, we only try to solve the integer version of the set-partitioning formulation at the root node of the branching tree, and every k nodes when exploring remainder of the tree. The choice of k depends on our preferences for a particular balance between desire for a good integer solution and dedication of computing resources.

3.5.2.2 Node selection strategy

In order to balance the emphasis on finding an optimal solution and quickly detecting integer solutions along the tree, we use an alternating strategy to select which node to explore next. Specifically, we maintain two synchronized queue data structures, a LIFO queue (stack) and a priority queue where nodes are ordered by the best bound potentials, i.e., objective function values of their parent nodes (the lower bound for the linear relaxation at the current node). At even steps, we fetch a node from the LIFO queue; at odd steps, we fetch a node from the priority queue.

3.5.2.3 Hierarchical integrality check

Let's take a look back at the branching rules we discussed in Section 3.5.1. Fixing $y^v_{t_i,t_j}$ to zero or one for some $t_i, t_j \in T$ and $v \in V$ is equivalent to making tests t_i and t_j (in this order) incompatible, or fixing the assignment of these two tests to this vehicle. Similarly, fixing x^v_t to zero or one is equivalent to forbidding the assignment of this test to this vehicle, or requiring it. In both cases, the branching constraints are only taking effect for the pricing problem associated with an individual vehicle,

which does not contain much useful information since we have multiple vehicles to consider in the pricing problem. If the final column generated by the pricing problem does not use vehicle v, then the newly added constraint implied by the branching rules is not useful.

Whenever we create branches in the tree, ideally we want to divide the feasible region in a more balanced way. Therefore, rather than checking for fractional $y^v_{t_i,t_j}$'s for every $t_i, t_j \in T,\ v \in V$, we instead define

$$z_{t_i,t_j} \equiv \sum_{v \in V} \left(y^v_{t_i,t_j} + y^v_{t_j,t_i} \right).$$

In other words, $z_{t_i,t_j} = 1$ if t_i, t_j are assigned to the same vehicle, regardless of their orders; and 0 otherwise.

It can be easily shown that if $y^v_{t_i,t_j}$ are all integers for all $v \in V$, then z_{t_i,t_j} must be integer, since the sum of integers must be an integer. On the other hand, if we detect a fractional z_{t_i,t_j} for some $t_i, t_j \in T$, we can conclude that there must be $y^v_{t_i,t_j}$ or $y^v_{t_j,t_i}$ that is fractional for some $v \in V$. In this case, we need to do further branching to eliminate fractional variable values. Instead of further detection of the fractional x variables which contribute to the fractional value of y variable, we can directly partition the feasible region into two branches: $z_{t_i,t_j} = 1$ and $z_{t_i,t_j} = 0$. The second constraint is equivalent to imposing an incompatibility relation between t_i and t_j in the following pricing problems, while the first constraint is equivalent to always assigning the two tests t_i and t_j together in every generated column.

Thus, we can partition the feasible region in a more balanced way, while preserving the structure of the pricing problems.

3.6 Numerical results

3.6.1 Data preparation and testing platform

We implemented our algorithms in Java 8 using IBM ILOG CPLEX 12.6 as the linear programming and MILP solver. Models were built using CPLEX Concert Technology API in Java. We did our own implementation of the logic of delayed column generation and branch-and-price algorithms, i.e., CPLEX was the only third-party software package used in our program. All parallel algorithms were implemented in a master-worker framework by using Java's threadpool feature. All algorithms were tested on a machine with Intel Xeon E1230 processor with 32GB RAM.

We tested our algorithms on two types of data instances: real and synthesized.

As part of the development of TP3S application, we collaborated with Ford to obtain data associated with 7 past vehicle programs (safety testing part) for which schedules were developed manually. For each of the vehicle programs, we had information on the portfolio of tests that need to be performed, vehicles available and their release times, and engineering information on the compatibility and precedence relations between different tests. Table 3.1 provides high-level characteristics of those programs. As we can see in Table 3.1, real instances vary in the number of total tests depending on the vehicle program. A new vehicle model that only involves incremental changes compared with previous year model requires a small set of tests since most of the designs have been thoroughly tested in previous years. In contrast, a brand new design may require a large portfolio of tests to be performed on the prototype vehicles for the purpose of validation. However, for all the instances we have seen, it is rare that the number of tests exceeds 100.

The value in the "Num Vehicles" column of the table is the cardinality of the set V and reflects the budget of vehicles made available for testing by the safety group. This number usually serves as an upper bound, albeit a fairly conservative

Table 3.1: The table provides data characteristics of problem instances used in our computational experiments.

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

one, on the number of vehicles ultimately required. An obvious selection for that upper bound is just the number of total tests in the vehicle program since we can always assume that a schedule that assigns each test to a dedicated vehicle is a valid one. The "Density" column captures the level of incompatibility between tests. It is computed as the percentage of 0's in the binary matrix $\mathbf{A}$ discussed in Section 1.2.1, i.e., $1-\sum_{i=1}^{|T|}\sum_{j=1}^{|T|} a_{ij}/|T|^2$. A higher density of the matrix $\mathbf{A}$ means that it is unlikely that many tests can share a vehicle, which is common in the crash testing procedure considering their destructive nature. Meanwhile, a lower density of incompatibility means more flexibility in the scheduling. In practice, we are often interested in solving instances where the vehicle specification requirements of test requests are relaxed. This is useful for engineers to measure the impact of engineering complexity on resources required for testing. Therefore, for each of the problem instances, we also considered a version where such specification constraints are relaxed. In Table 3.1, they are marked with "r" after the instance ID.

To better evaluate the performance of our algorithms, we also synthesized a wide

Table 3.2: Parameters controlled during the data instance synthesis process

Parameter	Range
Number of tests	10 – 300
Number of vehicles	8 – 240
Incompatibility density (L)	0.8, 0.85, 0.9, 0.95
Tightness of time window (F)	1.0, 1.5, 2.0, 2.5

range of data instances that resemble the real instances, as well as instances that were more extreme in terms of size, rarely seen in current reality, to better understand any limitations of our algorithms.

We varied several parameters during the instance generation process: (1) the number of tests; (2) the number of budgeted vehicles and their release times; (3) the compatibility and precedence relations between tests; (4) the tightness of the time windows. For (1), we generated a wide range of instances, ranging from 10 to 300 tests. The durations of tests were sampled from the empirical distribution that was based on tests from real instances 1–7. For (2), we set vehicle budget to be equal to 80% of the total test number (i.e., we assumed that at least 20% of the tests would share a vehicle with another test). For (3), after generating the tests, we set a target density level L and sampled the value of each entry of compatibility matrix $\mathbf{A}$ from a Bernoulli distribution with success probability $1 - L$. At the last step, we generated release times and milestones (due dates) for tests randomly, by controlling the time between the two endpoints via a "tightness factor" F. Setting $F = 1$ means that the time window is roughly equal to the average duration of the tests, which means there is little flexibility in shifting the execution of the test without causing any time penalty. Therefore, a low factor F means tighter time window constrains and less flexibility in terms of timing, and vice versa. Table 3.2 summarizes the choices of controlled parameters in the generation of the synthesized data.

By enumerating all combinations of the parameter values, we generated 256 synthetic data instances in addition to the 7 original and 7 relaxed instances we obtained

Table 3.3: Definition of size groups

Size group	Num. tests range
Small	< 30
Typical	30 – 60
Moderate	70 – 90
Large	100 – 150
Extreme	> 150

from Ford. We can partition these instances into 5 groups based on the number of tests: *small, typical, moderate, large*, and *extreme*, where *small* contains less than 30 tests, *typical* contains 30-60, *moderate* contains 70-90, *large* contains 100-150, and *extreme* contains 150-200 (see Table 3.3).

3.6.2 Results

We first tested the full enumeration approach applied to the **CTSP-MP**, as well as delayed column generation algorithm (without branch-and-price), on real instances. Table 3.4 shows the runtime of the two algorithms. The full enumeration algorithm solved most of the instances to optimality. However, we have two instances, 7r and 10r, where the number of variables exceeded available memory; we mark these cases as "−" in the table. The column generation algorithm is often slower, but it was able to handle all the instances, solving all but two of them to optimality (and coming within 3.5% and 0.3%, respectively, of optimality for the remaining two instances).

When testing the delayed column generation algorithm on the synthesized data instances, we set the time limit for the last step — solving **CTSP-RMP** with all columns generated while solving **CTSP-LRMP** — to 300 seconds. The objective function value of **CTSP-LRMP** provides a lower bound for **CTSP-MP**. Therefore, when measuring optimality, we compare the best incumbent we attained while solving **CTSP-RMP** to the optimal value of **CTSP-LRMP**.

In general, the delayed column generation algorithm scales well and can solve most

Table 3.4: The table provides run times (in seconds) for our full-enumeration and column-generation algorithms, where a dash indicates that the algorithm did not terminate.

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

instances in *small, typical, moderate,* and *large* groups to optimality. The average statistics are shown in Table 3.5. "Avg. RMP Gap" is the average of optimality gaps returned by the solver when solving the integer version of the **CTSP-RMP**. "Avg. Opt Gap" is the average of optimality gaps computed by comparing the best integer solution with the optimal objective function value of the relaxation **CTSP-LRMP**. As we can see from the table, except for the *extreme* group, the delayed column generation performed exceedingly well both in terms of solution times and solution quality. But as the number of tests grows larger, we are apparently not introducing enough variables into **CTSP-RMP**, and the integer problem becomes challenging to solve. The algorithm starts to struggle to find a good quality solution. We also recorded the percentage of the solved instances (which we defined as instances for which we can obtain a solution within 5% optimality gap) in Figure 3.10 for different size groups. Similarly, we are able to solve almost all instances in groups other than *large* and *extreme*. We show the dependence of optimality gaps and solution times on the number of tests in Figure 3.6. As we can see, the optimality gap increases

Table 3.5: Average performance of delayed column generation algorithm

Size group	Sol. time (sec)	Avg. RMP Gap	Avg. Opt Gap
Small	9.46E-03	0.00%	0.00%
Typical	1.46E+00	0.00%	2.11%
Moderate	3.54E+01	0.00%	2.40%
Large	2.50E+02	2.34%	3.96%
Extreme	2.18E+03	13.63%	17.10%

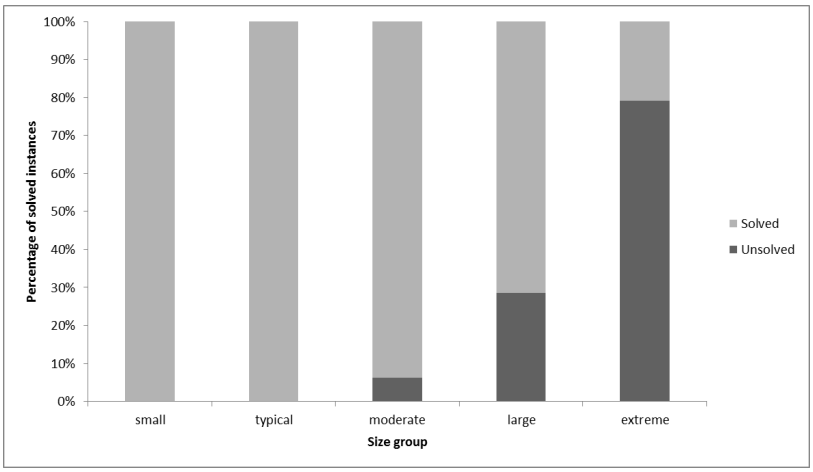


Figure 3.10: Percentage of instances solved (<5% optimality gap) by delayed column generation in each size group.

drastically when the number of tests exceeds 150.

Next, we tested the branch-and-price algorithm on synthesized instances. For the branch-and-price algorithm, the optimality gap can be computed as the relative difference between the best incumbent and the best lower bound, which is the lowest objective function value of the linear relaxations of all pending nodes. We terminated the algorithm after 1800 seconds. The average performance statistics are shown in Table 3.6, which includes average solution time, optimality gap, and number of tree nodes explored in each group. From the table, we can see the branch-and-price algorithm performs extremely well for all groups except *extreme*. Specifically, we can solve instances in *small, typical, moderate* almost to optimality. Additionally, we also compare the solution attained by branch-and-price algorithm with those attained by delayed column generation heuristics which corresponds to just solving the root

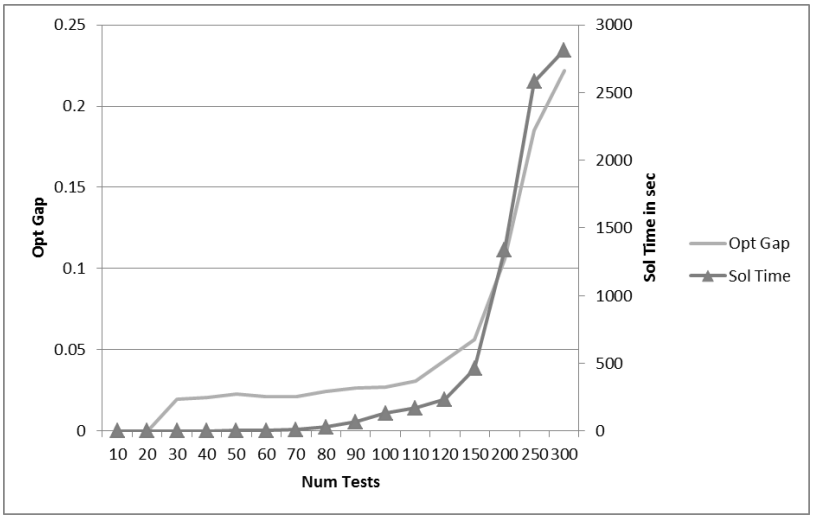


Figure 3.11: The optimality gap and solution time (in secs) versus the number of tests

Table 3.6: Average performance of branch-and-price algorithm

Size group	Sol. Time	Opt Gap	Nodes Exp.	Opt Gap Improv.	Obj Val Improv.
Small	7.25E-02	0.01%	1.848485	0.06%	0.01%
Typical	1.34E+02	0.12%	324.6508	1.99%	0.67%
Moderate	8.48E+02	0.82%	391.4167	1.58%	0.87%
Large	1.09E+03	3.27%	113.1875	0.94%	0.64%
Extreme	2.60E+03	11.89%	4.090909	2.70%	2.36%

node in the branching tree. "Opt Gap Improv." column of the table presents the absolute improvement in the optimality gap (%). This can be achieved either by finding a better incumbent or improving the lower bound to prove optimality. We also include the column "Obj Val Improv." containing the percentage improvement in the objective function value of the best feasible solution found by the two algorithms. As we can see, for *small* instance group, the improvements are trivial. But for *typical, moderate,* and *large* groups, we do improve the optimality gap significantly as well as succeeded in finding better solutions. A graphical comparison of the optimality gaps obtained by the branch-and-price algorithm and delayed column generation algorithm is shown in Figure 3.12. Even for *extreme* instance group, doing further branching can

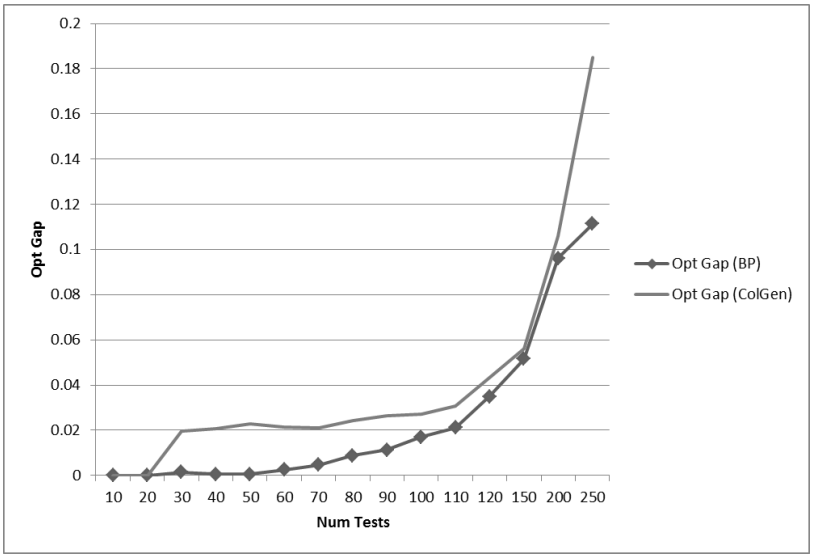


Figure 3.12: Optimality gap: branch-and-price vs. delayed column generation

still help to find better solutions compared with just solving the root node. Among all instances, branch-and-price algorithm improves the optimality gap of 69% of them. Among the 69% improved instances, in 62% of them the branch-and-price algorithm found a better incumbent for; in 7% of them branch-and-price improved the lower bound solely without finding a better solution.

For a complete results of using delayed column generation and branch-and-price algorithms to solve **CTSP**, we ask reader to refer to Table A.1 and Table A.2.

CHAPTER IV

Scheduling of safety crash tests under supporting resource constraints

4.1 Introduction

In Chapter III we considered a version of the test scheduling problem where prototype vehicles are the only resource required for the execution of tests. In that setting, the primary goal of optimization is to reduce the number of vehicles used, combined with an optional penalty function related to time metrics, used when treating time window constraints as soft ones (due dates) rather than hard ones (deadlines).

However, in reality, the execution of a test has requirements far more complex than just a prototype vehicle. For example, when conducting a crash test, the vehicle used will be delivered to a safety testing lab. At the lab, the vehicle is first equipped with different instruments and equipment such as sensors, dummies, cameras, weight ballasts, etc., in order to collect data during the crash, simulate injuries to vehicle occupants, and monitor vehicle deformation and dummy motion. For some types of equipment, e.g., dummies that are equipped with high precision sensors and microcomputers, the lab may have access to only a limited number of units — in some cases, only one. In these circumstances, if too many crash tests that require access to such a resource during their execution are scheduled to be conducted on the same

or consecutive days, it becomes impossible to finish them in a timely manner due to lack of access to sufficient resources. In these situations, we would spread out the execution of such tests to alleviate the "peak time" demand for the scarce resource.

Figure 4.1 gives an example of over-utilization of a resource. In this example with 5 tests, each test is requiring 1 unit of a particular resource during its execution. However, the daily supply for this resource is only 2 units. Therefore, the schedule proposed on the left, which would use 3 units of this resource on one of the days is not feasible. On the other hand, if we introduce some buffer time (delay of execution for a following test) between tests, then we can reduce the peak usage of the resource from 3 to 2, resulting in a feasible schedule on the right.

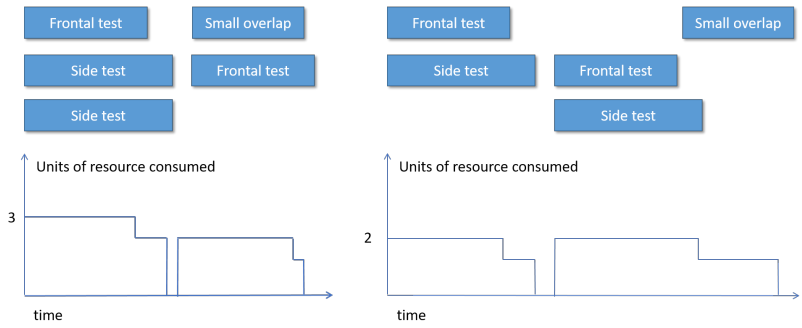


Figure 4.1: Example of resource over-utilization

In addition to equipment and instruments, there are other types of constrained resources in the testing process: manpower, facility availability, etc. In this chapter, we study a version of the crash test scheduling problem in which limited availability of such supporting resources is taken into account. Depending on the types of the crash tests considered, different subsets of resources may be needed. For example, different dummies are required in different tests, and some of the crash tests may not even require a dummy to be installed. A single test may also require different subsets of resources during different stages. For example, the vehicle might be delivered to different facilities for different types of preparation work, thus requiring different

facility resources.

In order to simplify the problem, we assume that a single resource is consumed during the execution of a crash test at a constant daily rate of 1 unit. We assume that the supply of this resource available on day e (for *epoch*) is R_e. Although simplifying, this assumption does not sacrifice a lot of generality since it can easily adapted to respect the different availabilities of multiple resources and stages of execution. For example, we can modify the constant consumption rate to represent the changing demands on the resource.

With the introduction of capacity constraints on a supporting resource, the goal is to find a schedule that minimizes the objective function that combines the cost of using prototype vehicles and a time penalty, as in the previous chapter, but now subject to the resource capacity constraint at each time epoch. We refer to this problem as the crash test scheduling problem under supporting resource constraints (**CTSPR**).

4.1.1 Literature review on scheduling under resource constraints

Resource constraints other than just the machines in the scheduling context frequently arise in practice, and thus there are many papers that study this variation of the problem. *Blazewicz et al.* (1983) gives a comprehensive review on this topic under different resource type and machine settings. Classification method and complexity analysis are given for the unit processing time special case. Optimality conditions and polynomial algorithms are given for this special case. More recently, *Hartmann and Kolisch* (2000); *Kolisch and Hartmann* (2006) give surveys of a wide range of heuristics on solving the Resource-Constrained Project Scheduling problem. Computational experiments have been conducted to compare the performance of various methods on a standard library of problem instances. *Garey and Johnson* (1975) examine the computational complexity of scheduling problems associated with a certain

abstract model of a multiprocessing system. It can be showed that under some special settings, when the number of processors is either 2 or 3, the problem can be solved with low order polynomial algorithms. But in general, even with one resource type, the problem is NP-complete and hard to solve. *Davis and Heidorn* (1971) propose an algorithm to solve the resource-constrained project network scheduling problem, where every task requires unit processing time. The approach is a form of bounded enumeration originally developed for the assembly balancing problem. Computational results on small instances are reported. *Chen* (2005) studies the problem where tasks are of various types and whenever a machine needs to switch from processing one type of work to another, a non-renewable resource is consumed; the supply of the resource is limited, and therefore there is a limit on the number of changes of work types. The paper proposes a heuristic to solve this type of problems.

Except heuristics, which are mostly commonly used in solving scheduling problems with resources constraints, there are other papers which explores combining several methods to solve this type of problems. *Christofides et al.* (1987) proposes a branch-and-bound algorithm for projects with resource constraints. The algorithm is based on the idea of using disjunctive arcs for resolving conflicts that are created whenever sets of activities have to be scheduled whose total resource requirements exceed the resource availability in some periods. Various lower bounding techniques including linear relaxation and Lagrangian relaxation are proposed. Computational results on instances up to 25 activities and 3 types of resources are reported. *Edis and Oguz* (2011) combines Lagrangian relaxation and Constraint Programming to solve the resource-constrained scheduling problem. The resource constraint is relaxed by introduction of Lagrangian multipliers and solved iteratively by a sub-gradient method for optimal values of multipliers. Then the partial solution obtained by solving the relaxation is fed into a constraint programming formulation to get a feasible solution.

In this chapter, we discuss extensions of the models and methods of Chapter III to this variation of the test scheduling problem. In addition to considering scheduling tests in a single vehicle program, we consider scheduling multiple programs whose testing periods may overlap, forcing them to share limited resources.

4.2 Mixed integer formulation

One way to incorporate resource constraints into the problem is to extend the mixed integer program (2.1)–(2.11) to keep track of resource usage in every epoch. Namely, we can discretize the time horizon into epoch slots and assign test starting times to these slots (rather than using continuous variables), enabling us to calculate total resource consumption in every epoch. Figure 4.2 gives an illustration of the resulting constrained assignment model. The green slots indicate the starting times of different tests assigned to the vehicles.

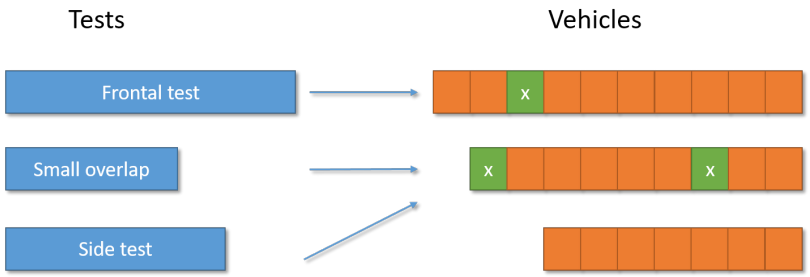


Figure 4.2: Time discretization assignment model

In this formulation, we define binary decision variables $x^e_{t,v}$, where $x^e_{t,v} = 1$ if we decide to assign test t to vehicle v, and the execution of t starts at epoch e.

Then the formulation is:

$$\text{minimize} \sum_{v \in V} c'_v u_v + \sum_{t \in T, v \in V} \sum_{e \in E} \theta^v_{t,e} x^e_{t,v} \tag{4.1}$$

$$\text{s.t.} \sum_{v \in V, e \in E} x^e_{t,v} = 1 \quad t \in T \tag{4.2}$$

$$\sum_{t \in T, e \in E} x^e_{t,v} \leq u_v \quad v \in V \tag{4.3}$$

$$x^e_{t,v} = 0 \quad e \in E : e \leq \max\{q_v, r_t\}, t \in T,\ v \in V \tag{4.4}$$

$$1 - x^e_{t_i,v} \geq \sum_{e \leq e' \leq e + p_{t_j}} x^{e'}_{t_j,v} \quad t_i, t_j \in T,\ e \in E \tag{4.5}$$

$$1 - x^e_{t_i,e} \geq \sum_{e' \geq e} x^{e'}_{t_j,v} t_i, t_j \in T : a_{ij} = 0,\ v \in V \tag{4.6}$$

$$\sum_{t \in T, v \in V} \sum_{e - p_t \leq e'} x^{e'}_{t,v} \leq R_e \quad e \in E \tag{4.7}$$

$$x^e_{t,v} \in \{0, 1\},\ u_v \in \{0, 1\}$$

As before, objective (4.1) is the total cost of the vehicles used and cost attributed to test completion tardiness $\theta^e_{t,v} = \max\{e + p_t - d_t, 0\}$, which is the tardiness of test t if it starts in epoch e. Also familiar are constraints (4.2), which make sure every test is covered, and (4.3), which forbid any assignments to a vehicle if it is not being used. (4.4) are the time window constraints, indicating that no test can start before it is released and vehicle to which it is assigned becomes available. (4.5) prevent two tests that are assigned to the same vehicle from overlapping with each other during their executions. For a time epoch e, we look back p_j periods and check if test t_j starts within this timeframe. If so, then the execution of t_j is ongoing and no test assigned to this vehicle should start its execution in epoch e. (4.6) enforce the ordering restriction for a pair of tests t_i and t_j where t_j cannot follow t_i on the same vehicle. Finally, (4.7) verify that resource consumption across all tests in each epoch is within available limits.

The above model contains only binary decision variables. The number of variables depends on the number of tests $|T|$, the number of vehicles $|V|$, and the length of time horizon $|E|$. In a typical problem instance, the length of planning horizon (taking into account the grace period following the latest due date) is usually around 1 year, i.e., on the order of 300 epochs. Therefore, we can see the timing horizon significantly amplifies the number of decision variables.

Numerical experiments have shown that this model is generally difficult to solve especially for large instances where more tests, or longer planning horizon, are involved. Therefore, we will approach this problem with an extension of the set-partitioning formulation of Section 3.3.2 instead.

4.3 Set-partitioning formulations

We want to adapt the set-partitioning formulation and the solution approach we used in Chapter III to work with the resource constraints. However, we need to revisit some of the assumptions made in developing the original set-partitioning model. In particular, in Section 3.3.2 we assumed that once the sequence of tests assigned to a particular vehicle has been decided, the tests will be executed essentially back-to-back (see formula (3.1) for calculation of test start times). However, when a supporting resource is involved, it is possible that a test that is otherwise ready for execution has to be delayed until the required resource becomes available.

In view of the above discussion, in addition to deciding which tests should be combined, and in which order, we also need to decide the starting times of tests. Hence, we extend the definition of a sequence to specify the execution timing decisions of the tests contained in it. Returning to the example of Figure 4.1, we have 2 schedules each using 3 sequences; the corresponding sequences on the left and right contain the same tests in the same order, but have different timing.

To summarize, here we define a (timed) sequence as an ordered set of tests together

with the start time of each test. To be valid, as before, the tests in the sequence have to be compatible and their order must be acceptable given the rehit rules; moreover, their execution start times must be in agreement with release times of the tests and the vehicle, and tests in the sequence may not overlap. Notationally, while the actual start times will depend on the vehicle to which the sequence is assigned, the timing decisions within the sequence can be defined by a sequence of shifts, or delays, $\Delta_{t_1}, \ldots, \Delta_{t_k}$ (here, tests in the sequences are executed in the order $t_1, \ldots, t_k$). When assigning these tests to vehicle v with release time q_v, the actual start time of each tests can then be computed as $s_{t_i} = q_v + \Delta_{t_i},\ i = 1, \ldots, k$. After the start times are determined, we can calculate the tardiness of each test and the resulting cost of assigning the sequence to the vehicle in the same way as in Chapter III.

Then, the scheduling problem reduces to deciding the assignments of these timed sequences to vehicles, subject to resource availability constraints in each epoch. Let Ω be the set of all timed sequences and E — the set of epochs, i.e., our planning horizon. The model is similar to the master problem in Section 3.4: we define binary decision variables $\lambda_{\omega,v}$ to represent assignments of (timed) sequences to vehicles, and formulate the problem as follows:

$$\textbf{CTSPR-MP} \quad \text{minimize} \sum_{\omega\in\Omega, v\in V} c_{\omega,v}\lambda_{\omega,v}, \tag{4.8}$$

$$\text{s.t.} \sum_{\omega\in\Omega:\ t\in\omega, v\in B} \lambda_{\omega,v} = 1, \quad t \in T, \quad (\pi_t), \tag{4.9}$$

$$\sum_{\omega\in\Omega} \lambda_{\omega,v} \leq 1, \quad v \in V, \quad (\rho_v), \tag{4.10}$$

$$\sum_{\omega\in\Omega, v\in V} \mathbf{1}^e_{\omega,v}, \lambda_{\omega,v} \leq R_e, \quad e \in E, \quad (\sigma_e) \tag{4.11}$$

$$\lambda_{\omega,v} \in \{0, 1\}$$

Objective (4.8) and constraints (4.9) and (4.10) are already familiar to the reader. Constraints (4.11) keep track of the resource consumption in each epoch. Here, we define $\mathbf{1}^e_{\omega,v} = 1$ if sequence ω uses the resource in epoch e when assigned to v (i.e., there exists some test t in ω such that $s^\omega_t \leq e \leq s^\omega_t + p_t$).

Similarly to the set-partitioning formulation in Chapter III, the number of sequences can be exponentially large, especially now that we have incorporated timing decisions as part of the sequence definition. In this case, it is unlikely that we can enumerate all the variables. Therefore, we rely on using delayed column generation for this model.

4.4 Delayed column generation algorithm

Following the ideas of Section 3.4, we will use delayed column generation to solve the linear relaxation of **CTSPR-MP**, and then re-impose integrality restrictions to find a feasible integer solution using only variables that were introduced in the column generation process.

As indicated in the formulation, we associate dual variables π_t, ρ_v, and σ_e with constraints (4.9), (4.10), and (4.11), respectively, in the linear relaxation of the master problem and its restrictions. Since we have modified our definition of a sequence and introduced a new set of constraints, we need to modify our pricing problem accordingly. One way of doing that is to formulate the pricing problem as a one vehicle version of **CTSPR-MILP** where time is discretized into epochs. However, this time-indexed formulation is difficult to solve even for a single vehicle, becoming prohibitively computationally expensive since it needs to be solved repeatedly.

Therefore, we propose two alternative approaches to solve the pricing problem: (1) using constraint programming, and (2) using a sequence-then-time strategy.

4.4.1 Constraint programming formulation of the pricing problem

In Constraint Programming (CP), each variable can be assigned values from specified range, or domain, which usually contains a finite number of options, and relationships between decision variables are stated in the form of logical constraints. The main goal of a CP problem is usually to find a solution that satisfies all constraints. CP solvers use a domain refinery method to reduce the domain of each decision variable as computation progresses. If the domain for a decision variable only contains a single value, then we can conclude that it has to be the final value of that variable; if the domain is empty, then we can conclude that the problem does not have a feasible solution; if there are multiple value candidates in the domain and the solver cannot decide which one is the true value, it will create multiple branches of the original problem, with the decision variable taking on each of the possible values in separate branches. A CP solver will perform this branching whenever it encounters a pending decision regarding the value of a variable. For some of the branches it creates, where decision variables are fixed, constraints might imply conflicts, in which case the solver needs to trace back to the branching point where this assignment was made and fathom this infeasible branch. This procedure is called back-propagation.

Here we will discuss using CP to solve the pricing problem in the delayed column generation algorithm for solving the linear relaxation of **CTSPR-MP**. Suppose π, ρ, and σ are the optimal values of dual variables associated with the constraints of the linear restricted master problem **CTSPR-LRMP** obtained by restricting the variables to a subset of all timed sequences. There are three types of decisions we need to make in the pricing problem: (1) which vehicle should be used; (2) what is the composition of the tests in the sequence; (3) when does each of these tests start. Again, we can decompose the pricing problem by vehicle (or vehicle group) as we did in Section 3.4.1. Once the vehicle (group) selection is fixed, each subproblem that corresponds to individual vehicle (groups) is independent of the others.

We propose a slot-based constraint programming model to select tests to form a sequence and to time them appropriately to minimize the reduced cost. Let K be the maximum number of tests can be performed on a single vehicle (this number can be derived based on rehit rules and timing considerations of the vehicle program). Then, there are K slots on the selected vehicle where we can assign tests. Notice that we do not have to use up all K slots. Hence, we introduce a dummy test t_0 which stands for "nothing" or "empty" if assigned to a slot — this way, each sequence will consist of exactly K tests. After deciding the assignments of tests to the K slots, the next step is to determine the start times of tests in each slots, to complete a timed sequence. The decision process is shown in Figure 4.3.

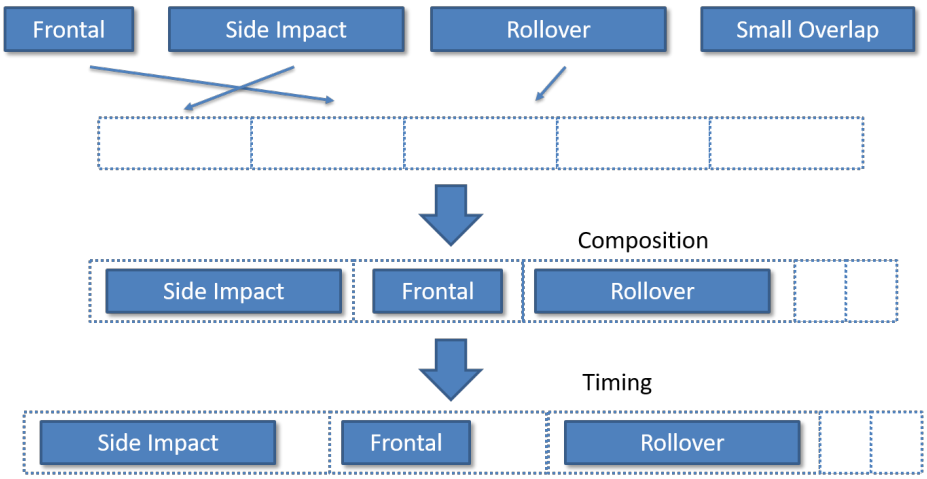


Figure 4.3: A slot based pricing model

Let $T_g \in T \cup \{t_0\},\ g = 1, \ldots, K$, be decision variables representing the choice of a test assigned to slot g in the sequence, and let integer variables $S_g \in E,\ g = 1, \ldots, K$ denote the start times of the test in each slot. We have the following formulation of

the pricing problem for vehicle v:

CSTPR-PP-CP

$$\text{minimize } c_v + \sum_{g=1}^{K} o_{T_g,S_g} - \sum_{g=1}^{K} \pi_{T_g} - \rho_v - \sum_{e\in E} \sigma_e \sum_{g} h^e_{T_g,S_g}, \tag{4.12}$$

$$\text{s.t. } \sum_{g=1}^{K} (T_g == t_i) \leq 1,\ t_i \in T, \tag{4.13}$$

$$\min\{q_v, r_{T_g}\} \leq S_g, \quad g = 1, \ldots, K, \tag{4.14}$$

$$S_{g+1} \geq S_g + p_{T_g}, \quad g = 1, \ldots, K-1, \tag{4.15}$$

$$(T_g == t_i) \rightarrow (T_{g'} != t_j),$$
$$g = 1, \ldots, K,\ g' = g+1,\ \ldots, K, t_i, t_j \in T : a_{ij} = 0 \tag{4.16}$$

$$(T_g == t_0) \rightarrow (T_{g'} == t_0), \quad g = 1, \ldots, K,\ g' = g+1, \ldots, K. \tag{4.17}$$

Objective function (4.12) is the reduced cost associated with assigning sequence constructed in this model to vehicle v, where $o_{T_g,S_g} = \max\{S_{T_g} + p_{T_g} - d_{T_g}, 0\}$, i.e., it is the tardiness of the test in slot g, and $h^e_{T_g,S_g} = 1$ if $S_{T_g} \leq e \leq S_{T_g} + p_{T_g}$, which tracks the penalty associated with using the resource in epoch e. Notice that o's and h's can be computed beforehand for every combination of values of T_g, S_g and T_g, S_g, e. (4.13) guarantee that each test is included in the sequence at most once. (4.14) ensure that tests start after their release times and the vehicle available time. (4.15) specify the precedence relations between two consecutive slots, where the test in slot g needs to be completed before the test at slot $g+1$ can start. (4.16) enforce the sequencing restriction if t_j cannot follow t_i, i.e., $a_{ij} = 0$ in the binary matrix $\mathbf{A}$. (4.17) indicate that once we decide to assign the dummy test t_0 to some slot, it means we are "closing" the sequence and assigning the dummy test to all the following slots as well. This helps to reduce the symmetry of the sequence composition that is caused by varying the position of the dummy test t_0 in the sequence.

Notice that there is a clear hierarchy in the decision process. If we decide the composition and order of the sequence, i.e., the values of T_g's, the timing decisions S_g are relatively easy to figure out, as we will see in the formulation (4.19)-(4.23). Therefore, we can enforce the above hierarchical search strategy during the solution process. Experiments have shown it is a useful strategy that can significantly improve computational times.

In the implementation, it is also useful to convert **CTSPR-PP-CP** into a feasibility problem by transforming the objective (4.19) into a constraint

$$c_v + \sum_{g=1}^{K} o_{T_g,S_g} - \sum_{g=1}^{K} \pi_{T_g} - \rho_v - \sum_{e \in E} \sigma_e \sum_{g} h^e_{T_g,S_g} \leq -\epsilon \tag{4.18}$$

for a small $\epsilon > 0$. The reason is that, unlike in the branch-and-bound algorithm for integer programming, the CP solution process does not provide lower bounds on the objective function values of various branches.

4.4.2 A sequencing then timing strategy to solve the pricing problem

As we described in Section 4.4.1, there is a clear hierarchical decision pattern to solving the pricing problem via constraint programming, since choosing the start times of tests in the sequence is relatively easy once the composition and ordering of the sequence has been determined. Therefore, when designing an algorithm for solving the pricing problem, we can take advantage of this hierarchy among the variables by decomposing the search process into stages: a sequence composition stage followed by a timing stage. Computational experiments have shown that this strategy can speed up the solution process.

However, one drawback of this strategy is that every time we solve the pricing problem, we need to reinitialize the search algorithm as if we have no prior information about the problem. Notice that only a small portion of the problem changes at each

iteration: only the coefficients in (4.18) change due to the changes in the values of the dual variables, while the decision region remains invariant across iterations. However, even after solving the pricing problem many times, the we do not retain information about what the feasible region for a valid column looks like. Therefore, instead of relying on constraint programming solvers to help us detect a negative reduced cost column, we discuss explicitly separating the test selection and sequencing decisions from the timing decisions, and develop algorithms that can take advantage of this structure.

4.4.2.1 Optimal timing of a given sequence of tests

In this subsection we will provide a pseudo-polynomial algorithm that uses dynamic programming to determine the start times of individual tests and the vehicle to assign this sequence of tests to so that the reduced cost is minimized, for a sequence consisting of a given ordered subset of tests.

Recall that the goal of the pricing problem is to find a sequence and a vehicle that minimizes the reduced cost

$$\sum_{v \in V} c_v u_v + \sum_{t \in T} \sum_{e \in E} o_t^e s_t^e - \sum_{t \in T} \pi_t x_t - \sum_{v \in V} \rho_v u_v - \sum_{t \in T} \sum_{e \in E} \sum_{j=e}^{e+p_t} \sigma_j s_t^j,$$

where $u_v,\ v \in V$, are binary decision variables representing which vehicle is chosen for the sequence; $x_t,\ t \in T$, are binary variables representing which tests to include in the sequence; and $s_t^e,\ t \in T,\ e \in E$ are binary variables which equal to 1 if test t starts processing in epoch e. There are 3 main consideration that are related to the timing decisions for individual tests: (1) the total tardiness penalty $\sum_{t \in T} \sum_{e \in E} o_t^e s_t^e$, which depends on the start times of individual tests; (2) the term related to resource usage $\sum_{t \in T} \sum_{e \in E} \sum_{j=e}^{e+p_t} \sigma_j s_t^j$, which also depends on start times; and (3) the selection of a vehicle for the sequence, which implicitly impacts the timing decisions by connecting

the earliest epoch a test can start to the release time of this vehicle. Meanwhile, the term $\sum_{t\in T}\pi_t x_t$ only depends on the composition of the sequence, and can be treated as a constant if the sequence is already fixed and given.

Based on the above observation, we can design a dynamic programming (DP) algorithm to determine the best start times for individual tests in a sequence of given composition.

Suppose the sequence contains K tests $t_1, t_2, \ldots, t_K$ (in this order). Define the state of the DP as (i, h), where $1 \le i \le K$ is the slot in the sequence for which we are currently deciding the start time and h is the earliest time the test at slot i can start processing. Define the value function $f(i, h)$ as the minimum cost of processing all tests in and after the ith slot, given the earliest starting time of test in the ith slot is h. This cost includes the tardiness of the tests and the penalty for using resources, as discussed in the above paragraph. The boundary conditions are

$$f(K, h) = \min_{s\ge h} (s + p_{t_K} - d_{t_K})^+ - \sum_{e=s}^{s+p_{t_K}} \sigma_e \text{ for any } h.$$

For the last test t_K in the sequence, if we start in epoch s, then the tardiness of this test is $(s + p_{t_K} - d_{t_K})^+$ and the penalty for using the resource for this test is the sum of dual variables σ from epoch s to $s + p_{t_k}$. The DP recursion is

$$f(i, h) = \min_{s\ge \max\{h, r_{t_i}\}} \left\{ (s + p_{t_i} - d_{t_i})^+ - \sum_{e=s}^{s+p_{t_k}} \sigma_e + f(i+1, s + p_{t_i}) \right\}.$$

For a test t_i in slot i, if we start the processing at time s, then the immediate cost (tardiness penalty and the cost of using the resource) can be calculated as $(s + p_{t_k} - d_{t_k})^+ - \sum_{e=s}^{s+p_{t_k}} \sigma_e$. Meanwhile, the earliest starting time for the next test in the sequence is $s + p_{t_i}$. By applying the recursive solution approach, we can compute values of $f(1, h),\ h \in E$. The complexity of the DP is related to the timing horizon

length $|E|$ and sequence length k, and is $O(k|E|)$.

We then use the values of $f(1,h),\ h \in E$, to help in deciding which vehicle should be assigned this sequence. In particular, for a vehicle v, the reduced cost of the column would be

$$c_v + f(1, q_v) - \sum_{i=1}^{K} \pi_{t_i} - \rho_v,$$

where $\sum_{i=1}^{K} \pi_{t_i}$ only depends on sequence composition, and q_v is the release day of vehicle v, as well as the earliest starting time for the first test in the sequence. Therefore, to minimize the reduced cost, we can pick vehicle $v^\star$ such that

$$v^\star = \arg\min_v \left\{-\rho_v + f(1, q_v) + c_v\right\}.$$

By applying the above procedures, given an ordered subset of tests, we can determine the best vehicle and timing choices for the tests in pseudo-polynomial time. Thus, to solve the pricing problem, we could exhaustively enumerate all ordered subsets of tests that are valid with respect to compatibility requirements and rehit rules, and then apply the timing and vehicle selection procedure to find the corresponding timed sequence with the smallest reduced cost.

As we discussed in Section 3.4.2, due to the destructive nature of the crash tests, the maximum number of rehits is fairly limited. Therefore, the enumeration of all possible valid ordered subsets of tests, can be computationally inexpensive, especially for instances with sparse matrices $\mathbf{A}$.

Following the ideas of Section 3.4.2, we can parallelize the search for the column with the most negative reduced cost. In particular, we can decompose the search for the best vehicle and timing choice by considering each ordered subset of tests separately. The illustration of this approach is shown in Figure 4.4.

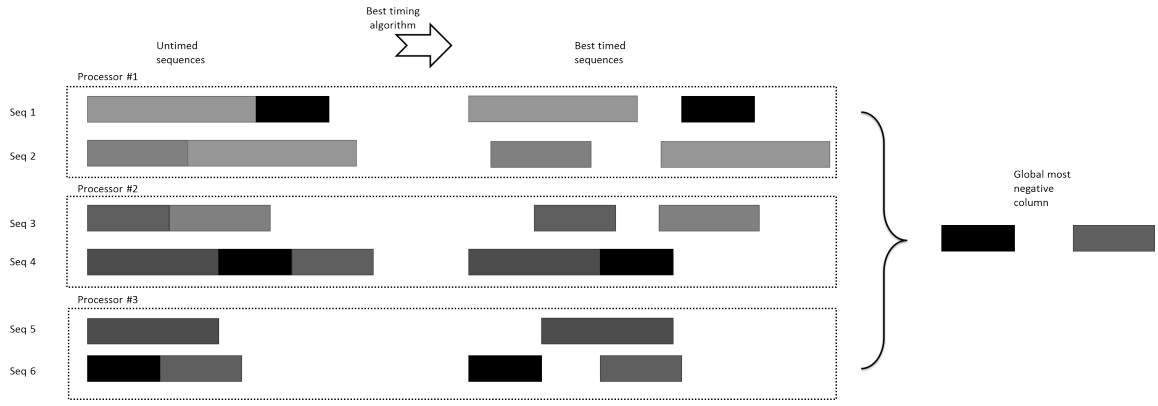


Figure 4.4: Offline pricing of all sequence compositions

4.4.3 Finding the initial set of variables

One key question in the delayed column generation algorithm is how to find the initial set of sequences to include in the restricted master problem, since the linear relaxation of the initial restricted master problem needs to be feasible for the algorithm to proceed.

In order obtain the initial set of sequences, we use a two-stage approach. First, we solve a resource-unconstrained version of the scheduling problem, i.e., **CTSP-MP**, using the approach described in Chapter III, which gives us an initial set of sequences where tests are performed back-to-back. Then we re-time the tests in these sequences to satisfy the resource constraints. Here, we use a constraint programming approach to re-time the tests.

Assume that after solving the resource-unconstrained version of the problem, we obtain a set of untimed sequences and vehicle assignments. Without of loss of generality, we assume the assignments to be $\omega'_1 \rightarrow v_1,\ \omega'_2 \rightarrow v_2,\ \ldots, \omega'_k \rightarrow v_k$, where $\omega'_i \rightarrow v_i$ indicates assigning an untimed sequence ω'_i to vehicle v_i. Then the next step is to time the tests subject to the resource constraints across multiple vehicles. Let the integer decision variables $s_t \in E,\ t \in T$ be the start times of tests, where E is the set of epochs (planning horizon). We can formulate and solve the problem using

ILOG CPLEX CP Optimizer (CPO) as follows:

$$\text{minimize} \quad \sum_{t \in T} o_t, \tag{4.19}$$

$$\text{s.t.} \quad r_t \leq s_t \leq d_t + o_t, \quad t \in T, \tag{4.20}$$

$$q_{v_i} \leq s_t \quad t \in \omega_i', \ i = 1, \ldots, k, \tag{4.21}$$

$$s_{t_l} + p_{t_l} \leq s_{t_m}, \quad t_l, t_m \in \omega_i' : t_l \to t_m, i = 1, \ldots, k \tag{4.22}$$

$$\sum_{t \in T : s_t \leq e \leq s_t + p_t} 1 \leq R_e, \quad e \in E, \tag{4.23}$$

$$s_t \in E.$$

Since vehicle usage has already been determined, the objective function (4.19) only includes the penalty associated with our timing decisions. (4.20) and (4.21) are temporal constraints for the earliest start time of each test. (4.22) describe the relationship between timing decisions for tests t_l and t_m implied by their order in the sequence. Finally, (4.23) are the resource constraints. Notice that constraints (4.23) are not linear. CPO provides for a special form of constraints, called global constraints, to enforce such relations across multiple variables. We can replace (4.23) with a global constraint called *cumulative* constraint, $\textit{cumulative}(s_1, s_2, \ldots, s_{|T|}, p_1, p_2, \ldots, p_{|T|}, e) \leq R_e, \ \forall e \in E$.

Since the goal of solving this constraint programming formulation is to obtain an initial set of timed sequences that guarantees feasibility of the linear restricted master problem, we can ignore the objective (4.19) and treat this problem as a pure feasibility problem.

4.4.4 A primal heuristic based on dual price of resources

After solving the linear relaxation of the master problem to optimality, we need to re-introduce integrality constraints $\lambda_{\omega,i} \in \{0, 1\}$ and solve the problem as a pure

binary optimization problem, as we did in Chapter III. This optimization problem is always feasible after we have introduced the initial set of columns that form a feasible schedule subject to the resource capacity constraints (4.11) as described in Section 4.4.3. However, this feasible solution is usually not particularly good since the sequences were not initially formed with an eye towards the scarce resource.

However, using a good initial feasible solution as a warm start helps the optimization solver to better fathom branches that lead to bad solutions in terms of objective function value in the branch-and-bound algorithm. It also helps to estimate the optimality gap at the early stage, which gives a straightforward intuitive metric that can be used to understand the progress of the optimizer. Therefore, in this section, we discuss how to use the dual variables π_t, ρ_v, σ_e to produce a feasible solution using heuristics.

When solving the linear relaxation of **CTSPR-MP**, we obtain dual information for constraints (4.9), (4.10) and (4.11) at every iteration. We use the dual information to guide the pricing problem to generate new columns to the RMP. The main motivation behind the process is that the dual variables π_t, ρ_v, σ_e capture so called "shadow prices" of each resource that is capacitated, if we view tests and vehicles as resources as well. For example, if in epoch e the number of tests in progress is lower than the capacity of the supporting resource, R_e, then by complementary slackness we know that the corresponding dual variable $\sigma_e = 0$, which encourages the pricing problem to generate new columns that use the resource in epoch e by charging a lower price (of 0) for this epoch. Similarly, π_t and ρ_v provide guidance for which tests to include and which vehicle to assign the sequence to, by setting different prices and rewards for resources (tests and vehicles).

Motivated by that, at the last iteration of the delayed column generation algorithm, when the linear relaxation is solved to optimality and no more new columns are to be generated, we can record the values of shadow prices associated with tests, vehi-

cles, and resources at the last iteration. These values indirectly capture the utilization of resources in a schedule. For example, an epoch when the supporting resource is not used to its full capacity can be viewed as free.

Then, for every possible untimed sequences, we apply the optimal timing algorithm as described in Section 4.4.2.1, using the dual variable information of π_t, ρ_v, σ_e. After doing that, we have a collection of sequence-to-vehicle assignments that minimize the reduced cost given their test compositions. We sort the entire collection by increasing order of reduced costs and scan over it to select assignments(columns) that cover as many tests as possible.

Whenever the supporting resource is used up at a certain epoch after we select a few columns, we update its dual price to a sufficiently high value to forbid generating further columns that involve using the resource at that epoch. Similarly, whenever the vehicles in a vehicle group are all used, we also update the corresponding dual price for the vehicle group to be large. Under those circumstances, the optimal timing algorithm will be re-triggered, and a new set of different columns where tests may be timed differently due the updates in the dual prices. Then we repeat the above process to add new columns, and update dual prices whenever needed, until all tests have been included at least once. The resulting set of columns compose a valid full schedule where constraints (4.11) and (4.10) are respected.

4.5 Scheduling multiple vehicle programs

Our numerical experiments in this chapter include extremely large instances that contain more than 100 individual tests (going up to several hundred). However, in reality, it is rare to have more than 100 crash tests performed for one individual vehicle development program; it is more typical to have around 50 tests, since most programs represent evolution of existing models that do not need to be subjected to a full battery of tests.

Besides demonstrating the scalability of our algorithm, solving extremely large instances has another important practical purpose. Although a single vehicle development program will not contain this many tests, it is common to have multiple vehicle models (for different product lines) developed simultaneously within the company. If testing phases of two vehicle programs do not overlap in time, the scheduling of those two programs can be performed independently. However, it is fairly common to have testing phases of two or more programs overlap with each other. Since these programs will have to share some resources (including instruments, dummies, time at the crash lab, etc., but typically excluding prototype vehicles), it is essential that test planners taking this into consideration when planning schedules for those overlapping vehicle programs.

One can always argue that scheduling for multiple vehicle programs does not fundamentally differ from solving a very large instance for a single program. However, we can come up with solution approaches that can explicitly take advantage of the partial independence of different vehicle programs, namely, decomposing instances corresponding to multiple program by program and by planning horizon.

4.5.1 Decomposition by the vehicle program

The first natural decomposition strategy we can think of is decomposition by vehicle program. Aside from the supporting resource constraints (4.11), scheduling tests for different programs can be done independently.

Suppose we are solving a multi-program instance of **CTSPR** using delayed column generation. One place where decomposition by vehicle program can be easily applied is in solving the pricing problem. In the pricing problem, the supporting resource constraints in the RMP (4.11) are dualized and represented in the definition of the reduced cost of a column. Therefore, there is no coupling between different vehicle programs in the pricing problem. We can set up an independent pricing problem for

each vehicle program, where the vehicle-program-specific pricing problem will only consider tests and prototype vehicles specific to this program.

We can solve each individual vehicle-program-specific pricing problem during the delayed column generation procedure either in parallel or sequentially. In the sequential setting, the time invested in solving all pricing problem scales linearly with the number of sub-vehicle-programs that are contained in the "combined" instance, assuming each individual program requires almost same amount of solution time and differences in terms of size and complexity are insignificant.

Our forthcoming computational experiments demonstrate that instances of similar sizes but corresponding to multiple programs require significantly less computation than single-program instance, especially in solving pricing problems.

4.6 Numerical results

4.6.1 Single program

We use a testing environment similar to the one described in Section 3.6.1. For simplicity, we only consider one supporting resource during the scheduling and we assume the availability of the resource is constant over time. We set the time limit to solve the integer version of **CTSPR-LRMP** to 900 seconds. We only tested the algorithm on *small, typical, moderate,* and *large* groups. The program encountered memory limit issues when trying to solve the *extreme* group because of the complexity of resources constraints. Table 4.1 summarizes the performance statistics of the algorithm on solving instances from various size groups. From the table, we can see that the algorithm performs well on all groups except *large*, where instances contain more than 100 tests. Figure 4.5 shows the percentage of solved and unsolved instances in each group, where we consider an instance to be solved if we can obtain a solution with optimality gap of 5% or less. We can solve almost all instances in groups other

Table 4.1: Average performance of delayed column generation for single-program instances of **CTSPR**

Size group	Sol. time (sec)	Avg. RMP Opt. Gap	Avg. Opt Gap
Small	5.10E+01	0.00%	0.61%
Typical	1.77E+02	0.00%	1.79%
Moderate	7.96E+02	1.29%	3.37%
Large	1.04E+03	6.17%	7.22%

than *large*, and half of the instances in *large*. Since the average optimality gap reported for this group in Table 4.1 is 7.22%, even instances that we consider unsolved are frequently quite close to the 5% threshold. The relatively poor performance on instances in *large* group might be due to the fact that we are using the same value of R_e (resource capacity) in all instances. As the number of tests increases while resource availability stays the same, it becomes increasingly more difficult to identify feasible schedules, and large tardiness penalties become unavoidable since tests have to be postponed due to lack of resources.

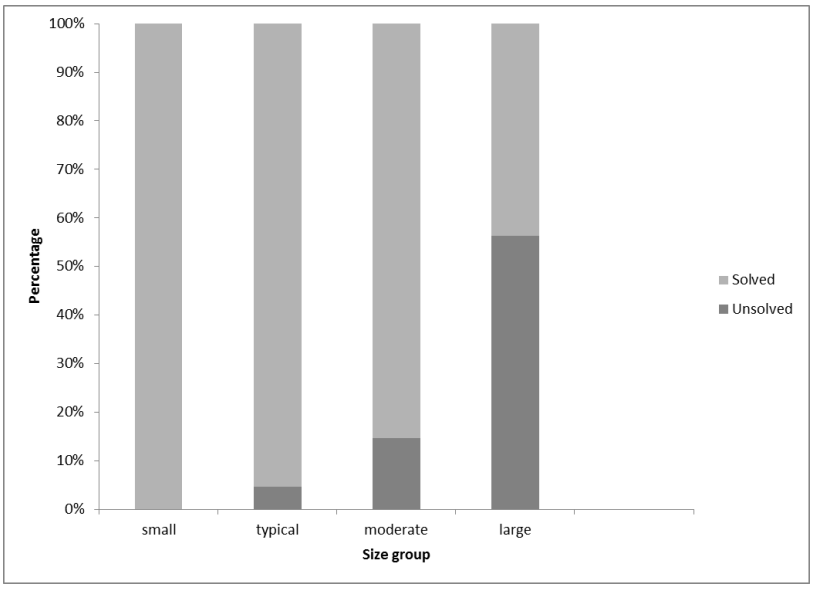


Figure 4.5: Percentage of solved and unsolved instances

We also plot the trend in average optimality gap as the size of the instances grows in Figure 4.6. For **CTSPR**, we achieve slightly worse scalability compared to its

resource-unconstrained counterpart in the previous chapter. Here, the turning point is around 120 tests.

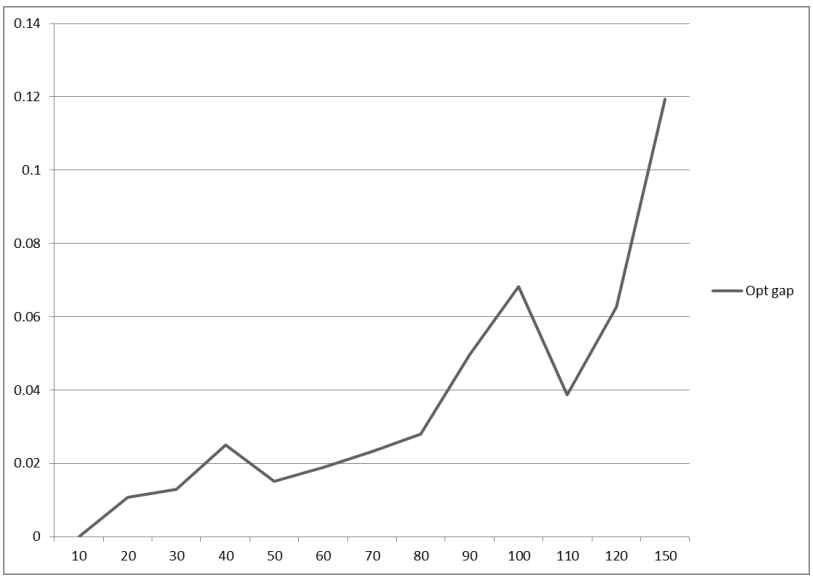


Figure 4.6: Optimality gap vs. number of tests

For a complete table of results of solving **CTSPR** using delayed column generation algorithm, we ask readers to refer to Table A.3 in the Appendix.

4.6.2 Multiple programs

We also tested our algorithm on instances formed by combining pairs of different instances to create larger instances representing two programs.

We considered scenarios when two vehicle programs completely overlap with each other, i.e., the start times of the programs are the same, and scenarios when they overlap only partially. To control the extent of the overlap, we introduced a parameter L which equals 1 if two vehicle programs start at the same time and therefore overlap with each other entirely; $L = 0$ if the execution horizons of two programs are independent; and $0 < L < 1$, which means that the second program starts $L \times |E^1|$ after the first program starts execution, if we assume the planning horizon of the first program is E^1.

The performance statistics are summarized in Table 4.2. The algorithm generally performs well when solving two-program instances, except when each program's instance comes from *large* group. However, this type of instances are rarely seen in reality since each large instance contains 100—150 tests; they were truly designed to test the computational limits of our algorithm rather than to be particularly realistic.

Table 4.2: Average performance of delayed column generation for two-program instances of **CTSPR**

Combination	Sol. Time	Avg. RMP Opt Gap	Avg. Opt Gap
Small + Small	1.78E-01	0.00%	0.00%
Small + Typical	2.74E+02	0.59%	0.66%
Small + Moderate	5.16E-01	0.00%	0.15%
Small + Large	4.72E+02	0.83%	0.89%
Typical + Typical	1.77E+01	0.00%	0.00%
Typical + Large	5.29E+02	3.68%	5.01%
Large + Large	6.52E+02	21.16%	34.40%

Quite intuitively, we also find that the algorithm performs better on instances where two programs are relatively independent, i.e., overlap only over a short time, as shown in Figure 5.1.

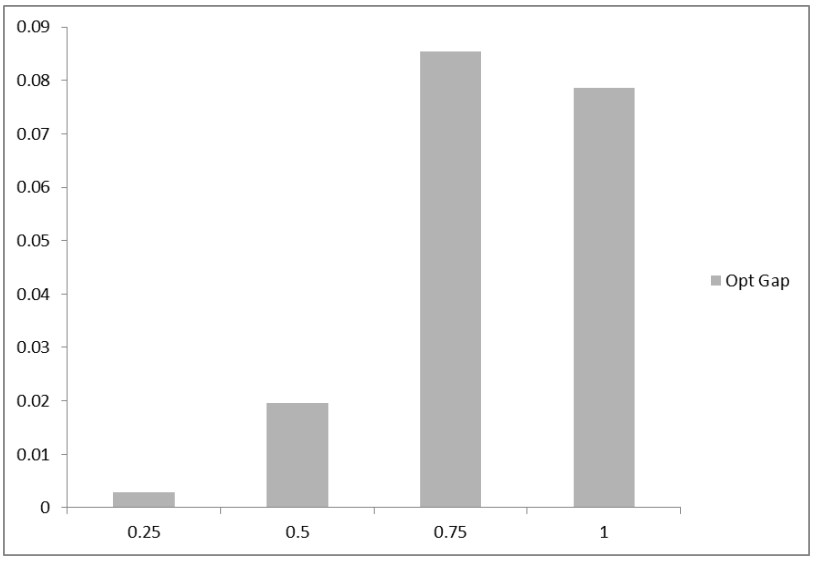


Figure 4.7: Average optimality gap for different values of overlap level L

For a complete table of results of solving multiple vehicle program scheduling, we

ask reader to refer to Table A.4 in the Appendix.

CHAPTER V

Conclusion

5.1 Conclusion

In this thesis, we discussed three types of scheduling problems that arise during the new vehicle model development stage at Ford.

In the first version, we consider schedules for all the tests from a wide variety of departments on a macro level. Due to the size of the problem instances, we mainly depend on using heuristics to attain high quality solutions.

In the second version, we develop schedules specifically for crash tests. The complex time window constraints and sequencing restrictions among those tests make this variation difficult to be solved just by heuristics. Additionally, due to the destructive nature of crash tests, a compact schedule for executing them is critical to the overall utilization of prototype vehicles for the full schedule. Therefore, more accurate models and solution approaches are necessary for solving this variation.

In the last variation of the problem, we introduce a new set of constraints, supporting resources constraints, into the model. The reason is that supporting resources can be the bottleneck for a schedule due to their limited supplies. Therefore, it is extremely important to consider resources other than just the prototype vehicles when scheduling crash tests. Based on the set-partitioning formulation we proposed in Chapter III, we propose an extension of the model where the timing decisions for in-

dividual tests are introduced explicitly into the definition of a sequence of tests. We also solve the scheduling problem for multiple vehicle programs, where the supporting resources constraints play a more vital role compared with the one for a single vehicle program.

In the numerical results sections of this thesis, we have shown the effectiveness and scalability of our algorithms. For all the problem instances that we obtained from Ford so far, we can always solve them to near-optimality with a small gap ($< 1\%$). Testing the algorithms on synthesized data instances, we can solve them up to 2 times the typical size of real instances within an optimality gap less than 5%. For those more extreme cases we generated to test our performance boundary, most of the time we were still able to attain feasible solutions within 10% of optimality, which shows that the algorithm is fairly future-proof in anticipation of solving more challenging scheduling problems that Ford might face in the future as a stricter compliance on occupant safety is adopted and more crash tests need to be performed in order to satisfy the stricter standards.

However, there is still significant space for improvements and topics that are interesting to explore under our current algorithm framework.

5.2 Future research directions

In this section, we discuss the areas where we can further improve our algorithms to achieve better performance and scalability. Also, we cover potential new variations of the scheduling problem which can be accommodated by extensions to our current algorithm framework.

5.2.1 Branch-and-price algorithm for scheduling under supporting resource constraints

In Section 3.5, we proposed a branch-and-price algorithm to solve **CTSP** exactly. From the numerical results, we do see that doing further branching on fractional variables helps to either improve the quality of the solution found by just solving the root node linear relaxation, or provide better estimates of the optimality gap of the solution (occasionally even proving its optimality) by improving the lower bound. In order to maintain the invariant structure of the pricing problems and produce more balanced trees, instead of directly branching on the composite variable $\lambda_{\omega,v}$, we build the relation between λ variables and the original variables $x_{t,v}$ and y_{t_i,t_j} in the **CTSP-MILP** formulation, and branch on the latter instead.

We can apply similar ideas to develop the branch-and-price algorithm to solve CTSPR exactly. Aside from the integrality check we do in Section 3.5, we would also check if

$$z_t^e = \sum_{\omega: t \in \omega, s_t^\omega = e, v} \lambda_{\omega,v} \tag{5.1}$$

is fractional or not by summing up all variables where the corresponding column contains t in the sequence and starts e at time epoch t.

By doing the integrality checks in Section 3.5, we can guarantee that there is no fractional assignments for a single test to different vehicles and consistent precedence orders between any two tests. By checking this additional rule (5.1), we can also guarantee that a single test is non-preemptive and has a fixed starting time on a vehicle.

However, fixing $z_t^e = 1$ has a significant impact on the decision region, while fixing $z_t^e = 0$ has relatively little impact. To better balance the tree, we can instead check integrality on an aggregate level, $g_t^e = \sum_{j \le e} z_t$, i.e, if test t has begun its execution before time epoch e. Thereby, fixing g_t^e to 1 or 0 requires the test to begin either

before or after t, and thus having approximately the same impact on the sub-decision region.

5.2.2 Decomposition by the planning horizon in solving multiple vehicle program scheduling

We discussed one decomposition strategy when solving instances involving multiple vehicle programs — decomposing the problem by the vehicle program when solving the linear relaxation of the set-partitioning formulation in Section 4.5. In addition, a further decomposition strategy is to decompose by execution horizon, namely to separate the execution horizon into different stages and solve a sub-scheduling problem for each stage. There are cases where two vehicle programs are happening concurrently, meaning they start almost at the same time and need to share the supporting resources throughout their execution periods. However, it is more common to have different vehicle programs that only partially overlap. There is usually some chronological order associated with each program, and although they overlap, a second vehicle program might not enter testing until the first program has been undergoing testing for a while.

To use an illustrative example, suppose we have two vehicle programs, 1 and 2, and program 2 begins testing 50 days after program 1 has started its testing. In this case, vehicle program 1 is utilizing the supporting resources by itself for the first 50 days in the planning horizon. After the first 50 day, the two programs need to work in a coordinated fashion once program 2 has also begun testing. It is possible that after, say, another 50 days program 1 will complete all of its tests, thus leaving the supporting resource solely for the use of program 2.

In the previous example, we can see that the planning horizon can be separated roughly into 3 stages: (1) the stage when program 1 is using all of the supporting resources; (2) the stage when programs 1 and 2 need to share resources; (3) the stage

when program 2 uses all of the resources. The most important of the three stages is stage (2), since it requires coordination between programs. If the detailed schedule of stage (2) is formed first, stages (1) and (3) are relatively independent of each other since one vehicle program is being considered during each stage. An illustration is provided in Figure 5.1.

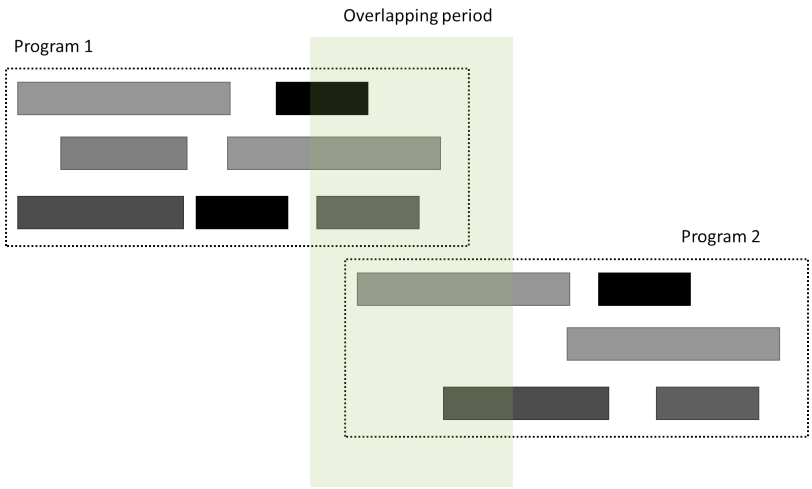


Figure 5.1: Two vehicle programs that partially overlap

One possible way to take advantage of such partial decomposition by planning horizon is to use a solution approach that combines a Lagrangian relaxation method and delayed column generation method.

Formally, let us assume that we have two vehicle programs, program 1 and 2, that do not start at the same time. Without loss of generality, we assume program 1 starts at time epoch 0, and 2 starts at epoch $e' > 0$. Suppose we have an estimate of when program 1 will wrap up (if time window constraints are hard ones, then it's the latest deadline for all tests in the vehicle program), and denote it by e''. If $e'' \leq e'$, then the scheduling for those two programs can be entirely decomposed into two independent scheduling problems. If not, we can decompose the planning horizon according to the stages we described above: stage $E^1 = \{e|\ 0 \leq e < e'\}$ where only tests from program 1 are being executed; stage $E^{\text{sharing}} = \{e|\ e' \leq e < e''\}$ where two programs

are sharing resources; and $E^2 = \{e|\ e \geq e''\}$ where only tests from program 2 are being executed.

For stages E^1 and E^2, we do not need to consider the resource sharing issue. But for E^{sharing}, two programs need to work in a coordinated way. Assume the set of tests, set of vehicles, and set of valid sequences for program i are T^i, V^i, and Ω^i, respectively. The full set-partitioning formulation is

$$\text{minimize} \quad \sum_{\omega\in\Omega^1, v\in V^1} c_{\omega,v}\lambda_{\omega,v} + \sum_{\omega\in\Omega^2, v\in V^2} c_{\omega,v}\lambda_{\omega,v} \tag{5.2}$$

$$\text{s.t.} \quad \sum_{\omega\in\Omega^i:\ t\in\omega,v} \lambda_{\omega,v} = 1, \quad t \in T^i,\ i = 1,2 \tag{5.3}$$

$$\sum_{\omega\in\Omega^i} \lambda_{\omega,v} \leq 1, \quad v \in V^i, i = 1,2 \tag{5.4}$$

$$\sum_{\omega\in\Omega^i, v} \mathbf{1}^e_{\omega,v}\lambda_{\omega,v} \leq R_e \quad e \in E^1 \cup E^{\text{sharing}} \cup E^2 \tag{5.5}$$

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

The only coupling relations between programs 1 and 2 is (5.5) when $e \in E^{\text{sharing}}$. Therefore, we introduce Lagrange multipliers $\sigma_e,\ e \in E^{\text{sharing}}$ and rewrite (5.2) as

$$L(\sigma) = \sum_{i=1,2} \left\{ \sum_{\omega\in\Omega^i, v\in V^i} c_{\omega,v}\lambda_{\omega,v} + \sum_{e\in E^{\text{sharing}}} \sigma_e(R_e - \sum_{\omega\in\Omega^i, v\in V^i} \mathbf{1}^e_{\omega,v}\lambda_{\omega,v}) \right\}.$$

For any given σ, the problem can be decomposed entirely into two independent pieces, where the program 1's scheduling happens during epochs in $E^1 \cup E^{\text{sharing}}$ and program 2's scheduling happens during epochs in $E^{sharing} \cup E^2$. During each individual horizon, the scheduling problem is equivalent to the one we have seen for a single vehicle program **CTSPR-MP** as described in Section 4.3, and can be solved efficiently by delayed column generation algorithm for moderate sized instances.

To obtain the best bound provided by $L(\sigma)$, we can implement a sub-gradient

method in order to search for the best value of $\sigma_e,\ e \in E^{\text{sharing}}$. At each iteration, we solve the individual scheduling problem for each program independently and combine two partial solutions to form a full solution for two programs. Then, we use the solution to obtain the sub-gradient and update σ accordingly. The process repeats until the σ value converges.

5.2.3 Scheduling with expediting resources

In some situations, resources such as labor can be used to expedite the execution of some tests. When preparing and instrumenting a vehicle, if we assign more engineers to work on the task, we may be able to reduce the time it takes to finish it, thus reducing the execution time. However, the improvements of the working efficiency are not always proportional to the amount of resources invested. If we assign 100 engineers to a task, it is unlikely we can finish the task in $\frac{1}{100}$th of its original length.

In reality, engineers at Ford usually work around a one-shift (day shift) schedule at the testing facilities. In some rare cases, if some tests turn out to be overwhelmingly critical and time-sensitive, there can be two groups of engineers who work around the clock (day and night shifts) assigned to those tests, in order to speed-up the execution process.

Here, in order to simplify the problem, we assume that at a given time epoch, we can have the option to double the execution speed of a task by assigning 2 shifts of labor to it. Notice in this case, the duration of a test is not a given time span (in epochs) anymore, but rather a value related to the above decision. Therefore, the complexity of the test (which we originally termed duration) is usually quantified as the amount of working hours required to finish the task (in shifts). For example, instead saying that a test is going to take 10 days to complete, we say that it requires 10 shifts of work to complete. If it is performed in the normal mode (1 shift a day) constantly, then it will be completed in 10 days, but if we assign 2 shifts of labor to

it every day, then its length is 5 days instead.

Because work hours are a limited resource, we cannot expedite every tasks we have. Instead, we can only concentrate resources on critical tests first, while the other ones available at the testing facility at the same time might need to wait their turns due to the scarcity. We also assume that once a test starts its execution, there has to be at least 1 shift of labor assigned to it at every epoch before it is completed, i.e., no preemptive execution.

Following the idea in Chapter IV, where we introduce timing decisions into the definition of a sequence, we can further expand the concept of a sequence by also introducing the working modes at each epoch for a test. For a test $t \in \omega$, aside from specifying its relative start time Δ_t, we also include an array of indicators to stand for the working modes at each time epoch, namely $m_{t,e} \in \{0, 1, 2\},\ e \in E.\ m_{t,e} = 0$ means no shifts are assigned to t at a relative (to the vehicle release day) epoch e; $m_{t,e} = 1$ or $m_{t,e} = 2$ means that 1 or 2 shift(s) are assigned to it at relative epoch e. Lastly, we have $\sum_{e \in E} m_{t,e} = p_t$, where p_t is the amount of work (in shifts) required to finish test t. When the vehicle to which the sequence is assigned to is determined, we can compute the absolute start times and work modes for each test in the sequence as $s_t = q_v + \Delta s_t$ and $m_{t,e} = m_{t,e-q_v}$. Then the set-partitioning formulation writes as

$$\text{minimize} \sum_{\omega,v} c_{\omega,v}\lambda_{\omega,v} \tag{5.6}$$

$$\text{s.t.} \sum_{\omega:\ t\in\omega,v} \lambda_{\omega,v} = 1, \tag{5.7}$$

$$\sum_{\omega} \lambda_{\omega,v} \leq 1, \tag{5.8}$$

$$\sum_{\omega,v} m^{\omega}_{t,e}\lambda_{\omega,v} \leq R_e \tag{5.9}$$

$$\lambda_{\omega,v} \in \{0, 1\}$$

where $m_{t,e}^{\omega}$ are the number of shifts (labor power) we assign to test t in the sequence ω as defined above.

To generate new variables, we also need to decide the number of shifts assigned to a test aside from its start time if we would like to include it in the sequence. We can adopt the dynamic programming formulation discussed in Section 4.4.2.1 to decide the work modes as well as starting times for tests. Define value function $g(i,h,p),\ i=1,\ldots,K,\ h\in E, p=0,\ldots,p_{max}$ similarly to Section 4.4.2.1, containing the terms in the reduced cost that depend on the timing decision. Let the state (i,h,p) be the current slot, current time, and the remaining work amount to complete the test correspondingly. The recursion is

$$g(i,h,p)=\mathbb{I}\{h>d_{t_i}\}+\min\begin{cases}-\sigma_e+g(i,h+1,p-1) & \text{assign 1 shift, if } p-1\geq 0\\ -2\sigma_e+g(i,h+1,p-2) & \text{assign 2 shifts, if } p-2\geq 0\end{cases} \tag{5.10}$$

The boundary conditions are:

$$g(i,h,p_{t_i})=\mathbb{I}\{h>d_{t_i}\}+\min\begin{cases}g(i,h+1,p_{t_i}) & \text{test is not started}\\ -\sigma_h+g(i,h+1,p_{t_i}-1) & \text{assign 1 shift}\\ -2\sigma_h+g(i,h+1,p_{t_i}-2) & \text{assign 2 shifts}\end{cases} \tag{5.11}$$

$$g(i,h,0)=g(i+1,h,p_{t_{i+1}}) \tag{5.12}$$

$$g(K,h,p)=\mathbb{I}\{h>d_{t_i}\}+\min\left\{-\sigma_h+g(K,h+1,p-1),2\sigma_h+g(K,h+1,p-2)\right\} \tag{5.13}$$

Besides the maximal number of tests that can be performed on a vehicle K, length of the planning horizon E, the algorithm also depends on the maximum shifts required for an individual test p_{max}.

5.2.4 Application to stochastic scheduling

Another practical variation of the scheduling problem is to consider uncertainty into the parameter values. For example, when deciding the portfolio of tests to conduct for a vehicle program, engineers may not have 100% confidence in how long a test will take even though the type and requirements of the test is known. Some vehicle parts required by the test may be delivered late thus affecting the progress of instrumenting the vehicle. Some mechanical work may be more complicated than expected and takes more time. A practical approach is to assume the duration of a test follows a probability distribution. In the most simplified example, we may assume the duration p_t for a test t can take values from a finite set, e.g., a triplet (p_t^1, p_t^2, p_t^3), where p_t^1 is the best-case (shortest) value, p_t^2 is the most likely value, and p_t^3 is the worst-case value, with some probabilities. We denote a realization of the test durations for all tests as $\xi \in \Xi$, where Ξ is the set for all realizations, which has cardinality $|T|^3$ for the discrete triangular case.

Suppose we want to solve **CTSP** (without resources constraints) under this stochastic test duration setting. If we follow the similar objective function definition, where we want to minimize a combination of vehicle usage and penalty function related to timing if a test is performed out of its execution window, the problem reduces to an expected value minimization problem. Because of the uncertain test durations, the time penalty of a test depends not only on its own duration realization, but also on its predecessors' duration realizations on the same vehicle. If we model the stochastic problem using formulation **CTSP-MILP**, then the decision region is stochastic and depends on the realization of the test durations ξ. Specifically, the start time decision variables $s_t,\ t \in T$, depend on ξ and are indeed $s_t(\xi)$. However, in **CTSP-MP** formulation, since the test composition of a variable $\lambda_{\omega,v},\ \omega \in \Omega, v \in V$ is determined, we can evaluate the expected time penalty for a sequence-vehicle assignment analytically and purely offline, i.e., $c_{\omega,v}(\xi)$ depends on the duration real-

ization but can be computed beforehand given the composition of ω and v. Then the stochastic counterpart can be written by just replacing the objective function by $\min \sum_{\xi \in \Xi} \mathbb{P}(\xi) \sum_{\omega \in \Omega, v \in V} c_{\omega,v}(\xi) \lambda_{\omega,v}$. This problem is no more difficult than a deterministic problem.

In another setting, instead of trying to minimize the expected time penalty, we assume a hard threshold on the maximum violation of the time window constraints for each test. However, due to the uncertainty of the test durations, we would like to satisfy this hard threshold constraint with a high probability, say 95%. Roughly speaking, among all the realization of durations, we can violate $0.05|\Xi|$ scenarios. If we view the maximum number of violated scenarios as a type of resource, the stochastic **CTSP** reduces to a deterministic **CTSPR** problem. We introduce an indicator $\mathbb{I}^{\xi}_{\omega,v},\ \omega \in \Omega, v \in V, \xi \in \Xi$ such that $\mathbb{I}^{\xi}_{\omega,v} = 1$ if the sequence-vehicle assignment violates the hard threshold constraint under duration realization ξ; and 0 otherwise. The resource capacity constraint is

$$\sum_{\xi \in \Xi} \mathbb{P}(\xi) \sum_{\omega \in \Omega, v \in V} \mathbb{I}^{\xi}_{\omega,v} \lambda_{\omega,v} \leq 0.05|\Xi|.$$

Notice that the indicators $\mathbb{I}^{\xi}_{\omega,v}$ can be evaluated offline for every sequence-vehicle combination beforehand because of the known test compositions. Therefore, solving a stochastic **CTSP** with a probabilistic constraint is equivalent to solving a **CTSPR** problem where the violated scenarios are a type of resource.

For the case of continuous distributions for durations or more general stochasticity, we can solve the optimization by Monte Carlo sampling and Sample Average Approximation (SAA) method, which transforms the continuous case into discrete realizations and similar solution strategy can be applied.

APPENDIX

APPENDIX A

Computational result tables

Table A.1: Numerical result for **CTSP** using delayed column generation

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols gen	Tardiness	Used vehicles	Obj val	time_spend(sec)	RMP gap	Opt gap
s0	10	8	0.82	28	2	2500.00	2	0	5	2500	3.32E-03	0.00%	0.00%
s1	10	8	0.82	28	2	2500.00	2	0	5	2500	4.44E-03	0.00%	0.00%
s2	10	8	0.82	28	2	2509.00	2	9	5	2509	4.48E-03	0.00%	0.00%
s3	10	8	0.82	28	2	2538.00	2	38	5	2538	5.47E-03	0.00%	0.00%
s4	10	8	0.90	20	1	3097.00	0	97	6	3097	2.44E-03	0.00%	0.00%
s5	10	8	0.90	20	1	3103.00	0	103	6	3103	3.23E-03	0.00%	0.00%
s6	10	8	0.90	20	1	3110.00	0	110	6	3110	2.75E-03	0.00%	0.00%
s7	10	8	0.90	20	1	3119.00	0	119	6	3119	6.41E-03	0.00%	0.00%
s8	10	8	0.95	15	1	4000.00	0	0	8	4000	2.12E-03	0.00%	0.00%
s9	10	8	0.95	15	1	4000.00	0	0	8	4000	2.02E-03	0.00%	0.00%
s10	10	8	0.95	15	1	4014.00	0	14	8	4014	2.03E-03	0.00%	0.00%
s11	10	8	0.95	15	1	4040.00	0	40	8	4040	2.21E-03	0.00%	0.00%
s12	10	8	0.98	12	-	-	-	-	-	-	1.70E-03	-	-
s13	10	8	0.98	12	-	-	-	-	-	-	1.61E-03	-	-
s14	10	8	0.98	12	-	-	-	-	-	-	1.60E-03	-	-
s15	10	8	0.98	12	-	-	-	-	-	-	1.58E-03	-	-
s16	20	16	0.80	102	9	3505.00	21	5	7	3505	3.10E-02	0.00%	0.00%
s17	20	16	0.80	102	9	3524.00	25	24	7	3524	3.82E-02	0.00%	0.00%
s18	20	16	0.80	102	7	3587.00	18	87	7	3587	2.81E-02	0.00%	0.00%
s19	20	16	0.80	102	8	3695.00	20	195	7	3695	3.20E-02	0.00%	0.00%
s20	20	16	0.84	84	4	4128.00	10	128	8	4128	1.36E-02	0.00%	0.00%
s21	20	16	0.84	84	7	4139.00	15	139	8	4139	2.08E-02	0.00%	0.00%
s22	20	16	0.84	84	4	4146.00	10	146	8	4146	1.38E-02	0.00%	0.00%
s23	20	16	0.84	84	9	4191.00	17	191	8	4191	3.00E-02	0.00%	0.00%
s24	20	16	0.89	63	2	5000.00	2	0	10	5000	6.83E-03	0.00%	0.00%
s25	20	16	0.89	63	1	5000.00	0	0	10	5000	7.53E-03	0.00%	0.00%
s26	20	16	0.89	63	2	5023.00	2	23	10	5023	8.06E-03	0.00%	0.00%
s27	20	16	0.89	63	3	5096.00	3	96	10	5096	8.63E-03	0.00%	0.00%
s28	20	16	0.95	42	1	6558.00	0	58	13	6558	3.95E-03	0.00%	0.00%
s29	20	16	0.95	42	1	6558.00	0	58	13	6558	4.10E-03	0.00%	0.00%
s30	20	16	0.95	42	1	6576.00	0	76	13	6576	4.04E-03	0.00%	0.00%
s31	20	16	0.95	42	1	6626.00	0	126	13	6626	4.62E-03	0.00%	0.00%
s32	30	24	0.77	235	17	4448.85	57	38	9	4538	3.21E-01	0.00%	2.00%
s33	30	24	0.77	235	19	4516.80	69	96	9	4596	3.77E-01	0.00%	1.75%
s34	30	24	0.77	235	13	4668.50	60	202	9	4702	2.82E-01	0.00%	0.72%
s35	30	24	0.77	235	14	4936.29	62	541	9	5041	3.24E-01	0.00%	2.12%
s36	30	24	0.81	198	13	4992.75	53	42	10	5042	1.49E-01	0.00%	0.99%
s37	30	24	0.81	198	13	5030.86	55	102	10	5102	1.64E-01	0.00%	1.41%
s38	30	24	0.81	198	12	5117.43	54	210	10	5210	1.58E-01	0.00%	1.81%
s39	30	24	0.81	198	14	5315.92	51	357	10	5357	1.98E-01	0.00%	0.77%
s40	30	24	0.87	148	10	5815.22	33	47	12	6047	2.23E-01	0.00%	3.99%
s41	30	24	0.87	148	7	5827.43	27	51	12	6051	2.37E-01	0.00%	3.84%
s42	30	24	0.87	148	8	5880.57	28	90	12	6090	2.07E-01	0.00%	3.56%
s43	30	24	0.87	148	9	6010.00	32	575	11	6075	7.45E-02	0.00%	1.08%
s44	30	24	0.94	84	3	7535.50	4	179	15	7679	1.38E-02	0.00%	1.90%
s45	30	24	0.94	84	3	7536.50	4	179	15	7679	1.40E-02	0.00%	1.89%
s46	30	24	0.94	84	4	7555.00	4	179	15	7679	1.62E-02	0.00%	1.64%
s47	30	24	0.94	84	4	7603.00	5	253	15	7753	2.61E-02	0.00%	1.97%
s48	40	32	0.78	392	19	5490.47	95	134	11	5634	1.51E+00	0.00%	2.61%
s49	40	32	0.78	392	21	5550.04	106	186	11	5686	1.57E+00	0.00%	2.45%
s50	40	32	0.78	392	19	5678.28	98	325	11	5825	1.72E+00	0.00%	2.58%
s51	40	32	0.78	392	21	6073.82	104	736	11	6236	2.12E+00	0.00%	2.67%
s52	40	32	0.83	312	10	6402.83	66	178	13	6678	5.87E-01	0.00%	4.30%
s53	40	32	0.83	312	16	6481.93	96	207	13	6707	9.95E-01	0.00%	3.47%
s54	40	32	0.83	312	16	6618.80	92	315	13	6815	1.32E+00	0.00%	2.96%
s55	40	32	0.83	312	13	6887.92	78	628	13	7128	6.94E-01	0.00%	3.49%
s56	40	32	0.88	233	11	7025.00	51	184	14	7184	2.15E-01	0.00%	2.26%
s57	40	32	0.88	233	11	7064.57	68	175	14	7175	2.32E-01	0.00%	1.56%
s58	40	32	0.88	233	12	7185.50	69	324	14	7324	3.19E-01	0.00%	1.93%

Table A.1: Numerical result for **CTSP** using delayed column generation

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols gen	Tardiness	Used vehicles	Obj val	time_spend(sec)	RMP gap	Opt gap
s59	40	32	0.88	233	13	7471.00	74	642	14	7642	4.87E-01	0.00%	2.29%
s60	40	32	0.94	131	4	9292.00	12	292	18	9292	2.85E-02	0.00%	0.00%
s61	40	32	0.94	131	5	9322.00	13	322	18	9322	3.43E-02	0.00%	0.00%
s62	40	32	0.94	131	5	9411.00	14	418	18	9418	4.47E-02	0.00%	0.07%
s63	40	32	0.94	131	5	9601.00	11	636	18	9636	4.65E-02	0.00%	0.36%
s64	50	40	0.79	565	18	6818.61	124	63	14	7063	3.62E+00	0.00%	3.58%
s65	50	40	0.79	565	21	6929.68	128	161	14	7161	3.93E+00	0.00%	3.34%
s66	50	40	0.79	565	21	7136.87	126	459	14	7459	4.01E+00	0.00%	4.51%
s67	50	40	0.79	565	21	7555.14	128	893	14	7893	4.14E+00	0.00%	4.47%
s68	50	40	0.84	451	15	7805.58	108	81	16	8081	1.59E+00	0.00%	3.53%
s69	50	40	0.84	451	14	7884.40	89	158	16	8158	1.67E+00	0.00%	3.47%
s70	50	40	0.84	451	16	8038.91	108	369	16	8369	2.08E+00	0.00%	4.11%
s71	50	40	0.84	451	16	8343.15	114	1045	15	8545	1.74E+00	0.00%	2.42%
s72	50	40	0.89	330	10	9094.78	72	172	18	9172	3.77E-01	0.00%	0.85%
s73	50	40	0.89	330	10	9149.00	66	228	18	9228	5.74E-01	0.00%	0.86%
s74	50	40	0.89	330	11	9309.00	79	370	18	9370	4.87E-01	0.00%	0.66%
s75	50	40	0.89	330	12	9576.83	73	1175	17	9675	5.78E-01	0.00%	1.03%
s76	50	40	0.95	184	5	11536.00	17	164	23	11664	1.26E-01	0.00%	1.11%
s77	50	40	0.95	184	6	11587.50	14	180	23	11680	2.63E-01	0.00%	0.80%
s78	50	40	0.95	184	5	11698.50	18	296	23	11796	1.89E-01	0.00%	0.83%
s79	50	40	0.95	184	5	11929.00	18	559	23	12059	1.67E-01	0.00%	1.09%
m0	60	48	0.80	783	22	7932.72	165	295	16	8295	7.14E+00	0.00%	4.57%
m1	60	48	0.80	783	26	8099.23	196	418	16	8418	8.46E+00	0.00%	3.94%
m2	60	48	0.80	783	24	8333.97	171	705	16	8705	8.31E+00	0.00%	4.45%
m3	60	48	0.80	783	23	8853.38	169	1223	16	9223	8.69E+00	0.00%	4.17%
m4	60	48	0.85	615	17	8626.33	131	388	17	8888	3.41E+00	0.00%	3.03%
m5	60	48	0.85	615	16	8789.67	134	555	17	9055	3.74E+00	0.00%	3.02%
m6	60	48	0.85	615	19	9051.00	146	822	17	9322	3.85E+00	0.00%	2.99%
m7	60	48	0.85	615	19	9543.86	143	1265	17	9765	4.37E+00	0.00%	2.32%
m8	60	48	0.90	434	16	10163.60	119	295	20	10295	8.80E-01	0.00%	1.29%
m9	60	48	0.90	434	16	10320.58	118	444	20	10444	1.07E+00	0.00%	1.20%
m10	60	48	0.90	434	16	10556.58	116	712	20	10712	1.35E+00	0.00%	1.47%
m11	60	48	0.90	434	13	11004.33	108	1150	20	11150	1.18E+00	0.00%	1.32%
m12	60	48	0.95	239	5	13613.00	22	613	26	13613	1.02E-01	0.00%	0.00%
m13	60	48	0.95	239	5	13619.00	24	619	26	13619	1.02E-01	0.00%	0.00%
m14	60	48	0.95	239	6	13669.00	22	669	26	13669	1.29E-01	0.00%	0.00%
m15	60	48	0.95	239	5	13830.00	20	1360	25	13860	1.70E-01	0.00%	0.22%
m16	70	56	0.80	1056	26	8996.02	215	313	18	9313	1.61E+01	0.00%	3.52%
m17	70	56	0.80	1056	25	9208.31	219	649	18	9649	2.87E+01	0.00%	4.79%
m18	70	56	0.80	1056	26	9529.25	221	897	18	9897	2.45E+01	0.00%	3.86%
m19	70	56	0.80	1056	30	10259.66	244	1668	18	10668	3.99E+01	0.00%	3.98%
m20	70	56	0.85	820	22	9822.65	191	121	20	10121	7.65E+00	0.00%	3.04%
m21	70	56	0.85	820	23	10036.28	197	370	20	10370	8.01E+00	0.00%	3.33%
m22	70	56	0.85	820	22	10366.77	192	642	20	10642	7.64E+00	0.00%	2.65%
m23	70	56	0.85	820	23	10976.23	194	1369	20	11369	8.97E+00	0.00%	3.58%
m24	70	56	0.90	567	15	11925.68	127	516	23	12016	2.80E+00	0.00%	0.76%
m25	70	56	0.90	567	16	12075.97	126	650	23	12150	1.60E+00	0.00%	0.61%
m26	70	56	0.90	567	14	12267.13	124	887	23	12387	2.15E+00	0.00%	0.98%
m27	70	56	0.90	567	13	12665.26	136	1362	23	12862	3.06E+00	0.00%	1.55%
m28	70	56	0.95	306	7	14888.00	44	413	29	14913	2.36E-01	0.00%	0.17%
m29	70	56	0.95	306	6	14980.50	35	512	29	15012	2.45E-01	0.00%	0.21%
m30	70	56	0.95	306	7	15164.25	38	697	29	15197	3.54E-01	0.00%	0.22%
m31	70	56	0.95	306	7	15614.71	41	1196	29	15696	4.51E-01	0.00%	0.52%
m32	80	64	0.80	1385	32	10041.06	289	6	21	10506	7.45E+01	0.00%	4.63%
m33	80	64	0.80	1385	31	10208.23	273	138	21	10638	6.52E+01	0.00%	4.21%
m34	80	64	0.80	1385	31	10610.08	281	538	21	11038	5.86E+01	0.00%	4.03%
m35	80	64	0.80	1385	30	11791.16	283	1739	21	12239	1.92E+02	0.00%	3.80%
m36	80	64	0.85	1070	22	11027.45	230	416	22	11416	1.45E+01	0.00%	3.52%
m37	80	64	0.85	1070	25	11247.63	233	680	22	11680	1.85E+01	0.00%	3.84%

Table A.1: Numerical result for **CTSP** using delayed column generation

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols gen	Tardiness	Used vehicles	Obj val	time_spend(sec)	RMP gap	Opt gap
m38	80	64	0.85	1070	23	11575.98	235	988	22	11988	1.45E+01	0.00%	3.56%
m39	80	64	0.85	1070	25	12378.81	262	2140	21	12640	1.52E+01	0.00%	2.11%
m40	80	64	0.90	730	17	13008.33	181	298	26	13298	5.95E+00	0.00%	2.23%
m41	80	64	0.90	730	17	13210.81	178	563	26	13563	6.36E+00	0.00%	2.67%
m42	80	64	0.90	730	18	13540.20	176	1286	25	13786	6.34E+00	0.00%	1.82%
m43	80	64	0.90	730	16	14123.24	189	1921	25	14421	6.00E+00	0.00%	2.11%
m44	80	64	0.95	390	7	16465.00	44	465	32	16465	3.05E-01	0.00%	0.00%
m45	80	64	0.95	390	9	16524.00	51	524	32	16524	4.09E-01	0.00%	0.00%
m46	80	64	0.95	390	7	16665.87	53	671	32	16671	4.15E-01	0.00%	0.03%
m47	80	64	0.95	390	7	17245.82	58	1316	32	17316	6.85E-01	0.00%	0.41%
m48	90	72	0.80	1731	25	11250.00	301	55	23	11555	1.36E+02	0.00%	2.71%
m49	90	72	0.80	1731	28	11376.39	307	307	23	11807	2.52E+02	0.00%	3.79%
m50	90	72	0.80	1731	31	11800.19	324	590	23	12090	8.69E+01	0.00%	2.46%
m51	90	72	0.80	1731	32	13242.09	331	2094	23	13594	3.07E+02	0.00%	2.66%
m52	90	72	0.85	1338	24	11978.83	270	137	25	12637	6.70E+01	0.00%	5.49%
m53	90	72	0.85	1338	24	12240.46	270	372	25	12872	6.23E+01	0.00%	5.16%
m54	90	72	0.85	1338	23	12697.41	251	870	25	13370	4.30E+01	0.00%	5.30%
m55	90	72	0.85	1338	25	13695.87	274	2243	24	14243	8.16E+01	0.00%	3.99%
m56	90	72	0.90	906	15	14181.44	188	562	28	14562	7.60E+00	0.00%	2.68%
m57	90	72	0.90	906	16	14332.39	204	567	28	14567	6.41E+00	0.00%	1.64%
m58	90	72	0.90	906	15	14688.31	188	990	28	14990	6.84E+00	0.00%	2.05%
m59	90	72	0.90	906	18	15390.66	218	2287	27	15787	7.34E+00	0.00%	2.58%
m60	90	72	0.95	496	8	18498.25	72	580	36	18580	1.18E+00	0.00%	0.44%
m61	90	72	0.95	496	7	18596.25	76	1172	35	18672	1.67E+00	0.00%	0.41%
m62	90	72	0.95	496	9	18764.52	68	1354	35	18854	1.47E+00	0.00%	0.48%
m63	90	72	0.95	496	7	19233.56	64	1851	35	19351	1.65E+00	0.01%	0.61%
l0	100	80	0.80	2143	22	12500.00	297	16	26	13016	3.62E+02	1.47%	4.13%
l1	100	80	0.80	2143	31	12581.69	378	100	26	13100	3.92E+02	2.11%	4.12%
l2	100	80	0.80	2143	35	13050.04	394	654	26	13654	4.11E+02	3.45%	4.63%
l3	100	80	0.80	2143	33	14698.16	406	2249	26	15249	4.06E+02	3.37%	3.75%
l4	100	80	0.85	1641	23	13126.70	297	211	27	13711	9.58E+01	0.00%	4.45%
l5	100	80	0.85	1641	27	13403.93	311	508	27	14008	1.23E+02	0.00%	4.51%
l6	100	80	0.85	1641	25	13894.42	305	1388	26	14388	8.33E+01	0.00%	3.55%
l7	100	80	0.85	1641	-	-	-	-	-	-	-	-	-
l8	100	80	0.90	1103	21	15363.81	281	847	30	15847	2.75E+01	0.00%	3.14%
l9	100	80	0.90	1103	21	15595.79	267	947	30	15947	1.33E+01	0.00%	2.25%
l10	100	80	0.90	1103	17	16074.94	238	1440	30	16440	1.15E+01	0.00%	2.27%
l11	100	80	0.90	1103	22	16962.84	279	2379	30	17379	1.63E+01	0.00%	2.45%
l12	100	80	0.95	610	12	19748.88	113	328	39	19828	2.03E+00	0.00%	0.40%
l13	100	80	0.95	610	11	19834.33	116	877	38	19877	1.78E+00	0.00%	0.22%
l14	100	80	0.95	610	11	20111.03	107	1130	38	20130	1.40E+00	0.00%	0.09%
l15	100	80	0.95	610	10	20808.09	104	1939	38	20939	3.55E+00	0.00%	0.63%
l16	110	88	0.80	2557	21	13750.00	314	0	30	15000	3.86E+02	8.23%	9.09%
l17	110	88	0.80	2557	36	13800.90	448	72	29	14572	4.61E+02	4.84%	5.59%
l18	110	88	0.80	2557	32	14320.76	410	630	28	14630	4.17E+02	0.00%	2.16%
l19	110	88	0.80	2557	37	16293.93	462	2585	29	17085	4.78E+02	4.50%	4.85%
l20	110	88	0.85	1945	28	14372.34	345	390	29	14890	1.84E+02	0.01%	3.60%
l21	110	88	0.85	1945	29	14657.18	372	669	29	15169	1.39E+02	0.00%	3.49%
l22	110	88	0.85	1945	26	15161.44	357	1165	29	15665	1.48E+02	0.00%	3.32%
l23	110	88	0.85	1945	32	16585.40	391	2872	29	17372	3.51E+02	3.15%	4.74%
l24	110	88	0.90	1329	18	17022.78	285	967	33	17467	3.87E+01	0.00%	2.61%
l25	110	88	0.90	1329	19	17243.87	296	704	34	17704	4.10E+01	0.00%	2.67%
l26	110	88	0.90	1329	19	17646.65	287	1632	33	18132	1.99E+01	0.00%	2.75%
l27	110	88	0.90	1329	20	18596.12	296	3066	32	19066	4.16E+01	0.00%	2.53%
l28	110	88	0.95	722	14	21404.13	154	1004	41	21504	4.30E+00	0.00%	0.47%
l29	110	88	0.95	722	10	21531.25	114	1108	41	21608	1.75E+00	0.00%	0.36%
l30	110	88	0.95	722	10	21874.50	108	1453	41	21953	2.05E+00	0.00%	0.36%
l31	110	88	0.95	722	12	22559.50	132	2219	41	22719	4.44E+00	0.00%	0.71%
l32	120	96	0.80	3027	19	15000.00	308	6	33	16506	4.12E+02	9.11%	10.04%

Table A.1: Numerical result for **CTSP** using delayed column generation

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols gen	Tardiness	Used vehicles	Obj val	time_spend(sec)	RMP gap	Opt gap
l33	120	96	0.80	3027	35	15023.11	492	32	32	16032	5.27E+02	6.21%	6.72%
l34	120	96	0.80	3027	35	15638.66	490	655	32	16655	5.40E+02	5.88%	6.50%
l35	120	96	0.80	3027	37	17884.20	515	2966	32	18966	5.61E+02	5.67%	6.05%
l36	120	96	0.85	2289	30	15189.20	410	128	32	16128	3.59E+02	3.44%	6.18%
l37	120	96	0.85	2289	29	15516.81	404	452	32	16452	3.60E+02	3.45%	6.03%
l38	120	96	0.85	2289	32	16139.38	432	1052	32	17052	3.68E+02	3.57%	5.65%
l39	120	96	0.85	2289	33	18064.68	443	3266	32	19266	3.72E+02	5.83%	6.65%
l40	120	96	0.90	1578	21	17459.04	317	402	35	17902	2.81E+01	0.00%	2.54%
l41	120	96	0.90	1578	21	17689.55	314	713	35	18213	5.49E+01	0.00%	2.96%
l42	120	96	0.90	1578	21	18246.64	317	1270	35	18770	3.93E+01	0.00%	2.87%
l43	120	96	0.90	1578	21	19666.83	351	3201	34	20201	9.64E+01	0.00%	2.72%
l44	120	96	0.95	847	11	22148.27	155	414	44	22414	4.43E+00	0.00%	1.20%
l45	120	96	0.95	847	13	22305.38	156	1069	43	22569	8.39E+00	0.01%	1.18%
l46	120	96	0.95	847	12	22720.51	145	1417	43	22917	4.53E+00	0.00%	0.86%
l47	120	96	0.95	847	13	23745.00	175	2546	43	24046	6.90E+00	0.00%	1.27%
l48	150	120	0.80	4702	19	18750.00	379	360	41	20860	6.61E+02	10.11%	11.25%
l49	150	120	0.80	4702	34	18820.00	661	87	41	20587	1.01E+03	8.57%	9.39%
l50	150	120	0.80	4702	36	19780.35	646	1160	41	21660	1.08E+03	8.65%	9.50%
l51	150	120	0.80	4702	40	22817.99	704	4478	41	24978	1.18E+03	8.64%	9.47%
l52	150	120	0.85	3559	35	18783.59	572	20	40	20020	4.98E+02	6.02%	6.58%
l53	150	120	0.85	3559	31	19159.43	502	419	41	20919	4.87E+02	8.16%	9.18%
l54	150	120	0.85	3559	31	20132.04	550	1456	41	21956	4.90E+02	8.18%	9.06%
l55	150	120	0.85	3559	34	22940.36	619	4442	40	24442	5.10E+02	6.08%	6.55%
l56	150	120	0.90	2430	25	21240.61	435	702	43	22202	3.41E+02	2.75%	4.53%
l57	150	120	0.90	2430	22	21672.29	398	1111	43	22611	3.37E+02	2.59%	4.33%
l58	150	120	0.90	2430	23	22426.43	449	2191	42	23191	3.40E+02	1.88%	3.41%
l59	150	120	0.90	2430	25	24319.58	475	4520	41	25020	3.45E+02	1.30%	2.88%
l60	150	120	0.95	1289	13	27021.96	254	771	53	27271	1.45E+01	0.00%	0.92%
l61	150	120	0.95	1289	14	27267.51	246	1014	53	27514	1.49E+01	0.00%	0.90%
l62	150	120	0.95	1289	13	27927.45	249	1669	53	28169	1.40E+01	0.00%	0.86%
l63	150	120	0.95	1289	16	29502.75	292	3839	52	29839	4.21E+01	0.00%	1.14%
l64	200	160	0.80	8254	20	25000.00	539	7	57	28507	2.02E+03	12.30%	14.03%
l65	200	160	0.80	8254	41	25274.24	951	760	56	28760	4.03E+03	12.12%	13.79%
l66	200	160	0.80	8254	42	27170.98	987	2658	56	30658	4.16E+03	11.37%	12.83%
l67	200	160	0.80	8254	42	31661.70	964	7546	59	37046	4.20E+03	14.53%	17.01%
l68	200	160	0.85	6230	23	25000.00	630	116	57	28616	8.16E+02	12.63%	14.46%
l69	200	160	0.85	6230	33	25378.69	785	520	58	29520	1.05E+03	14.00%	16.32%
l70	200	160	0.85	6230	35	27301.25	825	2705	57	31205	1.09E+03	12.50%	14.30%
l71	200	160	0.85	6230	37	31755.36	872	7639	61	38139	1.15E+03	16.72%	20.10%
l72	200	160	0.90	4226	29	26764.05	674	664	56	28664	4.52E+02	6.58%	7.10%
l73	200	160	0.90	4226	26	27530.84	657	1685	57	30185	4.39E+02	8.68%	9.64%
l74	200	160	0.90	4226	28	28899.23	669	3667	57	32167	4.49E+02	10.04%	11.31%
l75	200	160	0.90	4226	31	32347.79	731	7989	57	36489	4.69E+02	11.33%	12.80%
l76	200	160	0.95	2190	19	34107.74	479	995	67	34495	3.08E+02	0.00%	1.14%
l77	200	160	0.95	2190	16	34752.10	447	1652	67	35152	1.29E+02	0.00%	1.15%
l78	200	160	0.95	2190	18	35790.78	519	3046	67	36546	3.22E+02	1.54%	2.11%
l79	200	160	0.95	2190	20	38954.27	560	6966	65	39466	3.25E+02	0.44%	1.31%
l80	250	200	0.80	12901	22	31311.00	726	135	77	38635	6.20E+03	18.95%	23.39%
l81	250	200	0.80	12901	28	32094.00	942	2021	97	50521	7.95E+03	36.47%	57.42%
l82	250	200	0.80	12901	43	35279.07	1220	4967	92	50967	-	30.78%	44.47%
l83	250	200	0.80	12901	31	41337.50	1066	11571	78	50571	8.98E+03	18.26%	22.34%
l84	250	200	0.85	9753	25	31311.00	851	389	76	38389	1.92E+03	18.43%	22.61%
l85	250	200	0.85	9753	41	32115.75	1145	1552	74	38552	3.03E+03	16.69%	20.04%
l86	250	200	0.85	9753	39	35391.27	1135	4701	75	42201	2.96E+03	16.13%	19.24%
l87	250	200	0.85	9753	40	41390.69	1163	12662	71	48162	3.02E+03	14.05%	16.36%
l88	250	200	0.90	6619	28	32076.06	852	953	70	35953	7.34E+02	10.75%	12.09%
l89	250	200	0.90	6619	34	33191.21	927	2440	73	38940	8.22E+02	14.68%	17.32%
l90	250	200	0.90	6619	31	36088.28	926	5204	72	41204	7.89E+02	12.38%	14.18%
l91	250	200	0.90	6619	36	41779.88	1015	12048	71	47548	8.62E+02	12.12%	13.81%

Table A.1: Numerical result for **CTSP** using delayed column generation

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols gen	Tardiness	Used vehicles	Obj val	time_spend(sec)	RMP gap	Opt gap
l92	250	200	0.95	3434	19	40797.43	606	1174	82	42174	3.56E+02	3.14%	3.37%
l93	250	200	0.95	3434	20	41773.36	720	2372	82	43372	3.61E+02	3.48%	3.83%
l94	250	200	0.95	3434	23	43984.71	737	5939	79	45439	3.71E+02	2.91%	3.31%
l95	250	200	0.95	3434	20	49060.64	708	10789	79	50289	3.65E+02	2.36%	2.50%
l96	300	240	0.80	18486	21	37834.00	858	814	96	48814	-	22.49%	29.02%
l97	300	240	0.80	18486	39	39520.23	1355	3294	87	46794	-	15.54%	18.41%
l98	300	240	0.80	18486	34	44507.00	1371	9543	114	66543	-	33.12%	49.51%
l99	300	240	0.80	18486	28	52067.00	1118	16954	101	67454	-	22.81%	29.55%
l100	300	240	0.85	13895	22	37834.00	881	1090	95	48590	3.79E+03	22.13%	28.43%
l101	300	240	0.85	13895	43	39566.95	1488	3242	93	49742	7.29E+03	20.45%	25.72%
l102	300	240	0.85	13895	42	44566.21	1437	8394	99	57894	7.34E+03	23.02%	29.91%
l103	300	240	0.85	13895	44	52088.02	1461	16379	96	64379	7.48E+03	19.09%	23.60%
l104	300	240	0.90	9448	32	38149.05	1097	1534	95	49034	1.46E+03	22.16%	28.53%
l105	300	240	0.90	9448	33	40186.37	1155	3773	87	47273	1.50E+03	14.97%	17.63%
l106	300	240	0.90	9448	38	44989.49	1249	8878	101	59378	1.63E+03	24.22%	31.98%
l107	300	240	0.90	9448	33	52353.85	1164	16975	94	63975	1.48E+03	18.16%	22.20%
l108	300	240	0.95	4861	21	48018.72	875	1473	97	49973	4.36E+02	3.77%	4.07%
l109	300	240	0.95	4861	25	49635.36	974	4723	97	53223	4.59E+02	6.68%	7.23%
l110	300	240	0.95	4861	27	53398.01	1021	8788	93	55288	4.75E+02	3.41%	3.54%
l111	300	240	0.95	4861	22	60149.06	872	16286	95	63786	4.48E+02	5.60%	6.05%

Table A.2: Numerical result for **CTSP** using branch-and-rpice

inst_id	num_test	num_vehicle	density	tardiness	used_vehicle	obj_val	time_spent (sec)	opt_gap	root_obj	best bound	nodes
s0	10	8	0.82	0	5	2500	1.03E-02	0.00%	2500	2500	1
s1	10	8	0.82	0	5	2500	8.94E-03	0.00%	2500	2500	1
s2	10	8	0.82	9	5	2509	7.92E-03	0.00%	2509	2509	1
s3	10	8	0.82	38	5	2538	9.77E-03	0.00%	2538	2538	1
s4	10	8	0.90	97	6	3097	9.03E-03	0.00%	3097	3097	1
s5	10	8	0.90	103	6	3103	1.76E-02	0.00%	3103	3103	1
s6	10	8	0.90	110	6	3110	1.35E-02	0.00%	3110	3110	1
s7	10	8	0.90	119	6	3119	8.00E-02	0.00%	3119	3119	1
s8	10	8	0.95	0	8	4000	4.16E-03	0.00%	4000	4000	1
s9	10	8	0.95	0	8	4000	4.28E-03	0.00%	4000	4000	1
s10	10	8	0.95	14	8	4014	4.75E-03	0.00%	4014	4014	1
s11	10	8	0.95	40	8	4040	4.91E-03	0.00%	4040	4040	1
s12	10	8	0.98	0	-		4.03E-03	-	-	-	1
s13	10	8	0.98	0	-		2.82E-03	-	-	-	1
s14	10	8	0.98	0	-		2.98E-03	-	-	-	1
s15	10	8	0.98	0	-		3.90E-03	-	-	-	1
s16	20	16	0.80	5	7	3505	4.77E-02	0.00%	3505	3505	1
s17	20	16	0.80	24	7	3524	6.01E-02	0.00%	3524	3524	1
s18	20	16	0.80	87	7	3587	4.91E-02	0.00%	3587	3587	1
s19	20	16	0.80	195	7	3695	1.27E-01	0.00%	3695	3695	1
s20	20	16	0.84	128	8	4128	5.04E-02	0.00%	4128	4128	1
s21	20	16	0.84	139	8	4139	3.48E-02	0.00%	4139	4139	1
s22	20	16	0.84	146	8	4146	2.87E-02	0.00%	4146	4146	1
s23	20	16	0.84	191	8	4191	5.46E-02	0.00%	4191	4191	1
s24	20	16	0.89	0	10	5000	1.03E-02	0.00%	5000	5000	1
s25	20	16	0.89	0	10	5000	7.14E-03	0.00%	5000	5000	1
s26	20	16	0.89	23	10	5023	1.25E-02	0.00%	5023	5023	1
s27	20	16	0.89	96	10	5096	1.13E-02	0.00%	5096	5096	1
s28	20	16	0.95	58	13	6558	1.28E-02	0.00%	6558	6558	1
s29	20	16	0.95	58	13	6558	8.80E-03	0.00%	6558	6558	1
s30	20	16	0.95	76	13	6576	1.10E-02	0.00%	6576	6576	1
s31	20	16	0.95	126	13	6626	1.11E-02	0.00%	6626	6626	1
s32	30	24	0.77	24	9	4524	1.66E+00	0.23%	4448	4513	29
s33	30	24	0.77	60	9	4560	1.14E+00	0.48%	4516	4538	13
s34	30	24	0.77	187	9	4687	1.24E+00	0.04%	4668	4685	13
s35	30	24	0.77	461	9	4961	9.63E-01	0.08%	4936	4957	11
s36	30	24	0.81	14	10	5014	8.96E-01	0.10%	4992	5009	19
s37	30	24	0.81	45	10	5045	7.51E-01	0.06%	5030	5041	17
s38	30	24	0.81	152	10	5152	1.04E+00	0.35%	5117	5134	23
s39	30	24	0.81	352	10	5352	6.38E-01	0.09%	5315	5347	13
s40	30	24	0.87	491	11	5991	1.32E+00	0.01%	5815	5990	107
s41	30	24	0.87	465	11	5965	1.00E+00	0.09%	5827	5959	81
s42	30	24	0.87	481	11	5981	9.18E-01	0.11%	5880	5974	47
s43	30	24	0.87	575	11	6075	5.53E-01	0.07%	6010	6071	29
s44	30	24	0.94	179	15	7679	9.39E-02	0.14%	7535	7668	15
s45	30	24	0.94	179	15	7679	1.07E-01	0.10%	7536	7671	15
s46	30	24	0.94	179	15	7679	1.06E-01	0.20%	7555	7664	15
s47	30	24	0.94	253	15	7753	1.37E-01	0.19%	7603	7738	19
s48	40	32	0.78	79	11	5579	9.54E+00	0.06%	5490	5575	71
s49	40	32	0.78	161	11	5661	2.00E+01	0.03%	5550	5659	139
s50	40	32	0.78	313	11	5813	3.89E+01	0.02%	5678	5812	291
s51	40	32	0.78	701	11	6201	9.08E+01	0.00%	6073	6200	737
s52	40	32	0.83	536	12	6536	5.93E+00	0.05%	6402	6532	85
s53	40	32	0.83	627	12	6627	1.37E+01	0.01%	6481	6626	213
s54	40	32	0.83	748	12	6748	1.30E+01	0.03%	6618	6746	191
s55	40	32	0.83	506	13	7006	1.04E+01	0.00%	6887	7005	131
s56	40	32	0.88	133	14	7133	1.55E+00	0.11%	7025	7125	39
s57	40	32	0.88	172	14	7172	1.60E+00	0.01%	7064	7171	41
s58	40	32	0.88	288	14	7288	1.87E+00	0.07%	7185	7283	35

Table A.2: Numerical result for **CTSP** using branch-and-rpice

inst_id	num_test	num_vehicle	density	tardiness	used_vehicle	obj_val	time_spent (sec)	opt_gap	root_obj	best bound	nodes
s59	40	32	0.88	586	14	7586	3.19E+00	0.03%	7471	7584	61
s60	40	32	0.94	292	18	9292	3.35E-02	0.00%	9292	9292	1
s61	40	32	0.94	322	18	9322	4.29E-02	0.00%	9322	9322	1
s62	40	32	0.94	418	18	9418	1.02E-01	0.07%	9411	9411	5
s63	40	32	0.94	636	18	9636	1.01E-01	0.36%	9601	9601	5
s64	50	40	0.79	41	14	7041	3.63E+02	0.00%	6818	7040	1521
s65	50	40	0.79	133	14	7133	5.10E+02	0.00%	6929	7132	1849
s66	50	40	0.79	326	14	7326	4.49E+02	0.00%	7136	7325	1719
s67	50	40	0.79	772	14	7772	6.01E+02	0.69%	7555	7718	2151
s68	50	40	0.84	14	16	8014	1.08E+02	0.00%	7805	8013	805
s69	50	40	0.84	65	16	8065	8.77E+01	0.01%	7884	8064	547
s70	50	40	0.84	619	15	8119	1.77E+01	0.02%	8038	8117	83
s71	50	40	0.84	884	15	8384	1.08E+01	0.00%	8343	8383	41
s72	50	40	0.89	172	18	9172	4.76E+00	0.01%	9094	9171	57
s73	50	40	0.89	223	18	9223	4.51E+00	0.04%	9149	9219	43
s74	50	40	0.89	364	18	9364	6.24E+00	0.02%	9309	9362	65
s75	50	40	0.89	1163	17	9663	7.38E+00	0.01%	9576	9661	83
s76	50	40	0.95	148	23	11648	1.55E+00	0.03%	11536	11644	45
s77	50	40	0.95	180	23	11680	1.89E+00	0.02%	11587	11678	77
s78	50	40	0.95	296	23	11796	2.35E+00	0.01%	11698	11794	85
s79	50	40	0.95	559	23	12059	5.00E+00	0.02%	11929	12057	179
m0	60	48	0.80	206	16	8206	1.80E+03	1.45%	7932	8088	2733
m1	60	48	0.80	342	16	8342	1.80E+03	1.20%	8099	8243	2257
m2	60	48	0.80	1024	15	8524	1.35E+03	0.62%	8333	8471	1643
m3	60	48	0.80	1505	15	9005	6.02E+02	0.45%	8853	8964	723
m4	60	48	0.85	308	17	8808	8.62E+01	0.01%	8626	8807	181
m5	60	48	0.85	470	17	8970	9.60E+01	0.01%	8789	8969	191
m6	60	48	0.85	739	17	9239	1.29E+02	0.00%	9051	9238	271
m7	60	48	0.85	1184	17	9684	8.47E+01	0.00%	9543	9683	167
m8	60	48	0.90	290	20	10290	2.12E+01	0.01%	10163	10288	97
m9	60	48	0.90	442	20	10442	2.27E+01	0.01%	10320	10440	101
m10	60	48	0.90	658	20	10658	2.28E+01	0.05%	10556	10652	99
m11	60	48	0.90	1111	20	11111	3.13E+01	0.02%	11004	11109	141
m12	60	48	0.95	613	26	13613	2.11E-01	0.00%	13613	13613	1
m13	60	48	0.95	619	26	13619	1.74E-01	0.00%	13619	13613	1
m14	60	48	0.95	669	26	13669	2.02E-01	0.00%	13669	13669	1
m15	60	48	0.95	1341	25	13841	9.99E-01	0.00%	13830	13840	13
m16	70	56	0.80	193	18	9193	9.52E+02	1.00%	8996	9102	519
m17	70	56	0.80	337	18	9337	1.26E+03	0.25%	9208	9314	661
m18	70	56	0.80	682	18	9682	6.03E+02	0.92%	9529	9594	269
m19	70	56	0.80	1501	18	10501	1.80E+03	1.56%	10259	10339	899
m20	70	56	0.85	103	20	10103	8.25E+02	1.00%	9822	10003	909
m21	70	56	0.85	293	20	10293	8.55E+02	0.99%	10036	10192	897
m22	70	56	0.85	631	20	10631	1.63E+03	1.00%	10366	10525	1749
m23	70	56	0.85	1653	19	11153	7.82E+02	0.34%	10976	11115	777
m24	70	56	0.90	466	23	11966	2.66E+01	0.01%	11925	11965	53
m25	70	56	0.90	606	23	12106	1.89E+01	0.03%	12075	12102	35
m26	70	56	0.90	1344	22	12344	4.76E+01	0.03%	12267	12340	95
m27	70	56	0.90	1735	22	12735	3.35E+01	0.01%	12665	12734	59
m28	70	56	0.95	413	29	14913	1.54E+00	0.06%	14888	14904	9
m29	70	56	0.95	512	29	15012	2.11E+00	0.04%	14980	15006	13
m30	70	56	0.95	697	29	15197	1.61E+00	0.12%	15164	15178	9
m31	70	56	0.95	1191	29	15691	4.52E+00	0.01%	15614	15690	29
m32	80	64	0.80	3	21	10503	1.81E+03	4.31%	10041	10068	245
m33	80	64	0.80	437	20	10437	1.81E+03	1.68%	10208	10264	395
m34	80	64	0.80	764	20	10764	1.11E+03	0.99%	10610	10658	191
m35	80	64	0.80	1874	20	11874	1.27E+03	0.58%	11791	11806	143
m36	80	64	0.85	284	22	11284	1.80E+03	1.14%	11027	11156	925
m37	80	64	0.85	572	22	11572	1.80E+03	1.77%	11247	11370	863

Table A.2: Numerical result for **CTSP** using branch-and-rpice

inst_id	num_test	num_vehicle	density	tardiness	used_vehicle	obj_val	time_spent (sec)	opt_gap	root_obj	best bound	nodes
m38	80	64	0.85	838	22	11838	1.80E+03	1.09%	11575	11709	1019
m39	80	64	0.85	1994	21	12494	6.04E+02	0.24%	12378	12463	265
m40	80	64	0.90	293	26	13293	6.02E+02	0.88%	13008	13177	755
m41	80	64	0.90	901	25	13401	6.02E+02	0.37%	13210	13352	733
m42	80	64	0.90	1232	25	13732	6.02E+02	0.46%	13540	13669	759
m43	80	64	0.90	1839	25	14339	6.01E+02	0.56%	14123	14259	735
m44	80	64	0.95	465	32	16465	6.84E-01	0.00%	16465	16465	1
m45	80	64	0.95	524	32	16524	7.49E-01	0.00%	16524	16524	1
m46	80	64	0.95	671	32	16671	1.70E+00	0.03%	16665	16665	5
m47	80	64	0.95	1310	32	17310	7.82E+00	0.02%	17245	17306	25
m48	90	72	0.80	1	23	11501	1.81E+03	2.22%	11250	11251	197
m49	90	72	0.80	146	23	11646	1.81E+03	2.18%	11376	11397	213
m50	90	72	0.80	582	23	12082	1.82E+03	2.17%	11800	11825	199
m51	90	72	0.80	1980	23	13480	1.81E+03	1.68%	13242	13256	111
m52	90	72	0.85	285	24	12285	1.80E+03	1.54%	11978	12098	467
m53	90	72	0.85	606	24	12606	1.80E+03	1.92%	12240	12368	431
m54	90	72	0.85	994	24	12994	1.81E+03	1.29%	12697	12828	517
m55	90	72	0.85	2153	24	14153	1.80E+03	2.65%	13695	13788	407
m56	90	72	0.90	319	28	14319	6.02E+02	0.07%	14181	14308	353
m57	90	72	0.90	566	28	14566	6.04E+02	0.96%	14332	14427	329
m58	90	72	0.90	1342	27	14842	6.03E+02	0.38%	14688	14786	439
m59	90	72	0.90	2175	27	15675	9.55E+02	0.85%	15390	15542	585
m60	90	72	0.95	1079	35	18579	5.93E+01	0.00%	18498	18578	121
m61	90	72	0.95	1172	35	18672	5.17E+01	0.01%	18596	18670	105
m62	90	72	0.95	1354	35	18854	3.95E+01	0.00%	18764	18853	101
m63	90	72	0.95	1850	35	19350	8.31E+01	0.00%	19233	19349	171
l0	100	80	0.80	18	26	13018	1.41E+03	4.14%	12499	12500	41
l1	100	80	0.80	100	26	13100	7.53E+02	4.12%	12581	12581	5
l2	100	80	0.80	563	26	13563	7.70E+02	3.93%	13050	13050	5
l3	100	80	0.80	2169	26	15169	1.29E+03	3.19%	14698	14699	21
l4	100	80	0.85	526	26	13526	1.81E+03	2.39%	13126	13210	225
l5	100	80	0.85	771	26	13771	1.80E+03	2.16%	13403	13479	193
l6	100	80	0.85	1277	26	14277	1.81E+03	2.17%	13894	13973	217
l7	100	80	0.85	2524	26	15524	1.82E+03	2.60%	15102	15130	119
l8	100	80	0.90	598	30	15598	6.01E+02	0.76%	15363	15479	209
l9	100	80	0.90	848	30	15848	6.35E+02	0.79%	15595	15724	241
l10	100	80	0.90	1288	30	16288	6.34E+02	0.71%	16074	16172	223
l11	100	80	0.90	2302	30	17302	1.80E+03	1.14%	16962	17107	667
l12	100	80	0.95	328	39	19828	4.87E+01	0.01%	19748	19826	63
l13	100	80	0.95	877	38	19877	1.89E+01	0.01%	19834	19876	21
l14	100	80	0.95	1130	38	20130	8.56E+00	0.07%	20111	20116	9
l15	100	80	0.95	2416	37	20916	2.45E+02	0.00%	20808	20915	343
l16	110	88	0.80	42	29	14542	8.25E+02	5.76%	13750	13750	9
l17	110	88	0.80	50	29	14550	1.01E+03	5.42%	13800	13802	13
l18	110	88	0.80	548	29	15048	1.81E+03	5.03%	14320	14326	35
l19	110	88	0.80	2526	29	17026	1.16E+03	4.48%	16293	16295	17
l20	110	88	0.85	346	29	14846	1.86E+03	3.07%	14372	14403	63
l21	110	88	0.85	605	29	15105	1.81E+03	2.74%	14657	14702	127
l22	110	88	0.85	1105	29	15605	1.81E+03	2.55%	15161	15217	95
l23	110	88	0.85	3024	28	17024	1.81E+03	2.54%	16585	16602	65
l24	110	88	0.90	819	33	17319	1.81E+03	1.19%	17022	17114	397
l25	110	88	0.90	980	33	17480	8.60E+02	0.83%	17243	17336	177
l26	110	88	0.90	1541	33	18041	1.80E+03	1.64%	17646	17749	371
l27	110	88	0.90	2897	32	18897	1.81E+03	1.14%	18596	18683	375
l28	110	88	0.95	981	41	21481	5.51E+01	0.00%	21404	21480	43
l29	110	88	0.95	1106	41	21606	2.87E+01	0.01%	21531	21604	21
l30	110	88	0.95	1453	41	21953	8.80E+01	0.01%	21874	21950	71
l31	110	88	0.95	2212	41	22712	6.02E+02	0.25%	22559	22655	531
l32	120	96	0.80	0	33	16500	9.39E+02	10.00%	15000	15000	9

Table A.2: Numerical result for **CTSP** using branch-and-rpice

inst_id	num_test	num_vehicle	density	tardiness	used_vehicle	obj_val	time_spent (sec)	opt_gap	root_obj	best bound	nodes
l33	120	96	0.80	84	32	16084	1.33E+03	7.05%	15023	15024	15
l34	120	96	0.80	638	32	16638	1.12E+03	6.39%	15638	15638	9
l35	120	96	0.80	3048	32	19048	6.04E+02	6.51%	17884	17884	3
l36	120	96	0.85	100	32	16100	7.87E+02	6.00%	15189	15189	9
l37	120	96	0.85	444	32	16444	1.18E+03	5.74%	15516	15551	31
l38	120	96	0.85	1156	31	16656	1.92E+03	2.96%	16139	16177	31
l39	120	96	0.85	2996	32	18996	7.39E+02	5.16%	18064	18064	5
l40	120	96	0.90	386	35	17886	1.81E+03	1.65%	17459	17595	229
l41	120	96	0.90	587	35	18087	1.80E+03	1.54%	17689	17813	235
l42	120	96	0.90	1175	35	18675	1.81E+03	1.72%	18246	18358	211
l43	120	96	0.90	3088	34	20088	1.81E+03	1.73%	19666	19747	133
l44	120	96	0.95	321	44	22321	3.24E+02	0.02%	22148	22317	175
l45	120	96	0.95	982	43	22482	2.68E+02	0.00%	22305	22481	145
l46	120	96	0.95	1417	43	22917	6.04E+02	0.19%	22720	22874	335
l47	120	96	0.95	2897	42	23897	3.36E+02	0.00%	23745	23896	191
l48	150	120	0.80	54	41	20554	9.05E+02	9.62%	18750	18750	3
l49	150	120	0.80	108	41	20608	1.30E+03	9.50%	18820	18820	3
l50	150	120	0.80	1178	42	22178	1.37E+03	12.12%	19780	19780	3
l51	150	120	0.80	4651	41	25151	1.43E+03	10.23%	22817	22817	3
l52	150	120	0.85	47	40	20047	1.85E+03	6.72%	18783	18784	13
l53	150	120	0.85	353	41	20853	1.48E+03	8.82%	19159	19162	13
l54	150	120	0.85	1475	40	21475	1.16E+03	6.67%	20132	20132	7
l55	150	120	0.85	4424	40	24424	6.19E+02	6.47%	22940	22940	3
l56	150	120	0.90	872	43	22372	1.30E+03	5.16%	21240	21274	25
l57	150	120	0.90	1380	42	22380	1.37E+03	3.19%	21672	21687	17
l58	150	120	0.90	1783	43	23283	1.27E+03	3.77%	22426	22437	13
l59	150	120	0.90	4477	41	24977	1.44E+03	2.65%	24319	24332	19
l60	150	120	0.95	850	53	27350	6.03E+02	0.96%	27021	27089	89
l61	150	120	0.95	1012	53	27512	6.01E+02	0.58%	27267	27354	91
l62	150	120	0.95	1677	53	28177	6.05E+02	0.57%	27927	28015	87
l63	150	120	0.95	3762	52	29762	6.08E+02	0.63%	29502	29575	77
l64	200	160	0.80	132	60	30132	4.30E+03	20.53%	25000	25000	3
l65	200	160	0.80	358	57	28858	7.90E+03	14.18%	25274	25274	3
l66	200	160	0.80	2746	56	30746	7.59E+03	13.16%	27170	27170	3
l67	200	160	0.80	7958	58	36958	7.24E+03	16.73%	31661	31661	3
l68	200	160	0.85	8	56	28008	1.41E+03	12.03%	25000	25000	3
l69	200	160	0.85	578	56	28578	1.75E+03	12.61%	25378	25378	3
l70	200	160	0.85	2811	57	31311	1.82E+03	14.69%	27301	27301	3
l71	200	160	0.85	7978	57	36478	1.81E+03	14.87%	31755	31755	3
l72	200	160	0.90	608	58	29608	8.56E+02	10.63%	26764	26764	3
l73	200	160	0.90	1742	56	29742	8.63E+02	8.03%	27530	27530	3
l74	200	160	0.90	3599	59	33099	8.97E+02	14.53%	28899	28899	3
l75	200	160	0.90	8029	56	36029	9.44E+02	11.38%	32347	32347	3
l76	200	160	0.95	980	67	34480	1.56E+03	0.99%	34107	34142	17
l77	200	160	0.95	1682	67	35182	1.54E+03	1.20%	34752	34764	17
l78	200	160	0.95	3310	66	36310	9.79E+02	1.45%	35790	35790	9
l79	200	160	0.95	6911	66	39911	9.22E+02	2.46%	38954	38954	7
l80	250	200	0.80	545	74	37545	-	-	-	-	3
l81	250	200	0.80	1604	79	41104	-	-	-	-	3
l82	250	200	0.80	4912	83	46412	-	-	-	-	3
l83	250	200	0.80	11820	82	52820	-	-	-	-	3
l84	250	200	0.85	328	73	36828	4.74E+03	17.62%	31310	31310	3
l85	250	200	0.85	1024	73	37524	6.54E+03	16.84%	32115	32115	3
l86	250	200	0.85	5061	71	40561	6.99E+03	14.61%	35391	35391	3
l87	250	200	0.85	11621	81	52121	7.20E+03	25.93%	41390	41390	3
l88	250	200	0.90	1164	75	38664	1.67E+03	20.54%	32076	32076	3
l89	250	200	0.90	2635	74	39635	1.71E+03	19.41%	33191	33191	3
l90	250	200	0.90	5759	75	43259	1.73E+03	19.87%	36088	36088	3
l91	250	200	0.90	12150	74	49150	1.87E+03	17.64%	41779	41779	3

Table A.2: Numerical result for **CTSP** using branch-and-rpice

inst_id	num_test	num_vehicle	density	tardiness	used_vehicle	obj_val	time_spent (sec)	opt_gap	root_obj	best bound	nodes
l92	250	200	0.95	1300	81	41800	2.12E+03	2.42%	40797	40812	13
l93	250	200	0.95	2120	82	43120	6.02E+02	3.22%	41773	41773	3
l94	250	200	0.95	5688	79	45188	6.15E+02	2.74%	43984	43984	3
l95	250	200	0.95	10787	79	50287	6.03E+02	2.50%	49060	49060	3
l96	300	240	0.80	661	98	49661	-	-	-	-	3
l97	300	240	0.80	3015	91	48515	-	-	-	-	3
l98	300	240	0.80	11893	93	58393	-	-	-	-	3
l99	300	240	0.80	18389	110	73389	-	-	-	-	3
l100	300	240	0.85	894	97	49394	-	-	-	-	3
l101	300	240	0.85	2912	88	46912	-	-	-	-	3
l102	300	240	0.85	8280	95	55780	-	-	-	-	3
l103	300	240	0.85	18562	126	81562	-	-	-	-	3
l104	300	240	0.90	-	-	-	-	-	-	-	-
l105	300	240	0.90	1992	96	49992	1.11E+03	-	48018	-	3
l106	300	240	0.90	4881	95	52381	1.15E+03	-	49635	-	3
l107	300	240	0.90	8876	96	56876	1.15E+03	-	53398	-	3
l108	300	240	0.95	15778	94	62778	1.11E+03	-	60149	-	3
l109	300	240	0.95	-	-	-	-	-	-	-	-
l110	300	240	0.95	-	-	-	-	-	-	-	-
l111	300	240	0.95	-	-	-	-	-	-	-	-

Table A.3: Numerical results for **CTSPR**

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols Gen	Tardiness	Used vehicles	Obj val	Sol time	RMP gap	Opt gap
s0	10	8	0.82	28	2	2500.00	1	0	5	2500	3.08E+01	0.00%	0.00%
s1	10	8	0.82	28	2	2500.00	1	0	5	2500	3.94E+01	0.00%	0.00%
s2	10	8	0.82	28	2	2522.00	1	22	5	2522	3.31E+01	0.00%	0.00%
s3	10	8	0.82	28	2	2564.00	1	64	5	2564	2.92E+01	0.00%	0.00%
s4	10	8	0.90	20	1	3097.00	0	97	6	3097	2.84E+01	0.00%	0.00%
s5	10	8	0.90	20	1	3103.00	0	103	6	3103	3.15E+01	0.00%	0.00%
s6	10	8	0.90	20	1	3110.00	0	110	6	3110	2.49E+01	0.00%	0.00%
s7	10	8	0.90	20	1	3119.00	0	119	6	3119	1.86E+02	0.00%	0.00%
s8	10	8	0.95	15	1	4000.00	0	0	8	4000	2.09E+01	0.00%	0.00%
s9	10	8	0.95	15	1	4000.00	0	0	8	4000	3.92E+01	0.00%	0.00%
s10	10	8	0.95	15	1	4014.00	0	14	8	4014	1.92E+01	0.00%	0.00%
s11	10	8	0.95	15	1	4040.00	0	40	8	4040	2.77E+01	0.00%	0.00%
s12	10	8	0.98	12	-	-	-	-	-	-	1.74E+01	-	-
s13	10	8	0.98	12	-	-	-	-	-	-	8.02E+00	-	-
s14	10	8	0.98	12	-	-	-	-	-	-	6.66E+00	-	-
s15	10	8	0.98	12	-	-	-	-	-	-	5.11E+00	-	-
s16	20	16	0.80	102	7	3942.00	17	107	8	4107	1.04E+02	0.00%	4.19%
s17	20	16	0.80	102	8	3949.00	18	101	8	4101	1.06E+02	0.00%	3.85%
s18	20	16	0.80	102	7	3971.00	18	150	8	4150	1.41E+02	0.00%	4.51%
s19	20	16	0.80	102	10	4074.00	29	260	8	4260	1.33E+02	0.00%	4.57%
s20	20	16	0.84	84	5	4361.00	7	361	8	4361	1.52E+02	0.00%	0.00%
s21	20	16	0.84	84	5	4368.00	7	368	8	4368	7.92E+01	0.00%	0.00%
s22	20	16	0.84	84	3	4368.00	6	368	8	4368	6.44E+01	0.00%	0.00%
s23	20	16	0.84	84	4	4420.00	6	420	8	4420	8.46E+01	0.00%	0.00%
s24	20	16	0.89	63	2	5000.00	1	0	10	5000	2.89E+01	0.00%	0.00%
s25	20	16	0.89	63	1	5000.00	0	0	10	5000	3.07E+01	0.00%	0.00%
s26	20	16	0.89	63	2	5031.00	1	31	10	5031	3.09E+01	0.00%	0.00%
s27	20	16	0.89	63	1	5132.00	0	132	10	5132	3.82E+01	0.00%	0.00%
s28	20	16	0.95	42	1	6558.00	0	58	13	6558	2.07E+01	0.00%	0.00%
s29	20	16	0.95	42	1	6558.00	0	58	13	6558	2.51E+01	0.00%	0.00%
s30	20	16	0.95	42	1	6576.00	0	76	13	6576	2.34E+01	0.00%	0.00%
s31	20	16	0.95	42	1	6631.00	0	131	13	6631	2.24E+01	0.00%	0.00%
s32	30	24	0.77	235	7	4784.00	27	284	9	4784	2.12E+02	0.00%	0.00%
s33	30	24	0.77	235	11	4838.00	38	338	9	4838	2.81E+02	0.00%	0.00%
s34	30	24	0.77	235	15	4942.00	57	442	9	4942	3.76E+02	0.00%	0.00%
s35	30	24	0.77	235	12	5217.00	53	717	9	5217	2.60E+02	0.00%	0.00%
s36	30	24	0.81	198	9	5318.50	29	430	10	5430	3.13E+02	0.00%	2.10%
s37	30	24	0.81	198	10	5335.00	40	426	10	5426	2.06E+02	0.00%	1.71%
s38	30	24	0.81	198	11	5398.00	34	503	10	5503	3.85E+02	0.00%	1.95%
s39	30	24	0.81	198	9	5597.50	35	614	10	5614	2.05E+02	0.00%	0.29%
s40	30	24	0.87	148	8	5949.50	25	125	12	6125	1.40E+02	0.00%	2.95%
s41	30	24	0.87	148	6	5992.00	19	168	12	6168	1.14E+02	0.00%	2.94%
s42	30	24	0.87	148	7	6068.50	20	230	12	6230	1.54E+02	0.00%	2.66%
s43	30	24	0.87	148	9	6278.00	27	432	12	6432	1.33E+02	0.00%	2.45%
s44	30	24	0.94	84	3	7704.50	2	270	15	7770	1.34E+02	0.00%	0.85%
s45	30	24	0.94	84	3	7704.50	2	279	15	7779	1.23E+02	0.00%	0.97%
s46	30	24	0.94	84	3	7723.00	2	303	15	7803	8.60E+01	0.00%	1.04%
s47	30	24	0.94	84	3	7801.00	2	347	15	7847	7.66E+01	0.00%	0.59%
s48	40	32	0.78	392	20	5927.06	103	255	12	6255	1.99E+03	0.00%	5.53%
s49	40	32	0.78	392	19	5981.35	96	276	12	6276	2.03E+03	0.00%	4.93%
s50	40	32	0.78	392	24	6049.00	122	369	12	6369	2.26E+03	0.00%	5.29%
s51	40	32	0.78	392	21	6296.88	111	683	12	6683	2.19E+03	0.00%	6.13%
s52	40	32	0.83	312	12	6904.75	61	649	13	7149	1.36E+03	0.00%	3.54%
s53	40	32	0.83	312	15	6976.35	72	675	13	7175	9.35E+02	0.00%	2.85%
s54	40	32	0.83	312	19	7086.49	92	719	13	7219	1.11E+03	0.00%	1.87%
s55	40	32	0.83	312	19	7319.00	114	819	13	7319	4.82E+02	0.00%	0.00%
s56	40	32	0.88	233	11	7486.50	45	165	15	7665	3.30E+02	0.00%	2.38%
s57	40	32	0.88	233	10	7504.00	47	194	15	7694	4.29E+02	0.00%	2.53%
s58	40	32	0.88	233	14	7604.50	60	296	15	7796	5.23E+02	0.00%	2.52%

Table A.3: Numerical results for **CTSPR**

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols Gen	Tardiness	Used vehicles	Obj val	Sol time	RMP gap	Opt gap
s59	40	32	0.88	233	16	7899.50	73	585	15	8085	4.98E+02	0.00%	2.35%
s60	40	32	0.94	131	2	9782.00	3	282	19	9782	7.72E+01	0.00%	0.00%
s61	40	32	0.94	131	2	9802.00	4	302	19	9802	5.90E+01	0.00%	0.00%
s62	40	32	0.94	131	3	9816.11	5	318	19	9818	1.11E+02	0.00%	0.02%
s63	40	32	0.94	131	2	10005.50	3	521	19	10021	7.10E+01	0.00%	0.15%
s64	50	40	0.79	565	22	7370.57	146	610	14	7610	4.28E+03	0.00%	3.25%
s65	50	40	0.79	565	19	7450.28	144	808	14	7808	4.83E+03	0.00%	4.80%
s66	50	40	0.79	565	22	7592.16	163	814	14	7814	3.48E+03	0.00%	2.92%
s67	50	40	0.79	565	20	7908.96	130	1125	14	8125	3.18E+03	0.00%	2.73%
s68	50	40	0.84	451	15	8373.92	110	567	16	8567	2.48E+03	0.00%	2.31%
s69	50	40	0.84	451	17	8449.44	124	629	16	8629	1.71E+03	0.00%	2.13%
s70	50	40	0.84	451	17	8552.14	125	779	16	8779	2.01E+03	0.00%	2.65%
s71	50	40	0.84	451	20	8795.60	148	991	16	8991	2.28E+03	0.00%	2.22%
s72	50	40	0.89	330	15	9733.00	104	733	18	9733	4.57E+02	0.00%	0.00%
s73	50	40	0.89	330	11	9761.25	71	362	19	9862	8.67E+02	0.00%	1.03%
s74	50	40	0.89	330	15	9785.00	110	785	18	9785	4.10E+02	0.00%	0.00%
s75	50	40	0.89	330	17	10012.50	111	1016	18	10016	5.12E+02	0.00%	0.03%
s76	50	40	0.95	184	7	11855.00	27	355	23	11855	1.28E+02	0.00%	0.00%
s77	50	40	0.95	184	4	11883.00	7	383	23	11883	1.17E+02	0.00%	0.00%
s78	50	40	0.95	184	11	11970.00	26	470	23	11970	1.62E+02	0.00%	0.00%
s79	50	40	0.95	184	6	12227.00	14	727	23	12227	1.35E+02	0.00%	0.00%
m0	60	48	0.80	783	25	8693.19	201	538	17	9038	8.66E+03	0.00%	3.97%
m1	60	48	0.80	783	26	8853.89	219	1019	16	9019	6.62E+03	0.00%	1.86%
m2	60	48	0.80	783	30	9032.33	259	1279	16	9279	1.06E+04	0.00%	2.73%
m3	60	48	0.80	783	30	9383.49	241	1652	16	9652	1.06E+04	0.00%	2.86%
m4	60	48	0.85	615	20	9537.41	172	813	18	9813	3.31E+03	0.00%	2.89%
m5	60	48	0.85	615	19	9684.52	188	929	18	9929	4.75E+03	0.00%	2.52%
m6	60	48	0.85	615	23	9885.79	205	1160	18	10160	5.94E+03	0.00%	2.77%
m7	60	48	0.85	615	25	10278.13	221	1436	18	10436	5.86E+03	0.00%	1.54%
m8	60	48	0.90	434	19	10893.00	162	489	21	10989	1.72E+03	0.00%	0.88%
m9	60	48	0.90	434	19	11007.47	176	634	21	11134	1.71E+03	0.00%	1.15%
m10	60	48	0.90	434	19	11210.82	153	878	21	11378	1.80E+03	0.00%	1.49%
m11	60	48	0.90	434	20	11621.37	169	1253	21	11753	2.50E+03	0.00%	1.13%
m12	60	48	0.95	239	10	14024.75	47	199	28	14199	1.14E+03	0.00%	1.24%
m13	60	48	0.95	239	10	14066.50	52	215	28	14215	1.11E+03	0.00%	1.06%
m14	60	48	0.95	239	9	14173.89	42	316	28	14316	9.33E+02	0.00%	1.00%
m15	60	48	0.95	239	6	14538.89	20	672	28	14672	1.30E+03	0.00%	0.92%
m16	70	56	0.80	1056	25	9756.91	224	667	19	10167	1.81E+04	0.00%	4.20%
m17	70	56	0.80	1056	25	9907.36	253	825	19	10325	3.88E+04	0.00%	4.22%
m18	70	56	0.80	1056	33	10134.83	340	1067	19	10567	4.83E+04	0.00%	4.26%
m19	70	56	0.80	1056	33	10601.52	285	1666	19	11166	4.54E+04	0.00%	5.32%
m20	70	56	0.85	820	22	10620.79	232	495	21	10995	1.17E+04	0.00%	3.52%
m21	70	56	0.85	820	23	10757.15	239	1037	20	11037	2.45E+04	0.00%	2.60%
m22	70	56	0.85	820	28	11030.36	307	1239	20	11239	3.48E+04	0.00%	1.89%
m23	70	56	0.85	820	27	11518.16	263	1884	20	11884	3.99E+04	0.00%	3.18%
m24	70	56	0.90	567	21	13004.11	200	697	25	13197	4.70E+03	0.00%	1.48%
m25	70	56	0.90	567	17	13075.25	192	817	25	13317	5.73E+03	0.00%	1.85%
m26	70	56	0.90	567	24	13222.94	245	951	25	13451	7.70E+03	0.00%	1.72%
m27	70	56	0.90	567	23	13586.56	246	1716	24	13716	6.91E+03	0.00%	0.95%
m28	70	56	0.95	306	17	15430.00	121	507	30	15507	1.41E+03	0.00%	0.50%
m29	70	56	0.95	306	13	15488.92	95	575	30	15575	2.20E+03	0.00%	0.56%
m30	70	56	0.95	306	13	15705.80	95	732	30	15732	6.61E+02	0.01%	0.17%
m31	70	56	0.95	306	11	16035.20	100	1166	30	16166	4.38E+03	0.00%	0.82%
m32	80	64	0.80	1385	35	10650.02	386	561	21	11061	7.75E+04	0.00%	3.86%
m33	80	64	0.80	1385	38	10871.65	424	798	21	11298	1.83E+05	0.00%	3.92%
m34	80	64	0.80	1385	43	11189.18	526	1143	22	12143	3.18E+05	6.06%	8.52%
m35	80	64	0.80	1385	35	11852.24	384	1939	22	12939	3.15E+05	6.87%	9.17%
m36	80	64	0.85	1070	24	11977.48	295	903	23	12403	4.44E+04	0.00%	3.55%
m37	80	64	0.85	1070	25	12134.42	320	1056	23	12556	3.85E+04	0.00%	3.47%

Table A.3: Numerical results for **CTSPR**

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols Gen	Tardiness	Used vehicles	Obj val	Sol time	RMP gap	Opt gap
m38	80	64	0.85	1070	37	12378.89	419	1676	22	12676	8.28E+04	0.00%	2.40%
m39	80	64	0.85	1070	29	12814.15	355	2156	22	13156	9.12E+04	0.00%	2.67%
m40	80	64	0.90	730	18	13955.54	218	688	27	14188	1.09E+04	0.00%	1.67%
m41	80	64	0.90	730	19	14104.50	240	826	27	14326	9.35E+03	0.00%	1.57%
m42	80	64	0.90	730	20	14375.20	261	1148	27	14648	1.76E+04	0.00%	1.90%
m43	80	64	0.90	730	25	14605.02	340	1860	26	14860	3.27E+04	0.00%	1.75%
m44	80	64	0.95	390	10	17153.00	75	653	33	17153	7.68E+02	0.00%	0.00%
m45	80	64	0.95	390	17	17154.00	196	654	33	17154	1.03E+03	0.00%	0.00%
m46	80	64	0.95	390	13	17277.00	144	779	33	17279	1.04E+03	0.01%	0.01%
m47	80	64	0.95	390	15	17627.38	185	1682	32	17682	8.03E+03	0.01%	0.31%
m48	90	72	0.80	1731	35	11853.13	461	854	26	13854	3.24E+05	14.26%	16.88%
m49	90	72	0.80	1731	35	12098.21	451	802	25	13302	3.24E+05	8.13%	9.95%
m50	90	72	0.80	1731	46	12444.88	617	1164	23	12664	1.22E+05	0.00%	1.76%
m51	90	72	0.80	1731	41	13319.80	508	2108	28	16108	3.26E+05	17.12%	20.93%
m52	90	72	0.85	1338	25	12901.02	354	930	25	13430	1.45E+05	0.00%	4.10%
m53	90	72	0.85	1338	28	13125.00	393	1038	25	13538	1.71E+05	0.00%	3.15%
m54	90	72	0.85	1338	41	13499.52	512	1487	25	13987	2.23E+05	0.00%	3.61%
m55	90	72	0.85	1338	35	14051.30	440	2318	27	15818	3.11E+05	9.56%	12.57%
m56	90	72	0.90	906	26	15334.59	346	989	29	15489	5.66E+04	0.00%	1.01%
m57	90	72	0.90	906	25	15488.00	363	1229	29	15729	1.49E+05	0.01%	1.56%
m58	90	72	0.90	906	22	15857.34	345	1577	29	16077	4.06E+04	0.00%	1.39%
m59	90	72	0.90	906	30	16138.68	456	2851	27	16351	6.32E+04	0.00%	1.32%
m60	90	72	0.95	496	17	19452.08	223	522	38	19522	9.51E+03	0.00%	0.36%
m61	90	72	0.95	496	15	19546.57	144	640	38	19640	7.36E+03	0.00%	0.48%
m62	90	72	0.95	496	15	19696.44	194	1224	37	19724	5.25E+03	0.00%	0.14%
m63	90	72	0.95	496	20	19914.19	278	2524	35	20024	1.88E+04	0.00%	0.55%
l0	100	80	0.80	2143	36	13136.34	534	748	27	14248	3.43E+05	7.54%	8.46%
l1	100	80	0.80	2143	42	13444.30	601	966	30	15966	3.48E+05	15.64%	18.76%
l2	100	80	0.80	2143	52	13835.99	810	1628	26	14628	994.5113	5.20%	5.72%
l3	100	80	0.80	2143	56	14849.58	804	2985	26	15985	989.3598	6.99%	7.65%
l4	100	80	0.85	1641	30	14212.83	454	1002	29	15502	3.15E+05	7.07%	9.07%
l5	100	80	0.85	1641	37	14493.56	544	1169	29	15669	3.18E+05	6.54%	8.11%
l6	100	80	0.85	1641	43	14769.01	622	1675	30	16675	3.23E+05	10.87%	12.91%
l7	100	80	0.85	1641	44	15687.13	683	2795	33	19295	3.23E+05	18.56%	23.00%
l8	100	80	0.90	1103	34	17058.25	551	1816	31	17316	1.09E+05	0.00%	1.51%
l9	100	80	0.90	1103	25	17187.25	468	1030	33	17530	3.06E+05	0.69%	1.99%
l10	100	80	0.90	1103	26	17300.26	444	1612	32	17612	1.43E+05	0.01%	1.80%
l11	100	80	0.90	1103	33	17872.56	610	3226	32	19226	3.08E+05	6.75%	7.57%
l12	100	80	0.95	610	18	20723.88	280	1304	39	20804	1.12E+04	0.00%	0.39%
l13	100	80	0.95	610	19	20930.89	243	1018	40	21018	1.09E+04	0.00%	0.42%
l14	100	80	0.95	610	21	20975.27	302	584	41	21084	1.09E+04	0.00%	0.52%
l15	100	80	0.95	610	27	21613.80	466	2861	38	21861	3.04E+05	0.34%	1.14%
l16	110	88	0.80	2557	45	14401.24	697	806	28	14806	1010.068	2.61%	2.81%
l17	110	88	0.80	2557	53	14695.62	807	1158	29	15658	1020.988	6.12%	6.55%
l18	110	88	0.80	2557	66	15189.42	1096	1689	29	16189	1049.084	6.16%	6.58%
l19	110	88	0.80	2557	71	16489.68	1242	3453	29	17953	1058.823	8.12%	8.87%
l20	110	88	0.85	1945	31	15577.54	551	1247	30	16247	937.0979	3.68%	4.30%
l21	110	88	0.85	1945	37	15850.64	636	1534	30	16534	943.7575	3.99%	4.31%
l22	110	88	0.85	1945	45	16093.34	711	2479	29	16979	949.8954	4.68%	5.50%
l23	110	88	0.85	1945	54	17165.96	934	4141	29	18641	956.3347	7.64%	8.59%
l24	110	88	0.90	1329	33	18520.55	646	1324	35	18824	918.9542	1.24%	1.64%
l25	110	88	0.90	1329	32	18757.65	593	1642	35	19142	917.5644	1.77%	2.05%
l26	110	88	0.90	1329	32	18788.45	626	2268	34	19268	918.3043	1.77%	2.55%
l27	110	88	0.90	1329	38	19717.25	807	4432	33	20932	920.2834	5.42%	6.16%
l28	110	88	0.95	722	17	22686.56	319	1784	42	22784	106.9546	0.00%	0.43%
l29	110	88	0.95	722	16	22781.21	233	782	44	22782	11.33921	0.00%	0.00%
l30	110	88	0.95	722	15	22870.71	191	887	44	22887	18.88204	0.00%	0.07%
l31	110	88	0.95	722	27	23541.00	578	3377	41	23877	908.2491	1.17%	1.43%
l32	120	96	0.80	3027	45	15684.64	803	847	31	16347	1055.087	4.03%	4.22%

Table A.3: Numerical results for **CTSPR**

Inst id	Num tests	Num vehicles	Incomp density	Num seq	Iterations	Relax Obj val	Cols Gen	Tardiness	Used vehicles	Obj val	Sol time	RMP gap	Opt gap
l33	120	96	0.80	3027	61	16058.04	1094	1326	32	17326	1093.539	7.27%	7.90%
l34	120	96	0.80	3027	70	16689.47	1330	2453	32	18453	1128.526	9.52%	10.57%
l35	120	96	0.80	3027	72	18306.01	1541	4195	32	20195	1144.103	9.35%	10.32%
l36	120	96	0.85	2289	38	16497.45	637	1478	33	17978	955.9296	7.84%	8.97%
l37	120	96	0.85	2289	42	16862.66	800	1834	33	18334	967.2947	7.97%	8.73%
l38	120	96	0.85	2289	59	17336.66	1073	2755	33	19255	984.9424	9.83%	11.07%
l39	120	96	0.85	2289	71	18711.35	1475	4981	33	21481	1009.223	12.83%	14.80%
l40	120	96	0.90	1578	40	19109.75	790	1344	37	19844	927.0247	3.39%	3.84%
l41	120	96	0.90	1578	35	19383.37	756	2176	36	20176	928.8183	3.73%	4.09%
l42	120	96	0.90	1578	39	19619.76	854	2610	35	20110	931.2616	2.12%	2.50%
l43	120	96	0.90	1578	54	20891.31	1276	4604	36	22604	944.9096	7.50%	8.20%
l44	120	96	0.95	847	28	23936.57	623	2252	44	24252	910.7416	1.19%	1.32%
l45	120	96	0.95	847	22	23983.47	408	1073	46	24073	910.4647	0.16%	0.37%
l46	120	96	0.95	847	22	24066.80	455	1148	46	24148	910.064	0.17%	0.34%
l47	120	96	0.95	847	41	25248.04	1168	4504	43	26004	917.0442	2.67%	2.99%
l48	150	120	0.80	4702	70	19540.21	1525	1387	41	21887	1565.46	10.69%	12.01%
l49	150	120	0.80	4702	70	20195.63	1653	1859	43	23359	1568.07	13.52%	15.66%
l50	150	120	0.80	4702	101	21542.10	2520	4366	43	25866	1824.678	16.71%	20.07%
l51	150	120	0.80	4702	97	24581.87	2715	7924	42	28924	1834.026	15.02%	17.66%
l52	150	120	0.85	3559	50	19947.54	1105	1849	41	22349	1090.795	10.71%	12.04%
l53	150	120	0.85	3559	62	20633.65	1408	2477	43	23977	1176.945	13.86%	16.20%
l54	150	120	0.85	3559	87	21915.62	2054	4601	41	25101	1259.493	12.63%	14.53%
l55	150	120	0.85	3559	87	24777.03	2354	8913	45	31413	1305.364	21.13%	26.78%
l56	150	120	0.90	2430	50	22748.10	1245	1914	45	24414	990.0995	6.71%	7.32%
l57	150	120	0.90	2430	56	23183.88	1382	3480	45	25980	1008.054	10.71%	12.06%
l58	150	120	0.90	2430	58	24334.27	1562	5374	43	26874	1027.247	9.42%	10.44%
l59	150	120	0.90	2430	79	26628.25	2116	7761	44	29761	1081.903	10.45%	11.76%
l60	150	120	0.95	1289	34	28586.00	902	1474	56	29474	934.8881	2.88%	3.11%
l61	150	120	0.95	1289	38	28831.01	1178	2634	54	29634	938.297	2.48%	2.79%
l62	150	120	0.95	1289	37	29813.77	1318	4301	53	30801	942.9943	3.10%	3.31%
l63	150	120	0.95	1289	51	32299.33	1820	7537	53	34037	961.3962	5.02%	5.38%

Table A.4: Numerical result for solving multiple vehicle program scheduling

Inst_id	overlap	total_num_test	total_num_vehicle	iterations	relax_obj_val	cols_gen	obj_val	time_spent	time_spent (sec)	opt_gap	root_opt_gap
s13_s29	0.25	30	24	1	9005.00	186	9005		3.09E-01	0.00%	0.00%
s13_s29	0.5	30	24	1	9000.00	186	9000		9.91E-02	0.00%	0.00%
s13_s29	0.75	30	24	1	9005.00	186	9005		1.53E-01	0.00%	0.00%
s13_s29	1	30	24	1	9006.00	186	9006		1.50E-01	0.00%	0.00%
s13_s45	0.25	40	32	6	9985.50	593	10027		5.86E-01	0.00%	0.42%
s13_s45	0.5	40	32	7	9985.50	587	10027		4.29E-01	0.00%	0.42%
s13_s45	0.75	40	32	5	9985.50	578	10032		4.53E-01	0.00%	0.47%
s13_s45	1	40	32	8	9993.50	617	10054		4.70E-01	0.00%	0.61%
s13_s61	0.25	50	40	6	11147.00	1264	11147		3.72E-01	0.00%	0.00%
s13_s61	0.5	50	40	8	11137.00	1268	11137		4.07E-01	0.00%	0.00%
s13_s61	0.75	50	40	8	11159.00	1300	11159		4.41E-01	0.00%	0.00%
s13_s61	1	50	40	8	11254.00	1361	11254		4.26E-01	0.00%	0.00%
s13_s77	0.25	60	48	9	13211.00	2355	13211		6.91E-01	0.00%	0.00%
s13_s77	0.5	60	48	8	13175.00	2330	13175		7.30E-01	0.00%	0.00%
s13_s77	0.75	60	48	9	13209.00	2293	13209		8.11E-01	0.00%	0.00%
s13_s77	1	60	48	9	13318.00	2370	13318		7.62E-01	0.00%	0.00%
s29_s45	0.25	50	40	4	10997.50	735	11032		4.10E-01	0.00%	0.31%
s29_s45	0.5	50	40	5	11003.50	724	11032		4.49E-01	0.00%	0.26%
s29_s45	0.75	50	40	7	10997.50	762	11037		4.64E-01	0.00%	0.36%
s29_s45	1	50	40	6	11007.50	774	11051		4.62E-01	0.00%	0.40%
s29_s61	0.25	60	48	9	12152.00	1457	12152		4.77E-01	0.00%	0.00%
s29_s61	0.5	60	48	10	12142.00	1420	12142		5.08E-01	0.00%	0.00%
s29_s61	0.75	60	48	5	12164.00	1398	12164		3.90E-01	0.00%	0.00%
s29_s61	1	60	48	5	12274.00	1371	12274		3.99E-01	0.00%	0.00%
s29_s77	0.25	70	56	7	14216.00	2427	14216		6.97E-01	0.00%	0.00%
s29_s77	0.5	70	56	17	14185.00	2587	14185		9.32E-01	0.00%	0.00%
s29_s77	0.75	70	56	6	14220.00	2409	14220		7.05E-01	0.00%	0.00%
s29_s77	1	70	56	10	14352.00	2708	14352		8.36E-01	0.00%	0.00%
s45_s61	0.25	70	56	7	13190.00	1830	13190		8.16E-01	0.00%	0.00%
s45_s61	0.5	70	56	13	13184.00	1900	13184		9.38E-01	0.00%	0.00%
s45_s61	0.75	70	56	8	13205.00	1838	13205		1.01E+00	0.00%	0.00%
s45_s61	1	70	56	11	13300.00	1939	13300		1.04E+00	0.00%	0.00%
s45_s77	0.25	80	64	7	15256.00	2883	15256		1.42E+00	0.00%	0.00%
s45_s77	0.5	80	64	12	15255.00	3218	15255		1.68E+00	0.00%	0.00%
s45_s77	0.75	80	64	7	15279.00	2823	15279		1.36E+00	0.00%	0.00%
s45_s77	1	80	64	12	15471.00	3343	15471		2.59E+00	0.00%	0.00%
s61_s77	0.25	90	72	11	16415.00	3691	16415		1.26E+00	0.00%	0.00%
s61_s77	0.5	90	72	19	16452.00	3761	16452		1.59E+00	0.00%	0.00%
s61_s77	0.75	90	72	16	16448.00	4147	16448		1.57E+00	0.00%	0.00%
s61_s77	1	90	72	16	16725.00	4636	16725		2.11E+00	0.00%	0.00%
s13_m13	0.25	70	56	16	14678.00	3766	14678		9.06E+00	0.00%	0.00%
s13_m13	0.5	70	56	16	14747.00	3666	14747		7.78E+00	0.00%	0.00%
s13_m13	0.75	70	56	12	14794.00	3642	14794		1.05E+01	0.00%	0.00%
s13_m13	1	70	56	16	14917.40	4116	14983		7.02E+00	0.00%	0.44%
s13_m29	0.25	80	64	15	16204.00	5709	16204		6.14E+02	0.42%	0.42%
s13_m29	0.5	80	64	15	16350.00	5722	16350		6.15E+02	1.15%	1.15%
s13_m29	0.75	80	64	15	16364.00	5908	16364		6.15E+02	0.00%	0.00%
s13_m29	1	80	64	17	16664.00	6158	16664		2.97E+01	0.00%	0.00%
s13_m45	0.25	90	72	17	17769.00	8302	17769		6.04E+02	0.00%	0.00%
s13_m45	0.5	90	72	23	17918.00	7482	17918		6.05E+02	0.00%	0.00%
s13_m45	0.75	90	72	21	18006.00	7731	18006		6.05E+02	0.00%	0.00%
s13_m45	1	90	72	19	18120.29	8670	18154		1.40E+01	0.00%	0.19%
s13_m49	0.25	100	80	39	17076.00	18389	17076		6.19E+02	0.00%	0.00%
s13_m49	0.5	100	80	41	17450.00	14889	17450		6.24E+02	0.00%	0.00%
s13_m49	0.75	100	80	37	17515.16	16998	18914		6.20E+02	7.99%	7.99%
s13_m49	1	100	80	41	17525.28	20247	18294		6.22E+02	3.99%	4.39%
s29_m13	0.25	80	64	17	15682.00	3926	15682		9.75E+00	0.00%	0.00%
s29_m13	0.5	80	64	12	15765.00	3830	15765		3.20E+01	0.00%	0.00%
s29_m13	0.75	80	64	17	15798.00	3963	15798		1.39E+01	0.00%	0.00%

Table A.4: Numerical result for solving multiple vehicle program scheduling

Inst_id	overlap	total_num_test	total_num_vehicle	iterations	relax_obj_val	cols_gen	obj_val	time_spent	time_spent (sec)	opt_gap	root_opt_gap
s29_m13	1	80	64	19	15989.27	4233	16078		1.31E+01	0.00%	0.55%
s29_m29	0.25	90	72	17	17176.00	5953	17176		6.14E+02	0.00%	0.00%
s29_m29	0.5	90	72	21	17344.00	5851	17344		6.15E+02	0.00%	0.00%
s29_m29	0.75	90	72	16	17198.00	6020	17198		1.75E+01	0.00%	0.00%
s29_m29	1	90	72	19	17699.00	6428	17699		3.63E+01	0.00%	0.00%
s29_m45	0.25	100	80	16	18736.00	8433	18736		6.04E+02	0.00%	0.00%
s29_m45	0.5	100	80	23	18987.00	7774	18987		6.05E+02	0.00%	0.00%
s29_m45	0.75	100	80	19	19021.00	7791	19021		6.04E+02	0.00%	0.00%
s29_m45	1	100	80	25	19210.51	10289	19295		7.22E+01	0.00%	0.44%
s29_m49	0.25	110	88	38	18066.00	18529	18066		6.21E+02	0.00%	0.00%
s29_m49	0.5	110	88	47	18516.46	15307	18522		6.23E+02	0.03%	0.03%
s29_m49	0.75	110	88	40	18577.83	17584	19154		6.21E+02	3.10%	3.10%
s29_m49	1	110	88	46	18635.32	22383	19951		6.25E+02	6.84%	7.06%
s45_m13	0.25	90	72	14	16722.00	4237	16722		1.28E+01	0.00%	0.00%
s45_m13	0.5	90	72	14	16795.00	4211	16795		2.63E+01	0.00%	0.00%
s45_m13	0.75	90	72	13	16838.00	4330	16838		1.69E+01	0.00%	0.00%
s45_m13	1	90	72	23	16969.48	5225	17119		2.61E+01	0.00%	0.88%
s45_m29	0.25	100	80	15	18248.00	6191	18248		6.15E+02	0.00%	0.00%
s45_m29	0.5	100	80	15	18414.00	6175	18414		6.15E+02	0.00%	0.00%
s45_m29	0.75	100	80	19	18454.00	6613	18454		6.15E+02	0.00%	0.00%
s45_m29	1	100	80	24	18908.60	7563	18922		3.03E+02	0.01%	0.07%
s45_m45	0.25	110	88	15	19780.00	8776	19780		6.05E+02	0.00%	0.00%
s45_m45	0.5	110	88	24	20048.75	8151	20068		6.06E+02	0.10%	0.10%
s45_m45	0.75	110	88	23	20070.44	8590	20270		6.06E+02	0.99%	0.99%
s45_m45	1	110	88	28	20205.44	10930	20461		5.70E+02	0.00%	1.26%
s45_m49	0.25	120	96	37	19338.00	18838	19338		6.19E+02	0.00%	0.00%
s45_m49	0.5	120	96	43	19520.00	15848	19520		6.23E+02	0.00%	0.00%
s45_m49	0.75	120	96	43	19598.51	18145	22024		6.24E+02	10.38%	12.38%
s45_m49	1	120	96	44	19727.03	23874	21273		6.27E+02	7.31%	7.84%
s61_m13	0.25	100	80	15	17901.00	4962	17901		1.77E+01	0.00%	0.00%
s61_m13	0.5	100	80	17	18021.00	5148	18021		9.87E+01	0.00%	0.00%
s61_m13	0.75	100	80	16	18061.00	5554	18061		1.14E+02	0.00%	0.00%
s61_m13	1	100	80	19	18363.00	6070	18363		3.58E+01	0.00%	0.00%
s61_m29	0.25	110	88	16	19474.00	6998	19474		6.15E+02	0.00%	0.00%
s61_m29	0.5	110	88	22	19605.00	7021	19605		6.15E+02	0.00%	0.00%
s61_m29	0.75	110	88	19	19790.00	7222	19790		6.15E+02	0.00%	0.00%
s61_m29	1	110	88	21	20202.00	8740	20202		5.07E+01	0.00%	0.00%
s61_m45	0.25	120	96	17	21031.00	9609	21031		6.05E+02	0.00%	0.00%
s61_m45	0.5	120	96	31	21275.00	9046	21275		6.06E+02	0.00%	0.00%
s61_m45	0.75	120	96	26	21452.50	10156	21580		6.07E+02	0.59%	0.59%
s61_m45	1	120	96	36	21632.36	13505	22756		6.09E+02	5.10%	5.19%
s61_m49	0.25	130	104	37	20391.00	19741	20391		6.21E+02	0.00%	0.00%
s61_m49	0.5	130	104	47	20874.00	16479	20874		6.25E+02	0.00%	0.00%
s61_m49	0.75	130	104	43	21143.51	18944	21310		6.25E+02	0.79%	0.79%
s61_m49	1	130	104	54	21316.92	25354	36584		6.35E+02	42.06%	71.62%
s77_m13	0.25	110	88	11	20005.00	5938	20005		1.46E+01	0.00%	0.00%
s77_m13	0.5	110	88	24	20128.00	6191	20128		1.17E+02	0.00%	0.00%
s77_m13	0.75	110	88	19	20173.00	6663	20173		1.25E+02	0.00%	0.00%
s77_m13	1	110	88	28	20538.10	8249	20610		6.05E+02	0.24%	0.35%
s77_m29	0.25	120	96	15	21579.00	8032	21579		6.15E+02	0.00%	0.00%
s77_m29	0.5	120	96	22	21772.00	8201	21772		6.16E+02	0.00%	0.00%
s77_m29	0.75	120	96	27	21979.00	9094	21979		6.17E+02	0.00%	0.00%
s77_m29	1	120	96	29	22521.41	10909	22643		6.20E+02	0.54%	0.54%
s77_m45	0.25	130	104	16	23187.00	10572	23187		6.05E+02	0.00%	0.00%
s77_m45	0.5	130	104	34	23441.00	10388	23441		6.09E+02	0.00%	0.00%
s77_m45	0.75	130	104	40	23693.24	12293	23809		6.09E+02	0.49%	0.49%
s77_m45	1	130	104	42	23937.93	16098	27883		6.13E+02	14.38%	16.48%
s77_m49	0.25	140	112	38	22924.58	20792	23053		6.22E+02	0.56%	0.56%
s77_m49	0.5	140	112	50	23060.30	17998	23199		6.33E+02	0.60%	0.60%

Table A.4: Numerical result for solving multiple vehicle program scheduling

Inst_id	overlap	total_num_test	total_num_vehicle	iterations	relax_obj_val	cols_gen	obj_val	time_spent	time_spent (sec)	opt_gap	root_opt_gap
s77_m49	0.75	140	112	46	23176.58	21372	24070		6.31E+02	3.85%	3.85%
s77_m49	1	140	112	64	23362.43	30665	28556		6.51E+02	18.48%	22.23%
m13_m29	0.25	130	104	16	23196.00	9602	23196		6.21E+02	0.00%	0.00%
m13_m29	0.5	130	104	22	23674.00	9593	23674		6.21E+02	0.00%	0.00%
m13_m29	0.75	130	104	25	23800.84	10750	23874		6.21E+02	0.31%	0.31%
m13_m29	1	130	104	29	24092.18	13382	27687		6.23E+02	13.87%	14.92%
m13_m45	0.25	140	112	19	24867.00	11872	24867		6.12E+02	0.00%	0.00%
m13_m45	0.5	140	112	36	25200.51	12224	25315		6.12E+02	0.45%	0.45%
m13_m45	0.75	140	112	41	25256.00	14128	27875		6.17E+02	9.56%	10.37%
m13_m45	1	140	112	46	25594.47	18598	31002		6.21E+02	17.61%	21.13%
m13_m49	0.25	150	120	41	24346.92	22358	24748		6.27E+02	1.65%	1.65%
m13_m49	0.5	150	120	47	24818.21	19484	27827		6.37E+02	10.21%	12.12%
m13_m49	0.75	150	120	60	24931.38	23888	41839		6.48E+02	42.30%	67.82%
m13_m49	1	150	120	59	25411.30	31391	46513		6.57E+02	45.61%	83.04%
m29_m45	0.25	150	120	20	26661.00	14467	26661		6.22E+02	0.00%	0.00%
m29_m45	0.5	150	120	44	27198.19	14830	28222		6.30E+02	3.76%	3.76%
m29_m45	0.75	150	120	44	27275.73	18387	32506		6.32E+02	18.18%	19.18%
m29_m45	1	150	120	51	27661.48	21943	-		6.40E+02	-	-
m29_m49	0.25	160	128	41	26285.00	25036	26285		6.43E+02	0.00%	0.00%
m29_m49	0.5	160	128	51	26884.79	22120	40106		6.56E+02	36.56%	49.18%
m29_m49	0.75	160	128	56	27089.41	27109	49351		6.64E+02	47.27%	82.18%
m29_m49	1	160	128	67	27456.87	35148	-		6.90E+02	-	-
m45_m49	0.25	170	136	39	27572.46	26743	29535		6.29E+02	7.12%	7.12%
m45_m49	0.5	170	136	54	28033.83	25016	28760		6.50E+02	2.59%	2.59%
m45_m49	0.75	170	136	63	28528.64	31698	56194		6.68E+02	50.47%	96.97%
m45_m49	1	170	136	73	29430.53	39444	-		6.98E+02	-	-

BIBLIOGRAPHY

BIBLIOGRAPHY

Alidaee, B., and H. Li (2014), 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*, *11*(1), 294–301.

Balakrishnan, N., J. J. Kanet, and S. V. Sridharan (1999), Early/tardy scheduling with sequence dependent setups on uniform parallel machines, *Computers and Operations Research*, *26*(2), 127–141.

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance (1998), Branch-and-Price: Column Generation for Solving Huge Integer Programs.

Bartels, J.-H., and J. Zimmermann (2009), Scheduling Tests in Automotive R&D Projects, *European Journal of Operational Research*, *193*(3), 805–819.

Blazewicz, J., J. K. Lenstra, and A. R. Kan (1983), Scheduling subject to resource constraints: classification and complexity, *Discrete Applied Mathematics*, *5*(1), 11–24.

Cao, D., M. Chen, and G. Wan (2005), Parallel machine selection and job scheduling to minimize machine cost and job tardiness, *Computers and Operations Research*, *32*(8), 1995–2012.

Chelst, K., J. Sidelko, A. Przebienda, J. Lockledge, and D. Mihailidis (2001), Rightsizing and Management of Prototype Vehicle Testing at Ford Motor Company, *Interfaces*, *31*(1), 91–107, doi:10.1287/inte.31.1.91.9687.

Chen, J.-F. (2005), Unrelated parallel machine scheduling with secondary resource constraints, *The International Journal of Advanced Manufacturing Technology*, *26*(3), 285–292.

Chen, J.-F., and T.-H. Wu (2006), Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints, *Omega*, *34*(1), 81–89.

Cheng, T., and C. Sin (1990), A state-of-the-art review of parallel-machine scheduling research, *European Journal of Operational Research*, *47*(3), 271–292, doi: 10.1016/0377-2217(90)90215-W.

Christofides, N., R. Alvarez-Valdés, and J. M. Tamarit (1987), Project scheduling with resource constraints: A branch and bound approach, *European Journal of Operational Research*, *29*(3), 262–273.

Cieliebak, M., T. Erlebach, F. Hennecke, B. Weber, P. Widmayer, and B. A. C. Ed (2004), Scheduling With Release Times and Deadlines on A Minimum Number of Machines, in *Exploring New Frontiers of Theoretical Informatics; IFIP 18th World Computer Congress TC1 3rd International Conference on Theoretical Computer Science (TCS2004) 2227 August 2004 Toulouse, France*, vol. 155, pp. 209–222, Springer.

Coffman Jr, E. G., M. R. Garey, and D. S. Johnson (1996), Approximation algorithms for bin packing: a survey, in *Approximation algorithms for NP-hard problems*, pp. 46–93, PWS Publishing Co.

Davis, E. W., and G. E. Heidorn (1971), An algorithm for optimal project scheduling under multiple resource constraints, *Management Science*, *17*(12), B–803.

Desrosiers, J., and M. E. Lübbecke (2005), A primer in column generation, in *Column generation*, pp. 1–32, Springer.

Edis, E. B., and C. Oguz (2011), Parallel machine scheduling with additional resources: a lagrangian-based constraint programming approach, in *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pp. 92–98, Springer.

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

Finke, G., P. Lemaire, J. M. Proth, and M. Queyranne (2009), Minimizing the number of machines for minimum length schedules, *European Journal of Operational Research*, *199*(3), 702–705.

Garey, M. R., and D. S. Johnson (1975), Complexity results for multiprocessor scheduling under resource constraints, *SIAM Journal on Computing*, *4*(4), 397–411.

Gilmore, P. C., and R. E. Gomory (1961), A linear programming approach to the cutting-stock problem, *Operations research*, *9*(6), 849–859.

Gilmore, P. C., and R. E. Gomory (1963), A linear programming approach to the cutting stock problempart ii, *Operations research*, *11*(6), 863–888.

Hartmann, S., and R. Kolisch (2000), Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, *European Journal of Operational Research*, *127*(2), 394–407.

Heady, R. B., and Z. Zhu (1998), Minimizing the sum of job earliness and tardiness in a multimachine system, *International Journal of Production Research*, *36*(6), 1619–1632.

Kim, D. W., D. G. Na, and F. F. Chen (2003), Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective, *Robotics and Computer-Integrated Manufacturing*, *19*(1-2), 173–181.

Kolisch, R., and S. Hartmann (2006), Experimental investigation of heuristics for resource-constrained project scheduling: An update, *European journal of operational research*, *174*(1), 23–37.

Kravchenko, S. A., and F. Werner (2009), Minimizing the number of machines for scheduling jobs with equal processing times, *European Journal of Operational Research*, *199*(2), 595–600.

Kurz, M. E., and R. G. Askin (2001), Heuristic scheduling of parallel machines with sequence-dependent set-up times, *International Journal of Production Research*, *39*(16), 3747–3769.

Lavoie, S., M. Minoux, and E. Odier (1988), A new approach for crew pairing problems by column generation with an application to air transportation, *European Journal of Operational Research*, *35*(1), 45–58.

Lee, S., J. Turner, M. S. Daskin, T. Homem-De-Mello, and K. Smilowitz (2012), Improving fleet utilization for carriers by interval scheduling, *European Journal of Operational Research*, *218*(1), 261–269.

Limtanyakul, K., and U. Schwiegelshohn (2012), Improvements of Constraint Programming and Hybrid Methods for Scheduling of Tests on Vehicle Prototypes, *Constraints*, *17*(2), 172–203.

Lin, Y.-K., and F.-Y. Hsieh (2014), Unrelated parallel machine scheduling with setup times and ready times, *International Journal of Production Research*, *52*(4), 1200–1214.

Liu, C. (2013), A Hybrid Genetic Algorithm to Minimize Total Tardiness for Unrelated Parallel Machine Scheduling with Precedence Constraints, *Mathematical Problems in Engineering*, *2013*, 1–11.

Lübbecke, M. E., and J. Desrosiers (2005), Selected topics in column generation, *Operations Research*, *53*(6), 1007–1023.

Mehrotra, A., and M. A. Trick (1996), A column generation approach for graph coloring, *informs Journal on Computing*, *8*(4), 344–354.

Potts, C. N., and V. A. Strusevich (2009), Fifty years of scheduling: a survey of milestones, *Journal of the Operational Research Society*, *60*, S41–S68, doi: 10.1057/jors.2009.2.

Rabadi, G., R. Moraga, and A. Al-Salem (2006), Heuristics for the unrelated parallel machine scheduling problem with setup times, *Journal of Intelligent Manufacturing*, *17*, 85–97.

Radhakrishnan, S., and J. a. Ventura (2000), Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times, *International Journal of Production Research*, *38*(10), 2233–2252.

Reich, D., Y. Shi, M. Epelman, A. Cohn, E. Barnes, K. Arthurs, and E. Klampfl (2016), Scheduling crash tests at ford motor company, *Interfaces*, *46*(5), 409–423.

Ribeiro, C. C., and F. Soumis (1994), A column generation approach to the multiple-depot vehicle scheduling problem, *Operations research*, *42*(1), 41–52.

Savelsbergh, M. (1997), A branch-and-price algorithm for the generalized assignment problem, *Operations research*, *45*(6), 831–841.

Sawik, T. (2010), An integer programming approach to scheduling in a contaminated area, *Omega*, *38*(3), 179–191.

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

Sivrikaya-rifolu, F., and G. Ulusoy (1999), Parallel machine scheduling with earliness and tardiness penalties, *Computers and Operations Research*, *26*(8), 773–787.

Sterna, M. (2011), A survey of scheduling problems with late work criteria, *Omega*, *39*(2), 120–129.

Taillard, É. D. (1999), A heuristic column generation method for the heterogeneous fleet vrp, *RAIRO-Operations Research*, *33*(1), 1–14.

Yu, G., and G. Zhang (2009), Scheduling with a minimum number of machines, *Operations Research Letters*, *37*(2), 97–101, doi:10.1016/j.orl.2009.01.008.

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