Optimization tool dedicated to the validation process for the automotive industry

Auteur : Duquenne, Benjamin
Promoteur(s) : Duysinx, Pierre
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil mécanicien, à finalité spécialisée en technologies durables en automobile
Année académique : 2016-2017
URI/URL : http://hdl.handle.net/2268.2/3304

ATFE0013-1: MASTER THESIS

Academic promoter: PR. PIERRE DUYSINX
Industrial promoter: DIPL.-ING MARTIN LENZ

University of Liège

Faculty of Applied Sciences

Optimization Tool Dedicated to the Validation Process for the Automotive Industry

Graduation Studies conducted for obtaining the Master's degree in Mechanical Engineering by

DUQUENNE Benjamin

Year 2016 – 2017

Abstract

Validation is a key process in the automotive sector. It ensures that the developed product fulfills the customer-oriented requirements at the system level. However, due to the increasing flexibility and complexity in the design of modern vehicles, this process has become more and more constrained and requires a tool dedicated to its automation and optimization.

This project intended to offer the basis of this optimization tool. In order to do so, it investigated two major parts of the tool: the interface dedicated to the validation and the optimization algorithm supporting the scheduling of the validation tests. This double study was the skeleton of this work.

First, a study of the validation process and project management theory was done in order to characterize the problem. It belongs to the discrete and non-linear category. Then, the potentially suitable optimization algorithms were reviewed and two of them were selected for the scheduling problem: the *Genetic Algorithm* (GA) and the *Mixed Integer Distributed Ant Colony Optimization* (MIDACO).

Mockups of the future interface were created, following the characterization of the validation process. Concerning the scheduling optimization, neither algorithm handled an increase of degree of freedom well, but MIDACO revealed to scale and to sample the solution space much better than GA did. Additionally, GA was intrinsically much slower and MIDACO provided fully reproducible results. The second algorithm was also more suitable for dynamic variations of the schedule.

In conclusion, the studied scheduling model can be used for further developments of a whole tool dedicated to the characterization, automation and optimization of the validation process. From the two studied algorithms, the MIDACO solver is the most suitable one and can be used for a more complex validation model, closer to the reality of operations or responding to the validation need of future complex developments.

Acknowledgements

This thesis is the accomplishment of five years of hard work in order to become a lifelong dream, engineer, symbol of innovation and added-value.

First, I would like to thank both academic institutions that welcomed me during my engineering education, the Royal Military Academy for my bachelor and especially the University of Liège which was a strong pillar during my Master degree to get me ready to face the professional world.

Then, I would like to thank in particular my academic promoter, Professor Pierre DUYSINX, for his precious help during this thesis and especially for his support during all the steps of this master. I want to thank his assistant as well, Simon Bauduin, for his advice all along this final master year. I would like to thank all the other professors for their help along this journey, such as Doctor Maud Bay for her help in the characterization of this project.

Moreover, I have a special attention for Diplom-Ingenieur Martin LENZ, my industrial promoter, the VKA and FEV which welcomed me. This project is the consequence of their impulsion, being a whole part of a challenging industrial project, in which I am honoured to be implied. I also thank Martin LENZ for his help and his attention for the success of this project.

Finally, I want to thank my relatives. Nothing would have been possible without their support. I thank them for their unconditional help during the redaction and the courage they gave me during the five previous years.

Contents

List of Acronyms

ACO Ant Colony Algorithm

AI Artificial Intelligence

B&B Branch & Bound

CU Cost Unit

DA Dragonfly Algorithm

DVP Design Validation Plan

EA Evolutionary Algorithm

FMEA Failure Mode and Effect Analysis

FP Functional programming

GA Genetic Algorithm

GUI Graphic User Interface

HiL Hardware-in-the-Loop

IoT Internet of Things

JSS Job-Shop Scheduling

MiL Model-in-the-Loop

MINLP Mixed Integer Non-Linear Programming

MMA Method of Moving Asymptotes

OEM Original Equipment Manufacturer

PDF Probability Density Function

PERT Program Evaluation and Review Technique

PM Project Management

PSO Particulate Swarm Optimization

PTTB Powertrain Test Bench

PWT Powertrain

QMP Quality Management Principle

QMS Quality Management System

QM Quality Management

SA Simulated Annealing

SE Systems Engineering

SiL Software-in-the-Loop

SOC State of Charge

TS Time Slot

UML Unified Modeling Language

WOT Wide Open Throttle

List of Figures

List of Tables

Introduction

Bergmann & Paulweber (2014) have described the development of modern vehicles in the last few years as strongly evolving. In particular, the basic powertrain architecture has begun to change substantially for the first time in a century. Modern, and especially hybrid-powered vehicles require much more flexibility and complexity in the development and design than ever before. In addition to this, the reduction of the development time leads to an increasing need of an efficient validation process.

A good validation lies in the good scheduling of its validation tests, ensuring simultaneously the highest quality of validation at the lowest cost and/or duration. The purpose of this work is to offer the basis of a scheduling optimization tool destined for the validation process in the automotive industry. On the one hand, this tool will be composed of an interface where the different tests to schedule will be defined. On the other hand, an automatic scheduling algorithm will support the validation by providing a (near-)optimal planning. These objectives can be formulated in the following questions:

How is the validation process characterized?
How are defined the tests composing the validation process?
What information is necessary to define a validation test?
What is the best way to collect this information?
Under what form can the tests be scheduled?
How can the scheduling problem be modelled mathematically as a problem to solve?
What are the criteria to assess the quality of an optimization method?
What is the best algorithm to solve a scheduling problem?

In order to answer these questions, the characterization of the validation process supports this work by providing the information needed to define tests. The methodology is divided into two main parts. First, a way to collect the data presented in the problem characterization is proposed via an interface, respecting a given *client perspective*. Then, the mathematical model of the validation scheduling is expressed as a problem to minimize, with variables, objective functions and constraints. It leads to a discussion about the most suitable methods, from which the *Genetic Algorithm* (GA) and *Ant Colony Optimization* (ACO) will emerge.

This work is structured in five parts. The addressed validation scheduling problem is first completely characterized. Then, a literature review gives the necessary knowledge about validation and project management, followed by the state-of-the-art of optimization algorithms for complex problems. The resolution methodology follows, with a special interest for the interface of the tool, the scheduling mathematical model and the choice of resolution methods. The results of the scheduling optimization is discussed in function of the tested optimization algorithm and the most suitable one, between GA and ACO, for this application is highlighted.

1. Context and Project Description

The validation process in the automotive industry has always been scheduled following the usual methods used in *Project Management* (PM) like the famous PERT or CPM methods. However, they only provide passive project scheduling, tracking and reporting aids, so do commercial PM tools, whose computational capabilities are unsatisfying. Additionally, their user must do the scheduling "by hand" without confidence about the global optimization and the result is generally based on the manager experience.

First, this chapter describes the industrial context in which the validation takes place. It leads to a discussion about the purpose of this work and the necessity of a scheduling optimization tool. Then, a general formulation of the scheduling problem is given, following the study of Bartschi Wall (1996) on this subject and the literature review about validation and PM of the next chapter. Finally, the given scheduling problem is deeply characterized in order to provide a working basis for the rest of this work.

1.1 Context

Validation in the automotive industry can be defined as the determination of the degree of matching between the requirements, defined during the conception, and the actual vehicle. The validation process is naturally impacted by the increasing flexibility and complexity in the conception of modern vehicles. Additionally, it is not only performed on highly standardized test facilities but also on parallel facilities, facilities in networks and real world driving tests. The efficiency of the validation process is then measured by its ability to offer a high-quality validation in a short amount of time and at a reasonable cost.

To meet this need, an optimization of the resource planning is required in order to make the most efficient use of the test facilities. It will lead to a *Design Validation Plan* (DVP) that represents the best compromise for all stakeholders and all concerned business units.

Moreover, this optimized resource planning must still ensure a good execution of the validation and the verification of key aspects of a vehicle. In the past years, the price of a wrong validation revealed to be strongly higher than the price of the validation itself, if well executed. The automotive industry having to deal with long-term projects but restricted costs, this aspect is particularly important.

In this context, the goal of this project will be to offer a tool supporting the validation process of vehicles. On the one hand, the tool will consist of a *Graphical User Interface* (GUI) that will gather the data needed to perform the validation under the form of tests. On the other hand, the tool will offer a proactive scheduling optimization of those tests in order to benefit, globally, from the most convenient and powerful way to validate a vehicle.

1.2 General Formulation of the Scheduling Problem

Bartschi Wall (1996) proposes a generic definition of the scheduling problem which will be used as a basis for the rest of this work.

First, there is a distinction between *planning* and *scheduling* principles. Planning deals with the elaboration of the process (the validation process in this case). Therefore, it defines constraints to respect and objectives to fulfill. Then, the scheduling assigns some resources to activities, and vice versa. It also attributes one or several cost and duration values to all activities, with the best time allocation. This last part will be the main concern of this project and the next section will determine which part of the validation refers to planning issues (creation of the validation tests in this case) and scheduling issues (optimal scheduling of the tests).

In general, a scheduling problem is composed of:

- Activities to execute;
- Resources to perform them;
- Constraints to respect;
- Objectives to minimize.

The defined objectives must reflect the optimal schedule of activities. Some characteristics are inherent to scheduling problems in general. The relevant ones for this case are summarized hereunder:

- Resources can be temporally restricted;
- Some tasks have precedence constraints;
- A task can be interrupted or no interruption may be allowed;
- A task has different characteristics if different resources are allocated to it;
- The schedule can face dynamic variations when the resources, tasks or objectives change over the time.

In the next section, one will make a complete analogy with the planning and scheduling concepts. Moreover, a complete definition of the problem to be solved in this work will be given.

1.3 Characterization of the Given Problem

Characterizing the problem to solve is a significant part of this work. Therefore, the current validation process must be presented in order to give the frame of this work. Then, the complete scheduling problem addressed in this project is given, without getting into mathematical concerns. The presented problem must remain precise while keeping the level of flexibility as high as possible. The next goals will be to translate the problem in an optimization problem to solve, after a complete literature review and the state-of-the-art on this subject.

1.3.1 Characterization of the Validation Process

Following the previous definitions of scheduling and planning, the validation process itself can be referred to planning. Therefore, it will lead to the *Design Validation Plan* (DVP), whose concept will be deeply studied in Section 2.1.

The validation process, succinctly defined in Fig. 1.1, is used as a quality measurement of given requirements. Therefore, it must contain very precise indications of what is needed to validate a requirement of a given vehicle. Each requirement contains four distinct characteristics: the requirement description, the test cases (defining the test procedures), the validation type and the concerned prototype phase. An example of interface displaying the requirements information is given in Appendix B.1.1.

Requirements

Each requirement is clustered into categories of requirements like "performances", "OBD", "Energy", "Marketing", etc. and receives the following information:

- A *REQU SPEC Number* (Requirement Specific Number) such as "REQ_0000xx", which is used to characterize a given requirement in all manipulations of the validation process. It can be considered as a primary key in analogy with database management;
- A *name* written using the *Unified Modeling Language* (UML) such as "Perf_LevelRoad-0-30mphTime" for example;
- A *description* like the following example: "On Level road 0-30 mph should be attained in less than or equal to x sec with the Load equal to curb + 300 lbs and 100% to 0% SOC in Temp range 30° F to 100°F", where SOC stands for the *State of Charge* of a battery pack.

Test Cases

The test cases describe how to proceed the validation of the requirements. They contain:

- A *Description* of the test itself. For the previous example, it is "WOT acceleration (kick-down) in most sportive driving mode (with ICE running) from standstill to 30 mph" where *WOT* stands for *Wide Open Throttle*;
- The *Test conditions*. It can be considerations like level road, ambient temperature, no humidity, full SOC, preconditioned car, etc. ;
- An *Acceptance criteria* like "Acceleration time <= x sec" for the previously used example. It is linked to an equation and a signal measured during the test.

Prototype Phases

A requirement can be validated at multiple phases of the prototyping. Vehicle prototyping is divided in four phases, going from specifications to a prototype produced in small series. The different phases are presented in detail in Appendix A.1.1.

Validation Type

The validation type indicates the *facilities* on which the tests will be performed to validate a given requirement. This requirement can be validated multiple times, at multiple prototype phases and on different facilities. For example, a powertrain requirement can be virtually assessed by simulation in

Phase A and then validated on a test bench in Phase B. Following the operation management definition of Bartschi Wall (1996), facilities can be seen as resources in this particular case.

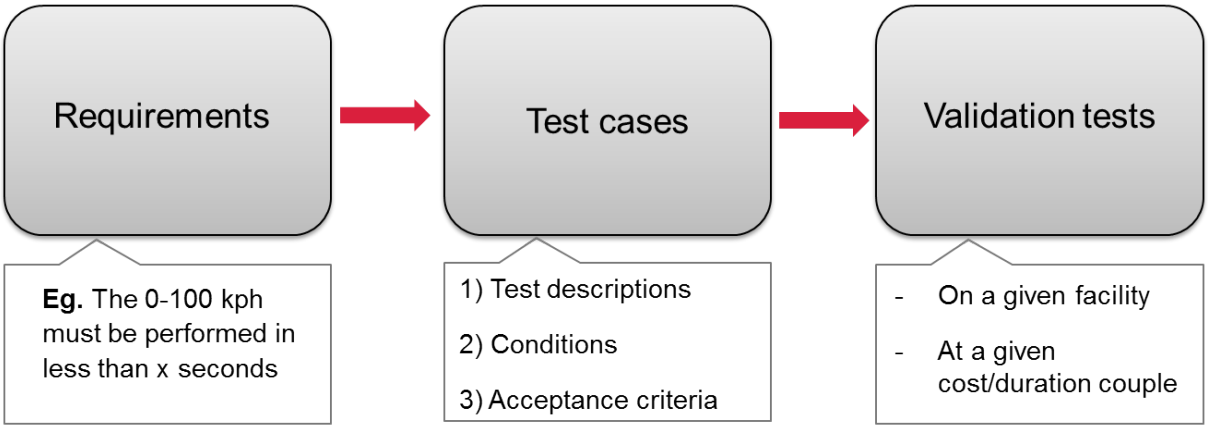


Figure 1.1: Summary of the validation definition.

The next section will be dedicated to the characterization of the scheduling problem linked to the validation process here described.

1.3.2 Basic Validation Scheduling Problem

This project aims to offer a solution for the optimization of the validation process. In practice, it consists in an automatic scheduling tool, where *tests* (referred as activities in the operation management theory) have to be scheduled in the (near-)optimal way in order to benefit from the best DVP. It implies a high quality of validation while minimizing the total cost and/or duration of the validation process under specific constraints. The goal of this section is to clearly identify the needs of the tool, while always keeping the highest level of flexibility. The next chapters will propose a solution to collect the necessary data and to solve the presented scheduling problem, summarized in Fig. 1.2.

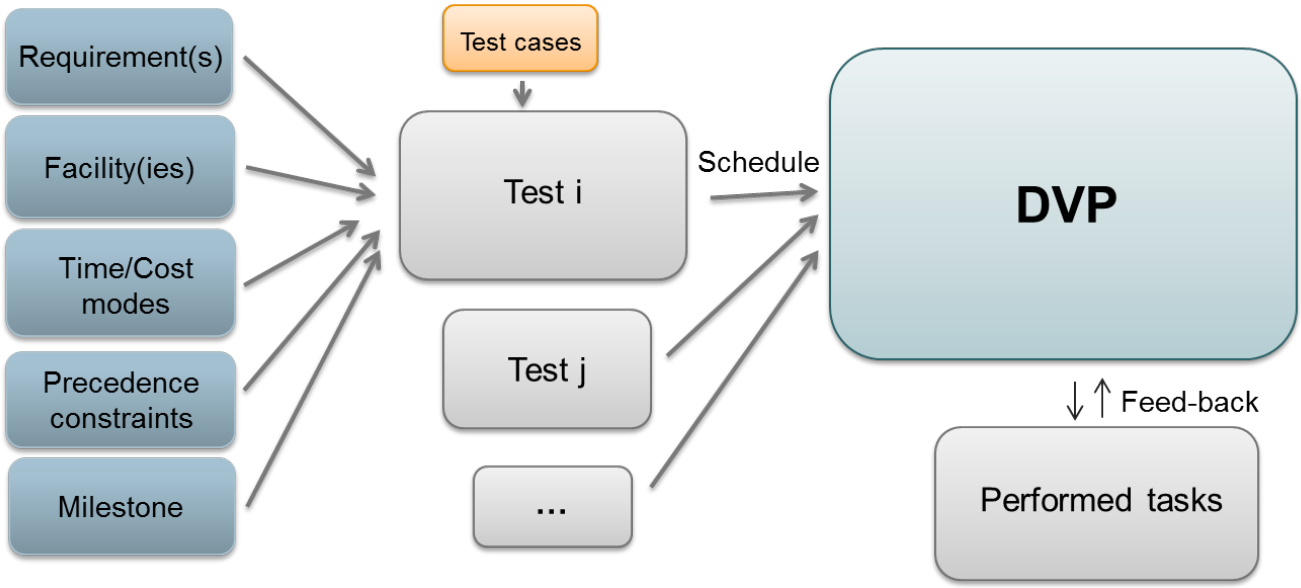


Figure 1.2: Summary of the scheduling optimization problem.

In this work, it is proposed to focus on tests, which can be seen as tasks to perform, rather than requirements. In this way, it will be possible to validate multiple requirements, a whole category of requirements or even requirements from different categories in only one test. A given test corresponds to a given *test facility* (simulation, test bench, chassis dynamometer, etc.). Naturally, only one facility at the time can be associated to a test since it cannot be performed on multiple facilities simultaneously. A test (cfr. the analogy with projects) corresponds to a single period of the schedule.

Then, the tests must be characterized by *time* and *cost* couples, exactly like projects as defined in Section 2.2.2. One assumes at this point that tests are defined by discrete time and cost values, estimated directly at the facility level. Each facility is described by as many cost/time integer values as resources combinations exist. For example, an acceleration test on a chassis dynamometer costs 500€ if the test is performed in 1 hour, 200€ if it is performed in 1.5 hours or 100€ in 2 hours, as shown in Fig. 1.3, depending on the allocated resources. This concept is called *crashing*.

The principle of having multiple time/cost combinations, called *modes*, will be deeply studied in Section 2.1. Only the integer values are available since they correspond to allocated resources (continuous values are not available). In this way, one does not take the resource allocation into account via the tool, but directly at the facility level.

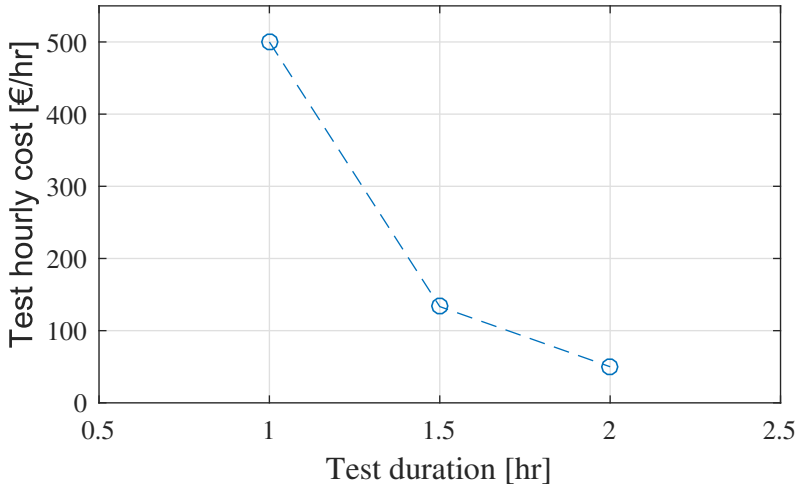


Figure 1.3: Concept of *crashing*, a facility hourly cost depends on the allocation of resources.

Moreover, the schedule must take two kinds of *constraints* into account. First, there are constraints on the tests, called *precedence constraints*, when some tests must be performed prior to others. For example, for a given prototype phase, the battery pack validation could have to be performed prior to validating the electric motor. Then, there are *availability constraints* on the facilities due to maintenance purposes, for example. Next to that, when a test is occurring on a facility, this facility is not available for another test on that period, even if there is no correlation between the two tests. Other constraints concern the milestone periods, described later in this chapter.

Finally, the built schedule must be able to deal with *dynamic variations* via feed-backs from the performed test. The schedule will be updated with the encountered time and cost during the test to constantly have the most optimized schedule. Additionally, a planned test can be cancelled while being validated if it appears to fail. In this case, the tool must propose the optimal solution, on the one hand, to benefit from the free remaining time. On the other hand, this cancelled test must be updated and scheduled in the best way. This type of problem is called an *online problem*, where there are ongoing changes in the activities, in opposition to offline problems, where the planned activities will not experience changes.

1.3.3 Milestone, Global Due Date and Maximum Budget

As for any other projects, the validation process is defined by either a global *due date* (or deadline), a maximum *budget*, or a mix of both when none of them is preferred, but the global result must be satisfying in every direction. the user must set his preference: minimizing the cost, the duration or a mix of both, defined as a *multi-objective problem*. In the case of a cost minimization, a global due date is defined, following the retro-planning principle. A maximum budget will be set when the duration is minimized.

Additionally, the tool can take as inputs a due date and a start date for each test to perform. One can see this characteristic as an extension to the precedence constraints. Instead of (or in addition to) setting precedence constraints on a given test, the user will set intermediary due dates, in order to deal with constraints on sub-issues. Moreover, the validation test could be planned during specific periods, such as seasons, defined by a starting and an ending time. This method is the well-known *milestone* method, described more precisely in Section 2.2.5.

Adaptive Constraints

However, the importance allocated to the global due date or intermediary due dates can vary. The last consideration is then about the level of constraint. On the one hand, some constraints, called *hard constraints*, must necessarily be respected, such as precedences or the due date of some critical tests. In this case, the solution area is restricted for solutions non-respecting these constraints. On the other hand, certain constraints could be flexible and then be respected if possible, with an additional cost if not respected. They are called *soft constraints* or *relaxed constraints* and result in a wider but more costly solution space. This concept is also known as *constraints relaxation*.

1.3.4 Facility Allocation Problem

In the previous problem characterization, facilities were seen as resources following the Bartschi Wall (1996) definition. The case of a unique facility allocated to each test was presented so no allocation of resources was needed. However, two points must be discussed:

- There is only one type of facility suitable for each test;
- There is only one facility per category (of facilities).

Actually, there can be redundancies in facilities. There are multiple facilities of a same family, like test benches, that are described by same or different time and cost values, following the size, age, etc. Several facilities can thus be suitable for a single test. The solution must then take them into account so that different tests having to be tested on a same kind of facility can now be performed in parallel. Each facility remains defined by its own unavailability constraints.

Then, a test schedule can be distributed among different facilities. If more than one facility is suitable for it, the highest level of optimization can be achieved by allowing a single test on multiple facilities. For example, a powertrain feature can be either validated on a brand new test bench or on an old one. It will result in higher costs but less time in the first case. Both test benches will have different availability periods. If the first the bench must be maintained in the middle of the test schedule, the validation process can switch to the second free facility. As a reminder, a less advantageous solution at the test level can lead to a more optimized solution at the DVP level.

1.3.5 Complete Test Definition

The complete definition of the validation tests is illustrated in Fig. 1.4, including the unavailability of facilities, the facility allocation problem and the adaptive constraints.

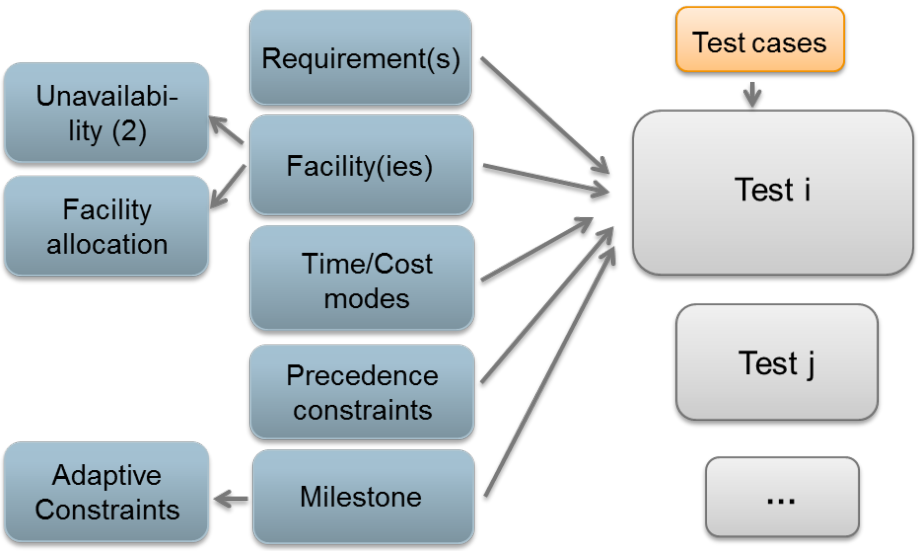


Figure 1.4: Summary of the complete test definition.

1.3.6 Desired Result

The optimal scheduled DVP is supported by a clear disposition of all tests illustrating their temporality and the facility supporting them. The most suitable schedule representation is the famous Gantt Chart, presented in Fig. 1.5. In the best case, the different tests would be clearly displayed, following a logical temporal continuity and the relevant information should be available easily.

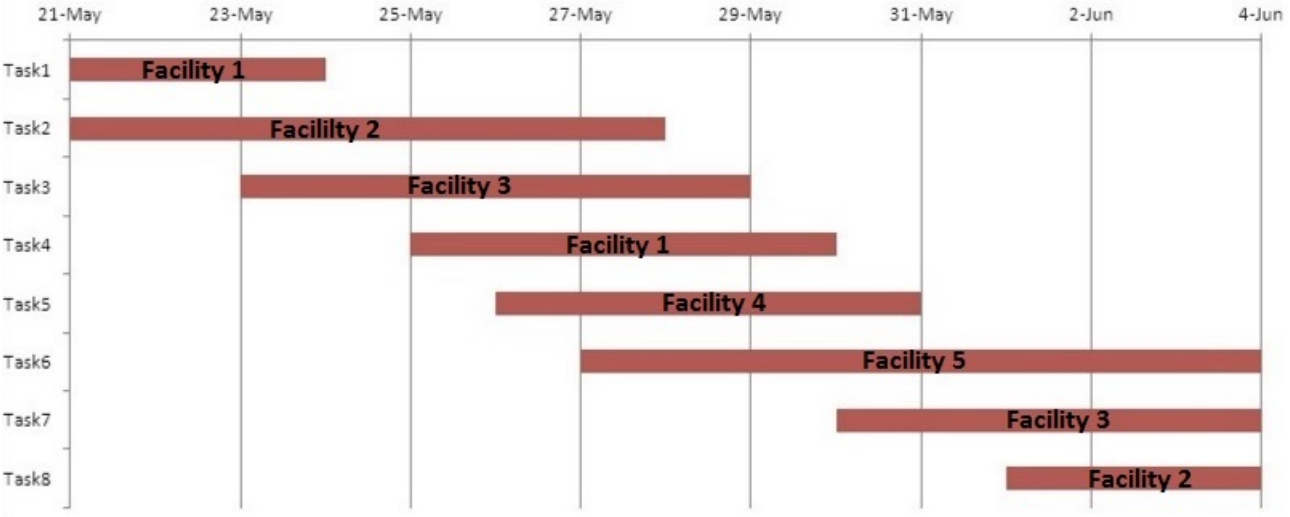


Figure 1.5: The well-known Gantt chart is used to display the validation schedule, where tasks can be seen as tests and facilities can be mentioned.

2. Literature Review and State of the Art

This chapter is dedicated to the theory behind the problem of the validation process optimization. It is divided into three distinct parts:

- The theory behind the validation process. In order to propose an efficient tool dedicated to the *validation plan*, whose role will be defined, it is necessary to clearly understand this concept and its conditions. Therefore, a global overview of it will be given;
- The theory of *Project Management*. The scheduling optimization being a major part of this field, this concept will be succinctly defined in order to highlight all relevant principles for this work;
- Scheduling optimization methods. In order to provide an acceptable DVP, different optimization algorithms can be implemented. The state-of-the-art of optimization algorithms for complex problems will be given, leading to a discussion on their suitability for this problem. Previous works studied this problem and will support this part.

After an exhaustive study of all mathematical models, Chapter 3 will determine the way of applying them to the validation process in the automotive industry.

2.1 Theory of Validation in Systems Engineering

Systems Engineering is the branch of Engineering dealing with complex systems as a whole, providing a multidisciplinary approach. Modern vehicles can be seen as systems since lots of embedded components act together in a constantly more complex but efficient union. In this section, the generic validation process will be deeply defined via the Systems Engineering approach. Two concepts are essential for the rest of this work and will be deeply explained: the *V-Model* and the *Design Validation Plan*. The rest of this section will be dedicated to verification and validation techniques and their correspondence with the validation in the automotive industry.

Systems Engineering Definition

Systems Engineering (SE), following Debbabi et al. (2010), deals with systems through their entire cycle, from their conception to their implementation. SE is a multidisciplinary discipline, considering systems as a whole, taking all aspects and components into account. It matches the increasing complexity of products and consists in a perfect approach for the automotive industry, and for hybrid vehicles in particular.

Every SE is specific. However, the combination software-hardware is a domain that must often be validated and provides a good theoretical representation of modern vehicles, embedding a lot of complex high-technology devices. This domain will be the support of this section.

Difference between *Verification* and *Validation*

The *Defense Modeling and Simulation Organization* (DMSO) provides the following definitions for the validation and verification processes.

- *Verification*: It is "the process of determining that a model implementation and its associated data accurately represent the conceptual description of the developers and specifications". The specifications refer to the component level, level that is less relevant for this work.

  "Are we building the system right?"

- *Validation*: It is "the process of determining the degree to which a model and its associated data provide an accurate representation of the real world from the perspective of the intended use of the model". The model is directly associated with the system level, the vehicle prototype in this case.

  "Are we building the right system?"

The reader must be aware of the difference in order to determine the level of abstraction needed for the development of this thesis.

2.1.1 V-Model

Bergmann & Paulweber (2014) have provided a complete explanation on how modern vehicles development has strongly evolved in the last few years. In particular, the basic powertrain architecture has begun to change substantially for the first time in a hundred years. Modern, and especially hybrid-powered, vehicles require much more flexibility in the development and design than ever before.

Due to this increasing complexity, their development is based on a *Total Validation* process rather than based on the usual architecture. To do so, a *V-Model* is used as a frame for this new type of development, as illustrated in Fig. 2.1. Initially created for software development, its use has been extended to all applications combining a high level of flexibility and validation, such as the modern automotive sector.

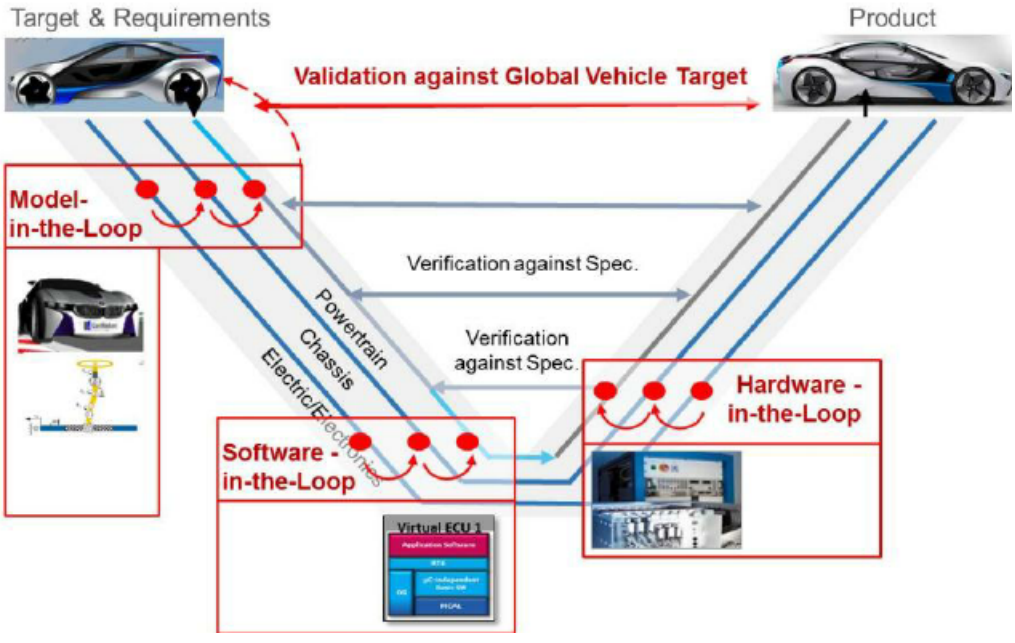


Figure 2.1: V-model - Total validation of modern vehicles, Bergmann & Paulweber (2014).

For the specific hybrid vehicle conception, the V-model is a top-down and then bottom-up process in terms of abstraction.

It begins with the *Requirements*, so does the validation plan. Then, the vehicle is modeled following different topologies to satisfy the requirements. Using *Model-in-the-Loop*, the model is adjusted until the requirements are satisfied, at the model level. The level of abstraction continues to decrease and the powertrains (ICE and/or electric), the chassis and the electronics are simulated thanks to a *Software-in-the-loop* (SiL) process, consisting in a loop at the software level to adjust the parameters to meet the specifications. It also ensures that the specifications have not been altered by the transcription into code. This concept will be defined in depth further.

Then, the level of abstraction increases by the simulation of given conditions at the hardware/vehicle level called the *Hardware-in-the-Loop* (HiL) technique. It consists in *Tests* (described by *Test cases*) to perform thanks to test benches, *Real World Driving Cycles* (RWDC), chassis dynamometers, etc. Finally, each step can be undertaken from the moment when the previous one is completed, what is the main drawback of this method.

At the end, the highest level of abstraction is reached by the prototype or commercialized vehicle, depending on the prototyping phase, and the validation assessment is done comparing the original requirements. During the whole validation process, the verification at the component level is performed in order to satisfy all the technical specifications.

2.1.2 Design Validation Plan

The *Design Validation Plan* (DVP) is basically the planning, thus with a temporal perspective, on which is based the validation of the design, following the requirements set at the beginning of the V-model. In practical terms, it is the output of the optimization tool described in Section 1.3. Following the scheduling definition of Section 1.2, the DVP can be seen as a mix of planning, describing how to perform the tests, and scheduling, providing a schedule of the validation tests. Fig. 2.2 provides a theoretical approach of a generic (basic) validation plan with an illustrating example encountered in the automotive industry.

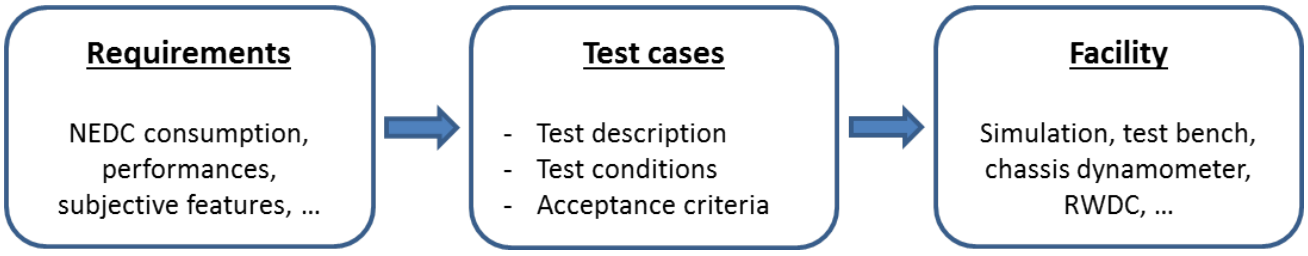


Figure 2.2: Theoretical approach of a generic (basic) DVP.

All DVPs consist in a list of requirements to test, test cases describing how to validate each requirement and the test facility used for validation. Moreover, the presented concept DVP is not only valid for automotive design.

However, this plan is not sufficient. The DVP must specify other features like the schedule, but also the cost and the concerned prototype phase (see Appendix A.1.1), as illustrated in Table 2.1 and Table 2.2. Although the DVP is a generic process, this specific example illustrate the kind of DVP encountered in the automotive industry, with three types of test facility (simulation, chassis dynamometer and RWDC). The tables are not closed since this plan is supposed to be extended to the end of the prototyping phase.

Requirement	**Testability [-]**			**Cost [€/hr]**			**Test Duration [hr]**		
	Sim.	Dyno	RWDC	Sim.	Dyno	RWDC	Sim.	Dyno	RWDC
Requirement 1	x	x		200	500		2	4	
Requirement 2	x	x		150	300		3	6	
Requirement 3	x	x	x	150	250	800	5	10	8
...									

Table 2.1: Example of an advanced DVP illustrating the validation costs and duration.

Table 2.2 represents an example of the scheduling of the validation plan, resulting from the optimization problem expressed in Section 1.3. The global output of the optimization tool should take a similar form.

Requirement	**Prototype Phase**											
	Proto A				Proto B				Proto C			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Requirement 1	Sim.				Dyno							
Requirement 2		Sim.							Dyno			
Requirement 3				Sim.			RWDC				Dyno	
...												

Table 2.2: Example of an advanced DVP illustrating the validation schedule, where Q stands for quarter.

2.1.3 Verification and Validation Techniques

There exist three *Verification and Validation* (V&V) techniques in SE (Debbabi et al., 2010):

- *Model checking*, which is suited for software and hardware applications;
- *Static Analysis*, used for software before either testing or model checking. It is a good way to provide lean programs faster to verify;
- *Empirical Methods*. In order to use this technique for validation purposes, one needs to possess *software metrics*. Software metrics is a way to measure that a software has a given property for the customer. It is not always a measurement in itself but provides a way of assessing the concerned property. Those metrics will be an important part of the future of automobile, especially with the breakthrough of connected platforms. It is already the case for the truck sector where software metrics is a major subject of study. The difficulty of metrics is to define the relevant ones for the customer.

  The metrics will be used as "signals" for the tool, with a given equation of the type $metrics <= given\ value$. If the return value is true, the given requirement is validated.

V&V techniques can be classified into four categories:

- *Informal* techniques, based on human interpretation and subjectivity, such as audit, inspection or review;
- *Static* techniques, based on the static implementation, without executing the model with a computer. Their goal is to verify the structure of the models and must be applied in addition to dynamic techniques;
- *Dynamic* techniques, based on the code execution by a machine. The outputs and the model are verified during the execution. The major difficulty of those techniques is the additional code needed to collect this data. It is the type used for the concerned validation process;
- *Formal* techniques, based on the mathematical principles and proofs. It includes model checking and theorem proving.

SE, previously based on traditional documents, is now mainly based on new model conception. The main advantage of modeling is the support for early V&V techniques, as illustrated along this work.

Debbabi et al. have elaborated the validation of SE through the five procedures explained hereunder.

First, the *Inspection* is a procedure where the design is compared with the standards. It fully depends on human factors such as organization, planning or data preparation. It relies on documented procedures and becomes more and more inappropriate with increasing complexity.

Second, the *Testing* procedure verifies the operability, supportability and performance capability to a given expectation. The drawback is that errors are either discovered lately, or even remain invisible for the testing person.

Third, the *Simulation* is considered. It is currently the best solution for V&V techniques, in order to decrease the testing costs and drawbacks. However, the number of simulation cycles is rapidly increasing due to the increasing complexity.

Fourth, *Reference Model Equivalence Checking* is used as a verification technique. It makes possible to compare a behavioral model with a reference one, the golden model. It is not sufficient in itself since it does not provide any insurance concerning invisible errors.

Finally, one has *Theorem Proving*, which verifies the mathematical principles behind the code, as explained before.

To conclude, the presented principles and methods behind the V&V techniques will be used during the validation process in order to provide a reliable result for the concerned vehicle, which can be considered as a system following the generic definition. In particular, model checking and empirical methods, with the use of software metrics, are relevant in this case.

Validation and Verification processes are key aspects of a global *Quality Management System* (QMS) and are valid under certain norms, depending on the targeted market. However, these notions are not necessary for this work and are available in Appendix A.1.2 and A.1.3.

2.2 Project Management Theory

The validation process can be seen as a *Project Management* (PM) problem. The goal of this section will be, first, to define exactly the involved concepts and the optimization problem to address. The usual methods used in schedule development, PERT and CPM will be presented in order to highlight their limits. Then, modelling techniques will be succinctly discussed, as well as the milestone method. The end of this chapter will deal with optimization algorithms for complex problems potentially suitable for this scheduling applications.

2.2.1 Project Management Definition

The *Project Management Body Of Knowledge* (PMBOK) from the Project Management Institute is the reference book for the PM problem definition and provides good practices in this field.

Definition of a Project

Following the PMBOK, a project is :

"A temporary endeavor undertaken to create a unique product, service, or result. The temporary nature of projects indicates that **a project has a definite beginning and end**. The end is reached when the objectives of the project have been achieved or when the project is terminated because its objectives will not or cannot be met, or when the need for the project no longer exists.[...] **Every project creates a unique product, service, or result**.[...] An ongoing work effort is generally a repetitive process that follows an organization's existing procedures. In contrast, because of the unique nature of projects, **there may be uncertainties** or differences in the products, services, or results that the project creates. Project activities can be new to members of a project team, which may necessitate **more dedicated planning than other routine work**. In addition, **projects are undertaken at all organizational levels**. A project can involve a single individual or multiple individuals, a single organizational unit, or multiple organizational units from multiple organizations."

The uniqueness, the temporary nature, the multi-level aspect and the planning necessity of projects are clear evidences of the analogy with the validation process, as described in Section 1.1. Validation corresponds to a project following the PMBOK definition, while tests correspond to activities following Bartschi Wall (1996).

Definition of Project Management

Following the PMBOK, project management can be concisely defined as "the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements." Moder et al. (1983) have also provided a six-step methodology to define project management:

1. Project Planning,
2. Time and Resource Estimation,
3. Basic Scheduling,
4. Time and Cost Trade-offs,
5. Resource Allocation
6. Control (leading to the dynamic variations discussed in Section 1.1).

All these six steps must be handled by the tool described in Section 1.1. In Chapter 3, the analogy with the given problem will be clearly highlighted in order to create a mathematical model of it. The most significant aspects are the *requirements*, which appear in both validation and project management processes.

2.2.2 Optimization Problem in Project Management

The PM optimization problem statement is the following: when does each activity or sub-part of a project have to be operated? Following which time and cost strategy? The resulting schedule must be the best compromise between the *time*, *cost* and *constraints*. Swanson (1973) has provided an intuitive model of the PM problem, which is the subject of this section.

Time and Cost

Swanson (1973) has defined activities by their *normal* and *crash* time or cost. The normal time defines the duration of an activity at the smallest cost. Crashing the activity results in a smaller duration but at higher cost, as illustrated in Fig. 2.3. In this case, a basic linear model is assumed, with a linear approximation between the normal and the crashing cost.

However, this linear model is not always right. Most of the time, the possible values are either discrete, not linear, or a combination of multiple models, depending on the project conditions. The objective of Fig. 2.3 is only to give a perspective of the time versus cost problem.

The *schedule compression*, i.e., the fact of performing projects in a smaller time but at higher costs, here presented can be done by two methods: *crashing* or *fast tracking*. The compression done by crashing consists in adding resources to shorten the project duration. Fast tracking is a way to perform some sequences of a same project in parallel. The scheduling problem presented in Chapter 1 implies both methods, crashing at the test level and fast tracking at the global validation level.

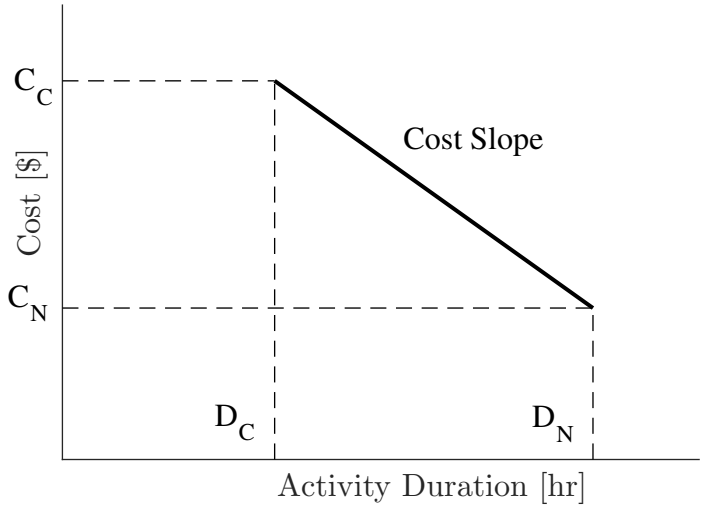


Figure 2.3: Schedule compression by *crashing*: the normal *N* or crashing *C* time and cost depend on the allocated resources.

Finally, there are overhead costs, corresponding to additional costs due to projects performed out of the deadline or needing more resources than they should. The optimal schedule must take them into account if they are defined at the beginning. For example, it could be more advantageous to complete a project out of the deadline if the resulting costs are lower than crashing costs.

Constraints

Apart from time and cost, the projects can be limited by other constraints. An activity can have a necessary prerequisite, called *precedence*, to complete prior to starting the concerned activity. Automotive validation tests for instance, depends on the concerned car part. Certain parts must be validated

prior to other ones when they a significant impact on their operation. Other constraints (other than purely temporal ones) can be found, depending on the application and the resources, and must be identified before the project definition and scheduling.

2.2.3 Schedule Development

Defining the schedule is one of the main concerns of project management. Two major methods have been developed in the late fifties: PERT and CPM. They are references for PM and are still often used. The book of Moder et al. (1983) on this subject will be the main reference for this part.

Both methods include (Chang et al., 2001):

1. A technique for tasks and requirements description;
2. A method to specify the relationships between tasks;
3. A description of the available resources.

The difference between scheduling and planning, presented in the DVP section, is that a plan defines all requirements that must be done with corresponding conditions and constraints, while scheduling specifies the way and the time of completion.

PERT Method

The *Program Evaluation and Review Technique* (PERT) was developed by the Navy in 1958 when missiles productions involved so many resources and constraints that an efficient method needed to be built. It was also an opportunity to reduce the delays and costs, respectively 40% and 70%, higher on average than the most optimistic expectation (Moder et Al., 1983).

This technique is suitable for both planning and control of time, thanks to a probabilistic model, whose variable is an *event*, defined by two end time predictions illustrated in Fig. 2.4: optimistic and pessimistic. It is then suitable for high precision and/or unpredictable projects like R&D. The path whose duration is the longest is called the *critical path* and it also corresponds to the project duration. This path is characterized by equal pessimistic and optimistic time predictions. However, crashing activity is not possible with PERT.

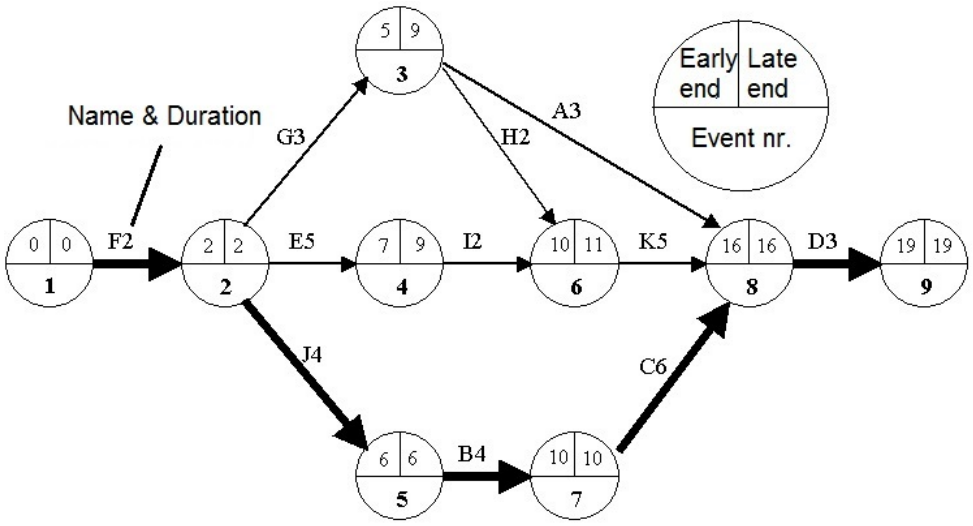


Figure 2.4: Example of a PERT diagram. The critical path and activities are represented by the thick line (Itel, 2015).

Critical Path Method

Based on the critical path principle, the *Critical Path Method* (CPM) is a statistical technique for well defined, predictable and repetitive activities like construction or production, permitting to control cost and time simultaneously. Instead of an event, the variable is an *activity*. The model is deterministic, based on only one time, but crashing is now possible. The CPM is illustrated in Fig. 2.5.

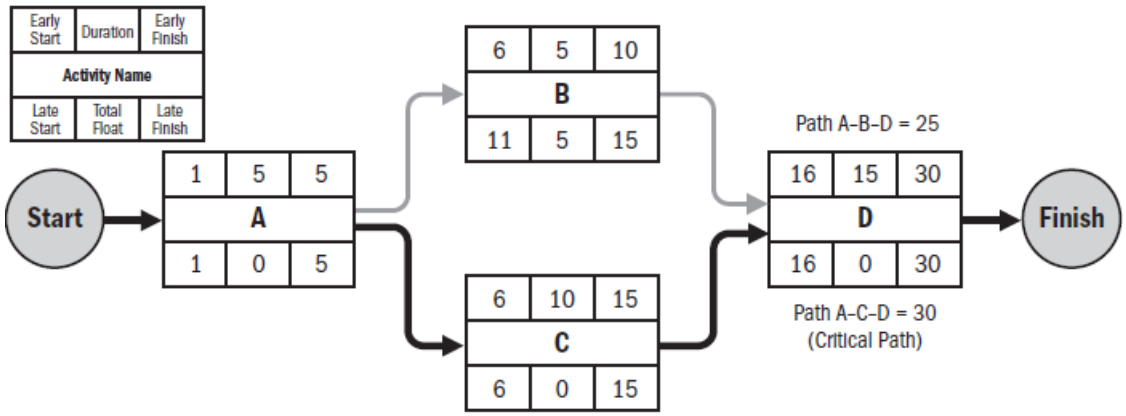


Figure 2.5: Example of a CPM diagram (*PMBOK*, 2013).

CPM extension: Critical Chain

The *Critical Chain Method* (CCM) is an extension of the CPM, which has been created to deal with uncertainties. In order to do so, it adds duration *buffers*, as shown in Fig. 2.6, which are statistically determined. At the end of the critical chain, one has a project buffer to protect the target date. There are also feeding[1] buffer on auxiliary chains, protecting them from slipping on the critical chain. The main advantage of this method is to only manage remaining buffers, whose size depends on the uncertainty.

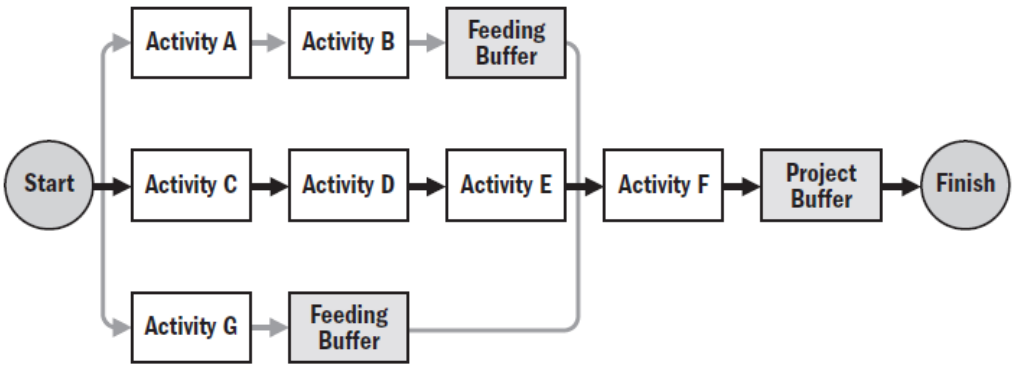


Figure 2.6: Example of a CCM diagram (*PMBOK*, 2013).

Discussion about Scheduling

The presented scheduling methods have been references in Project Management for a long time. However, they only provide passive project scheduling, tracking and reporting aids, so do commercial PM tools, whose computational capabilities are unsatisfying. Additionally, their user must do the scheduling "by hand" without confidence about the global optimization and the result is generally

[1]The word *feed* comes from the delays that "feed" the critical chain since the total duration increases.

based on the manager experience.

The goal of the validation tool presented in Section 1.1 being to become as generic, automatic and optimal as possible, these methods are not fulfilling and more modern optimization algorithms will be presented in this chapter. This project will then treat scheduling as an assignment function, whose purpose will be to present a time schedule.

2.2.4 Modelling techniques

As described in the PMBOK (2013), there are two major modelling techniques for Project Management: *Simulation* and *What-if Scenario Analysis*.

Simulation

It consists in modelling thanks to the simulation of different projects durations with different sets of activities. It can use the principle of variation of the costs and durations, as already explained, with a probabilistic distribution of these inputs for example. At the end, one has a probabilistic distribution of the total project cost. This technique will be used in this scheduling approach.

What-if Scenario Analysis

"What if the situation represented by scenario 'X' happens?". This technique is very interesting for feedback modeling. It makes possible to model a proactive PM method and will be subject to a further work.

2.2.5 Milestone

The final aspect of the PM theory is about the continuous scheduling respect. Andersen (2006) has described the milestone method as a powerful way to ensure that the project is on time when combined with the PERT and CPM methods.

Retroplanning, i.e., scheduling taking the due date as a basis, cannot be used with only one reference, otherwise, variations and uncertainties will have a too strong influence. Therefore, sequences can be defined along the scheduling by setting deadlines to key tasks in order to ensure the scheduling respect at these steps. The steps composing the milestone are defined as *points* in the time and not *periods* in the time. Therefore, they cannot be used as events or buffers (see Section 2.6). However, following Anderson (2006), a major objective of the milestone is to make possible to determine whether a task is performed or not (leading to a dynamic rescheduling, or not).

A final objective of the milestone method is to define periods during which the activity must be performed. These periods define the lower and higher boundaries for this activity, not its beginning or ending. In the particular case of validation in the automotive industry, key parts of a vehicle must be tested in real conditions during the most extreme periods of the year, such as winter or summer.

2.3 Scheduling Optimization Methods

Following Bartschi Wall (1996), there are two categories of schedule-related problems: scheduling-only and joint scheduling/assignment problems. The first one only consists in scheduling different tasks of a project. The second one deals with sequences, duration and resource allocation. The two most-known scheduling/assignment problems are *project and job-shop scheduling*. The first part of this section will be dedicated to the study of these problems in order to help the characterization of the addressed problem. Moreover, it will make possible to compare the results of this work with the literature related to them in a chapter dedicated to it.

Garey et al. (1976) have described the scheduling problem of this work (see Section 1.1) as a combinatorial NP-complete problem. This specific kind of problem is characterized by a verification in a polynomial time, i.e., it is possible to verify a proposed solution at a reasonable computational cost. However, the major difficulty comes from determining feasible solutions for this problem, which is the main purpose of this work. In the worst cases, it leads to exponential computational costs or even insoluble problems. The need of powerful optimization algorithms appears, in order to solve the problem or to approximate the solution. These algorithms will be the second concern of this section.

Optimization methods intend to minimize/maximize an *objective function* that represents the problem. This principle stays valid for all optimization problems. Suitable methods for scheduling applications are divided into two different categories:

- *Optimum seeking procedures*, which intend to find the global optimal, the global minimum of the objective function in this case;
- *Heuristic methods*, that produce near-optimal solutions in a reasonable amount of time in case of bigger sized problems.

Both categories will be separately studied by focusing on aspects that can impact the resolution of the scheduling problem.

The mathematical model elaborated in Chapter 3 will consider the given problem as a specific kind of *Mixed Integer Non-Linear Programming* (MINLP) problem. These problems will be studied in the third part of this section and the methods usually used to solve them will be presented. Finally, the challenges of scheduling problems will be explained, leading to a discussion providing advises about the choice of the optimization method.

2.3.1 Similar Scheduling Problems

Two famous problems propose a mix of scheduling and assignment issues: project and job-shop scheduling. Both are resource-constrained, meaning that not all tasks can potentially be done in parallel due to limited resources. One can also cite the well-known *Travelling Salesman*, but the analogy with the studied problem is too weak.

Project Scheduling

This particular problem is defined by one project composed of a set of tasks to perform. Some tasks are constrained by *precedences*: a task cannot be started until one or other multiple tasks are done. All the others can be planned in parallel. Each task is described by procedures, time and cost relationships. The most general goal is to minimize either the cost or the total duration. The concept is illustrated in Fig. 2.7 under a precedence diagram.

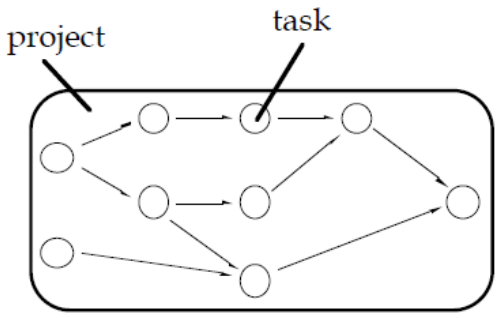


Figure 2.7: Concept of project scheduling - The constraints of precedence are illustrated by lines between the concerned tasks (Bartschi Wall, 1996).

This concept can be extended to *multi-modal* problems, where each task is described by different *modes*. A mode, in operation management, is defined by a particular way to operate a task. In the studied problem, multiple modes appear either by schedule compression (different cost/time values) of a given facility or by allocating multiple facilities, leading to different test procedures. The problem of these modes comes from the additional degrees of freedom, leading to more combinations of solutions, leading to more computational time.

Multi-modal problems include both planning and scheduling principles. This is the reason why the validation is scheduled under the name of Design Validation *Planning*, involving multiple modes.

Job-Shop Scheduling

This second typical scheduling problem is the *Job-Shop Scheduling* (JSS) problem, which is used as a basis for production line plannings. It consists in a work order where a set of jobs must be planned and each of them contains a set of tasks to perform. Each task has only one predecessor, one execution mode (uni-modal problem) and one has generally multiple different resources (machines or workers) compatible with each task. The job-shop scheduling problem is illustrated in Fig. 2.8.

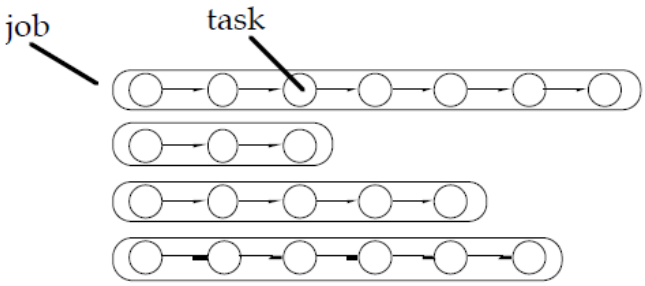


Figure 2.8: The Job-Shop Scheduling Problem consists in uni-modal jobs containing a set of tasks with only one precedence task (Bartschi Wall, 1996).

Problem Discussion

The problem characterized in Chapter 1 is undoubtedly more similar to the project scheduling problem. The different tests to perform can be seen as tasks to schedule with multiple precedences, making it impossible to organize in jobs. Additionally, each task being defined by several values of time and cost, the given problem refers to a multi-modal problem, comparable to a multi-modal project scheduling. However, the facility allocation described in Section 1.3.4 can be seen as a subgroup of a job shop scheduling problem. In summary, this work is a combination of both project and job-shop scheduling, with a predominance for project scheduling.

2.3.2 Optimum Seeking Methods

This first class of solution is also called *exact methods*. They provide the exact solution (or global optimum) if it exists. In the case no exact solutions exist, it is possible to implement a way to provide some indications. Their main problem is the needed computational time that can be prohibitive in case of big problems, containing many constraints and problems with lots of degrees of freedom. The most-known exact method is linear programming and will be the subject of this study.

(Integer) Linear Programming

A *linear programming* (LP) problem is an optimization problem described by a linear objective function and linear constraints. A (mixed-)integer linear programming problem contains discrete variables. Ferguson (2003) has provided the following representative two-variables and four-constraint example

$$\begin{aligned} &\max x_1 + x_2 \\ x_1 \geq 0, &x_2 \geq 0 \\ x_1 + 2x_2 &\leq 4 \\ 4x_1 + 2x_2 &\leq 12 \\ -x_1 + x_2 &\leq 1 \end{aligned}$$

where $x_1 + x_2$ is the objective function. Being a two variables problem, it can be represented on a 2D polyhedron with sides built from the constraints, as shown in Fig. 2.9. The exact optimum is located in $(\frac{8}{3}, \frac{2}{3})$.

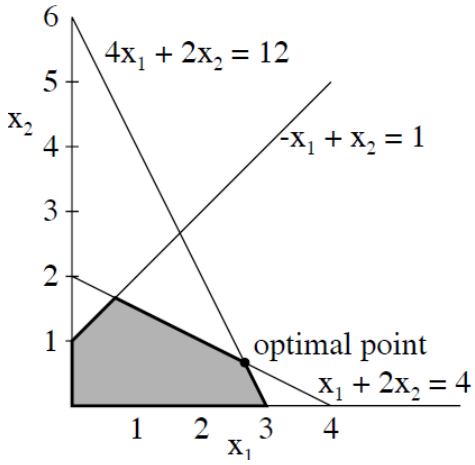


Figure 2.9: A two-variable linear programming (LP) problem can be represented on a 2D polyhedron (Ferguson, 2003).

The example illustrated in Fig. 2.9 can be generalized in a matrix form, where the complexity of the problem is translated in the size of the matrices:

$$\max \boldsymbol{x}^T \boldsymbol{b} = x_1 b_1 + \ldots + x_m b_m \tag{2.1}$$

$$\boldsymbol{x}^T \boldsymbol{A} \geq \boldsymbol{c}^T \tag{2.2}$$

There are two main linear programming methods, for which Robere (2012) has given the following explanations:

1. The *simplex method*, which consists in starting at a feasible point of the polyhedron and then travelling along the edges from vertices to vertices, thanks to the *pivot* operation, in a way

that the objective function is constantly decreasing. In the case of bad pivoting operations, the *cycling* phenomenon occurs and the algorithm is not able to stop at the best answer.

2. The *interior point method*, that goes into the polyhedron (the feasible region) and not only around the sides. It requires more computational time than the simplex method but provides finer results. Nevertheless, the simplex method has generally been more used in practice.

The presented methods have complementary and hybrid methods used to adapt them to the complexity of different kinds of problems.

2.3.3 Metaheuristic Methods

The main drawback of exact methods is the required computational time that can be prohibitive for large problems like validation. While they intend to find the exact solution, metaheuristics aim to provide a feasible solution, which is not systematically the global optimum, depending on the problem. The improvement lies in the lower computational cost, as well as the finer tuning of resolution.

The second advantage of metaheuristic methods is their ability to adapt themselves to more complex problems to define. Additionally, they can be stochastic, providing different results every time the algorithm is run, or deterministic with the same coherent result at each use. They can also be hybridized between different heuristic methods, leading to the term *meta*.

Metaheuristics generally do not permute tasks until a satisfying one is found. Indeed, most of them follow three steps:

- Planning, defining all needed resources;
- Sequencing, providing a solution respecting the precedences;
- Scheduling, minimizing the objective function.

Finally, they are divided into 2 main categories. First, *Spread Algorithms*, that attempt to determine an optimal solution by using a set of potential solutions defined by their own objective function value. The second one is called *Neighbourhood Algorithms*, that only use one potential solution, adapting it, evaluating its objective function and creating a neighbour to the previous solution.

Following Dréo et al. (2011), the most important characteristics that metaheuristics share are:

- Their stochastic aspect, which is an asset for the *combinatorial explosion* of possibilities in combinatorial problems;
- They can be adapted either for continuous or discrete problems;
- They all are derived from physical or biological theories (biomimetic optimization);
- Their settings need to be tuned precisely, as experienced in Section 3.3.1, as well as a high computational time, even if it is better than for exact methods.

In this section, the main metaheuristics will be presented, with a focus on the *Evolutionary Algorithm*, *Ant Colony Optimization*, *Particulate Swarm Optimization* (spread algorithms) as well as the *Simulated Annealing* and *Tabu Search* (neighbourhood algorithms).

Evolutionary Algorithms

Evolutionary Algorithms (EA) are optimization algorithms inspired from the biological evolution of species (Dréo et al. 2011). The most-known of them, the *Genetic Algorithm* (GA), will be here described because of the vast literature and the numerous implementations available.

The given problem has N solutions, some of them being local optima and other global optima. These solutions can be seen has a population of N individuals which have a given performance (or fitness) measured by an objective (or fitness) function. The purpose of GA is to make the population evolve by successive generations increasing the global fitness of the population by two main mechanisms inspired from the Darwin Theory:

- *Selection*, making competitive individuals reproduce;
- *Mating* or *Reproduction*, creating higher-potential offspring by mixing, recombination and characteristics variations of parents.

In practice, individuals are a list of integers for combinatory problems, real numbers, binary numbers or a mix of different kinds. The fittest individual(s) of the population at a given time is the solution of the solver. In particular, GA are very efficient for binary problems, because of the strong similarity between binary numbers and genes.

Changing from one generation to the other goes through four phases: selection of the fittest individuals, mating (reproduction), fitness assessment and substitution. The mating process consists in operators, *crossover* and *mutation*, applied on copies of the selected parents. The substitution phase replaces initial individuals, the weakest ones, by the new ones. This principle is illustrated in Fig. 2.10.

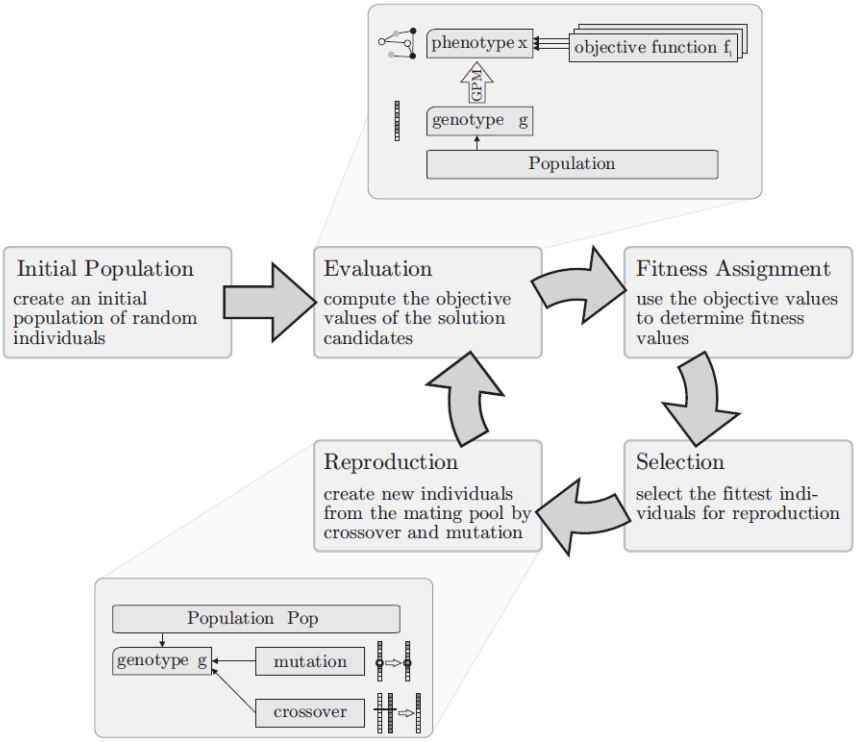


Figure 2.10: Genetic Algorithms (GA) consist in populations that evolve following the biological evolution (Weise, 2009).

With the increase of computational power and the increase of cores number of CPUs, parallel computing has become available. This parallel computing makes possible to evolve multiple populations in the same time, with migration between them to add another degree of evolution between different ethnic groups, by *migration*. All evolution processes, migration, crossover and mutations can be deterministic or probabilistic processes, whose law can be chosen. The main drawback of GA is the computational time for complex problems, which can become prohibitive, due to the simultaneous fitness assessment of large sets of solutions.

Ant Colony Algorithms

This second spread algorithm is inspired from the capacity of ant colonies to solve complex problems collectively despite the limited individual capacity of their members.

Dréo et al. (2011) have defined the *Ant Colony Optimization* (ACO) as a method based on the collaboration between ants to get food. They always find the same path (deterministic process), which is also the shortest one. It results from an indirect communication between ants via the environment called *stigmergy*. Each ant drops a chemical substance called *pheromone* on its way and all members of the colony orient themselves towards the area with the highest pheromones concentration, which also indicates the shortest path.

A major advantage of stigmergy is the capacity to face obstructions efficiently by finding a new shortest path rapidly, as illustrated in Fig. 2.11. Obstructions can be seen as dynamic variations (see Section 1.3) in the operation management point of view.

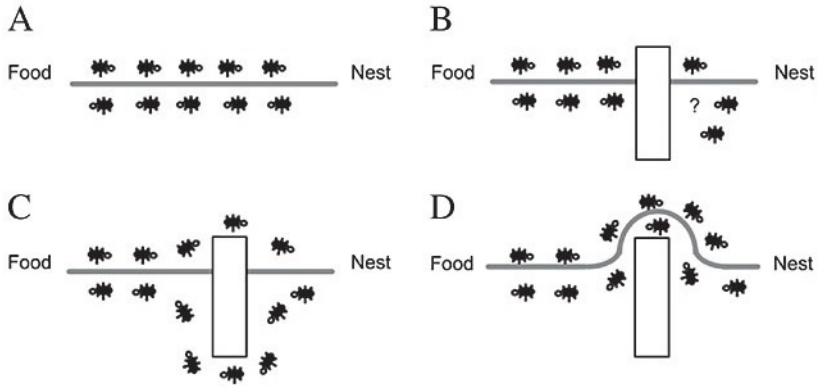


Figure 2.11: Ant colonies always find the shortest path to food thanks to stigmergy. [A] Ants follow the shortest path between the nest and the food; [B] An obstruction occurs; [C] Ants decide to go right and left with the same probability; [D] The right way being the shortest one, pheromones are dropped more rapidly and their concentration increases (Cerasela & Nechita, 2016).

Additionally, this kind of algorithms has the following characteristics:

- A high intrinsic parallelism;
- A high flexibility to environment modifications;
- Robustness to weaknesses among the ants;
- Decentralization: the intelligence is spread to all ants;
- Self-organization of the colony.

Finally, Dréo et al. (2011) have indicated ACO for problems with dynamic variations and needing a high failure tolerance. However, few powerful implementations are available due to the youth of the method.

Particule Swarm Optimization

Marini & Walczak (2015) have provided an introduction to the concept of *Particulate Swarm Optimization* (PSO). This method arose from the limitation of the *Artificial Intelligence* (AI) due to the centralization of the control at the individual level. The goal of PSO was then to introduce sociality contribution between individuals thanks to this spread metaheuristic algorithm.

PSO has been inspired by the behaviour of gregarious species like fishes, bees, dragonflies, birds, etc. The concept of swarm has been generalized to homogeneous and simple agents doing individually elementary tasks but interact (semi-)stochastically between themselves to work globally without central control.

Like for ACO, each agent taken individually is not efficient, but the swarm is globally capable of solving difficult problems. Like ACO again, the movement of each agent is not only the result of its own best solution memory, but the movement is also influenced directly by the best solution found by the other agents.

Like other spread algorithms, this one begins with a population (a swarm) of N candidate solutions $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}\}$ where $\overline{x_i} = [x_{i1} x_{i2} ... x_{iD}]$ is the position vector in a D-dimensional space of the i^{th} agent, where D is the number of variables. Its position at each moment is defined by

$$\overline{x_i}(t+1) = \overline{x_i}(t) + \overline{v_i}(t+1)$$

where $\overline{v_i}(t+1)$ contains the velocity components of the i^{th} agent at the next instant, that will influence its direction. This velocity vector is the result of three influences:

- Own inertia of the agent;
- Cognitive behavior of the agent, defining its tendency to go back to its personal best (p) solution;
- Social influence, defining the tendency of the agent to follow the global best (g) position of the swarm.

These influences are gathered in the velocity definition,

$$\overline{v_i}(t+1) = \overline{v_i}(t) + c_1(\overline{p_i} - \overline{x_i}(t))\mathbf{R_1} + c_2(\overline{g} - \overline{x_i}(t))\mathbf{R_2}$$

where the c coefficients are the acceleration constants and the $\mathbf{R}$ matrices are random diagonal matrices representing the stochastic behavior of the social and cognitive influences. The basic algorithm is given in Fig. A.2 and an animation in Fig. 2.12 illustrates the PSO operation for the simple problem of minimizing $(x-15)^2 + (y-20)^2$.

There are multiple implementations and hybrid forms of the PSO algorithm, implementing different forms of interactions, inspired from different species. However, a very promising one, the *Dragonfly Algorithm* (DA) (Mirjalili, 2016) simulates the surviving behavior of dragonflies and provides good results for different sorts of difficult optimization problems, especially for multi-objective problems.

Figure 2.12: Animation illustrating the PSO operation for the simple 2-D example $\min((x-15)^2 + (y-20)^2)$ (Elshamy, 2014).

Simulated Annealing

Dréo et al. (2011) have provided an analogy between optimization problems and the energy state minimization of magnetic material called *Spin Glass* illustrated in Fig. 2.13. The energy state of spin glass can be seen as a landscape with local and global minima. Similarly to the annealing method, the goal is to obtain the most stable state at the minimum energy state. In order to do so, the material must, first, reach a high temperature. Then, this temperature is slowly decreased until the most stable state is obtained. The *Simulated Annealing* (SA) method translates this thermodynamic problem in an algorithm where the energy state is the objective (or fitness) function to minimize. A control parameter is also introduced, a fictitious temperature, that will indicate the transformations to perform.

This neighbourhood algorithm, summarized in Fig. 2.14, starts from an initial configuration, i.e., a given solution at a given "high" temperature. An elementary transformation (similar to the mutation in GA) is applied leading to a new configuration and an energy variation ΔE. If ΔE is negative, then this configuration is preserved. If ΔE is positive, then the probability for this configuration to be maintained follows a stochastic law depending on the chosen algorithm. This process is reiterated at constant temperature until a sufficient number of elementary transformations is achieved, corresponding to a local or global thermodynamic equilibrium. Then, the temperature is decreased again and a new serie of transformations is applied, until the global minimum state is reached.

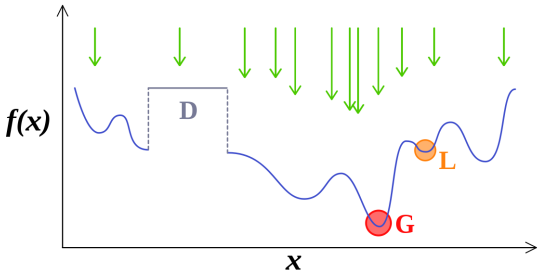


Figure 2.13: The simulated annealing (SA) translates the energy state minimization of the Spin-glass in an optimization algorithm. *G*, *L* and *D* stand for, respectively, global minimum, local minimum and discontinuity (Dréo et al., 2011).

The main reasons to opt for SA are generally its flexibility and its ease to implement. Unfortunately, it reveals to request a very fine tuning, what is a general drawback of all metaheuristics and an efficient temperature law. Additionally, SA requests most of the time a prohibitive computational time for heavy optimization problems.

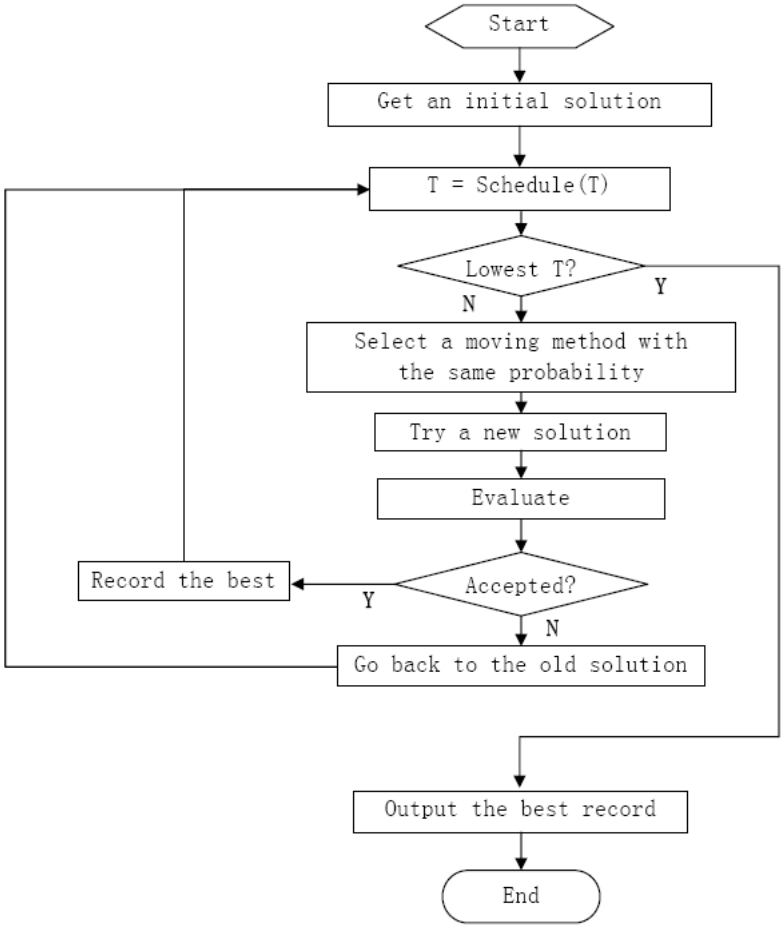


Figure 2.14: Basic Simulated Annealing (SA) algorithm, Sheng & Takahashi (2012).

Tabu Search

Tabu Search is an optimization algorithm inspired from the human memory, adapting its solutions by what it learned from the past. Like Simulated Annealing, it works with a single configuration at the time that evolves over the iterations, making it a neighbourhood algorithm. The transition from a configuration i to a configuration $i+1$ is made by two steps:

- A set of neighbours, $V(i)$, is made of all available solutions by a single elementary movement from i. However, if the set is too big, it can be reduced;

- Each member of $V(i)$ is assessed thanks to an objective function and the best configuration of $V(i)$ becomes the $i+1$ configuration, even if its fitness is lower than the one of i. This specificity makes possible not to get stuck in local minima.

In order to avoid a cyclic operation, consisting in selecting a configuration already encountered, one sets a list of forbidden movements, the *tabu* configurations, containing all the movements already performed. It simulates a short-term memory.

A long-term memory is simulated via two mechanisms based on the recurrence of events:

- *Intensification* that investigates deeper some promising areas of solutions;
- *Diversification* which is the ability of the algorithm to explore unexpected areas to widen the spectre of solutions.

Tabu Search benefits from less parameters to tune than Simulated Annealing and provides better results in some cases, but both intensification and diversification rise more complexity. Additionally, simulating memory increases the degree of freedom, what leads to the inability to analyze mathematically the method.

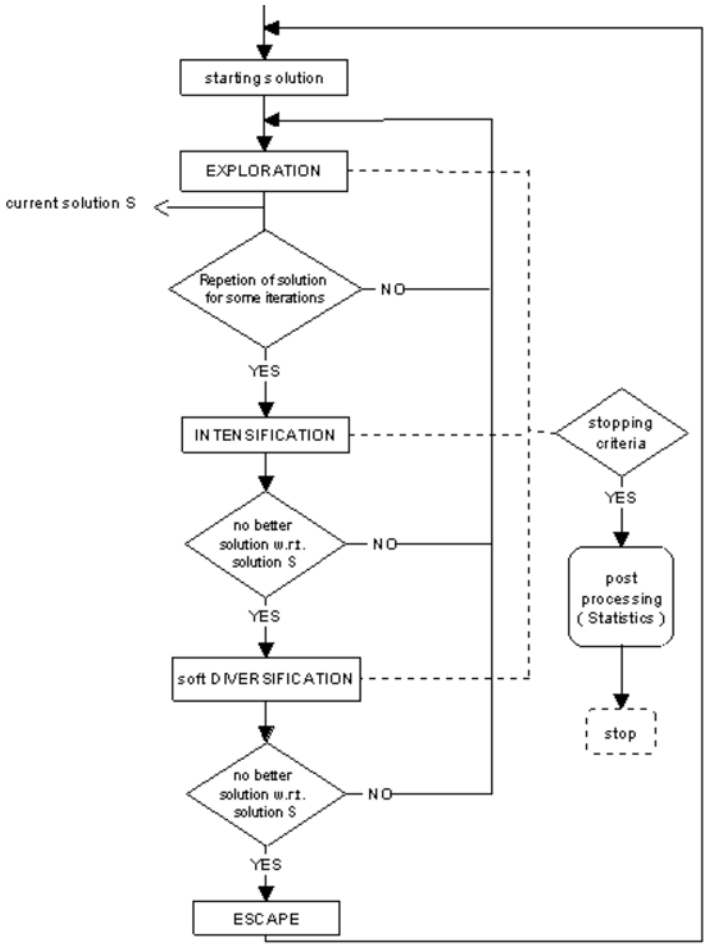


Figure 2.15: The Tabu Search (TS) method simulates the memory process, Cortadella (2000).

2.3.4 Mixed Integer Non-Linear Problems

Mixed Integer Non-Linear Programming (MINLP) is a field of optimization that has many applications in multiple domains. These problems are defined by discrete, potentially linear variables and non-linear constraints, like the concerned scheduling problem, as developed in the next chapter. These problems are difficult to solve, especially when they are non-convex (Schlüter et al., 2012), unlike the problem presented in Fig. 2.9. This non-convexity occurs in some scheduling cases, like problems without due date.

Following Floudas (1995), the major difficulty in MINLP is the lack of optimality conditions, meaning that there is few "improving feasible direction". These problems, as well as combinatorial ones, must compare the fitness value of all solutions, without indications of improvement, leading to

quasi-partial enumerations of the tested solutions.

Two methods will be here discussed: the classical Branch and Bound method and the promising optimization software MIDACO. The other methods presented previously can be adapted to MINLP but were not dedicated to these specific problems.

Branch and Bound

The most intuitive solution for combinatorial problems, similar to the *Brute Force* method, would be to enumerate all possible combinations and picking the fittest one. However, it would lead to a very high computational time. *Branch & Bound* (B&B) is an optimization method for combinatorial and MINLP based on a smart enumeration of solutions in the set of all admissible solutions. As said before, there is no optimality conditions. The principle is to prove the optimality of a solution by partitioning the solution space by eliminating the non-improving solutions.

The B&B execution can be represented by a tree structure (see Fig. 2.16), containing branches and bounds with a common root containing all solutions for the optimization problem. For a minimization problem like the scheduling one, the method must benefit from:

- A method to define a lower bound for the fitness assessment, like the cost for example, for all partial solutions;
- A strategy for dividing, creating *branches*, the research space into smaller spaces, called *bounds*;
- A method to define an upper bound for at least one partial solution.

The algorithm obeys to the following rules:

1. Build the common root of the tree, containing the set of all solutions by relaxing the integer constraint. The optimality condition is applied, i.e., checking the equality between the lower and the upper bound (of the root in this case). If the optimality condition is verified, the process is stopped;
2. The root is divided in multiple subgroups;
3. Recursive process: the B&B method is applied on the subgroups. If the optimality condition is verified, a solution is valid but not necessarily optimal globally;
4. This potentially optimal subgroup will not be divided if its lower bound is higher than the lower bound of another solution, since the global optimum will not be part of this group. This method eliminates valid, yet not globally optimal solutions of the research space.

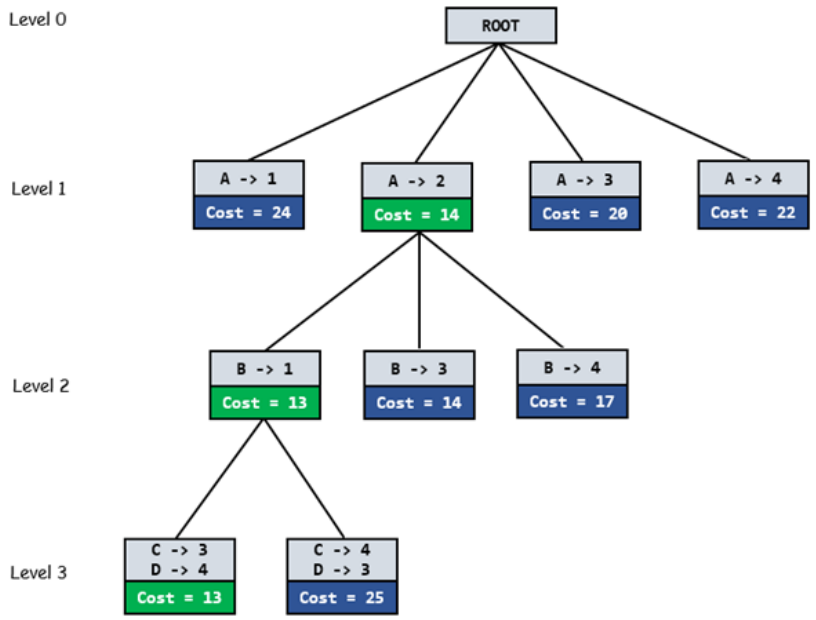


Figure 2.16: Tree structure for a Job-Shop Scheduling problem using B&B (Bhatia, 2013).

While particularly suited for MINLP and combinatorial problems, B&B is not a heuristic itself. Rebaïne (2005) has provided applied examples of the B&B method for resource allocation problems, Job-Shop scheduling problems or for the Travelling Salesman problem.

Mixed Integer Distributed Ant Colony Optimization (MIDACO)

The *Mixed Integer Distributed Ant Colony Optimization* (MIDACO) software is a solver developed originally for spacial applications by the *European Spatial Agency*. Spatial activities provide generally many scheduling solutions due to the highly constraining frame that Space represents.

It is based on an ACO algorithm, with an extension to integer variables. The algorithm samples the solution space thanks to a *probability density function* (PDF), based on multiple kernels rather than parameters, making possible to estimate the density of probability of improvement in any points. The solution space sampling is illustrated in Fig. 2.17 for integer variables.

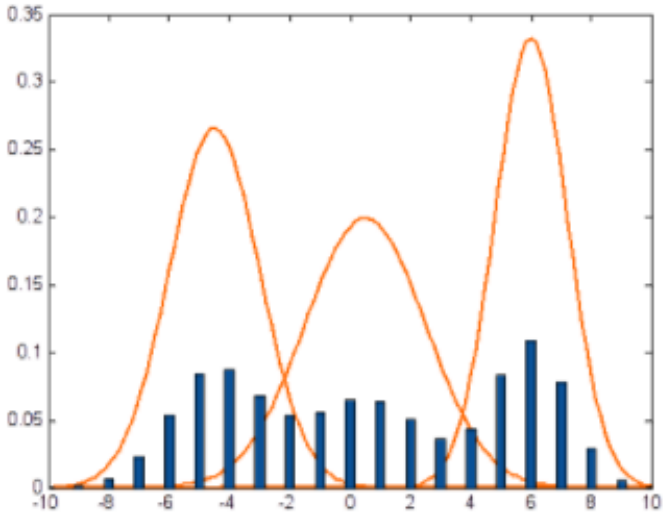


Figure 2.17: The MIDACO algorithm is able to sample an integer decision space thanks to a multi-kernel PDF (MIDACO User Manual, 2016).

MIDACO works as a *black-box* optimizer, providing a high level of freedom, minimizing a function $f(x)$ constrained by $g(x)$ (in)equalities, whose variables x can be integer and/or continuous variables. Additionally, this algorithm is able to solve big non-linear and non-convex problems, up to more than 10.000 variables and thousands of constraints. A massive parallelization is also possible for a more effective computational time. Implementing an ACO method, its results are deterministic.

Like all evolutionary-like algorithms, MIDACO does not guarantee to provide the global optimum. However, Schlüter et al. (2012) have provided a complete study of the efficiency of the MIDACO method on more than 100 benchmark MINLP. For scheduling problems in particular, it has revealed to be significantly more efficient than all other methods (and B&B in particular) in terms of computational time and the ability to find the global optimum.

2.3.5 Challenges of Scheduling Problems

Bartschi Wall (1996) has presented three different challenges for optimization problems that are particularly relevant for scheduling problems. Scheduling applications are affected by scaling (the size of these problems), their dynamic nature, their sparse and restricted solution space. The *No Free Lunch Theorem* is also relevant for scheduling problems and will be presented in this section.

First, scheduling is sensitive to the increase of the size of the problem, i.e., the number of tasks, modes, resources and constraints. There are two possibilities to deal with scaling difficulties:

- Pruning the search space to highlight more promising areas;

- Specifying more precisely the search space in order to remove irrelevant search areas.

However, both methods are difficult to implement technically.

Second, the dynamic nature of real scheduling applications can be seen either as uncertainties, or dynamic variations during the project execution. Uncertainties are unpredictable events due to failures in the requirements or resources. Dynamic variations are used to reflect the schedule execution precisely in the schedule prediction but also to deal optimally with the dynamic nature of real problems. In this way, the schedule will be optimal accordingly to the actual operations and not to the project prediction.

Beside its size, the sparseness of the solution space is a challenge, as well as the number of restricted areas. A restricted area corresponds to an unfeasibility for which the schedule must be either extended, if possible, or the algorithm must "jump" to other areas. In the first case, it leads to a less satisfying result, in the second one, it results in more computational time. Unfortunately, barely any heuristics are capable to detect rapidly restricted areas of the search space.

Finally, Wolpert & Macready (1997) have defined the *No Free Lunch Theorem* as "For any algorithm, any elevated performance over one class of problems is offset by performance over another class". It means that finding a solution for any method requires the same computational cost, averaged over all classes of problems. This last statement results in the need to create, or to tailor, a method for the given problem, or to find the method that will have satisfying performances for the given class of MINLP. The no free lunch theorem is also valid for other domains like economics.

2.3.6 Methods - Discussion

In this literature review, some promising optimization methods have been presented, going from the "basic" linear programming to the very technical MIDACO. To conclude this chapter, some indications can be given for orientating the choice of the resolution method for the scheduling problem characterized in Chapter 1.

Arostegui (2006) has studied the performances of the Genetic Algorithm, Simulated Annealing and Tabu Search on combinatorial problems. It has revealed that Tabu Search has good performances. It is also the case for the two other ones but their performances are more specific to the given problem, and, especially to the parameters of the chosen method.

Dréo et al. (2011) have stated that spread metaheuristics, like GA, PSO or ACO, are generally more powerful than neighborhood metaheuristics. They are also less subject to local minima confinement. However, they generally request more computational time and they are more sensitive to fine of from their parameters. Among these methods, ACO in particular has shown very good results (Xiao et al., 2013) for MINLP and tasks scheduling applications. This method is also the most suitable one to deal with dynamic variations.

Some methods are more mature, depending on the given problem and the given tool used to perform the optimization. GA is the most mature method, with a very general implementation and a hands-on tuning of its parameters. Also, a lot of scheduling optimization applications have been solved using this heuristic and they provide a large source of cases. ACO has a powerful MINLP implementation, MIDACO (not free), that is also easy to adapt to any kinds of problem and has shown very encouraging results.

PSO has implementations on the tool used for this work but no found implementations were able to deal with MINLP. Branch & Bound is also a very promising algorithm but suffers from a lack of generality in its implementation and requires to be adapted to each problem. There exists other specific programs dedicated to the optimization in this industry. They will not be studied in this work but a description is available in Appendix C.1.1.

3. Methodology

This chapter is dedicated to the methodology used to solve the problem presented in Chapter 1. It will be divided into three main parts:

- The interface of the tool;
- The mathematical model of the scheduling problem;
- The choice of algorithms and their corresponding implementation.

The literature review of Chapter 2 will be used as a frame for the characterization of the problem and the state-of-the-art will provide the potential resolution methods.

The first section will specify the needs of the interface used to create the DVP. The purpose of this interface is to gather the data needed for the tests scheduling. It will go through all the requirements of the schedule itself, from the target of the DVP to the precise definition of the constraints. This study will lead to mockups of the interface presented in Chapter 4.

The second section of this study will be dedicated to the mathematical model used to solve the scheduling problem. It will begin with the translation of the basic scheduling problem in a mathematical problem to optimize. Then, the facility allocation case will be included, followed by the constraints relaxation and some indications about dynamic variations and multi-objective optimization will be given.

Finally, it will lead to the choice of the most relevant algorithms studied in Section 2.3, during the state-of-the-art review. The way to implement the chosen algorithm for scheduling applications will be described for three different cases: scheduling without facility allocation, with facility allocation and adaptive constraints.

The next chapter will display the results of this methodology. It will first give the mockups of the interface, followed by the results of the scheduling optimization for each chosen algorithm, implementing the different cases addressed in this study.

3.1 Optimization Tool Interface

This section intends to present the tool objectives and specifications in order to explain how to gather the information described in Section 1.3 in the most flexible and generic way. The solution proposed to the user of the DVP optimization tool is a *Graphic User Interface* (GUI), in which he will be able to enter the needed information in a user-friendly way.

After studying the interface, a mathematical model will be proposed in Section 3.2 as an optimization problem. This model will be constructed thanks to the objectives, constraints and variables

defined precisely via the interface. The GUI can be seen as the link between the project management problem and the mathematical model.

3.1.1 Tool Objectives

The tool must offer three different main services:

1. Creating a DVP, in which the user defines tests;
2. Updating a DVP, providing dynamic variations about the aborted tests or tests already performed;
3. Constraining a DVP;
4. Displaying an existing DVP and its relevant information.

These four objectives must meet different needs that will be deeply defined in this section. The resulting GUI mockups of Section 4.1 will provide a way to execute the presented specifications, respecting a certain client perspective.

Creating a DVP

When the user creates a DVP, he must actually define the tests composing the DVP. As shown in Fig. 2.2, the tests will be composed of:

- A requirement or a set of requirements to validate;
- Test facility(ies) on which the test will be performed;
- The test case providing a description of the test, the conditions and the acceptance criteria.

Additionally, the different facilities suitable for each test must be described by different modes, i.e., different cost/duration values. The goal of the DVP is defined as a time-minimization and/or a cost-minimization problem.

Updating a DVP

Dynamic variations, i.e., feeding the schedule with factual information about the predicted tests is a major factor of the efficiency of the tool. The feedback must satisfy two cases:

- The test has been aborted and must be rescheduled;
- The test has been completed and must be precisely settled in the DVP.

The first case is a real challenge for scheduling problems since the user has to define the level of change he accepts, i.e., the number of other tests that are perturbed by the rescheduling. The second one ensures the optimality of the global schedule, by fixing the actual schedule and time/cost value of the performed test. In this way, the other scheduled tests will be marginally rescheduled in function of parameters to define at the model level.

Constraining a DVP

The GUI must implement a way to define the following constraints:

- The unavailability periods of facilities;
- The milestone periods;
- The maximum budget and/or the maximum duration;
- Precedences between tests.

These constraints do not have to appear necessarily on the same menu but the way to define them must be as intuitive as possible.

Displaying a DVP

The DVP must be displayed focusing on the schedule. The whole schedule must be available, as well as the tests data. The way to display the relevant tests must be intuitive, with the set of all concerned tests available. The user must access easily the test case of each test he wants to perform. Other pieces of information like the planned schedule for a given test, its computed cost, the chosen facility, etc. must be displayed as well.

3.2 Scheduling Mathematical Model

The method used to collect the relevant data has been presented. It means that the kinds of data available are known precisely. This section will specify under what form the information of the tests will be stored in order to be able to use it for the mathematical model. At the same time, the basic mathematical model will be presented in detail. It will be extended to the facility allocation, constraint relaxation, dynamic variations and multi-objective optimization cases.

Finally, the choice of the algorithm used to solve the mathematical problem will be discussed following the suitability of the methods presented in Section 2.3. It will lead to the implementations of two chosen algorithms and their parameters will be presented.

In the next chapter, the results of this methodology will be presented, supported by working examples illustrating three different cases: the basic validation scheduling, scheduling with allocation of facilities and adaptive constraints implementation.

3.2.1 Problem characterization

Before solving any optimization problems, it must be correctly addressed. For characterizing it, a precedence diagram is the most suitable option to represent the given scheduling problem, as shown in Fig. 3.1.

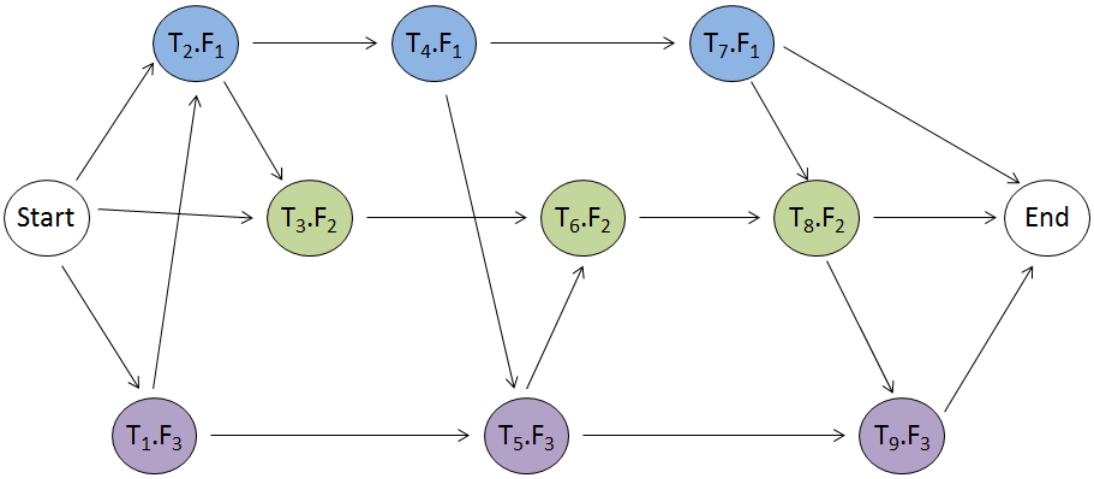


Figure 3.1: Characterization of the scheduling problem.

A complete problem characterization has been done at the beginning of this work (see Chapter 1). However, a succinct explanation of the precedence diagram is presented below.

It is first characterized by a *start time*, and an *end time* that is preferably smaller than a given *due date*. The precedence diagram is composed of scheduled tests. Arrows are located between tests and represent *precedences* between tests, either due to precedence constraints or because these tests are performed on a same facility. Finally, the chart is split into rows, which individually represent a given facility j on which a test i is performed, under the form $T_i.F_j$.

3.2.2 Basic Scheduling Equations

The way to represent the data and how to translate the scheduling problem in a mathematical optimization problem is here presented. It will follow the following order: available data, objective

functions, unknowns, milestone periods and variables. Cost and/or time optimization will be both studied. These equations do not include either facility allocation or dynamics variations, which will be subject of a separated study.

Data

By using the tool described in Section 3.1, the user provides for each test:

- The different cost and duration values for each facility, describing at what cost a test is performed in a given duration: for a $Test_i$, the j^{th} mode is given by the couple $(Test_{i,cost_j}, Test_{i,time_j})$. These values result from the schedule compression by crashing explained in Section 2.2.2;
- Precedence constraints, defining the tests to perform before the concerned test;
- The facility and the corresponding unavailability periods for this facility.

It leads to three matrices, one vector and one array: the cost matrix C, the corresponding duration matrix D, the precedence matrix P, the facility vector F and an unavailability array U.

The **C** and **D** matrices both have one described test per line and a mode per raw. For N tests composed of maximum M modes, one has:

$$\mathbf{C} = \begin{bmatrix} Test_{1,cost_1} & Test_{1,cost_2} & \dots & Test_{1,cost_M} \\ Test_{2,cost_1} & Test_{2,cost_2} & \dots & Test_{2,cost_M} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ Test_{N,cost_1} & Test_{N,cost_2} & \dots & Test_{N,cost_M} \end{bmatrix} \mathbf{D} = \begin{bmatrix} Test_{1,time_1} & Test_{1,time_2} & \dots & Test_{1,time_M} \\ Test_{2,time_1} & Test_{2,time_2} & \dots & Test_{2,time_M} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ Test_{N,time_1} & Test_{N,time_2} & \dots & Test_{N,time_M} \end{bmatrix} \tag{3.1}$$

If a cost/duration couple is not defined (not all tests have the same number of modes), its cost/duration is set to infinite, leading to an infinite cost if chosen, as described later.

The precedence matrix **P** has one line per test as well, but its rows are composed of the precedence tests numbers. When there is no precedence, a zero is set. The facility vector $\overline{F}$ contains one facility per test, corresponding to the lines of the **C**, **D** and **P** matrices. For N tests, one has:

$$\mathbf{P} = \begin{bmatrix} Test_{1,precedence_1} & Test_{2,precedence_1} & \dots & Test_{N-1,precedence_1} \\ Test_{1,,precedence_2} & Test_{2,precedence_2} & \dots & Test_{N-1,precedence_2} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ Test_{1,precedence_N} & Test_{2,precedence_N} & \dots & Test_{N-1,precedence_N} \end{bmatrix} \quad \overline{F} = \begin{bmatrix} Facility_1 \\ Facility_2 \\ . \\ . \\ . \\ Facility_N \end{bmatrix} \tag{3.2}$$

If $Test_i$ must be performed before $Test\,j$, one will write a i at the j^{th} line. For example, one considers a three-test problem, for which Test 2 must be performed after Test 1 and Test 3 must be performed after Test 1 and Test 2. It leads to the following precedence matrix and facility vector:

$$\mathbf{P} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 2 \end{bmatrix} \quad \overline{F} = \begin{bmatrix} Facility_{Test_1} \\ Facility_{Test_2} \\ Facility_{Test_3} \end{bmatrix} \tag{3.3}$$

Again, there is no link between the precedences and the facilities, which are two distinct problems, and there can be repetitions in the facilities ($Test_i$ can be performed on the same facility as $Test_j$).

Finally, the facilities that are unavailable (for maintenance purposes for example) at certain periods are listed in an array with the corresponding periods. One period is represented by a 2-sized vector containing the starting and the ending time of the unavailable period. Table 3.1 shows an example of two facilities, described by two periods of unavailability each.

$U =$

Facilities	**Periods**	
	Start	End
$Facility_1$	1	5
	10	12
$Facility_2$	2	4
	7	8

Table 3.1: The unavailability periods of facilities are listed in the U array.

In this particular example, the first facility is not available from the first to the fifth and from the tenth to the twelfth time slot.

Objective functions

The objective (or fitness) function is the function to minimize. In this case, it can be either the cost function, the time function or a combination of both (the problem is then called a multi-objective optimization problem). The two functions are given by:

- The cost function, C_{tot}, to minimize for N tests of maximum M modes:

$$\min C_{tot} = \sum_{i=1}^{N} \sum_{j=1}^{M} m_{ij} C_{ij} \tag{3.4}$$

- The time function, D_{tot}, to minimize for N tests of M modes:

$$min\, D_{tot} = \min(\max_{1 \leq i \leq N} f_i) \text{ and } f_i = s_i + \sum_{j=1}^{M} m_{ij} D_{ij} \tag{3.5}$$

  where s_i and f_i are, respectively, the start time and finish time of the ith test. This particular case is a well-known *MinMax problem* where the goal is to minimize the worst case.

The coefficients m_{ij} are called the modes coefficients of Test i. The analogy with the mode concept in project scheduling (see Section 2.3.1) can be made. They are defined by

$$m_{ij} = \begin{cases} 1, & \text{if mode}_j \text{ of Test}_i \text{ is executed;} \\ 0, & \text{otherwise.} \end{cases} \tag{3.6}$$

Their role is to choose one and only one cost/duration value for each test so that

$$\forall i, \sum_{j=1}^{M} m_{ij} = 1 \tag{3.7}$$

Linear Constraints

First, there are *precedence constraints*: If $\text{Test}_{i'}$, noted $T_{i'}$, is a predecessor of Test_i, noted T_i, the end time of $T_{i'}$ must be lower than the start time of T_i:

$$\begin{aligned} &\text{if } i' \in P_i = \{i' : T_{i'} = \text{ predecessor of } T_i\} \\ &\Rightarrow s_i > f_{i'} = s_{i'} + \sum_{j=1}^{M} m_{i'j} D_{i'j} \end{aligned} \tag{3.8}$$

Then, there are *mode unavailabilities*: If a mode is not defined (infinite cost and duration), it value must be set to zero:

$$\text{If } C_{ij} = D_{ij} = \infty \Rightarrow m_{ij} = 0 \tag{3.9}$$

Non-Linear Constraints: Resources Availability

As said in Section 1.3, facilities can be seen as resources following the operations management definition. The resources in this case can be unavailable due to two reasons:

- A resource can be inoperable because of maintenance purposes or used for other projects;

- If a resource is used for a test, it is not available anymore for a project performed in parallel.

In any cases, the test must begin after the end of the unavailability period, f, or it must finish before the beginning of the unavailability period, s. For $e \in \{1, ..., E\}$, where E is the number of unavailability periods, one has the exclusion variable x_e such that

$$x_e = \begin{cases} 0, & \text{If } T_i \text{ begins after the end of unavailability period} \\ 1, & \text{If } T_i \text{ ends before the beginning of unavailability period} \end{cases}$$

so that the constraint is expressed as

$$x_e(s_i - f) + (1 - x_e)(s - f_i) > 0 \tag{3.10}$$

The exclusion variable is multiplied by other variables, creating a non-linear constraint. One can extend the previous sentence to the following conclusion: in optimization problems, logical disjunctions (commonly called *OR*) lead to non-linear constrained problems.

Milestone

To deal with the milestone continuous scheduling respect, time bounds can be set on each test, so that Test i is constrained by a upper bound UB and a lower bound LB,

$$s_i > LB \quad \& \quad s_i + \sum_{i=1}^{N} \sum_{j=1}^{M} m_{ij} D_{ij} < UB$$

where the strict inequalities are due to the use of time slots. In order to implement the milestone, a bound matrix B is created for N tests,

$$\mathbf{B} = \begin{bmatrix} LB_1 & UB_1 \\ LB_2 & UB_2 \\ . & . \\ . & . \\ . & . \\ LB_N & UB_N \end{bmatrix}$$

If there is no constraint on the start date, $LB_i = 1$ (first time slot) and if there is no constraint on the test due date, $UB_i = DD$, where DD is the global due date (last tolerated time slot).

This way of interpreting the milestone also makes possible to correlate some tests to specific periods, like winter tests. In that particular case, $LB = 21/12/2017$ and $UB = 20/03/2018$.

Variables

The value of the variables minimizing the objective function the most while respecting the constraints will be the output of the algorithm. For $i \in \{1,...,N\}$, $j \in \{1,...,M\}$ and $e \in \{1,...,E\}$, the variables are the modes m_{ij}, the starting times s_i and the exclusion variables x_e. For N tests, M modes and E exclusion variables, the list of variables will be:

$$\overline{Variables} = [m_{11}, m_{12}, ..., m_{NM}, s_1, s_2, ..., s_N, x_1, x_2, ..., x_E]$$

These values will make possible to directly calculate the duration/cost of each test and the computed schedule. The modes and exclusion variables are binary values, while the starting times are discrete variables due to the use of time slots instead of hours, days, months or quarters. For example, a test that occurs between 00:00 and 01:00 (1 hour) begins on time slot 1 and finish on time slot 1.

3.2.3 Facility Allocation Problem

The goal of this section is to solve the facility allocation problem introduced in Section 1.3.4. As Bartschi Wall (1996) explains, "Schedules assign resources to tasks (or tasks to resources) at specific times". Previously, one assumed that only one facility was assigned for each test and this allocation of resources was then irrelevant. In this section, more than one facility can be allocated to each test. This allocation is restricted to the facility itself and not extended to the human resources, allocated at the facility level.

Problem Definition

The problem definition differs from the previous one with the definition of the cost, duration and facility matrices. The facility vector is now a facility matrix whose number of rows corresponds to the maximum number of different facilities per test. A facility matrix is illustrated in Eq. 3.11, for N tests and maximum L different facilities per test. When a test has less than this number of possible facilities, a 0 is set. There still can be redundancies in the facilities, i.e., different tests can use the same facility.

$$\mathbf{F} = \begin{bmatrix} Facility_{1,Test_1} & . & . & . & Facility_{L,Test_1} \\ Facility_{1,Test_2} & . & . & . & Facility_{L,Test_2} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ Facility_{1,Test_N} & . & . & . & Facility_{L,Test_N} \end{bmatrix} \tag{3.11}$$

The cost and duration matrices become three-dimensional matrices, whose number of layers corresponds to L different facilities per test, as explained for the facility matrix. Each layer of these matrices represents the different modes (time/cost values) of the tests. For the same example, it leads

to a L-layer cost/duration matrix, as shown in Eq. 3.12 for the cost.

$$\mathbf{C_1} = \begin{bmatrix} Test_{1,cost_1} & \dots & Test_{1,cost_M} \\ Test_{2,cost_1} & \dots & Test_{2,cost_M} \\ . & . & . \\ . & . & . \\ . & . & . \\ Test_{N,cost_1} & \dots & Test_{N,cost_M} \end{bmatrix} \dots \mathbf{C_L} = \begin{bmatrix} Test_{1,cost_1} & \dots & Test_{1,cost_M} \\ Test_{2,cost_1} & \dots & Test_{2,cost_M} \\ . & . & . \\ . & . & . \\ . & . & . \\ Test_{N,cost_1} & \dots & Test_{N,cost_M} \end{bmatrix} \tag{3.12}$$

The mathematical model presented in Section 3.2.2 remains valid, with a modal unity adapted to L layers and M modes: $\forall i, \sum_{l=1}^{L} \sum_{j=1}^{M} m_{ijl} = 1$. One single mode of one single facility can be selected for each test.

Non-Linear Constraints

The non-linear constraints due to the unavailability of a facility (see Eq. 3.10) must be adapted. Now, this unavailability is conditioned to the choice of the facility for a given test: the facility must be selected to be under the unavailability constraint. In Section 3.2.2, for all tests i, one had $\sum_{j=1}^{M} m_{ij} = 1$. This is adapted for multiple facilities, with

$$\sum_{j=1}^{M} m_{ijl} = \begin{cases} 0, & \text{if Test } i \text{ is not performed on the facility } l; \\ 1, & \text{if Test } i \text{ is performed on the facility } l. \end{cases}$$

As a reminder, there are two cases of resources unavailability:

- A facility is not available due to reasons like maintenance, closure, etc. For the e^{th} unavailability period, the i^{th} test and the l^{th} facility, the constraint is then written as

$$x_e(\sum_{j=1}^{M} m_{ijl})(s_i - f) + (1 - x_e)(\sum_{j=1}^{M} m_{ijl})(s - f_i) > 0, \tag{3.13}$$

  where s and f are, respectively, the beginning and the end of the unavailability period.

- Two tests, T_1 and T_2, can be performed on a same facility. The unavailability is now subject to the following constraint for the l^{th} facility:

$$x_e(\sum_{j=1}^{M} m_{1jl} \times \sum_{j=1}^{M} m_{2jl})(s_1 - f_2) + (1 - x_e)(\sum_{j=1}^{M} m_{ijl} \times \sum_{j=1}^{M} m_{2jl})(s_2 - f_1) > 0. \tag{3.14}$$

Eq. 3.14 and 3.13 will be computed in any cases but will only be constraining in the case where both tests choose the same facility. One has thus $\sum_{j=1}^{M} m_{1jl} = \sum_{j=1}^{M} m_{2jl} = 1$.

Objective Functions

The global duration and global cost can be extended to the case of L facilities allocated, for N tests and M modes as follows:

$$C_{tot} = \sum_{l=1}^{L} \sum_{i=1}^{N} \sum_{j=1}^{M} m_{ijl} C_{ijl}, \tag{3.15}$$

$$D_{tot} = \max_{1 \le i \le N} f_i \text{ and } f_i = s_i + \sum_{l=1}^{L} \sum_{j=1}^{M} m_{ijl} D_{ijl}. \tag{3.16}$$

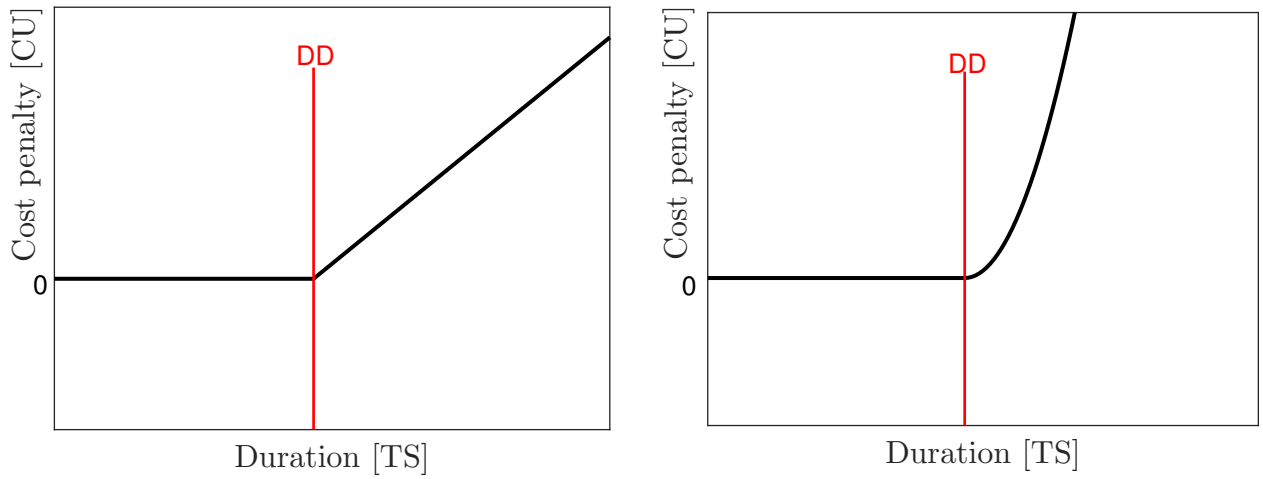
3.2.4 Constraints Relaxation

This section intends to solve the problem posed in Section 1.3.3. Previously, the problem was considered as hard-constrained, by imposing intermediary due dates on each test and a global due date. In the case of non-respect of these constraints, the algorithm did not converge. With *soft constraints*, the convergence is insured even for non-admissible solutions but at a higher cost. The analogy with an uneven landscape can be made: hard constraints are forbidden geographical zones, while soft constraints can be seen as high and difficult (yet admissible) areas like hills. This concept is also called *constraints relaxation*.

Constraints Relaxation Methods

There exist multiple ways to relax constraints. The most famous relaxation laws are the linear and quadratic relaxations illustrated in Fig. 3.2, where CU and TS stand for, respectively, *cost unit* and *time slot*. The linear one is generally used in the industry, with a constant penalty on each day delayed after the due date *DD*. The quadratic one is used in optimization problems where the relaxation must be limited by a prohibitive additional cost to ensure a convergence near the constraints.

Svanberg (1987) has proposed the well-known *Method of Moving Asymptotes* (MMA), in which the curvature of the curve increases (by moving the asymptote) in function of the deviation from the constraint. It has shown for the last two decades convincing results in structural optimization.



(a) Linear Relaxation.

(b) Quadratic Relaxation.

Figure 3.2: The linear and quadratic constraint relaxation are the two most-known laws.

Overrun Costs

When the optimization goal is to minimize the cost of the validation, a global due date is defined. On the contrary, when the goal is to minimize the global duration of the validation, a maximum budget is set. In multi-objective optimization, both constraints must be defined.

In case of overrun of a intermediary/global due date, the user (or contract) specifies additional costs called *overrun costs*. The linear case is chosen with $\Delta C = W \times O$, where W is the weight W [CU/TS] and O the overrun [TS]. The overrun can be calculated knowing the *end* of test or DVP, E, and the intermediary/global *due date*, DD:

$$O = \begin{cases} E - DD, & \text{If } E \geq DD; \\ 0, & \text{else.} \end{cases} \tag{3.17}$$

When the duration is minimized, there is no additional time considered but only a maximum budget as a fixed constraint.

Implementation

The adaptive constraints implementation still needs the definition of a minimum start date and a due date. Moreover, it needs to define the weighting factor defining the additional cost linked to the overrun of an intermediary or global due date. In order to do so, the bound matrix **B** is adapted to receive the weighting factor of each of the N tests. For Test_i and the global due Date DD, the weighting factor is, respectively, W_i and W_G.

$$\mathbf{B} = \begin{bmatrix} LB_1 & UB_1 & W_1 \\ LB_2 & UB_2 & W_2 \\ . & . & . \\ . & . & . \\ . & . & . \\ LB_N & UB_N & W_N \end{bmatrix} \overline{DD} = \begin{bmatrix} DD & W_G \end{bmatrix}$$

By convention, $W_i = 0$ when the time range is hard-constrained, while W_i is a scalar when the time range is soft-constrained. The cost function is then defined, for N tests, M modes and L different facilities per test, by

$$C_{tot} = \sum_{l=1}^{L} \sum_{i=1}^{N} [\sum_{j=1}^{M} m_{ijl} C_{ijl} + W_i \times O_i] + W_G \times O_G \tag{3.18}$$

where W_G, O_G and O_i are, respectively, the global weighting factor, the global overrun and the intermediary overruns. The implementation itself is impacted in the linear constraints definition, where the constraints matrix (defined in Eq. 3.19) must be adapted by removing the constraints for which $W_i \neq 0$, otherwise the hard constraint will still be applied. This rule can generalized:

A constraint in the constraints matrix will be a hard constraint while a constraint in the objective function will be a soft constraint, also called relaxed constraint.

3.2.5 Dynamic variations

As discussed in Section 2.3.5, the dynamic nature of scheduling problems is a major difficulty of this project. The implementation of dynamic variations will face three major concerns:

1. The level of rescheduling: when a validation test has been aborted, how many other tests can be scheduled? The higher the level of rescheduling, the higher the disturbance among the other scheduled tests. However, the level of global optimization will also be higher;

2. The choice of the method. If only the selected tests will be rescheduled, it will be a usual scheduling optimization method (GA, PSO, etc.) of the concerned tests with the other ones kept scheduled. If the whole scheduling is concerned, then a more suitable method for dynamic variations is prescribed, like an ant colony algorithm;

3. In function of the importance of a test, the rescheduling will vary. If a test is located on the critical path, a variation will imply a global variation of the whole validation process. Therefore, an additional method must be implemented, determining the critical path automatically and then adapting the rescheduling method.

The difficulty of the dynamic nature mainly lies in the disturbance allowed of the schedule due to the updates. If only small changes are acceptable in case of rescheduling, then a deterministic

algorithm like ACO is preferred, in order to provide almost the same schedule at constant parameters. Another possibility is to fix the tests that are not wanted to be rescheduled but it decreases the level of automation and optimality.

However, a dynamic scheduling is a sizeable work. Therefore, it will be the concern of a future thesis. Additional implementations are proposed in Section 6.5.

3.2.6 Multi-objective Optimization

Some optimization problems are more evolved and require to optimize several aspects of the problem. In this case, it can be relevant to optimize the compromise between the time and the cost, which can be two contradictory objectives. In order to do so, instead of optimizing either the global cost or the global duration, both can be optimized, leading to a two-objective optimization problem. The objective function will be a vector of fitness functions. Other objectives can be relevant and are discussed in Section 6.2.

Deb (2011) has defined the purpose of a multi-objective optimization method to provide the Pareto front of the two objective functions, as shown in Fig. 3.3 for a scheduling problem. The pareto front is the frontier in the solution space where a fitness function value cannot be improved without downgrading another one.

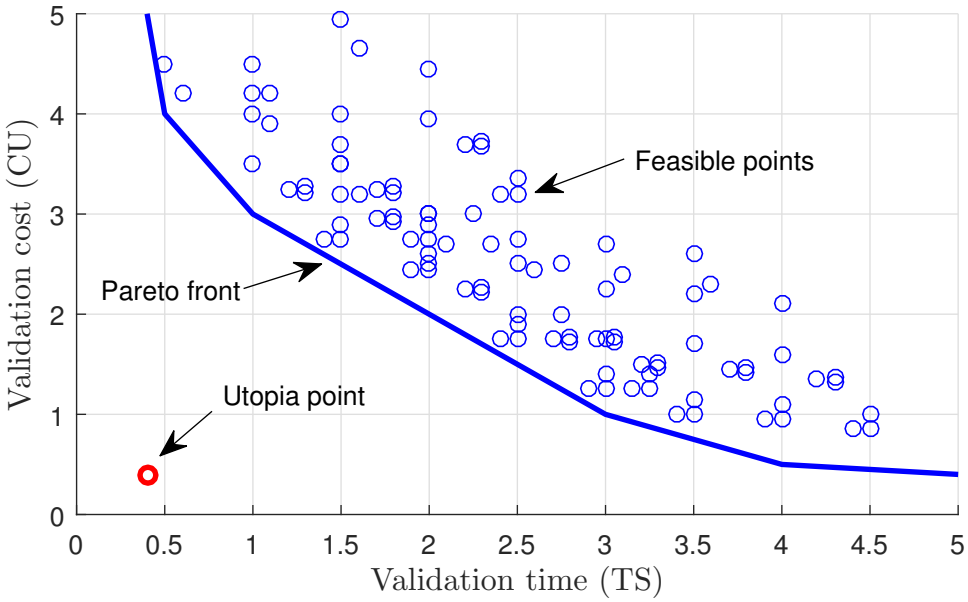


Figure 3.3: A Multi-Objective Optimization method intends to define the Pareto front.

Like basic optimization methods, the main parameters to tune are the definition of the elite individuals, to ensure simultaneously the best compromise between elitism and diversity. The *Pareto Fraction* determines the number of individuals on the Pareto front (the elite) while the *Distance* from the front preserves diversity by imposing the distance from the Pareto Front of the solution areas that the algorithm will consider.

3.3 Optimization Algorithms: Choice and Implementation

The choice of the method to solve a problem results from a compromise between *what is possible* and *what is the most suitable* solution. *What is possible* mainly depends on the solver, while *the most suitable solution* results from the literature review in Chapter 2. In practice, the critical condition of the chosen optimization method is to be able to deal with MINLP, which is a concept defined in Section 2.3.4.

The *Matlab* solver has a Genetic Algorithm implementation, `ga`, for which the difficulty is located in the fine-tuning of the parameters, such as the migration, the mutation or the crossover, and the problem description through constraints and an objective function. Additionally, `ga` implements a linear programming method to provide initial populations respecting the linear constraints, before optimizing them and making them respect non-linear constraints. The parallel computing made available by `ga` is also an asset for large problems like this one. However, this parallel computing can sometimes only compensate the slowness of spread algorithms (Weise, 2017), due to the assessment of large sets of solutions.

Following Weise again, GA is suitable for scheduling activities. Scheduling is generally the result of a constructive process from a manager. This process takes then place in a restrictive area of solutions, while GA algorithms will be an open process leading to solutions strongly different to what a human can provide. For these reasons, the GA method will be chosen first to implement the complete problem in order to have working scheduling solutions. This algorithm will not probably be the real best one, but its purpose is to be rapidly working.

Other algorithms are competitive for scheduling problems and MINLP, such as B&B, ACO, Simulated Annealing (Dréo 2011) or PSO. In particular, ACO have provided good results for scheduling problems (Merkle et al., 00) and its adaptation to MINLP, MIDACO, outperforms all other methods, including B&B (Schlueter et Al., 2012). However, not all methods have a compatible implementation with MINLP at the moment on Matlab, being only suitable for continuous problems. This is not the case here, due to binary variables (modes and exclusion variables) and discrete starting times. The choice is made to test the MIDACO solver to compare the results with the Matlab `ga` solver.

Both `ga` and MIDACO have a multi-objective optimization solver. For `ga`, it is called `gamultiobj`, while MIDACO is able to deal directly with multiple fitness functions. Unfortunately, no implementation for discrete problems has been found for the moment using `gamultiobj` but well for MIDACO. No comparison will be possible between both methods.

3.3.1 Implementation using GA

This section deals with the concrete implementation of the mathematical model using the `ga` solver. It will go through the mathematical aspects and provide a solution of implementation.

In the next chapter, the results of this implementations will be displayed using working examples. They will illustrate the potential suitability of this solver for large and complex scheduling problems.

Linear Constraints

The linear constraints are expressed under the matrix form:

$$\mathbf{A}\bar{\mathbf{x}} \leq \bar{\mathbf{b}} \tag{3.19}$$

where $\overline{\mathbf{x}}$ is the vector containing the variables, i.e., the modes, the starting times and the exclusion variables. $\mathbf{A}$ is the matrix of coefficients for linear constraints and $\overline{\mathbf{b}}$ is a vector containing linear constraints, such as precedences, due date(s), maximum budget or constraints on modes.

Non-linear Constraints

Non linear constraints are gathered in a vector of the type $\overline{\mathbf{c}} \leq \overline{\mathbf{0}}$. This vector contains the implementation of the Eq. 3.10, Eq. 3.14 or 3.13.

Fitness Functions

The fitness function is the function assessing each solution of the algorithm (population in the case of a genetic algorithm). It also express what has to be minimized, depending on the DVP objective:

- The cost, the fitness function is then the total cost, expressed in Eq. 3.18, a function whose variables are the modes;
- The validation duration, the fitness function is the finish time of the latest test to end, expressed in Eq. 3.16 a function whose variables are the modes and starting times.

GA parameters

The Genetic Algorithm can be adjusted via different kinds of parameters, defining the evolution of the populations, as explained theoretically in Section 2.3.3. The relevant parameters will be defined and the final choice will be presented in Section 4.2.1. Since the problem is constrained by non-linear and integer constraints, not all factors are available.

The crossover operation cannot be defined because the solver does not allow to modify it in case of integers, but the *crossover fraction*, i.e., the fraction of the population at the next generation, not including elite children, that is created by the crossover function. Concerning the mutation law, the *adaptive feasible* one is fixed for this case, leading to adaptive directions in function of past results but respecting linear constraints without any margins.

The *elite proportion* specifies how many individuals in the current generation are guaranteed to survive to the next generation. These individuals are considered as the elite. The validity of the parameters configuration comes from the good compromise between elitism and diversity of the chosen individuals. On the one hand, more elitism will lead to more difficulty to converge to a solution but this solution will be more probably the global optimum. On the other hand, more diversity will make the algorithm converge faster but the solution will be, statistically, less globally optimum.

The fitness function scaling converts the raw fitness scores that are returned by the fitness function to values in a range that is suitable for the selection function. There is a choice between a ranking, selecting an individual in function of its rank (more elitism), a proportion, selecting it in function of a proportional scaling (mix between elitism and diversity), or following a top scaling with an equal value of fitness for all best individuals (more diversity).

The size of the populations can be chosen, as well as the number of populations processing simultaneously. It permits to explore a larger area of solutions and, thanks to migration whose frequency and proportion can be defined, it also permits to mix different populations in order to converge more rapidly.

It is also possible to specify a parallel computing, other than the original one of Matlab, making a clever use of the available CPUs, and to vectorize the fitness functions and constraints to enhance the computational time. Finally, two different algorithms, Augmented Lagrangian and Penalty Method, are implemented to solve non-linear constraints. These two methods replace the constrained problem by a serie of unconstrained problems, while simultaneously a penalty is added to the fitness function (see the analogy with Section 3.2.4). The difference between them is the additional term used in the Augmented Lagrangian method enhancing the convergence.

3.3.2 Implementation using MIDACO

It appears (see Section 4.2) that the main performance of GA is to be capable of solving any kinds of optimization problem, but they are not the best solution to solve MINLP. Following the *No Free Lunch* theorem, it is more interesting to study a method renowned for its particular performance for this specific kind of problem, ACO, and its adaptation to MINLP in particular, MIDACO.

Constraints Definition

The MIDACO solver offers a particularly easy way to define the equality, inequality, linear and non-linear constraints. The N constraints are all gathered in the vector $\overline{\mathbf{g}} = [g_1, g_2, ..., g_N]$, so that $\overline{\mathbf{g}} = \overline{\mathbf{0}}$ or $\overline{\mathbf{g}} \leq \overline{\mathbf{0}}$. Additionally, MIDACO makes possible to define constraints not only with variables like

$$g_i = 3x_2 + 5x_5 \geq 0$$

but also with values defined previously such as the cost, the global duration, etc. It leads to constraints like the maximum budget constraint

$$g_i = Budget_{max} - C_{tot} \geq 0$$

defining the constraint on the total cost being lower the maximum allocated budget.

Fitness Functions

The fitness functions are defined in the same way as for the `ga` solver, using Eq. 3.18 and 3.16. However, multi-objective optimization is here possible, with the functions being defined in a vector, containing all fitness functions to minimize.

Additionally, there is a second problem that multi-objective optimization can address: the minimum starting time. Apart from minimizing the cost and/or the duration of the validation process, the user can specify that the DVP must start as soon as possible, leading to the minimizing of the global starting time S:

$$\min S = \min_{\forall i} s_i \tag{3.20}$$

Parameters

Schlueter and Munetomo (2016) have described MIDACO as a very powerful tool to deal with large sets of difficult constraints. In order to deal with large-scale complicated problems, it is advised to proceed by *cascade*, exploiting the following parameters at each step:

- The initial *Seed*, determining the sequence of sampling of random numbers from the generator. The interesting aspect of the seed is that, for a same seed, the solver can be run multiple time, it will provide reproducible result (similar to the stigmergy concept defined in Section 2.3.3);

- The *Focus*, especially for highly non-linear problems. The focus defines how large is the search area around the starting point. At the first step, it is interesting to set it to 0 so that it is absolutely free. Once a first (not necessarily globally optimal) solution is found, the focus can be increased to 100 or 1000, following the size of the solution space: the larger space, the higher focus effect, but it can lead to be stuck in a local minimum;

- The *Ants* and *Kernel* simultaneously and with great caution. MIDACO has a proactive approach concerning these two parameters by adapting them dynamically to the given problem, but the user can still fix them. *Kernel* defines the number of kernels in the PDF illustrated in Fig. 2.17. *Ants* is the number of ants MIDACO generates at each iteration.

Again, as in all optimization problems, the quality of the implementation will be in the compromise between diversity, synonym of convergence, and elitism, synonym of global optimum. Apart from these parameters, the *Character* of the problem defines the kind of faced problem, combinatorial in this case.

4. Results

The problem to address has been presented in Chapter 1. It consists in developing a tool dedicated to the gathering of the information concerning the validation process and scheduling the tests composing it. The methodology used to solve it has been presented in Chapter 3 and this chapter intends to present the results.

The results are divided into three main parts, resulting from the structure of the methodology, considering at the same time the interface and the algorithm supporting the tool:

- The mockups of the interface, illustrating the different menus used to collect the needed data and to schedule the validation tests;
- The (near-)optimal scheduling using the implementation of Genetic Algorithms, `ga`;
- The (near-)optimal scheduling using the implementation of Ant Colony Optimization, *MIDACO*.

The scheduling results will follow three steps of resolution: the basic scheduling problem (see Section 1.3.2), scheduling with allocation of facilities and adaptive constraints. Simultaneously, the parameters of each solver will be studied in order to propose improvements depending on the concerned problem.

Finally, the scheduling results will be discussed following the computational cost of finding a solution, the convergence (represented by the ability to find a feasible solution), the optimality and the respect of constraints. These measures of performance aim to highlight the suitability of each algorithm for the scheduling application of this work. The computed validation will be presented following *Time Slots* (TS) and *Cost Units* (CU) for, respectively, the total duration and cost of the validation process.

4.1 Optimization Tool Interface

The tool is presented as a user-friendly GUI, whose goal is to gather the information needed to create the different tests composing the DVP. It consists of four main menus and two support menus guiding the user along the definition of the requirements, tests cases and boundary conditions of the scheduling:

- The *Home* Menu, dispatching the user of the tool to three other menus;
- The *Create DVP/Add Test* Menu, used to define the tests and test cases;
- The *Update existing DVP* Menu, where tests will be updated for rescheduling purposes, in order to implement *dynamic variations*;

- The *Display DVP* Menu, used to display the DVP, under the form of a Gantt Chart, and data about tests;

- The *Test facilities* and *Test constraints* menus.

An example of a real GUI is provided as well and is described in Appendix B.1.1. It shows how the validation process begins, by selecting a requirements file and displaying their information.

4.1.1 GUI Main Menus

First, the *Home* Menu (see Fig. 4.1) directs the creator/user of the tool towards menus where he can either create a DVP (or add tests to an existing one), update an existing DVP or display a computed one.

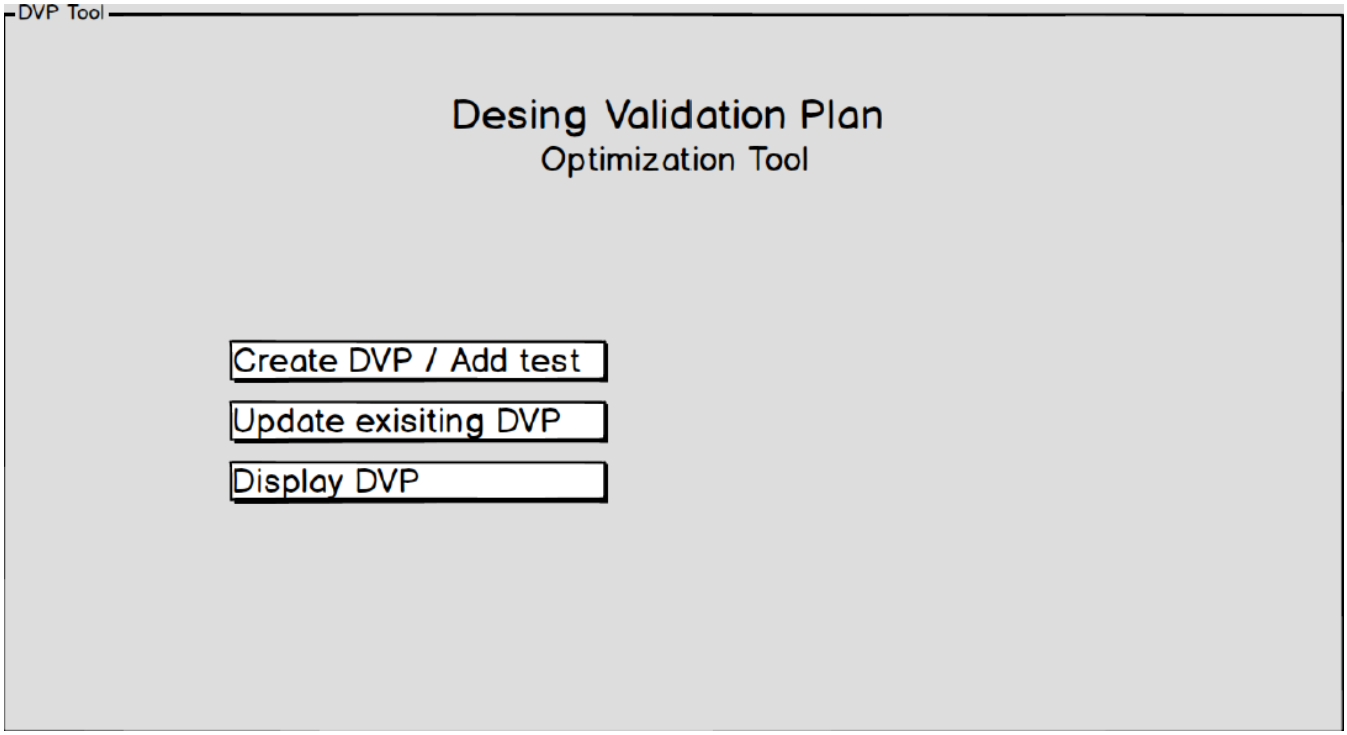


Figure 4.1: GUI *Home* Menu.

When the user must create a DVP or to add tests to an existing one, it begins by selecting the requirement(s) or category of requirements to validate in one test. The facility(ies) on which the test can be performed is selected from a file where a set of facility has been previously defined, with the corresponding modes. Then, the test cases are defined and the DVP or test can be created, with the choice to minimize either the cost or the duration. Finally, constraints can be added and the schedule can be displayed. This process is illustrated in Fig. 4.2.

The concept of dynamic variations has already been presented. These variations must be defined in the GUI, via the *Update* Menu shown in Fig. 4.3. If a case has been performed, it is efficient to enter the actual scheduling data and the actual cost mode. Additionally, if a test has not been completed or has failed, it must be deleted and replaced by a new one in the most convenient way, meaning without strongly disturbing all other tests.

Finally, the schedule is displayed following the most convenient manner, a Gantt chart, as discussed in Section 1.3.6. In addition to the schedule, this menu can also display the information concerning the test cases, defining the test procedures, the schedule and the modes selected by the optimization algorithm for each test.

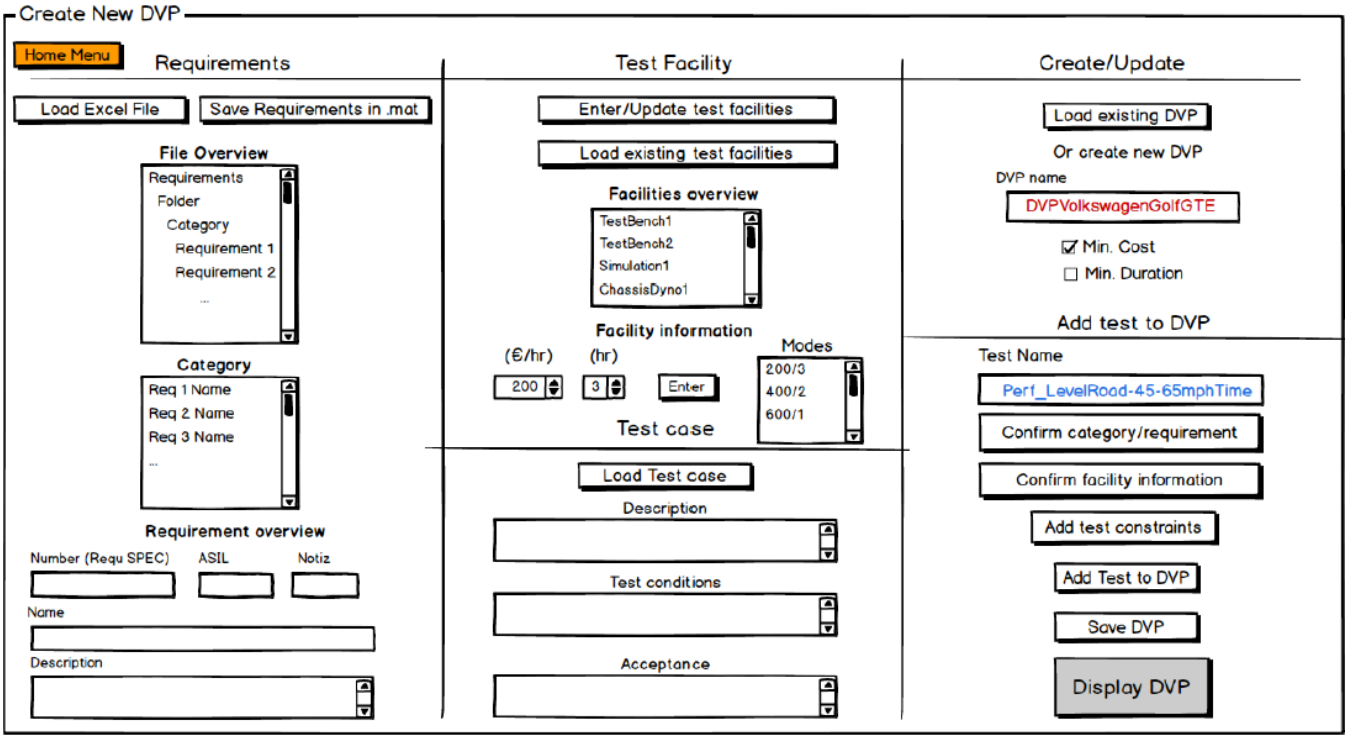


Figure 4.2: GUI *Create DVP/Add test* Menu.

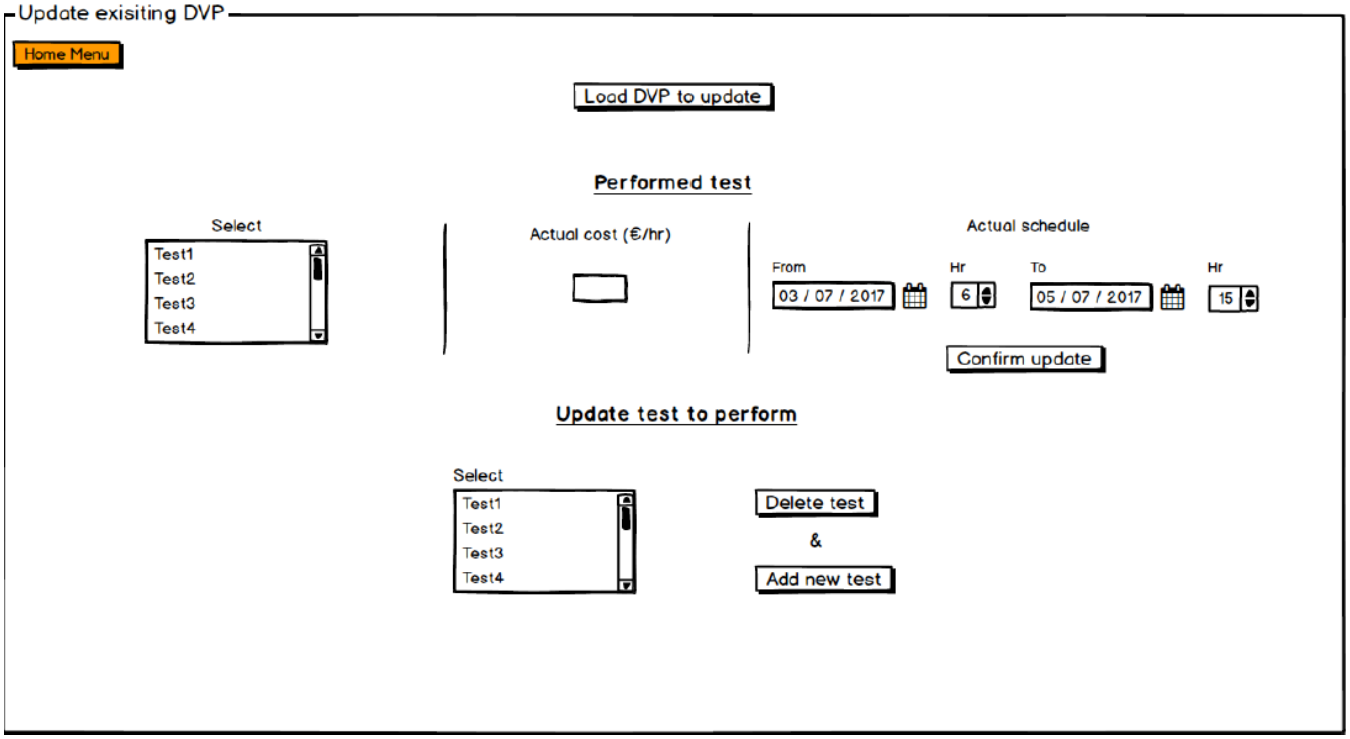


Figure 4.3: GUI *Update existing DVP* Menu.

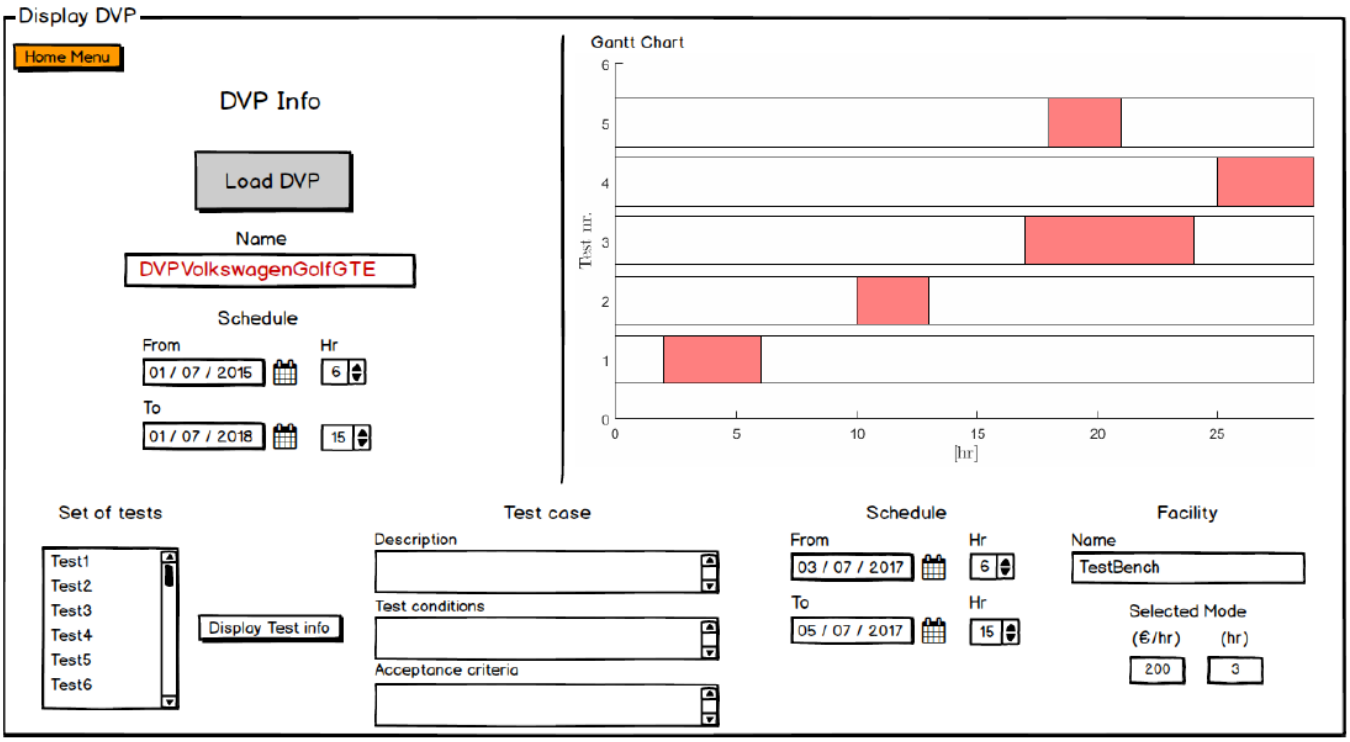


Figure 4.4: GUI *Display DVP* Menu.

4.1.2 GUI Support Menus

Each facility is added to an existing or new facility file, by entering its relevant information. The information will be the name of the facility, a description, the location etc. Moreover, the unavailability periods will be defined, where the facility is not available for other purposes than for validation tests. The GUI menu in which the facilities are described is shown in Fig. 4.5.

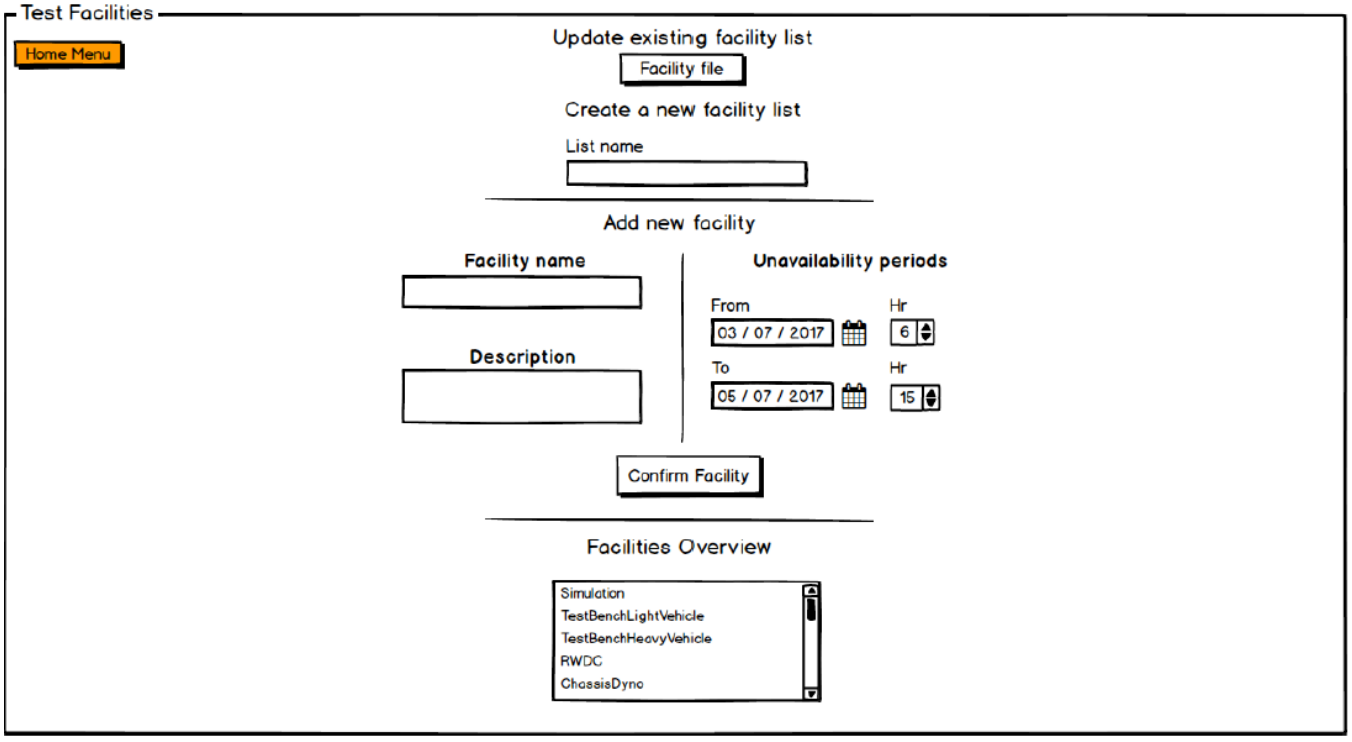


Figure 4.5: GUI *Test facilities* Menu.

Finally, the constraints on each test must be defined, as illustrated in Fig. 4.6. They cover precedences and maximum cost, if the validation duration is minimized. Certain tests have a personal due date, but most of them have the global due date of the validation. Consequently, if a concerned test does not have a maximum personal budget, it has the maximal budget proposed by its modes.

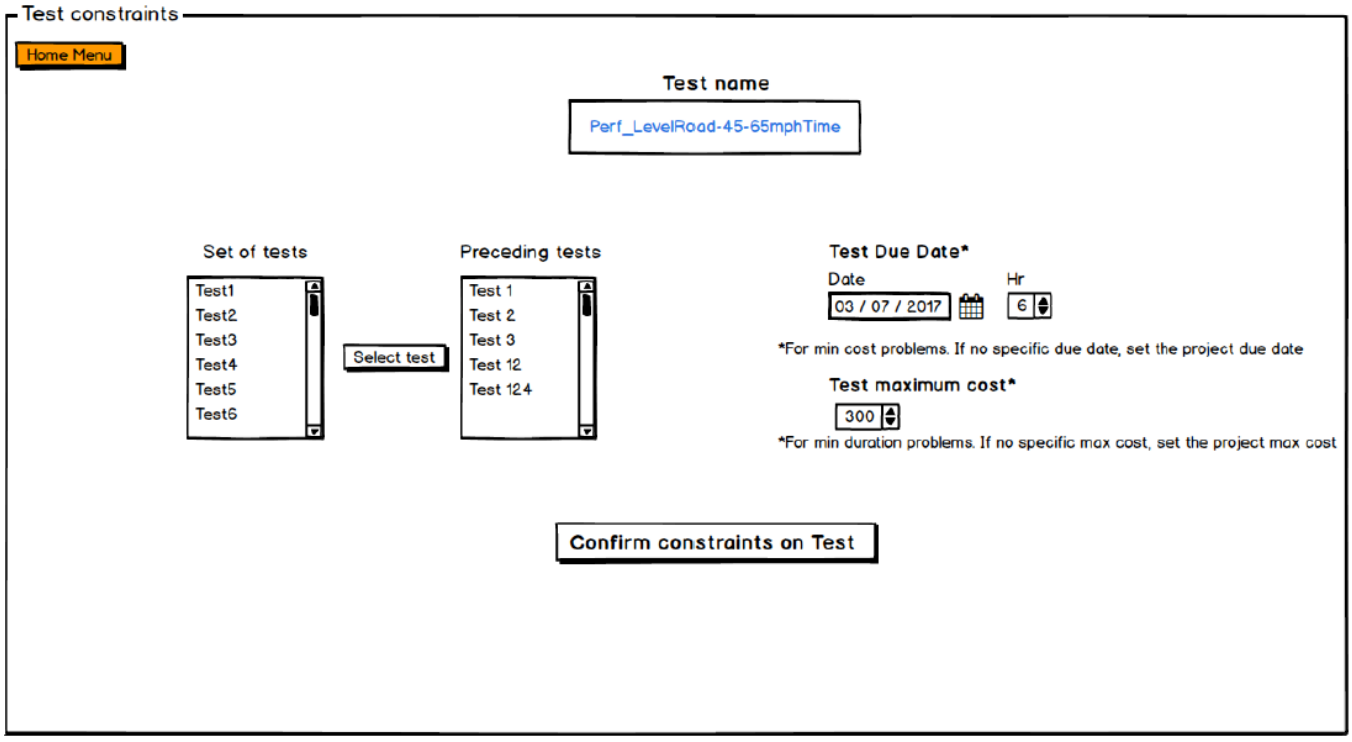


Figure 4.6: GUI *Test Constraints* Menu.

4.2 Scheduling Optimization by GA

The scheduling problem is the major concern of this work. It has been addressed deeply in Chapter 1 and the methodology used to solve it has been presented in Section 3.2. This section intends to display the results using the the implementation presented in Section 3.3.1 in three steps: without allocation of facilities, with allocation of facilities and with adaptive constraints.

For each case, an extension of a basic working example is used in order to highlight strengths and weaknesses of GA. At the same time, the influence of the parameters, the scalability (the ability to deal with big problems), the optimality (the deviation from the global optimum) and the convergence (the ability to find a feasible solution) is studied.

4.2.1 Scheduling Without Facility Allocation

The basic study, without facility allocation, consists in scheduling validation tests that only correspond to a unique facility. It gives the opportunity to study the performances of the GA algorithm for a simple example and to highlight the suitability of GA for simple scheduling applications. Additionally, this configuration will be used to study the scalability, i.e. the ability to schedule a high number of tests, of GA.

Working Example

In this case, the facility matrix is a $N \times 1$ vector, with only one facility per line, allocated to each one of N tests. One considers the following highly constrained 6-test and 4-facility example:

$$\mathbf{C}[\mathbf{CU}] = \begin{bmatrix} 200 & / \\ 200 & 100 \\ 250 & 150 \\ 300 & 150 \\ 400 & 200 \\ 350 & 200 \end{bmatrix} \quad \mathbf{D}[\mathbf{TS}] = \begin{bmatrix} 1 & / \\ 3 & 5 \\ 5 & 7 \\ 2 & 4 \\ 3 & 7 \\ 5 & 8 \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} 2 & 3 & 5 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\overline{F} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 3 \\ 4 \\ 2 \end{bmatrix} \quad U = \begin{bmatrix} 1 & \begin{bmatrix} 8 & 10 \end{bmatrix} \\ 3 & \begin{bmatrix} 15 & 17 \end{bmatrix} \\ 4 & \begin{bmatrix} 10 & 16 \end{bmatrix} \\ & \begin{bmatrix} 9 & 50 \end{bmatrix} \end{bmatrix} \quad \mathbf{B}[\mathbf{TS}] = \begin{bmatrix} 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \end{bmatrix}$$

It leads to complex characterization diagrams, as illustrated in Fig. 4.7.

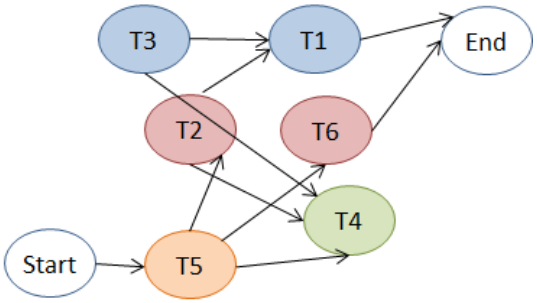


Figure 4.7: Characterization diagram of the working example.

GA parameters

The following parameters (explained in Section 3.3.1) result from a best compromise between computational time, convergence, width of the solutions area, elitism and diversity:

- The crossover fraction is set to 85% and the 35% fittest individuals (elite) are chosen at each generation and a top scaling is preferred;
- 5 subpopulations of 100 individuals each;
- Migration of individuals between populations every 5 generations, and 30% of the population migrates at each migration. A small interval of generations between migration makes possible to investigate promising areas more rapidly;
- The non-linear constraints are solved by the Augmented Lagrangian Method with an initial penalty factor of 100;
- The original parallel processing is more efficient. The vectorization of the function/constraint was not successful either. The given implementation is not concerned by the improvements proposed by parallelization, as explained by Weiss (2017).

They are satisfying in the convergence point of view, conditioned to the feasibility of the constraints, all seen as hard constraints in this current study. This is shown in Fig. 4.8, where the best individual of each generation of each subpopulation is represented by a blue dot. During the first generations, none of the subpopulations is able to find a valid solution and the value of the fitness function converges to zero. This phenomenon stops when the first efficient mutation/crossover happens going to a better solution area of the solution space. From this moment, each migration from other subpopulations will investigate this new promising area. The five subpopulations are clearly visible and the best solution area, 1550 CU, is privileged at the end, with all subpopulations located at this solution.

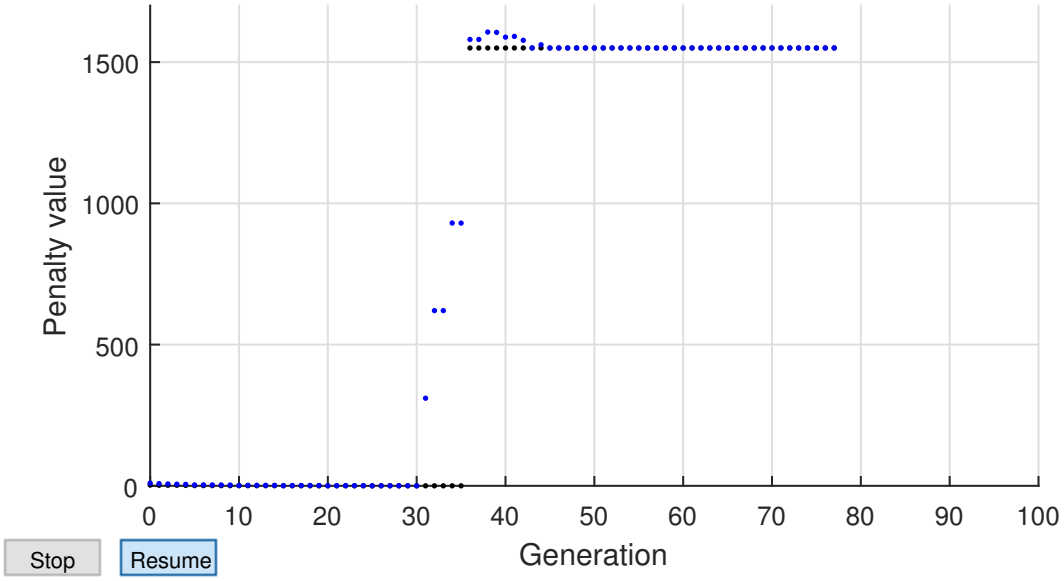


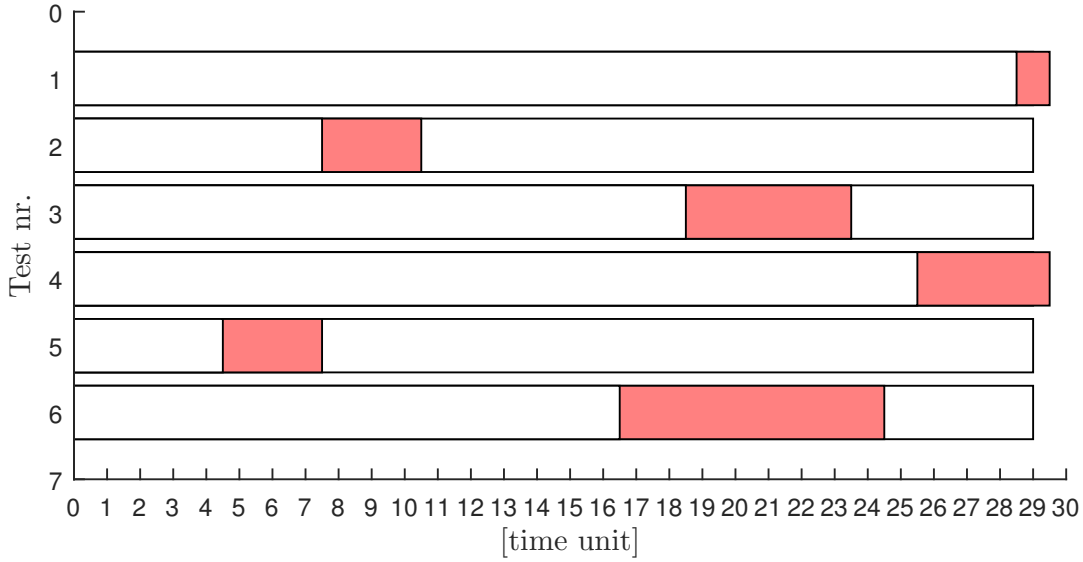
Figure 4.8: GA: Evolution of the best individuals of five subpopulations.

However, the results are not fully satisfying, for reasons that do not depend on the settings but more to the genetic algorithm itself. As illustrated all along this study, GA is slow, not particularly suited for this scheduling problem and has the tendency not to seek the global optimum.

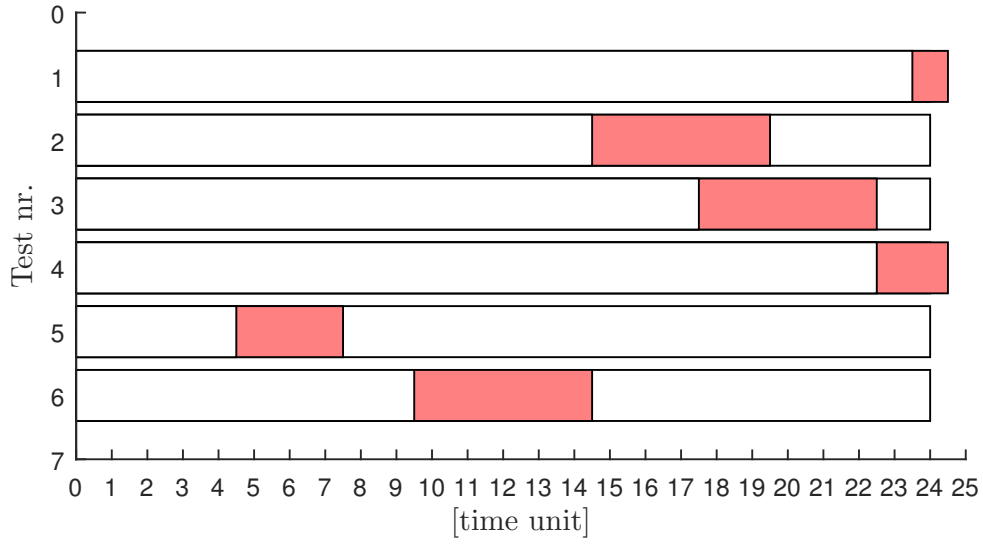
Results

First, the total cost is minimized. For this given example, the due date of each project will be the global due date of the validation project, 30 TS (the first slot being hour/day/quarter 1), seen as a hard constraint by the algorithm. The case where the global due date is seen as a soft constraint will be studied in Section 4.2.3. The result is displayed in Fig. 4.9a, with a total cost of 1400 CU and a duration of 29 TS.

Then, the minimization of the total duration is studied. The problem of optimality, i.e., the deviation from the global optimum, rapidly occurs: if no deadline is set for the same problem, the solution proposed by the algorithm is a validation in 70 TS, more than the duration for the cost minimization case (with $DD = 30$ TS). Due to this lack of optimality, the same due date, 30 TS, is imposed and it results in a validation requiring 24 TS (17% less time) and 1600 CU (14% more expensive), what is a logical result.



(a) Cost minimization, 1400 CU in 30 TS.



(b) Time minimization, 1600 CU in 24 TS.

Figure 4.9: The basic scheduling example using `ga` already highlights the suitability of the solver for this problem.

Besides the lack of optimality, the results provided by GA are probabilistic. The consequence is a scheduled validation that is not reproducible, even with the same parameters of resolution. The non-reproducibility is problematic for multiple reasons, and for dynamic variations in particular, as explained in Section 3.2.5.

Convergence and Optimality

Genetic algorithms were presented in Section 2.3.3 as methods making possible not to get stuck in local minima. To verify this assumption, the working example has been optimized 50 times in order to measure

- The number of attempts to find a feasible solution, representing the convergence ability.
- The mean distance from the global optimum, representing the optimality;
- The variability of the solutions, described by their standard deviation.

For the cost minimization of the basic working example, the global maximum is 1700 CU and the global minimum is 1000 CU (determined by MIDACO) in 29 TS.

The solution space is considered as sparse and constrained, due to large unavailability periods and short due dates. Over the 50 runs, the performances of the algorithm are mixed:

- 90% of the 50 trials converge at the first attempt;
- The mean deviation from the global optimum is $\overline{\Delta C} = 443.18$ CU (or 44.32 %);
- The standard deviation is $\sigma_C = 94.50$ CU.

This variation is due to the stochastic aspect of the heuristic approach but is not satisfying following three considerations. First, a large computational cost, what is experienced here, is only acceptable if the optimization result can be assumed as the global or, at least, a near-global minimum (44.32 % of deviation is not considered as near-optimal). Second, the deviation itself, from the global minimum, is significantly too volatile to be considered as reliable. Finally, the given six-test problem is very simple. this phenomenon is then supposed to increase with a higher number of tests, modes and constraints.

The performances are partially explained by the compromise between elitism of the designated individuals and the diversity of the considered areas of solution. The systematic convergence is an asset of this configuration and is due to the defined diversity. More elitism leads to a higher computational time, not acceptable for an example that simple, and less convergence leading sometimes to solutions non-respecting the constraints. Bartschi Wall (1996) has had the same problem of variation for heavily-constrained resources, as shown in Fig. 5.1a.

Computational Time and Scalability

Both cases need approximately 20 seconds, depending on the available CPUs. This amount of time is acceptable, yet it is supposed to increase significantly for a bigger number of tests, modes and especially more non-linear constraints.

Fig. 4.10 illustrates the scaling problem of Genetic Algorithms by comparing the computational time in function of the number of scheduled tests. The computational time increases quadratically between 3 and 30 tests, with 35% less time on average when the non-linear constraints are not considered. For non-linear constraints concerns, the proportion between linear and non-linear constraints

has been kept constant in order to compare like with like. Following the quadratic fitting, for 1000 tests, the computation would take 22 hours, without taking facility allocation into account, synonym of additional modes, and/or adaptive constraints.

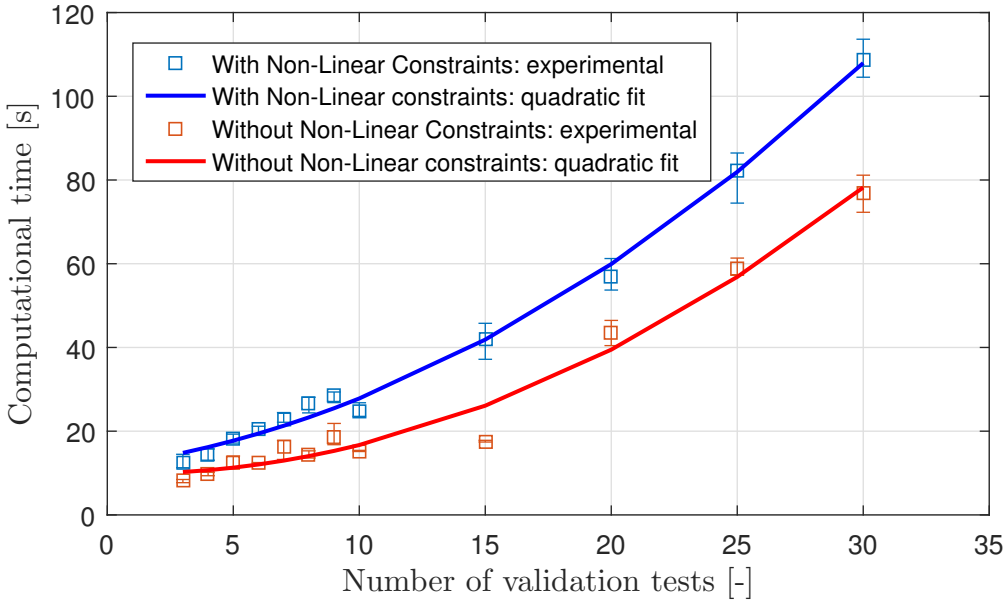


Figure 4.10: The computational time using `ga` increases quadratically.

Additionally, the convergence becomes uncertain starting from 20 scheduled tests, increasing with higher numbers of tests, with several attempts needed regardless of the parameters. The bad scaling of this scheduling problem can be explained by this non-exhaustive list of reasons:

1. The Genetic Algorithm is intrinsically slow (Weise, 2017), due to the fitness assessment of all individuals and due to the mutation process;
2. Combinatorial problems are generally slower than other kinds of problems;
3. NP-hard problems like this one are known to have an exponential execution time for most solvers;
4. The presence of non-linear constraints slows the optimization process considerably. This phenomenon is visible thanks to the linear programming implementation of `ga` used to solve the linear constraints rapidly prior to the non-linear ones.

Starting Times Problem

Scheduling by retro-planning, i.e., setting a due date rather than a starting date, is applied in this work. However, as illustrated in Fig. 4.9a and Fig. 4.9b, the schedule does not start as soon as it can. It is not contradictory to minimizing the objective: the start date is not always linked to the end date or total cost, but it is not the most suitable solution in practice.

The first solution could be to shift the planning towards the first time slot. However, this is not always possible because of the unavailability periods. The second possibility would be a multi-objective optimization, minimizing in the same time the total cost/duration and the beginning of the schedule. The multi-objective optimization is discussed in Section 3.2.6.

4.2.2 Scheduling With Facility Allocation

The problem of allocating different facilities to a same test has been modeled in Section 3.2.3. In practice it leads to additional modes, but these modes do not all correspond to the same facility. Moreover, it leads to additional non-linear constraints, conditioned to the fact that the facility is chosen, as expressed in Eq. 3.14 and 3.13. Mathematically, it results in an even sparser and more restricted solution space.

Working example

The same six-test problem is studied, extended to a second facility for three of the tests to schedule. This choice comes from the need for comparison and to ensure the feasibility of the problem.

$$\mathbf{C_1}[CU] = \begin{bmatrix} 200 & / \\ 200 & 100 \\ 250 & 150 \\ 300 & 150 \\ 400 & 200 \\ 350 & 200 \end{bmatrix} \quad \mathbf{C_2}[CU] = \begin{bmatrix} 300 & 100 \\ 350 & 150 \\ 200 & / \\ / & / \\ / & / \\ / & / \end{bmatrix} \quad \mathbf{D_1}[TS] = \begin{bmatrix} 1 & / \\ 3 & 5 \\ 5 & 7 \\ 2 & 4 \\ 3 & 7 \\ 5 & 8 \end{bmatrix} \quad \mathbf{D_2}[TS] = \begin{bmatrix} 1 & 3 \\ 1 & 3 \\ 6 & / \\ / & / \\ / & / \\ / & / \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} 2 & 3 & 5 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} 1 & 3 \\ 2 & 1 \\ 1 & 4 \\ 3 & 0 \\ 4 & 0 \\ 2 & 0 \end{bmatrix} \quad U = \begin{bmatrix} 1 & \begin{bmatrix} 8 & 10 \\ 15 & 17 \end{bmatrix} \\ 3 & \begin{bmatrix} 10 & 16 \end{bmatrix} \\ 4 & \begin{bmatrix} 9 & 50 \end{bmatrix} \end{bmatrix} \quad \mathbf{B}[\mathbf{TS}] = \begin{bmatrix} 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \end{bmatrix}$$

This example is intentionally the same on the first layer as the one presented in Section 4.2.1. The difference comes from Tests 1, 2 and 3, which benefit from an additional two-mode facility. The precedences do not vary, as well as the unavailability periods of facilities.

Result

The cost minimization is illustrated in Fig. B.3, available in Appendix B.2.1. The computation lasts 115 s (almost 600 % increase compared to single-facility tests) for a computed cost of 1250 CU (12.5 % of deviation from the optimum). However, a problem of convergence rapidly occurs: the algorithm must run six times before getting this solution, even with a greater diversity (elitism of 35% of the fittest individuals).

Concerning the global duration minimization, the same problem occurs with the need of nine attempts to converge the solution illustrated in Fig. B.4. As seen previously, the global optimality of the solver is not satisfying: without due date, the minimum global duration found is 65 TS, which is far from the 28 times slots here presented.

These results are unsatisfying following three reasons:

- The convergence is not ensured anymore, what would be acceptable only if the solution was the (near-) global optimum. This is not the case;
- Neither more diversity, nor more elitism helps to achieve a better convergence or optimality;
- This phenomenon is supposed to increase with higher scale and more complicated problems, this problem being particularly simple.

The problems raised in Section 4.2.1 are still present with the facility allocation and are even amplified. It shows that GA deals badly with higher degrees of freedom. These results are contradictory to the study of Bartschi Wall (1996), in which the multi-modal scheduling performs well, even helping the algorithm to converge by the addition of possible solutions.

4.2.3 Adaptive Constraints

As described in Section 1.3.3, it can be interesting to define the different due dates (intermediary and global ones) as soft constraints, meaning that they can be missed in return for additional cost. However, constraints relaxation means a wider and less sparse solution space, with costlier areas where it was previously restricted. With an optimum-seeking method, this feature has no effect since the global optimum remain unchanged.

Working Example

The problem characterized in Section 4.2.2 is used again with an adapted global due date, $\overline{DD}$, and **B** matrix:

$$\mathbf{B}[TS] = \begin{bmatrix} 1 & \infty & 0 \\ 1 & \infty & 0 \\ 1 & \infty & 0 \\ 1 & \infty & 0 \\ 1 & \infty & 0 \\ 1 & \infty & 0 \end{bmatrix} \quad \overline{DD}[TS] = \begin{bmatrix} 30 & 5000 \end{bmatrix}$$

The choice is made to define all tests in the same unconstrained interval, $[1,\infty[$ TS, without weighting factors. However, the global due date remains 30 TS, like the previous studied example (known to be feasible), with an additional cost of 5000 CU per time slot. This means that the total cost will be 5000 CU higher for every time slot higher than 30 for the global validation process. This prohibitive value has been chosen to highlight the optimality/convergence of the solver. The feasibility of this problem has been proven in Section 4.2.2.

Results

The solution space being wider, but also bumpier (particularly in terms of cost), the results can be discussed in three different ways, implementing

- More diversity (elite fraction of 35% of the population), supposed to converge more rapidly, but being not globally optimal, leading sometimes to the non-respect of the constraints. This value of elite fraction was used in the previous problem and it made possible to solve this problem respecting all the constraints;

- More elitism (elite fraction of 5%), supposed to show problems of convergence, but being more globally optimal, avoiding costly solutions;

- A compromise between elitism and diversity (elite fraction of 15%).

Elitism and diversity can be managed via other options but the elite fraction is the most effective one. The solutions are summarized in Table 4.1 and illustrated in Fig. 4.11.

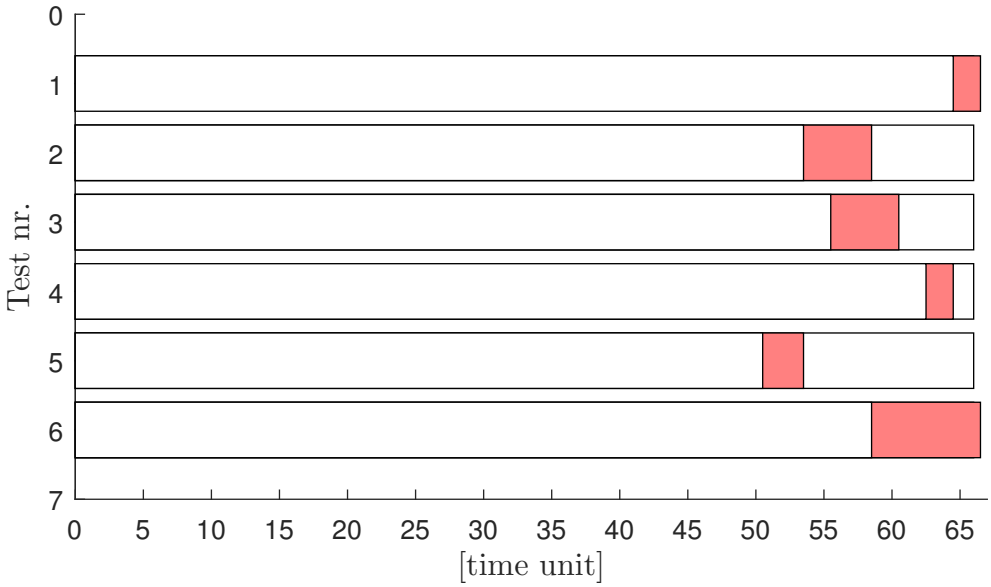
Elite Fraction [%]	Attempts [-]	Comput. Time [s]	Validation Cost [CU]	Validation Time [TS]
35	1	103	186550	68
5	1	173	171500	63
15	2	66	**1300**	**27**

Table 4.1: GA adaptive constraints study - A mix between diversity and elitism provides very encouraging results.

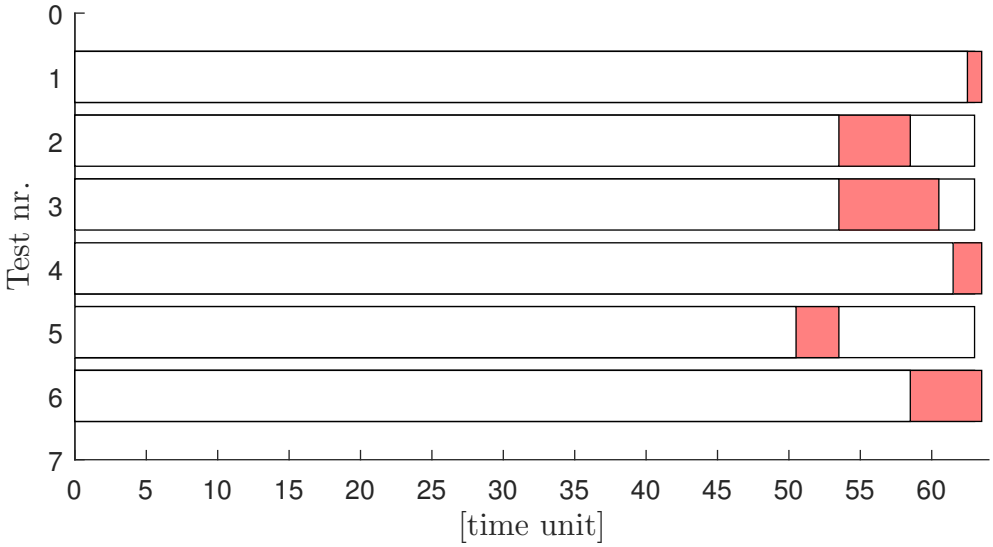
Relaxing the constraints leads to a wider and less sparse solution space. As expected, the convergence gets improved, requiring only two attempts. In case of strong elitism, the convergence requires a lot of computational time, due to the bad scalability of GA, and it shows the tendency to get stuck in local minima (non-respecting the constraints). In case of strong diversity, the algorithm converges more rapidly, but it leads to a larger additional cost. In both cases, the algorithm takes the easy choice of going out the unavailability periods so that it only has to respect precedences.

However, a compromise between diversity and elitism gives satisfying results respecting the constraints. In addition, the computational cost, convergence and optimality are satisfying. This result is particularly interesting since there were convergence problems with the previous studied case with hard constraints, even with several diversity/convergence parameters. GA is then assumed to "jump" badly in the solution space, what is a challenge of scheduling problems (Bartschi Wall, 1996), as discussed in Section 2.3.5.

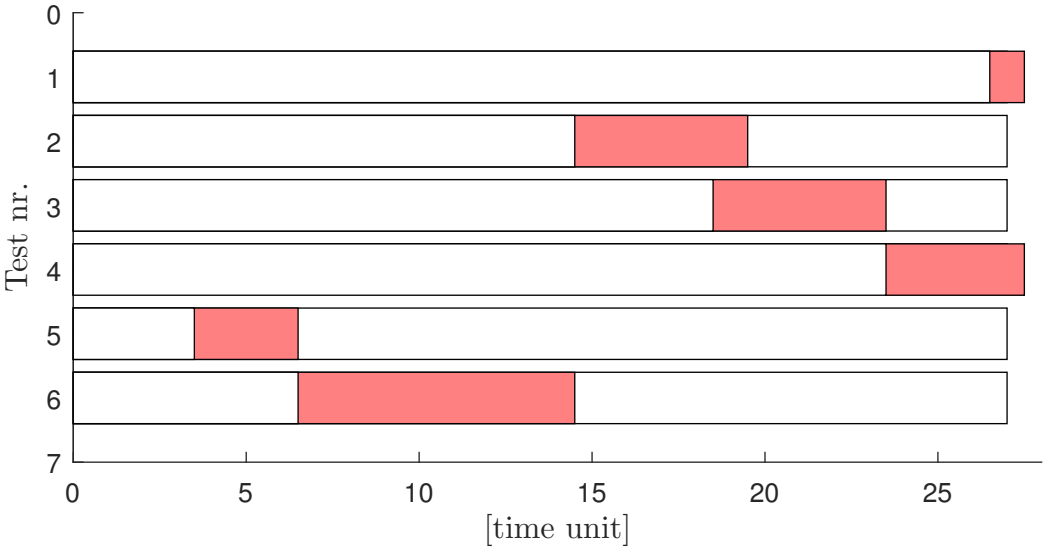
To conclude, this problem highlights the importance of having a very fine tuning of the parameters in order to benefit both from global optimality and convergence ability. This tuning fully depends from the problem and requires then the habit from the user and some attempts to get a satisfying result. Additionally, relaxing the constraints is a good solution to help the algorithm to converge, conditioned, again, to the good tuning of the parameters.



(a) Elite Fraction of 35%.



(b) Elite Fraction of 5%.



(c) Elite Fraction of 15%.

Figure 4.11: GA adaptive constraints study - A mix between diversity and elitism provides very encouraging results.

4.3 Scheduling optimization by MIDACO

This section intends to display the scheduling results using MIDACO. As done for GA, the results will be divided into three parts: scheduling without facility allocation, with facility allocation and with adaptive constraints. The methodology has been presented in Section 3 and the implementation is described in Section 3.3.2.

MIDACO being more powerful than `ga`, the following additional subjects can be studied: cascade optimization, optimality in function of the computational cost and multi-objective optimization. These additional features are included in the three studies except from the multi-objective optimization, which is subject to dedicated study.

Additionally, since MIDACO is compatible with multi-objective optimization, each optimization minimizes the global cost and/or duration and the starting time simultaneously.

4.3.1 Scheduling Without Facility Allocation

Scheduling without allocating facilities is a good introduction to the optimization performances of an algorithm. It makes possible to measure its ability to scale in terms of number of tests while keeping the number of modes limited. Additionally, the different features offered by the solver can be explored. The next point of interest will be the implementation of facility allocation.

Working Example

For comparison purposes, the same working example is used than in the GA study, without facility allocation. It consists in six tests, corresponding to only one of the four available facilities and each of them must be completed before 30 TS. A summary is available hereunder:

$$\mathbf{C}[CU] = \begin{bmatrix} 200 & / \\ 200 & 100 \\ 250 & 150 \\ 300 & 150 \\ 400 & 200 \\ 350 & 200 \end{bmatrix} \quad \mathbf{D}[TS] = \begin{bmatrix} 1 & / \\ 3 & 5 \\ 5 & 7 \\ 2 & 4 \\ 3 & 7 \\ 5 & 8 \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} 2 & 3 & 5 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\overline{F} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 3 \\ 4 \\ 2 \end{bmatrix} \quad U = \begin{bmatrix} 1 & \begin{bmatrix} 8 & 10 \\ 15 & 17 \end{bmatrix} \\ 3 & \begin{bmatrix} 10 & 16 \end{bmatrix} \\ 4 & \begin{bmatrix} 9 & 50 \end{bmatrix} \end{bmatrix} \quad \mathbf{B}[TS] = \begin{bmatrix} 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \end{bmatrix}$$

Results

While `ga` is unable to reach the global optimum, 1000 CU for the cost minimization, MIDACO is able to determine it at a higher computational cost, around 100 s (compared to 20 s for `ga`). However, it is capable to determine a local minimum, 1550 CU, in only 5 seconds. The influence of the optimality on the computational cost is the concern of the next study. The globally optimal schedule is displayed in Fig. 4.12.

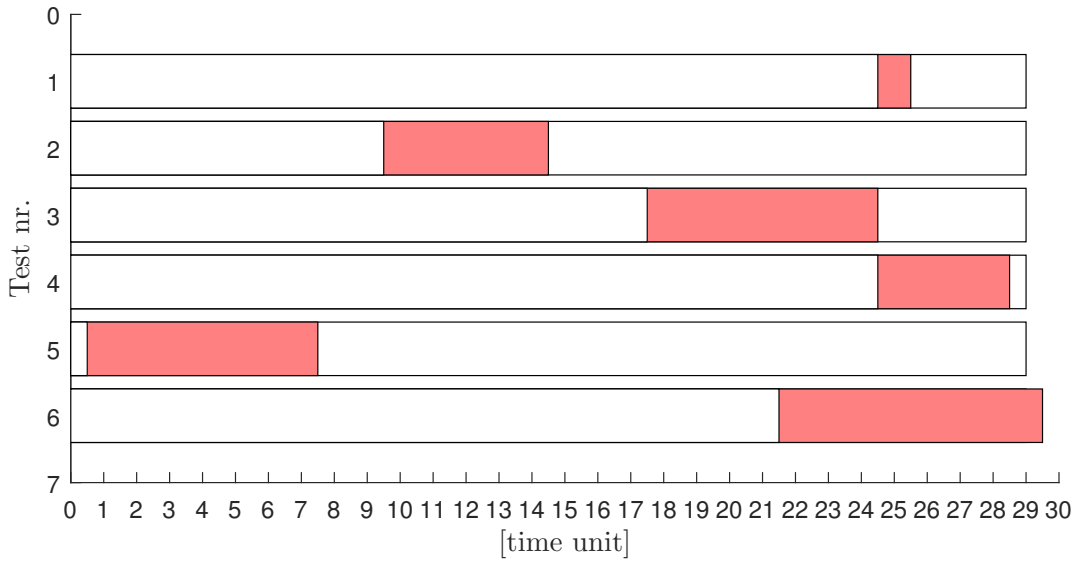


Figure 4.12: Global optimum (cost minimization) of the working example using MIDACO, 1000 CU in 29 TS.

Computational Time

First, the working example is solved using different stopping criterion values in order to evaluate the optimality, represented by the deviation from the global optimum in function of the computational time. In the mean time, the influence of the seed, i.e., the sequence of random number generated by the solver, on the computational time can be studied. During one scheduling optimization, the same seed can be used at each ACO run, providing reproducible results for a same seed and a same computer, respecting the stigmergy principle defined in Section 2.3.3. This last remark was one of the main advantages of ACO for scheduling problems following Dréo et al. (2011).

As shown in Fig. 4.13, the closer to the global optimum, the higher computational time is needed for the MIDACO solver. However, this additional time increases drastically near to the global optimum.

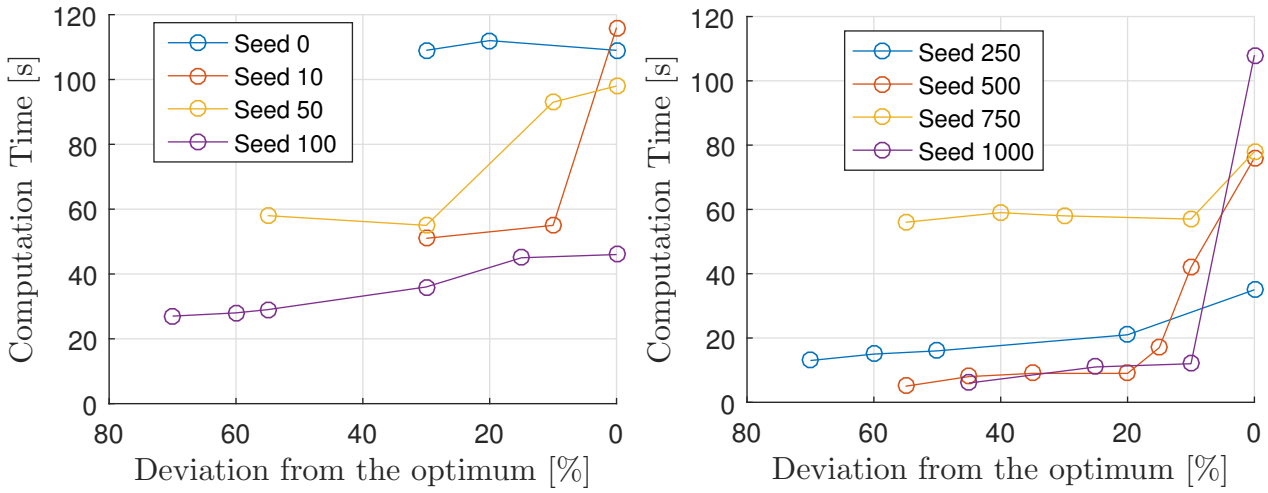
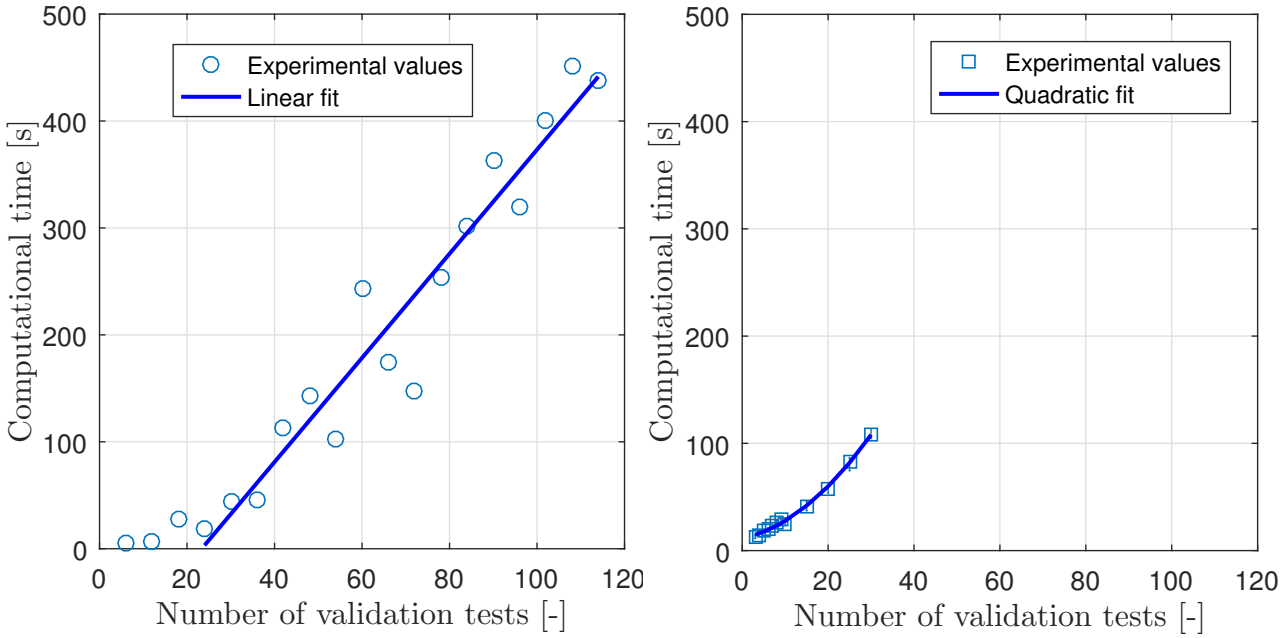


Figure 4.13: MIDACO optimality, computational time in function of the deviation from the global optimum.

Moreover, the effect of the seed is threefold:

- At higher seeds, the number of working combinations is higher, demonstrating a better sampling of the solution space for high seed parameters;
- The computational time for non-global optimum is lower at high seeds, but significantly higher at smaller seeds (except for 750). The opposite is true for the global optimum: MIDACO performs faster for lower seeds. Between the global maximum (70%) and 10% of deviation from the global optimum, there is a plateau effect for which the computational cost remains unchanged despite the increase of optimality;
- There is a specific seed value, 250 in this case, for which MIDACO performs well for any desired optimality, having a plateau effect from 70% to 10%.

Then, the scalability of MIDACO is measured in the same way the GA study did. Therefore, the computational time is measured in function of the number of scheduled tests until the first convergence, using a seed of 500. Again, the proportion between linear and non-linear constraints has been kept constant as well as the level of constraints on resources. Fig. 4.14a illustrates the scalability from 6 to 114 tests to schedule. Note that only one run for each scheduling is needed and the results as well as the computational time are fully reproducible.



(a) MIDACO: Linear relation and good scalability.

(b) GA: Quadratic relation and bad scalability.

Figure 4.14: Scalability of the two studied optimization algorithms: Computational time in function of the number of tests to schedule.

Contrary to GA, MIDACO shows a linear relation between the number of tests and the computational time, except for low numbers of tests where there is no big influence. Moreover, the convergence is satisfying, with only one attempt needed, no matter the number of tests. It explains the high number of tests in Fig. 4.14a compared to Fig. 4.14b (scaled version of Fig. 4.10). Thanks to this good scalability, scheduling 1000 tests is now feasible and would require 80 minutes to converge to a feasible solution. However, this value is theoretical since it does not include facility allocation, resulting in additional modes, and adaptive constraints.

Multi-Stage Optimization

MIDACO implements an easy way to perform a multi-stage optimization (also called optimization in cascade) in order to decrease the required computational time. The idea is, instead of using a constant compromise between diversity and elitism, to begin the research with a high degree of diversity. Once, it converges to a solution, one increases the elitism while decreasing the diversity to investigate a promising area. The process is composed of three steps described below:

1. Using the lower bounds as initial solution with a focus of 0;
2. Getting a first solution, respecting the constraints but not being the global optimum;
3. Using that solution as initial solution and running the algorithm with a higher focus, 100 or higher, to decrease the global computation time.

The same six-tests working example, with a seed of 1000 is solved. As shown in Fig. 4.15, the effect of solving the problem in cascade is powerful to improve the computational cost for finding the global optimum, 386% lower than in a single-step execution. The seed value can also be adapted during the cascade in order to vary the sampling of the solution space.

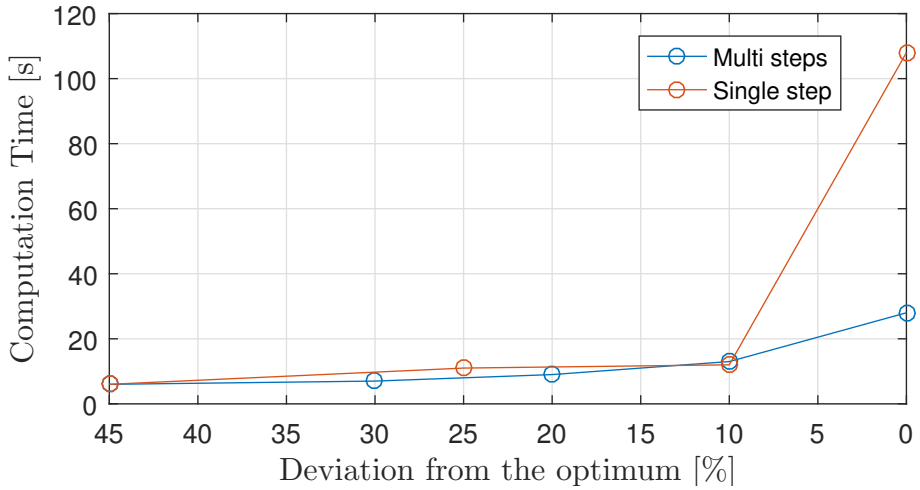


Figure 4.15: The multi-stage optimization decreases the computational cost to determine the global optimum (seed 1000).

4.3.2 Scheduling With Facility Allocation

MIDACO has revealed to be an efficient tool to solve highly-constrained and/or big-scaled problems. However, as discussed in Section 5.2.1, a high number of modes can have a negative or positive impact on the performances of a solver. This section intends to evaluate the performances of MIDACO to deal with additional modes via the allocation of multiple facilities for certain tests.

Working Example

The same working example as for the GA study is used for comparison purposes. The level of constraints remains the same as the previous example, but additional modes are introduced in order to put forward the ability of MIDACO to deal with additional degrees of freedom, represented by the facility allocation.

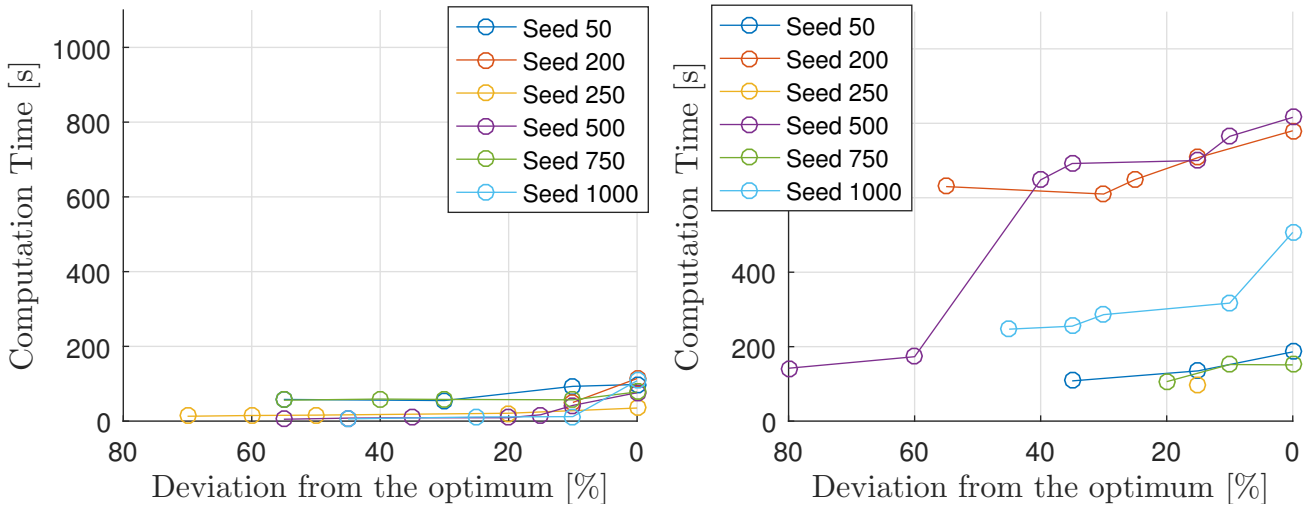
$$\mathbf{C_1}[CU] = \begin{bmatrix} 200 & / \\ 200 & 100 \\ 250 & 150 \\ 300 & 150 \\ 400 & 200 \\ 350 & 200 \end{bmatrix} \quad \mathbf{C_2}[CU] = \begin{bmatrix} 300 & 100 \\ 350 & 150 \\ 200 & / \\ / & / \\ / & / \\ / & / \end{bmatrix} \quad \mathbf{D_1}[TS] = \begin{bmatrix} 1 & / \\ 3 & 5 \\ 5 & 7 \\ 2 & 4 \\ 3 & 7 \\ 5 & 8 \end{bmatrix} \quad \mathbf{D_2}[TS] = \begin{bmatrix} 1 & 3 \\ 1 & 3 \\ 6 & / \\ / & / \\ / & / \\ / & / \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} 2 & 3 & 5 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} 1 & 3 \\ 2 & 1 \\ 1 & 4 \\ 3 & 0 \\ 4 & 0 \\ 2 & 0 \end{bmatrix} \quad U = \begin{bmatrix} 1 & \begin{bmatrix} 8 & 10 \end{bmatrix} \\ 3 & \begin{bmatrix} 15 & 17 \end{bmatrix} \\ 4 & \begin{bmatrix} 10 & 16 \end{bmatrix} \\ & \begin{bmatrix} 9 & 50 \end{bmatrix} \end{bmatrix} \quad \mathbf{B}[TS] = \begin{bmatrix} 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \\ 1 & 30 \end{bmatrix}$$

Results

The computational time in function of the optimality is illustrated in Fig. 4.16, with different seed parameters. Fig. 4.16a displays the results already discussed in the previous section, compared the results for facility allocation in Fig. 4.16b.

The additional computational cost resulting from the additional modes is significant, from 190% to 470%, depending on the seed. Second, the computational time, like in the previous study, strongly increases when determining the global optimum compared to local optima. However, the most powerful seed values are different, 750 for this example. Moreover, the number of determined solutions, representing the quality of the sampling of the solution space, does not necessarily increase with higher seed. It proves that the seed values are problem-dependent, especially on the number of modes.



(a) Without facility allocation.

(b) With facility allocation.

Figure 4.16: MIDACO performances for facility allocation - Computational time in function of the deviation from the global optimum.

Multi-stage Optimization

The computational cost can be decreased by using a multi-stage optimization. The global optimum can be easily achieved with a computational time decreased of 18% compared to a one-step optimization. The parameters used are summarized in Table 4.2.

Focus [-]	Seed [-]	Validation Cost [CU]	Deviation from optimum [%]	Computational Time [s]
0	50	1350	35	115
10000	50	1300	30	131
10000	50	1250	25	258
100000	500	1150	15	397
10000	50	1000	0	416

Table 4.2: MIDACO multi-stage optimization for facility allocation.

The scalability and the influence of the modes have been studied in the previous sections. The effect of a larger space solution will be the next point of interest.

4.3.3 Adaptive Constraints

As discussed during the GA study, relaxing the constraints widens consequently the solution space. Therefore, the effect on a solver can be either positive or negative, depending on its ability to determine the global optimum and its ability to converge rapidly. This will be the point of this section, followed by a multi-objective optimization.

Working Example

For comparison purposes, the same six-test example is used, with two possible facilities to allocate for three tests. As studied for GA, the tests constraints on the tests are, first, fully relaxed, being scheduled in the range $[1,\infty[$. Additionally, a partial relaxation, between $[1,50[$, is also studied. In both cases, the global due date, 30 TS, remains valid but with a prohibitive additional cost of 5000 CU per late time slot:

$$\mathbf{B}[TS] = \begin{bmatrix} 1 & \infty & 0 \\ 1 & \infty & 0 \\ 1 & \infty & 0 \\ 1 & \infty & 0 \\ 1 & \infty & 0 \\ 1 & \infty & 0 \end{bmatrix} \quad \text{or} \quad \mathbf{B}[TS] = \begin{bmatrix} 1 & 50 & 0 \\ 1 & 50 & 0 \\ 1 & 50 & 0 \\ 1 & 50 & 0 \\ 1 & 50 & 0 \\ 1 & 50 & 0 \end{bmatrix} \quad \text{and} \quad \overline{DD}[TS] = \begin{bmatrix} 30 & 5000 \end{bmatrix}$$

Results

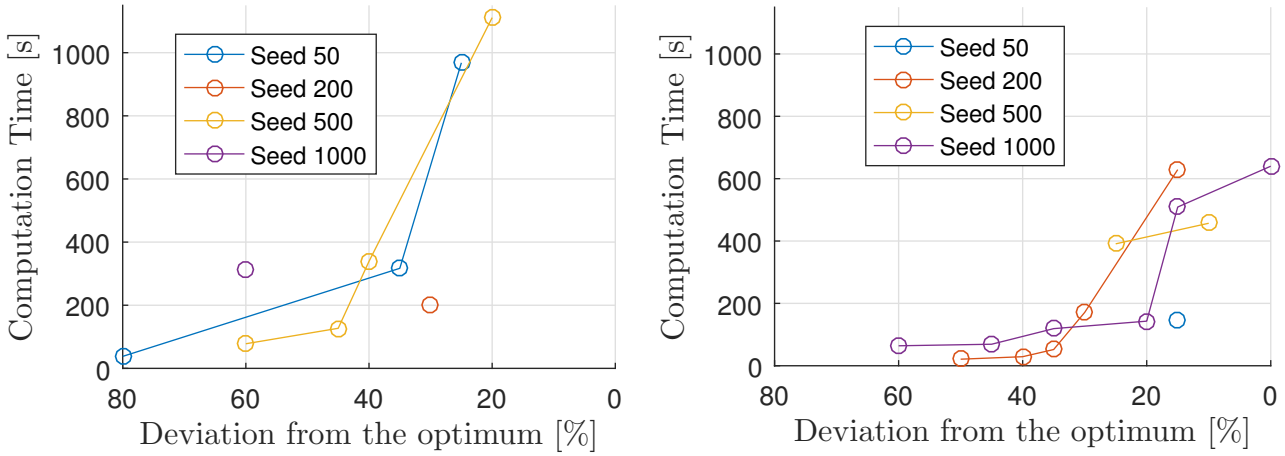
First, relaxing the constraints has a positive effect on the computational cost for direct convergence, i.e., first solution respecting the hard constraints, which is not the case for the global due date. As expected, the larger the relaxation, the lower the computational cost and vice versa:

- The full relaxation requires only three seconds but the computed validation duration is greater than 10 000 TS;

- The partial relaxation requires 11 seconds but plans the validation in 47 TS.

These solutions not respecting the global due date, this low computational cost must be weighed with the due date respect case. When the due date is respected, illustrated in Fig 4.17a, a problem of convergence is observed in the full constraint relaxation case (non-convex problem). For any seed values, it is not possible to determine the global optimum, but even most of local optima observed in Fig. 4.13 are not determined. The few ones found by the algorithm require a high computational cost.

On the contrary, when the relaxation of constraints is partial, shown in Fig. 4.17b, the computational time is consequently lower. the convergence and the sampling of the space are enhanced. However, it is still worse than for hard constraints. It proves that MIDACO has no difficulty to "jump" from area to area, which was one of the main difficulties of scheduling problems (Bartschi Wall, 1996) presented in Section 2.3.5.



(a) Full constraints relaxation.

(b) Partial constraints relaxation.

Figure 4.17: Relaxing the constraints leads to a bad convergence of MIDACO.

Multi-stage Optimization

In order to improve the computational cost of determining the solutions, a better sampling of the solution space can be executed, using the multi-stage optimization. This optimization process is completely described in Table B.2 and B.1 in Appendix B.3.1, respectively for the partial and full relaxation case. It is also summarized in Fig. 4.18, taking the hard-constrained case into account. From the three cases, the partial relaxation appears to be the most efficient one.

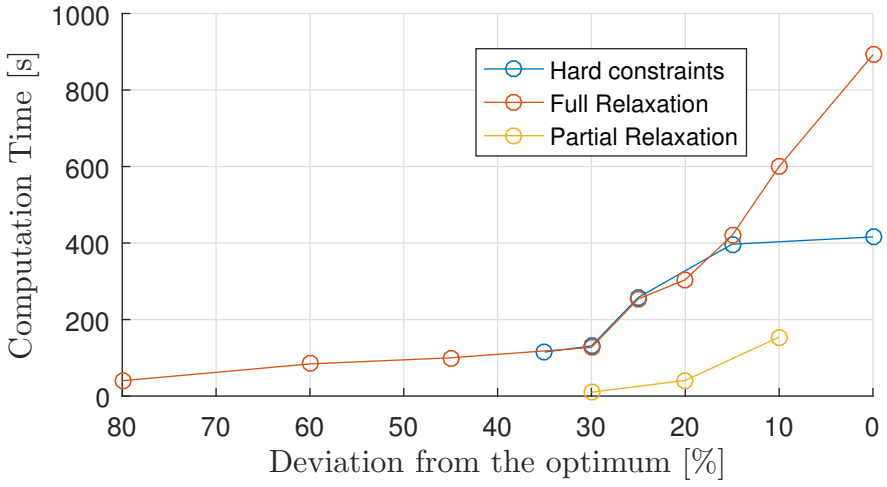


Figure 4.18: Comparison of performances for different constraints implementations using a multi-stage optimization.

However, the partial relaxation optimization always gets stuck in a local minimum, regardless of the parameters. Concerning the comparison between the full relaxation and the hard constraints, the performances are very close, except for the global optimum, which requires 225% more computational time without hard constraints. This observation is the opposite than the one made with GA. Consequently, one can state that MIDACO is a good algorithm to "jump" from different areas of the solution space, what was one of the main challenges of scheduling problem discussed in Section 2.3.5. It is then advised to use hard constraints rather than hard ones with MIDACO.

4.3.4 Multi-Objective Optimization

The final study concerns the multi-objective optimization proposed by MIDACO. The multi-objective optimization has been presented as a way to deal with problems that intend to minimize more than one feature, the time and the cost in the case, but also the starting time.

Working example

In Section 3.2.6, a way to represent the best compromise solutions was presented under the form of a *Pareto Front*. It represents the combinations of cost and time that cannot improve one objective function without downgrading the other one. In order to clearly illustrate the Pareto front, the same example is solved using an updated milestone and global due date of 50 TS (hard constraints). These values are intentionally larger in order to present more intermediary solutions:

Results

The simultaneous optimization of the global cost and duration leads to the Pareto front illustrated in Fig. 4.19. In addition to the Pareto front, its displays the edges of the set of feasible solutions, which can be defined as *the set of values that cannot improve the value of a fitness function while improving the other one*. At the extreme right of the front, it is possible to compact the schedule without deteriorating the cost. On the contrary, at the extreme left of the front, it is possible to reduce the total cost without deteriorating the total duration of the validation.

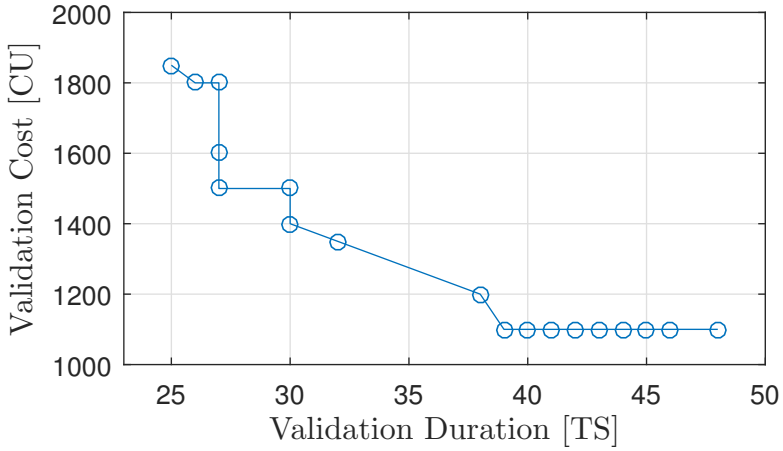


Figure 4.19: Pareto Front of the working example for the simultaneous minimization of the validation duration and cost (seed 500).

For the more compromising areas, the contradiction between the global duration and cost appears clearly after a certain level of optimization, 38 TS for this example. The user is able to choose the most convenient compromise for his application. In terms of computational cost, it is not higher than for a single-objective optimization of the cost.

5. Synthesis and Analysis

The first goal of this chapter is to synthesize the results observed in Chapter 4, following the three main subjects: the interface, the optimization using GA and the MIDACO study. The strengths and weaknesses of the two algorithms will be then analyzed in order to orientate the choice of the user in function of the encountered scheduling problem. After the choice of the most suitable algorithm for this application, some improvements will be proposed. These improvements will concern the implementation of the problem, in order to improve the computational performances. The next chapter will also propose improvements concerning the mathematical model, in order to bring it closer to the reality of validation.

5.1 Optimization Tool Interface - Key Aspects

The tool consists of a GUI, composed of 6 menus, helping the user to precisely define the relevant data for the composition of the validation tests. These menus were presented under the form of mockups, with a working example used to display the different requirements of the validation plan.

The user is able to create a new DVP or to complete an existing one, by:

- Selecting the requirements covered by a test;
- Selecting the facilities on which it can be performed;
- Defining the tests cases.

Then, the constraints applied on this test are expressed, followed by precedences and milestone. There is a specific menu dedicated to the facility definition where its unavailability periods are defined. Finally, there is menu dedicated to the display of the results of the scheduling tool, under the convenient form of a Gantt Chart.

5.2 Optimization Algorithms - Key Results

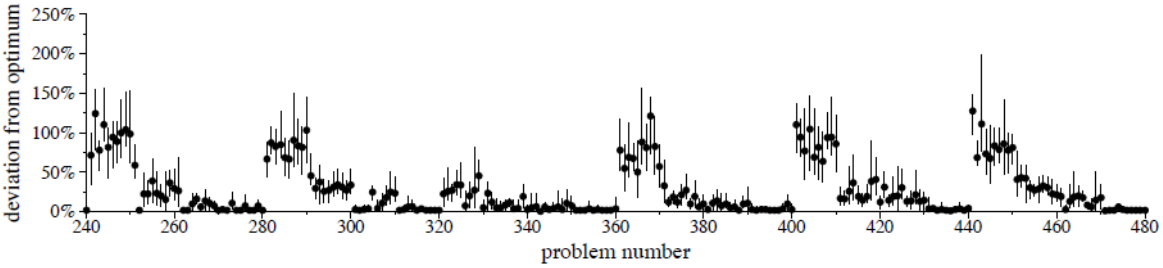
The scheduling problem was presented under the form of a mathematical problem to optimize, following to the global duration and/or cost. Two metaheuristic optimization algorithms have been tested, the *Genetic Algorithm*, based on the biological evolution of species, and the *Ant Colony Optimization* algorithm, based on the stigmergy between ants. Each method will be discussed individually since they have not always been able to perform in the same way for a given problem. Nevertheless, their key characteristics will be highlighted to discuss their ability to deal with scheduling problems. Finally, the most suitable method will emerge for the resolution of a complex validation scheduling.

5.2.1 GA - Key Results and Discussion

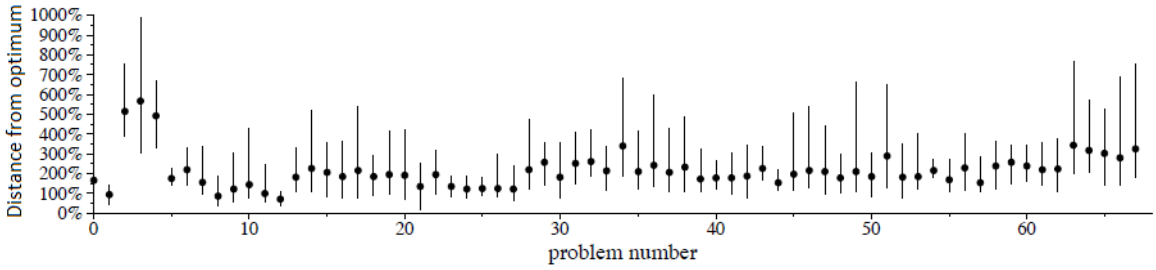
Three case were studied: the basic scheduling without facility allocation, facility allocation and adaptive constraints. These specific cases were supposed to put forward the matching between the `ga` solver and the scheduling optimization. The first case was used to determine the algorithm parameters ensuring the convergence and it led to results where the time or the cost was well minimized. However, the study revealed that GA was intrinsically slow, not particularly suited for combinatorial and non-linear problems. Additionally, it scaled badly in terms of number of tests, modes and resources. A negative impact of 35% in average was also observed on the computational cost by non-linear constraints.

A problem of reproducibility occurred, with different results for a same problem and same parameters. This is not suitable for dynamic variations (presented in Section 3.2.5) because only the concerned test can be rescheduled. Additionally, GA appeared too generalist, i.e., able to solve all kinds of problems, but not very powerful for MINLP (cfr. No Free Lunch Theorem). Concerning the relaxation of constraints, the bad global optimality of GA was a major concern and it revealed the need of a very fine tuning of parameters. However, with a good tuning, the convergence improved compared to the hard-constrained case. This last observation indicated that GA is a bad "jumper" (see Section 2.3.5) but it is indicated for problems including adaptive constraints. To conclude, the bad scalability, the slowness and the tuning of the parameters are the main challenges of GA, depending on the problem itself, but also on the way the constraints are defined.

These results can be discussed in comparison to the study made by Bartschi Wall (1996). In that study, GA has performed well for multi-modal scheduling, without hurting the performances. It is contradictory with the bad scaling observed in the facility allocation of this current work. It is likely due to a different modal definition. Nevertheless, Bartschi Wall has had similar problems for tightly-constrained resources, as shown in Fig. 5.1a. This observation has also been made in this work for constraints relaxation. In his JSS study (see Fig. 5.1b), he has put forward the bad parallel ability of GA, comparable to problems with a high level of precedences, such as this one. Bartschi Wall has also concluded that tuning the algorithm parameters can help solving the scheduling problem, but he expects less impact than this work did.



(a) Effect of constrained resources on 500 problems.



(b) Effect of precedences on 70 problems.

Figure 5.1: Performances of Genetic Algorithms (Bartschi Wall, 1996).

5.2.2 MIDACO - Key Results and Discussion

The MIDACO solver, implementing an ACO algorithm, was evaluated for several implementation scenarios. It went from a basic single-facility scenario to a complex multi-objective facility allocation scheduling optimization and adaptive constraints.

First, the scheduling problem, at constant degrees of freedom and non-linear constraints, scaled well, as expected by Schlueter & Munetomo (2016). This linear scalability bodes well for large scheduling optimization such as validation in the automotive industry. However, the performances were negatively impacted by a higher degree of freedom, represented by multiple facilities leading to additional modes.

Second, most of optimization algorithms are not suited for heavily-constrained problems. Nevertheless, MIDACO performed well for this case in this study thanks to a good sampling ability of the solution space. The counterpart of this characteristic was the need to be guided, via an optimization in cascade, if a high degree of optimality was required at a reasonable computational cost. It also concerned soft-constrained (non-convex) problems for which MIDACO was less powerful than for hard-constrained ones, because of its good ability to "jump", following the discussion in Section 2.3.5.

A powerful feature of ACO was the full reproducibility, at constant seeds, of its results, thanks to the principle of stigmergy. The stigmergy is also, following Dréo et al. (2011), what ensures a good compatibility between ACO and dynamic variations. In an analogy with the path illustrated in Fig. 2.3.3, a dynamic variation can be seen as an obstacle, that the algorithm will circumvent, without disturbing the rest of the path or schedule.

Then, MIDACO itself was convenient to help the user to set the wanted level of optimality, compared to the computational cost. It was also able to determine the global optimum, unlike GA, as expected by Schlueter et al. (2012). Additionally, it provided features whose goal was to ensure the convergence at a reasonable computational cost, such as the powerful multi-stage optimization. In return, this optimization in cascade was more subject to local optima convergence.

In conclusion, MIDACO has the powerful implementation of the choice between diversity and elitism. In the first case, it will lead to a better sampling of the solution space but a lower convergence. In the second case, it will lead to a better convergence but also to the tendency to get stuck in local minima. ACO being suited for scheduling activities, helped by the clever dynamic evolution of the number of ants and kernel by the MIDACO solver, it results in a satisfying optimality of the solutions in a short amount of time.

5.2.3 Choice of the Method

This study puts forward the excellent performances of MIDACO to solve a MINLP problem like this one, in terms of scalability, ease of implementation and optimality. In addition to the reproducibility of the results at constant parameters, the optimization in cascade makes possible to choose the degree of optimality and to decrease the computational cost of determining the global optimum. Then, the easy tuning of the parameters influencing the compromise between elitism and diversity is a strong asset of resolution. It leads to the choice between a fast convergence and a high optimality, which is represented by the deviation from the global optimum, depending on the required computational cost. Finally, the efficient discrete multi-objective optimization reveals to be particularly suitable for this application.

The last descriptions of MIDACO promotes the choice of this method for any scheduling activities. Actually, GA can be a more suitable scheduling optimization algorithm for two specific cases: slightly-constrained and multi-modal problems, which can be solved efficiently by GA, as well as problems for which the adaptive constraints are important and must be taken into account in the resolution.

5.3 Improvements

This section is dedicated to the computational improvements of the implemented method. It refers then more to mathematical concerns than project management issues. This last subject will be discussed in Chapter 6. This study begins with mathematical concerns that could lead to improvements and new propositions of algorithms to implement. Another way to consider the problem is proposed in order to increase the computational efficiency of the method. It finishes with other programs relevant for this application.

5.3.1 Non-Linear Constraints

A major drawback of MINLP is the non-linearity and non-convexity of the constraints. As shown in Fig. 4.10, the non-linearity results for `ga` in a significantly higher computational cost, and this result is also valid for most of optimization algorithms. Moreover, this phenomenon is supposed to increase with a greater number of unavailability periods and facilities.

In order to get rid of non-linear constraints, there exists specific methods to transform them in linear constraints, like a Taylor serie for example. However, this problem being discrete, this choice is impossible. The second way is to perform a first optimization with an exclusion variable equal to 1 (see Eq. 3.13 and 3.14) and a second one with an exclusion variable equal to 0. The best result is then kept.

The last proposition of non-linearity resolution is not fully satisfying for two reasons. First, it removes a degree of freedom since the exclusion variable can vary for one test to the other in a same optimization. Second, it requires to run the algorithm two times while the computational cost is already heavy.

5.3.2 Implementation Improvements

Two possible improvements on the code itself are here presented. The first one concerns the vectorization of the variables, while the second one intends to improve the computation of the initial solution.

Vectorization

`ga` has revealed not to be fully suitable for big scheduling applications. However, the *vectorization* of the algorithm could lead to significant improvements if a higher level of parallelization is provided. This principle of vectorization consists in providing a set of several variables in the fitness functions, constraints, etc. in order to test simultaneously several combinations of solutions. This method would be complementary to the parallel processing presented in Section 4.2.1.

Initial Solution

A difficulty for both algorithms has been to find a first feasible solution, even the worst one, in a small amount of time. The convergence ability of an algorithm has been one of the main concerns of

this work. If it is ensured more rapidly, the computational effort of the algorithms would be dedicated to the improvement of the solution.

A solution would be to automatically make a matching between the milestone periods of the tests and the resources availability. In addition to be relatively easy to implement, the computational time of this solution would be more acceptable than a stochastic combination. This solution is currently being implemented in a complementary work and the first results have demonstrated a linear relationship between the required computational time and the number of tests to schedule.

5.3.3 New Problem Definition

Another reason of the slowness of the optimization method (in both cases) has been the freedom it had, due to modes and multiple facilities per test. One can derive it as a general rule:

The price to pay for a high level of freedom in optimization is a high computational cost.

A solution would be to let the chosen algorithm making combinations of numbers between 1 and the number of tests. Each of these combinations would correspond to a schedule provided by a scheduler (working as a black box), as well as a ranking (the objective function) of this combination. The combinations would be easy to implement with very few constraints. The bottleneck would lie in the scheduler in this case, more efficient in terms of computational time but benefiting from less freedom of resolution, since the scheduler will follow a human-designed law.

5.3.4 Tailor-made Algorithm

MIDACO has been defined several times as powerful tool for MINLP. However, other algorithms, and B&B in particular (see Section 2.3.4), are efficient for discrete problems but require a tailor-made implementation. It would work as a *Brut Force* method, fully combinatorial, whose combinations would be only integers respecting the bounds. However, it would benefit from a significantly restricted solution space and a smarter definition of the combinations.

5.3.5 Other Optimization Programs

Several programs are suitable for optimization applications and are used in the industry. Among others, one can cite the famous *JuliaOpt*, *CPLEX* and *IPOPT* optimizer. They are presented in Appendix C.1.1.

6. Perspectives

Ford (2009) defines a model as a *substitute* for a real system. Therefore, the closer the model is to the reality of operations, the more efficient will be the resolution of the scheduling problem. This work has intended to provide a model, where tests were constrained under precedences, milestone, budget, due dates, and multiple facilities that could be allocated to them following other constraints. However, this model of scheduling can be extended to several implementations in order to bring it closer to the reality of operations in the validation process.

First, tests can be defined more precisely, following their priority and own objective(s). Then, the objectives of the global validation can vary from the cost/time minimization and the optimality can be expended to multiple projects. The resources can also be used with more freedom than only defined by modes and unavailability. The particular case of tests performed in parallel is studied and the principle of dynamic variations is extended to further applications. Finally, the impacts of these implementations is discussed in terms of computation.

6.1 Advanced Tests Definition

The presented scheduling process allocated the same importance to all scheduled tests. Additionally, the mode selection for each test was only geared towards the global optimization of a validation project. However, distinctions between the different tests can be made, on their priority and their objective.

Priority of Tests

The milestone method makes possible to define intermediary deadlines for specific tests. Beside this, it would be relevant to prioritize some tests, following the impact of their validation. Some requirements have a high importance of validation when a whole system is based on it, and a bad quality of validation could lead to a strongly negative impact on the global cost. The battery, for instance, is a key part of the electric package of a hybrid powertrain, for which the validation is particularly important for future developments.

Distinct Test Objective

Beside different priorities and global objectives, tests can be driven by their own objective. For example, depending on the need, a test can be driven by its cost, duration or quality of validation (see Section 6.2). It will lead to a different choice of mode or a different allocation of resources. At the same time, the global objective of the project is still valid.

6.2 Quality Assessment

As explained in Section 1.1, the total cost and/or duration of the validation process are not the only matters of interest. The *quality* of the validation itself can be an objective of the scheduling as well. The quality can be assessed following different forms:

- The quality in its very essence, i.e., the matching between the physical result and the model. In order to assess it, the user of the tool could define grades to a test in function of its compliance with the requirements, based on the quality of the resources allocated to it, the qualification of the workers performing the test and the test duration;

- The quality of the scheduling itself: the unfilled delays, the starting date compared to the desired one, etc.

The major difficulty in high quality objective is the assessment of the quality, which is dependent on the problem, and to translate it in a value to be measured.

6.3 Multi-Project Optimization

As observed in this work, the global optimum does not necessarily correspond to the optimum at the test level, due to constraints for example. This observation can be extended to different validation projects. Instead of optimizing the global duration and/or cost of a vehicle validation, the global optimum of several validation projects could be sought.

The problem definition would remain mainly unchanged, with as many cost/duration/facility matrices as projects to schedule. Nevertheless, the fitness functions must be adapted to represent the importance allocated. The following possibilities are considered:

- If the different validation projects are not all of equal importance, a weighted sum of the objective function of each project would relate the relative importance;

- If a multi-objective optimization is available, in addition to a global optimization, each project could still have its own objective function (time/cost minimization), independently to the other projects;

This list is not exhaustive and multiple combinations of these objective functions is also possible, requiring a multi-objective optimization solver.

6.4 Advanced Resources Definition

Facilities have only been defined by unavailability periods and different modes, leading to different cost/duration values, at the test level. However, this description is incomplete because the facilities are also described by resources, build-up procedures, mobile resources and facility networks. This section presents additional characteristics and suggests how to take them into account.

6.4.1 Advanced Resources Allocation

In Chapter 3, the cost/time discrete values have been determined directly at the facility level, leading to discrete modes. The level of freedom can be drastically increased, by allocating the resources (human, financial, etc.) via the tool itself, knowing their availability and adequacy. The problem would become a full *Joint Scheduling/Assignment Problem* (Bartschi Wall, 1996), where the schedule

of the activities must be determined, as well as a complete assignment of resources to them. Job-Shop Scheduling, Advanced Portfolio Management or Project Scheduling with Assignment are well-known problems of this kind.

Implementation

The implementation presented here stands for the human resources. The core competencies of each worker can be listed in an *Worker* array, W, with the concerned facilities and a percentage describing the compliance between the worker and the concerned facility, as shown in Table 6.1.

$W =$

Worker	Facility nr.	Compliance [%]
$Worker_1$	1	80
	3	50
$Worker_2$	4	20
	1	70

Table 6.1: The level of compliance between workers and facilities are stored in the W array.

In this example, the second worker is 70% compliant with the first facility. The algorithm would have to select the worker as done previously with modes, with consequences on the quality of the validation, but also the duration and the cost, depending on the compliance of the different workers.

6.4.2 Setup and Uninstallation Procedures

One assumed from the beginning that a facility was available to perform a validation test as soon as the starting time in the schedule mentioned it. In practice, the facility must be prepared via a *setup* procedure, in order to receive the concerned module to validate. After the validation is performed, this module must be uninstalled.

The setup and the uninstallation are both characterized by a duration and a cost, due to the resources requisition. However, this additional cost and duration are only valid if the facility must be prepared, i.e., it has not been prepared in the past without uninstalling. It could lead to unusual situations where a better facility with respect to the objective is not selected, because a less suitable one already benefits from the setup.

This feature can be implemented in an array presented like the unavailability array in Table 3.1, the *Setup Array S*. A two-facility example is given in Table 6.2. In this example, each facility is described by two setup modes but they are conditioned to the need of setup. There is no setup needed if it is still available.

$S =$

Facilities	Costs	
	Setup Cost [CU]	Uninstallation Cost [CU]
$Facility_1$	200	150
	250	120
$Facility_2$	230	450
	700	480

Table 6.2: The setup and uninstallation costs are stored in the S array.

6.4.3 Mobile Resources

In addition to fixed facilities, there are additional resources (human, energetic or facilities) required to perform a whole test or some parts of it. When multiple facilities are suitable for one test, some of them can be self-sufficient (but generally more expensive), while others (generally cheaper) can be used until a certain point and will require an additional resource after then.

This problem can be illustrated by chassis dynamometers for a 125-hp vehicle. A chassis dynamometer is supported by a fan. However, certain chassis dynamometers have a fan suitable until 100 hp, while others, more expensive, are suitable for higher-power test. The power curve and the mobile resource problem is illustrated in Fig. 6.1.

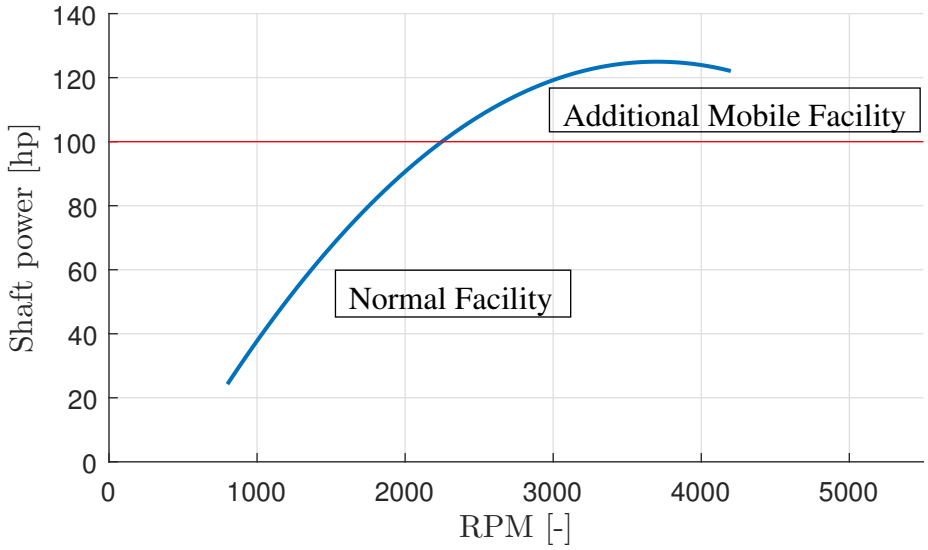


Figure 6.1: Mobile facilities are used for more demanding parts of tests.

The major part of a powertrain validation is in the range used during the NEDC or WLTP cycle, not necessitating any other mobile resources. However, the powertrain must also be validated at some point at WOT, necessitating a 125-hp-compatible fan. A mobile resource is also linked to setup and uninstallation procedures. The decision to allocate a bigger facility or a smaller one with the need of an additional resource must be taken into account when allocating a facility to a test.

Implementation

The choice of taking a mobile resource, or not, must be seen as the choice of a mode by the algorithm. Therefore, the choice is between a self-sufficient facility or a smaller one combined with a defined additional resource. This imposes an adaptation of the F matrix, which becomes now a F array, as shown in the example in Table 6.3. Besides the facility, it displays the needed mobile resource(s) for each test.

$F =$

Facilities	**Mobile Resource**	**Facilities**	**Mobile Resource**
$Facility_1$	[1 2]	$Facility_4$	/
$Facility_2$	[1]	$Facility_3$	[5 8]

Table 6.3: The mobile resources appear in the Facility array F.

In this particular example, the Test 1 can either deal with the first facility, with the support of the first or the second mobile resource, or the fourth facility without any mobile resources. Additional U and S arrays can be created, dedicated to the mobile resources definition.

6.4.4 Facility Network

In this work, one assumed that each test was considered individually, performed on only one facility, with precedences to respect. Nevertheless, a strong interest lies in the parallel testing of several components organized in network, leading to the organization of the facilities in network. HiL has demonstrated a strong interest for this kind of configuration.

As shown in Fig. 6.2, the global validation of a hybrid powertrain is a typical example of facilities necessitating to be used in parallel. In order to do so, the battery pack is connected to the e-motor, which is also connected to an axle. Simultaneously, the ICE is connected to the electric powertrain via the hybrid transmission. Finally, the global network is operated by a virtual drive simulator, simulating driving cycles.

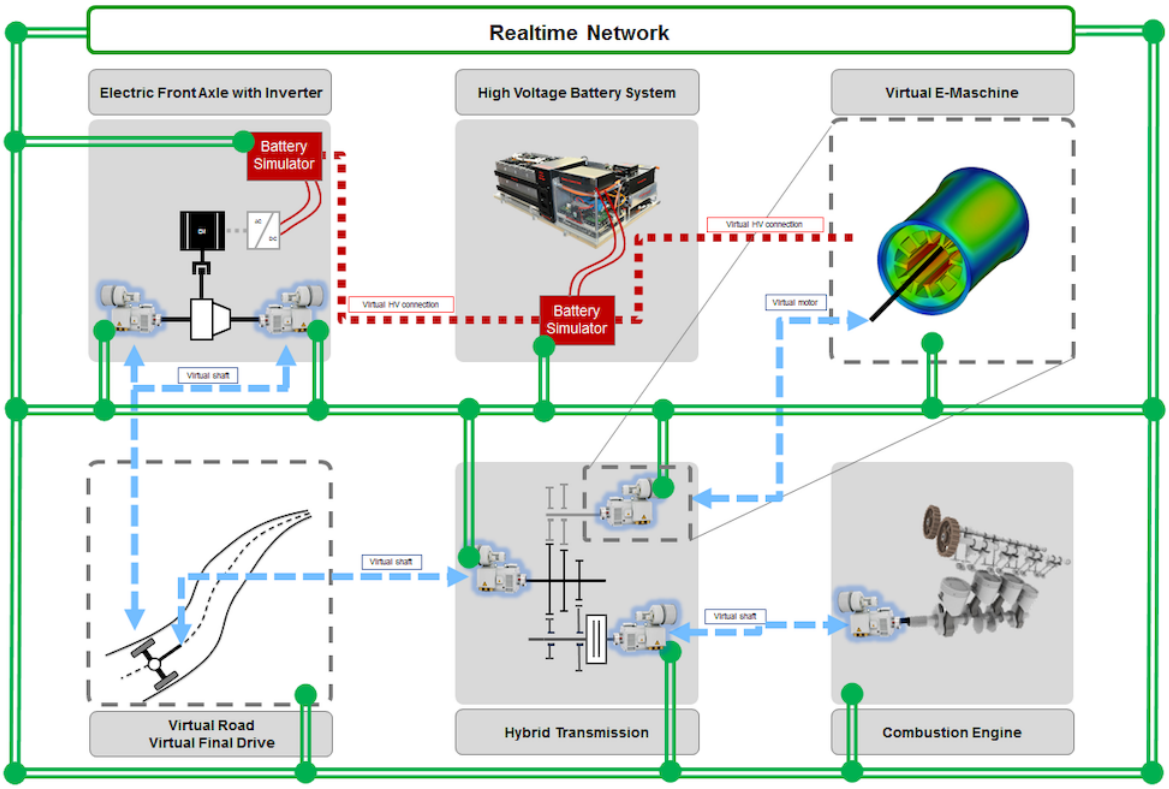


Figure 6.2: Network of Facilities, *Center for Mobile Propulsion*, RWTH Aachen.

To implement networks of facilities, the simultaneity of testing must be ensured. Therefore, two possibilities exist: either the simultaneity is considered as a single test with several facilities, or multiple tests with the same starting and ending date. The second choice is the most adapted to the current implementation, with a *Parallel Matrix*, defined exactly as the precedences matrix defined in Eq. 3.2.

6.5 Dynamic Variations

Dynamic variations are the consequences of the feedbacks resulting from the application of the validation tests in practice. A basic implementation has already been proposed in Section 3.2.5 but this section intends to extend the application of dynamic variations.

6.5.1 Automatic Critical Path Determination

The critical path represents the set of linked activities for which the total duration is the longest. Its length determines then the total duration of the validation process. The rescheduling of the tests situated on this path will certainly have a negative impact on the respect of the due dates, as illustrated in Fig. 2.5 and 2.6.

Therefore, an automatic critical path determination could preventfrom planning tests which present a risk of validation failure. These tests should be mentioned at the beginning of the optimization process in order to constrain their scheduling on the critical path. Additionally, *buffers* could be added in order to ensure the respect of the due date.

Buffers

The second and complementary proposed way to deal with uncertainties due to dynamic variations is the addition of buffers, as shown in Fig. 2.6. The *Project Buffer* is the easiest one to implement since the global due date only has to be adapted to respect it, and it is assumed that it will protect the critical path to exceed its limit.

The *feeding buffers* are more delicate to implement because they must ensure the respect of the temporary due dates while being small enough not to slip on the critical chain. Therefore, the algorithm should determine by itself the biggest buffers possible on auxiliary chains without exceeding the duration of the the critical path.

6.6 Discussion

The extensions presented in this chapter have the advantage to bring the scheduling model closer to the reality of the validation process. However, there are two possible drawbacks. First, the improvements will lead to tighter constraints, as mobile resources (with their own unavailability constraints), facility networks or dynamic variations do. Second, it will create a significant number of additional degrees of freedom. The algorithm will have more options to deal with, particularly for an advanced resources allocation of workers.

These two difficulties highlight, again, the need of a tool able to deal with multiple expectations. MIDACO revealed to be efficient for tightly constrained and big-scaled problems, so it is expected to be suitable for additional constraints linked to facility networks, for example. Nevertheless, additional modes have a negative impact on its performances, so mobile resources and advanced resources definition will require a significant amount of time to solve the optimization problem taking them into account.

Conclusion

The increasing complexity and number of norms, in addition to the shortening of the development time of modern vehicles leads to the need of optimizing the validation planning. Simultaneously, the automotive industry must guarantee a high level of quality. This work intended to provide a working basis for a tool dedicated to the optimization of the validation process in this specific sector. It addressed the validation following two main subjects: the graphic interface, where the validation of a vehicle is defined, and the scheduling optimization algorithm supporting the tool.

First, the problem was characterized as a scheduling application, where *validation tests* had to be optimally scheduled, in order to minimize the cost and/or duration of the global process. These tests were characterized by facilities on which they can be performed, requirements they were supposed to fulfill, modes, precedences and milestone periods. Additionally, the facilities were defined with unavailability periods. Finally, dynamic variations were defined as a way to give feedbacks from the tests, in order to provide the most optimal and consistent schedule with the reality of operations.

Then, a literature review was performed in order to study the concepts behind the validation and project management in general. It helped to characterize the problem to address as a *Mixed Integer Non-Linear Programming* (MINLP) problem and to create a model of it. The state-of-the art of optimization algorithms for complex problems was also reviewed and metaheuristics appeared to be the most suitable kind of methods for solving this problem. In particular, *Genetic Algorithms* (GA) and *Ant Colony Optimization* (ACO) were described as suitable for scheduling applications like the one of this work.

The objectives of the interface were defined following the needs of the validation process studied during the literature review. A mathematical model of the scheduling was elaborated and the methodology used to solve it was presented under the form of an objective function to minimize and constraints to respect. Two solvers emerged from the state-of-the-art, `ga` (Matlab tool) and MIDACO, implementing, respectively, a genetic and an ant colony algorithm suitable for discrete and non-linear problems.

The objectives of the interface resulted in mockups of the different menus directing the user all along the validation definition. A real GUI was even implemented, displaying the different requirements of a modern vehicle. Then, a complete study of the optimization algorithms was performed in order to highlight the suitability of GA and ACO using working examples, from a basic case, with only one facility per test, to a complex multi-modal example with allocation of facilities and adaptive constraints (also called relaxed constraints).

From both algorithms, MIDACO revealed to be the most efficient one. Its scalability, its ability to converge to a feasible solution at a small computational cost, its multi-stage optimization and its optimality make MIDACO very suitable for this specific problem. However, GA appeared to be able to provide a working solution for multiple kinds of problems, including MINLP, but it was generally not the most efficient one. Additionally, MIDACO revealed to be a good "jumper" in the solution

space, what makes it more suitable for hard-constrained problems, unlike GA that performed better with constraint relaxation.

The problem addressed in this work can be extended much further. The constant increasing complexity of development in the automotive sector has been the frame of this work. However, the society is only at the very beginning of a real mobility revolution, defined by three main domains of development:

- The *electrification of the powertrain* will accelerate and the disappearance of (pure) ICEs within the next two decades is ensured in the Western World (Korda, 2017);

- *Autonomous driving* will spread over all OEMs and categories of vehicle. This revolution is an opportunity for the automotive sector but it is also a great challenge in terms of development and norms, and thus in terms of validation requirements;

- The *Internet of Things* (IoT) is conquering the transportation industry with connected platforms, intelligent maintenance, tailor-made services, communication between vehicles, etc. This is synonym of additional developments, as well as additional embedded systems, and thus synonym of more validation and verification tests to perform.

This revolution will lead to more complex and bigger mathematical models to solve, with more and more degrees of freedom in the resolution. At what computational cost will it be possible to ensure a satisfying level of flexibility in the scheduling ? Will the evolution of optimization algorithms follow these challenging trends in the automotive industry?

To conclude, in a near future, a major part of the validation process will be done directly by the client. How can the validation, and particularly the efficient scheduling of its tests, guarantee the quality of this after-sales validation? What methods can be used in order to include this way of validating in a tool like the one of this work? How to model this kind of validation?

A. Literature Review and State of the art - Appendix

A.1 Theory of validation in Systems Engineering

A.1.1 Prototype phases in the automotive industry

Daimler AG provides information about the implementation of softwares in the vehicle industry (Houdek, 2013), leading to more complex systems as defined earlier.

The prototyping of vehicles (hybrid or not) in the automotive industry goes through different phases, depending on the development state of progress and functionality of items within the system. These phases are defined hereunder and illustrated in Fig. A.1:

- The *A-sample phase* or *Specification phase* is the limited-functionality prototype. It is still almost impossible to drive it;
- The *B-sample phase* is a prototype, fully drivable and highly mature;
- The *C-sample phase* is not a prototype anymore but a sample produced in small series. It is now fully functional;
- The *D-sample phase* is the same product as the C-sample, but provided to OEM's for design purposes.

Those notions will be used during the optimization of the validation plan.

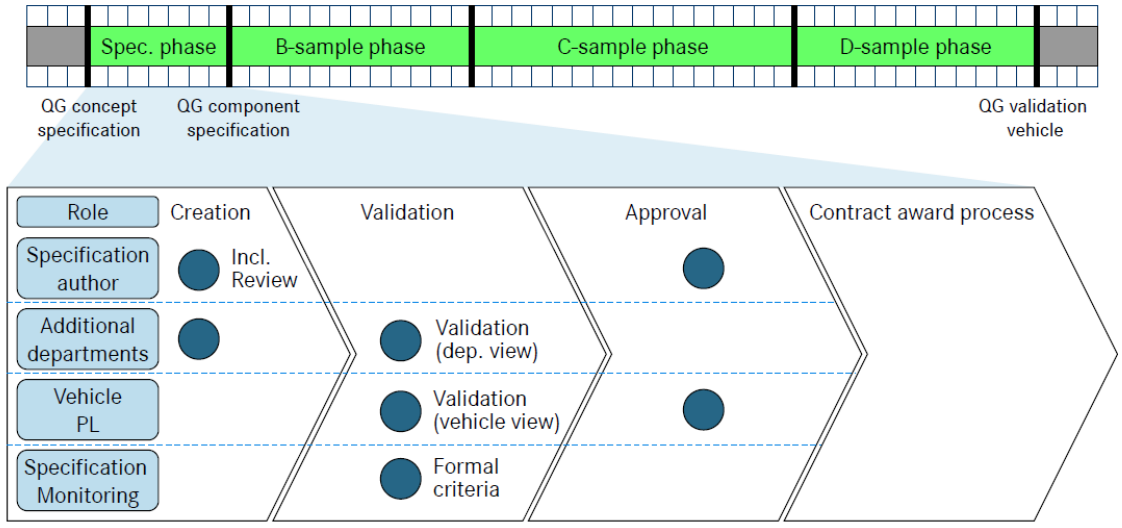


Figure A.1: Simplified excerpt of the Daimler Passenger Cars development process (Houdek, 2013).

A.1.2 Quality Management system

Validation and Verification processes are key aspects of a global *Quality Management System* (QMS). The *International Organization for Standardization* (ISO) is the international reference for *Quality Management* (QM). it defines *Quality* as:

"The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs". Clients can thus consider that a product has a high quality if it complies with their requirements.

In the ISO perspective, QM is divided into seven principles, *Quality Management Principles* (QMPs):

1. Customer focus
2. Leadership
3. Engagement of people
4. Process approach
5. Improvement
6. Evidence-based decision making
7. Relationship management

The best-known QMS norm is the ISO 9000 serie. This serie is currently composed of 3 norms: ISO 9000:2015 (essential principles), ISO 9001:2015 (requirements) and ISO 9004:2009. It consists in twelve steps, going from the market study to the recycling of the systems (in case of SE). The sixth step concerns the validation process.

A.1.3 Relevant norms for Validation

Following the ISO institution, a norm consists in "documents that provide requirements, specifications, guidelines or characteristics that can be used consistently to ensure that materials, products, processes and services are fit for their purpose". A previous thesis (Anand, 2016) made a complete research about relevant norms for the validation process in the specific automotive industry. They are listed hereunder and should be taken into account at the moment to define tests and test cases for the validation process.

ISO 15288

The ISO 15288 norm is called "Systems and software engineering - System life cycle processes" and provides standards for all life cycles of systems, including vehicles following the generic definition of SE. It provides four types of processes (technical, project, agreement and enterprise) and thus development processes, to apply at any level in the hierarchy of a system's structure.

ISO 26262

It is a "Road vehicles - Functional safety" norm. It is related to the safety of electrical and/or electronic systems of vehicle. This must be taken into account at Concept phase, Product development phase (for both Hardware level and Software level), and Integration and Testing phase. All these phases are concerned by the validation.

EAST ADL

EAST ADL stands for *Embedded Automotive Systems Architecture Description Language*. It is an extension of the partnership *AUTomotive Open System ARchitecture* (AUTOSAR) used to standardize the software architecture for automotive *Electronic Control Units* (ECU). The process of description of the architecture is divided into 4 phases namely: Vehicle level, Analysis level, Design level and Implementation level.

A.2 Scheduling Optimization Methods

A.2.1 Metaheuristic Methods

In the main text, the following metaheuristic methods have been presented:

- Genetic Algorithms;
- Particulate Swarm Optimization;
- Ant Colony Optimization;
- Tabu Search;
- Simulated Annealing;
- Branch & Bound;

Among those, Marini & Walczak (2015) propose a basic PSO algorithm displayed in Fig. A.2.

1. Initialization. For each of the N particles:
   a. Initialize the position $\mathbf{x}_i(0)$ $\forall i \in 1{:}N$
   b. Initialize the particle's best position to its initial position $\mathbf{p}_i(0)=\mathbf{x}_i(0)$
   c. Calculate the fitness of each particle and if $f(\mathbf{x}_j(0)) \geq f(\mathbf{x}_i(0))$ $\forall i \neq j$ initialize the global best as $\mathbf{g}=\mathbf{x}_j(0)$
2. Until a stopping criterion is met, repeat the following steps:
   a. Update the particle velocity according to equation (5):
      $$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + c_1\left(\mathbf{p}_i - \mathbf{x}_i(t)\right)\mathbf{R}_1 + c_2\left(\mathbf{g} - \mathbf{x}_i(t)\right)\mathbf{R}_2$$
   b. Update the particle position according to equation (4):
      $$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$$
   c. Evaluate the fitness of the particle $f(\mathbf{x}_i(t+1))$.
   d. If $f(\mathbf{x}_i(t+1)) \geq f(\mathbf{p}_i)$, update personal best: $\mathbf{p}_i=\mathbf{x}_i(t+1)$
   e. If $f(\mathbf{x}_i(t+1)) \geq f(\mathbf{g})$, update global best: $\mathbf{g}=\mathbf{x}_i(t+1)$
3. At the end of the iterative process, the best solution is represented by $\mathbf{g}$.

Figure A.2: Basic PSO algorithm (Marini & Walczak, 2015).

B. Results - Appendix

B.1 Optimization Tool Interface

B.1.1 Interface Example

An example of a GUI is given in the `fcn_gui.m` file. It illustrates how to represent the requirements of a vehicle to validate in an user-friendly manner.

First, run the `fcn_gui.m` function without any inputs. The window shown in Fig. B.1 appears and proposes you to select a given file of requirements.

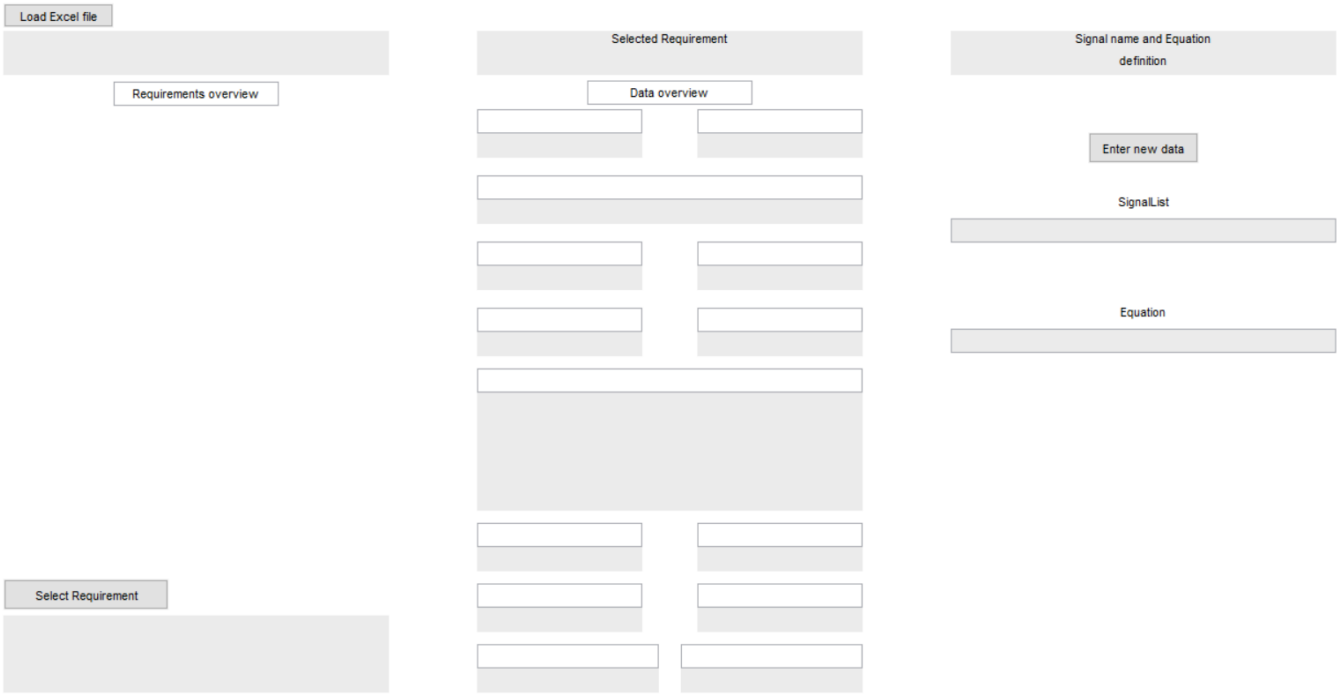


Figure B.1: The menu that appears when the GUI is run.

The set of requirements, defined with Instep, is loaded via the button *Load Excel file*. Then, the user can double-click on each folder and sub-folders to display all the requirements contained in the different categories. Once the category is chosen, the user double-clicks on the desired number of the type Req_i and then clicks on the *Select Requirement* button. This procedure is explained in Fig. B.2

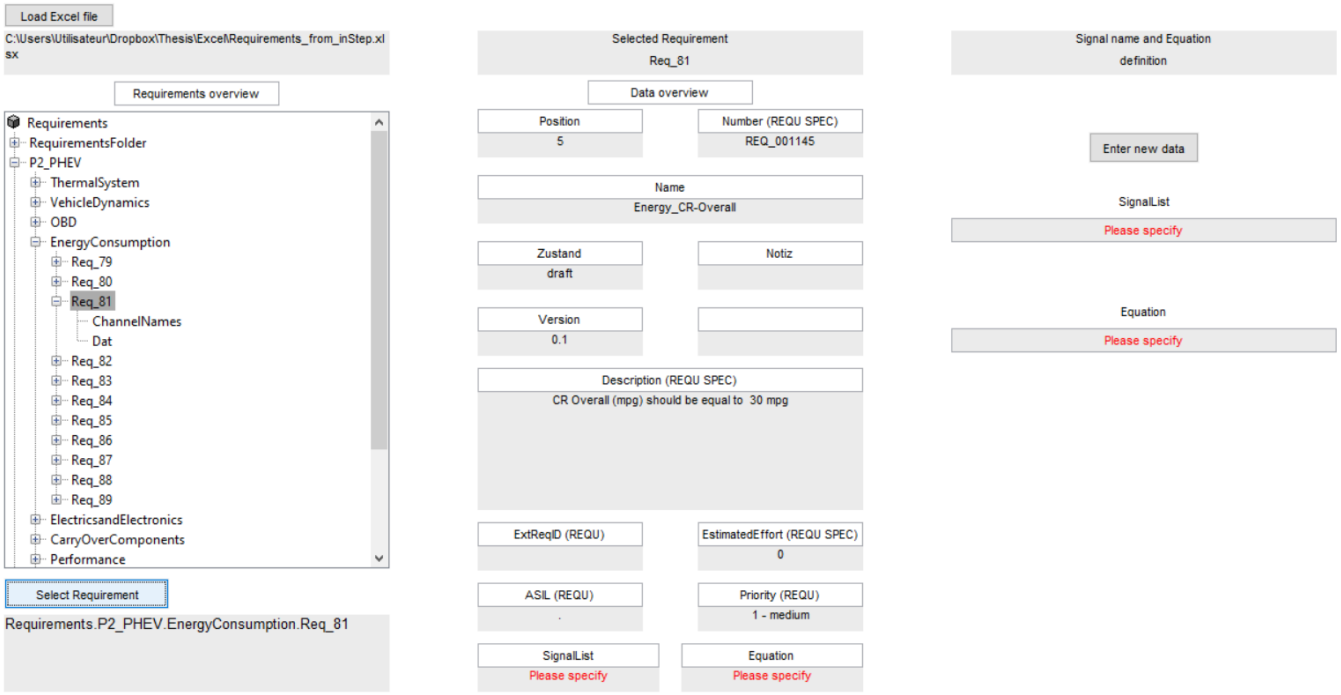


Figure B.2: The information of the chosen requirement to validate can be displayed.

B.2 Scheduling optimization by GA

B.2.1 Scheduling With Facility Allocation

The next examples are used to illustrate the duality between cost and duration minimization for same conditions, considering multiple facilities. In the first case, the global duration will be the highest possible, 29 TS, while the second one will perform the validation is less time but at 35% higher cost.

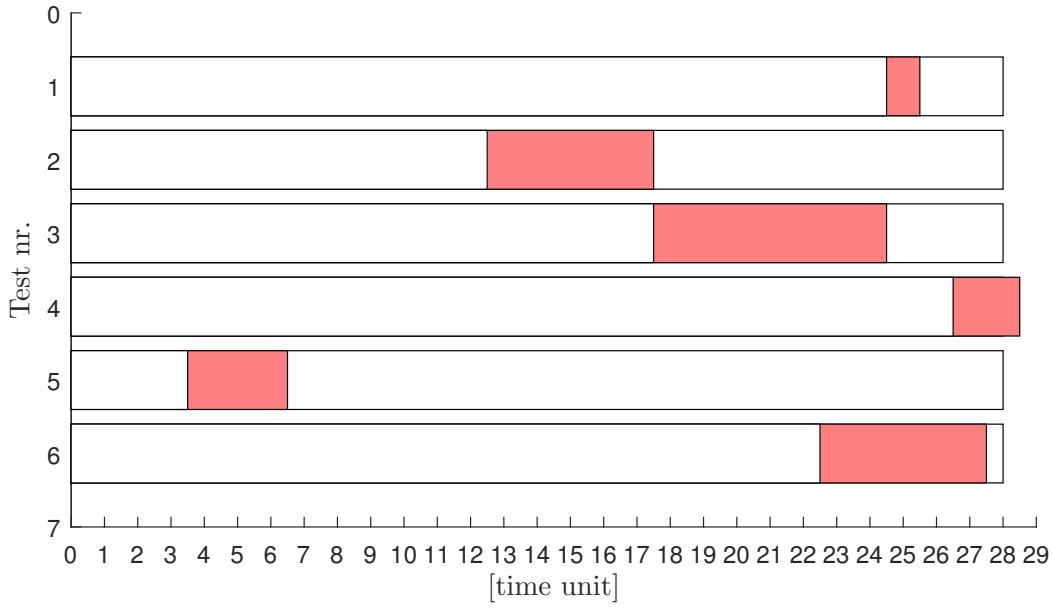


Figure B.3: Scheduling problem with facility allocation using `ga` (min cost): cost minimization, 1250 CU in 28 TS.

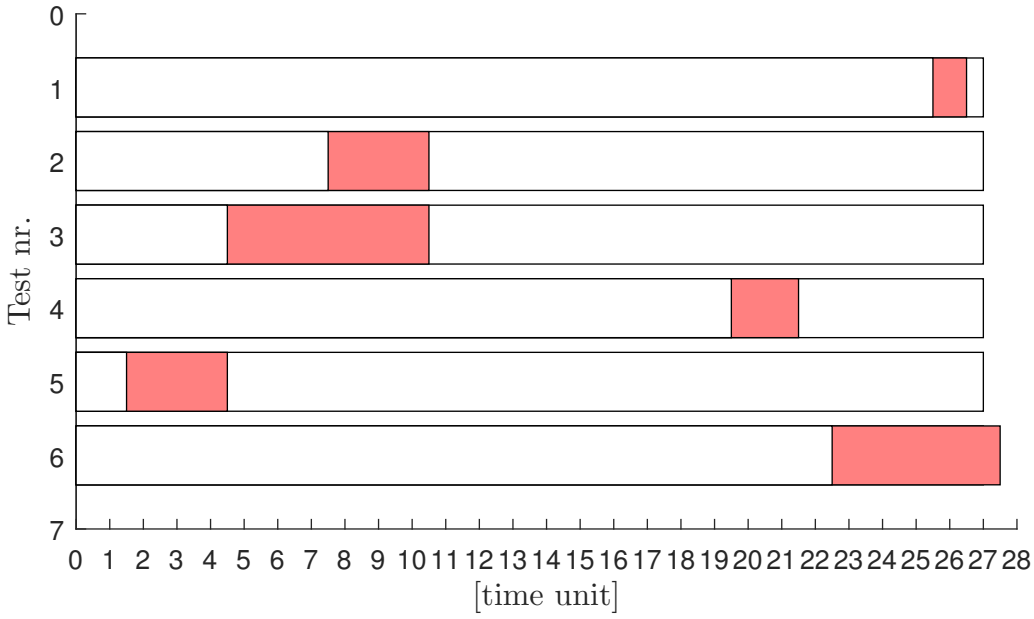


Figure B.4: Scheduling problem with facility allocation using `ga` (min time): time minimization, 1650 € in 27 time slots.

B.3 Scheduling optimization by MIDACO

B.3.1 Adaptive Constraints

The multi-stage optimization is used to narrow the search space progressively during the optimization process. This is particularly suitable for relaxed constraints, synonym of a wider solution space. Table B.1 and B.2 displays the parameters used all along the process.

Focus [-]	Seed [-]	Validation Cost [CU]	Computational Time [s]
0	0	1300	10
100000	0	1200	41
100000	0	1100	154

Table B.1: MIDACO multi-stage optimization for partial constraints relaxation.

Focus [-]	Seed [-]	Validation Cost [CU]	Computational Time [s]
0	50	1800	40
1000	500	1600	84
100000	500	1450	100
100000	500	1300	127
100000	500	1250	254
100000	50	1200	304
150000	50	1150	420
150000	50	1100	600
150000	50	1000	892

Table B.2: MIDACO multi-stage optimization for full constraints relaxation.

C. Synthesis and Analysis - Appendix

C.1 Improvements

C.1.1 Other Optimization Programs

Several programs are suitable for optimization applications and are used in the industry. Among others, one can cite the famous *JuliaOpt*, *CPLEX* and *IPOPT* optimizer

JuliaOpt

Written in *Julia*, it contains several optimization packages, like the *Global Optimization Library* of Matlab. However, it contains more specific solvers, that are able to solve linear and/or non-linear, convex and/or non-convex and discrete and/or continuous problems. The advantage of these solvers is to make the must efficient use of the No Free Lunch Theorem, by using a dedicated solver instead of a generalist one, like `ga`, synonym of greater efficiency.

CPLEX

Developed by *IBM*, CPLEX is a flexible high-performance mathematical programming solver for linear programming, mixed integer programming, quadratic programming, and quadratically constrained programming problems, well known in all kinds of industries.

It is able to perform a large parallelization of the computing (even on several computers). It is also able to deal with millions of variables and constraints, what is a good asset for several projects, of a thousand of validation tests and modes. It contains lots of a parameters to tune and is recognized for its ability to quickly detect unfeasibilities.

IPOPT

Ipopt stands for *Interior Point OPTimizer* is a Julia package for large non-linear optimization, implemented in the same way as MIDACO does. Unfortunately, no publications on this software for MINLPs have been published.

Bibliography

[AJKK06] Marvin A. Arostegui Jr., Sukran N. Kadipasaoglu, and Basheer M. Khumawalab. An empirical comparison of tabu search, simulated annealing, and genetic algorithms for facilities location problems. *International Journal of Production Economics*, 103:742–754, 2006.

[And] E. S. Andersen. Milestone planning—a different planning approach. *PMI® Global Congress 2006—Asia Pacific, Bangkok, Thailand*.

[Bha13] Sanjiv K. Bhatia. Branch and bound | set 4 (job assignment problem). Pratical course, 2013.

[BP14] Alexander Bergmann and Micheal Paulweber. Smart Systems Enable Automotive Powertrain Developments. Vienna, March 2014.

[BW96] Matthew Bartschi Wall. *A Genetic Algorithm for Resource-Constrained Scheduling*. PhD thesis, Massachusetts Institute of Technology, 1996.

[CCZ01] Chang, Christensen, and Zhang. Genetic algorithms for project management. *Annals of Software Engineering*, 11:107–139, 2001.

[CN16] Gloria Cerasela and Elena Nechita. Solving fuzzy tsp with ant algorithms. 2016.

[Cor00] Prof. Jordi Cortadella. Tabu search method, 2000.

[Deb11] Kalyanmoy Deb. Multi-objective optimization using evolutionary algorithms: An introduction. *Department of Mechanical Engineering, Indian Institute of Technology Kanpur*, 2011. KanGAL Report Number 2011003.

[DHJ+10] Debbabi, Hassaïne, Jarraya, Soeanu, and Alawneh. *Verification and Validation in Systems Engineering*. Springer, 1 edition, 2010. ISBN: 978-3-642-15227-6.

[DPST11] Dréo, Pétrowski, Siarry, and Taillard. *Métaheuristique pour l'optimisation difficile*. Eyrolles, 3 edition, 2011. ISBN: 978-2-212-11368-6.

[Els14] Wesam Elshamy. animated simulation of particles searching for the minima of a simple function. Matlab Community, 2014.

[Fer03] Thomas S. Ferguson. *Linear Programming - A Concise Introduction*. Department of Mathematics, University of California at L.A., 2003.

[Flo95] Christodoulos A. Floudas. *Nonlinear and Mixed-Integer Optimization*. Oxford University Press, 1 edition, 1995. ISBN: 0-19-510056-5.

[For09] Andrew Ford. *Modeling the Environment*. Island Press, 2 edition, 2009. ISBN: 978-1597264730.

[fS15] International Organization for Standardization. *Quality management principles*. ISO, 2015. ISBN: 978-92-67-10650-2.

[GJS76] Michael R. Garey, David S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1:117–129, 1976.

[HD04] Hick Hannes and Klaus Denkmayr. Probabilistic safety assessment and management. chapter The Load Matrix — a method for optimising powertrain durability and reliability test programmes, pages 1627–1635. Springer, 2004. ISBN: 978-1-85233-827-5.

[Hou13] Dr. Frank Houdek. Managing large scale specification projects. In *19th international conference on requirement engineering: Foundation for Software Quality*. Daimler AG, 2013.

[Ite15] Pierre-François Itel. Gestion de projet: Résumé de cours niveau master 1, 2015.

[Kor17] Robin Korda. Automobile : comme hulot, ces pays rêvent de la fin du diesel et de l'essence. *Le Parisien*, 2017.

[Kum16] Anand Kumar. Electrified powertrain system design process. Master's thesis, Lehrstuhl für Verbrennungskraftmaschinen RWTH Aache University, Germany, 2016.

[Mir16] S. Mirjalili. Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete and multi-objective problems. *Neural Computing & Application*, 27:1053–1073, 2016.

[MMPH98] Maltby, Morello, Perret, and Hopkin. *Checklist for Preparing a Validation Plan*. Transport Telematics Support & Consensus, 1998.

[MPD83] Moder, Phillips, and Davis. *Project Management with CPM, PERT and Precedence Diagramming*. Van Nostrand Reinhold Company Inc., 3 edition, 1983. ISBN: 0-442-25415-6.

[MW15] F. Marini and B. Walczak. Particle swarm optimization (pso). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015.

[Nav04] E. Navet. When does the iso 9000 quality assurance standard lead to performance improvement? assimilation and going beyond. *IEEE Transactions on Engineering Management*, 51:352 – 363, 2004.

[PMI13] PMI. *A Guide to the Project Management Body of Knowledge*. Project Management Institute, Inc., 5 edition, 2013. ISBN: 978-1-935589-67-9.

[Reb05] Djamal Rebaïne. La méthode de branch and bound. Chapter 6, 2005.

[Rob12] Robert Robere. Interior point methods and linear programming. *University of Toronto*, 2012.

[SGR12] M. Schlueter, M. Gerdts, and Jan-J. Rückmann. A numerical study of midaco on 100 minlp benchmarks, 2012.

[SM16a] M. Schlueter and M. Munetomo. *MIDACO Solver - User Manual*. IIC, Hokkaido University, Japan, 2016.

[SM16b] M. Schlueter and M. Munetomo. Numerical assessment of the parallelization - scalability on 200 minlp benchmarks. *IEEE Congress on Evolutionary Computatio*, 149:830–837, 2016.

[ST12] Yiqiang Sheng and Atsushi Takahashi. *A Simulated Annealing Based Approach to Integrated Circuit Layout Design*. Marcos de Sales Guerra Tsuzuki, 2012. ISBN: 978-953-51-0767-5.

[Sva87] Krister Svanberg. The method of moving asymptotes - a new method for structural optimization. *International Journal for Numerical Methods in Engineering*, 24:359–373, 1987.

[Swa73] Swanson. Linear programming: an approach to critical path management. *Project Management Quarterly*, 4:14–21, 1973.

[Wei] Alan Weiss. Improving performance with parallel computing. Matlab documentation on Global Optimization Toolbox. Accessed: 2017-05-30.

[Wei09] Thomas Weise. Global optimization algorithms – theory and application. Faculty of Computer Science and Technology of the Hefei University, June 2009.

[Wei17] Thomas Weise. Why you should use evolutionary algorithms to solve your optimization problems (and why not). *Institute of Applied Optimization (IAO)*, 2017.

[WM97] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[XAT13] Jing Xiao, Xian-Ting Ao, and Yong Tang. Solving software project scheduling problems with ant colony optimization. *Computers Operations Research*, 40(1):33 – 46, 2013.